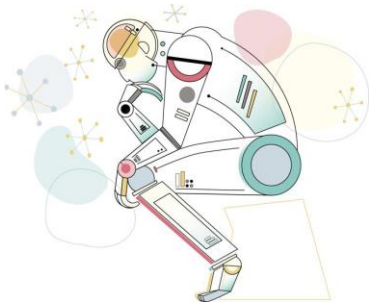
21 Hottest Research Papers On Computer Vision and Machine Learning



These papers provide a breadth of information about powerful convolutional neural networks that is generally useful and interesting from a Data science perspective.

Contents

11. A Deep Visual Correspondence Embedding Model for Stereo Matching Costs
12. Conditional Random Fields as Recurrent Neural Networks
13. Local Convolutional Features with Unsupervised Training for Image Retrieval
14. FlowNet: Learning Optical Flow with Convolutional Networks
15. Active Object Localization with Deep Reinforcement Learning
16. Deep Neural Decision Forests
17. Im2Calories: towards an automated mobile vision food diary
18. DeepBox: Learning Objectness with Convolutional Networks
19. Flowing ConvNets for Human Pose Estimation in Videos
20. Understanding deep features with computer-generated imagery
21. Visual Tracking with Fully Convolutional Networks

HD-CNN: Hierarchical Deep Convolutional Neural Network for Large Scale Visual Recognition

Zhicheng Yan[†], Hao Zhang[‡], Robinson Piramuthu[*], Vignesh Jagadeesh[*], Dennis DeCoste[*], Wei Di[*], Yizhou Yu[⋆]
[†]University of Illinois at Urbana-Champaign, [‡]Carnegie Mellon University
[*]eBay Research Lab, [⋆]The University of Hong Kong

Abstract

In image classification, visual separability between different object categories is highly uneven, and some categories are more difficult to distinguish than others. Such difficult categories demand more dedicated classifiers. However, existing deep convolutional neural networks (CNN) are trained as flat N-way classifiers, and few efforts have been made to leverage the hierarchical structure of categories. In this paper, we introduce hierarchical deep CNNs (HD-CNNs) by embedding deep CNNs into a category hierarchy. An HD-CNN separates easy classes using a coarse category classifier while distinguishing difficult classes using fine category classifiers. During HD-CNN training, component-wise pretraining is followed by global finetuning with a multinomial logistic loss regularized by a coarse category consistency term. In addition, conditional executions of fine category classifiers and layer parameter compression make HD-CNNs scalable for large-scale visual recognition. We achieve state-of-the-art results on both CIFAR100 and large-scale ImageNet 1000-class benchmark datasets. In our experiments, we build up three different HD-CNNs and they lower the top-1 error of the standard CNNs by 2.65%, 3.1% *and* 1.1%, *respectively.*

1. Introduction

Deep CNNs are well suited for large-scale learning based visual recognition tasks because of its highly scalable training algorithm, which only needs to cache a small chunk (mini-batch) of the potentially huge volume of training data during sequential scans (epochs). They have achieved increasingly better performance in recent years.

As datasets become bigger and the number of object categories becomes larger, one of the complications that come along is that visual separability between different object categories is highly uneven. Some categories are much harder to distinguish than others. Take the categories in CIFAR100 as an example. It is easy to tell an *Apple* from a *Bus*, but harder to tell an *Apple* from an *Orange*. In fact, both *Apples* and *Oranges* belong to the same coarse category *fruit and vegetables* while *Buses* belong to another coarse category *vehicles 1*, as defined within CIFAR100. Nonetheless, most deep CNN models nowadays are flat N-way classifiers, which share a set of fully connected layers. This makes us wonder whether such a flat structure is adequate for distinguishing all the difficult categories. A very natural and intuitive alternative organizes classifiers in a hierarchical manner according to the divide-and-conquer strategy. Although hierarchical classification has been proven effective for conventional linear classifiers [38, 8, 37, 22], few attempts have been made to exploit category hierarchies [3, 29] in deep CNN models.

Since deep CNN models are large models themselves, organizing them hierarchically imposes the following challenges. First, instead of a handcrafted category hierarchy, how can we learn such a category hierarchy from the training data itself so that cascaded inferences in a hierarchical classifier will not degrade the overall accuracy while dedicated fine category classifiers exist for hard-to-distinguish categories? Second, a hierarchical CNN classifier consists of multiple CNN models at different levels. How can we leverage the commonalities among these models and effectively train them all? Third, it would also be slower and more memory-consuming to run a hierarchical CNN classifier on a novel testing image. How can we alleviate such limitations?

In this paper, we propose a generic and principled hierarchical architecture, *Hierarchical Deep Convolutional Neural Network* (HD-CNN), that decomposes an image classification task into two steps. An HD-CNN first uses a coarse category CNN classifier to separate easy classes from one another. More challenging classes are routed downstream to fine category classifiers that focus on confusing classes. We adopt a module design principle and every HD-CNN is built upon a building block CNN. The building block can be chosen to be any of the currently top ranked single CNNs. Thus HD-CNNs can always benefit from the progress of single CNN design. An HD-CNN follows the coarse-to-fine

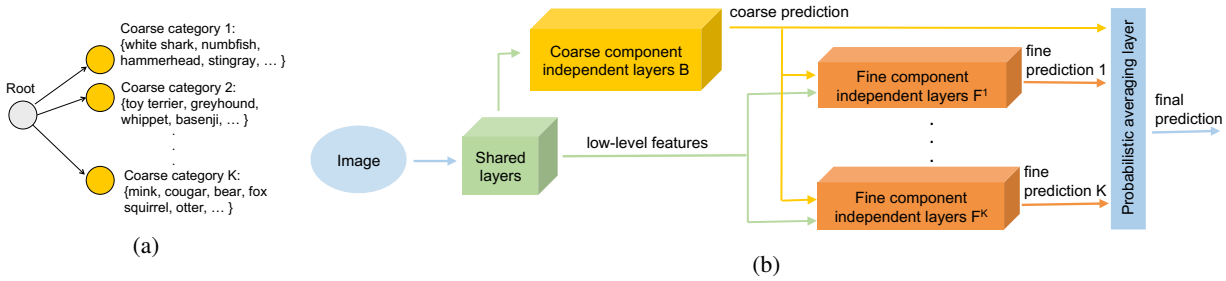


Figure 1: (a) A two-level category hierarchy where the classes are taken from ImageNet 1000-class dataset. (b) Hierarchical Deep Convolutional Neural Network (HD-CNN) architecture.

classification paradigm and probabilistically integrates predictions from the fine category classifiers. Compared with the building block CNN, the corresponding HD-CNN can achieve lower error at the cost of a manageable increase in memory footprint and classification time.

In summary, this paper has the following contributions. First, we introduce a novel hierarchical architecture, called HD-CNN, for image classification. Second, we develop a scheme for learning the two-level organization of coarse and fine categories, and demonstrate various components of an HD-CNN can be independently pretrained. The complete HD-CNN is further fine-tuned using a multinomial logistic loss regularized by a coarse category consistency term. Third, we make the HD-CNN scalable by compressing the layer parameters and conditionally executing the fine category classifiers. We have performed evaluations on the medium-scale CIFAR100 dataset and the large-scale ImageNet 1000-class dataset, and our method has achieved state-of-the-art performance on both of them.

2. Related Work

Our work is inspired by progresses in CNN design and efforts on integrating a category hierarchy with linear classifiers. The main novelty of our method is a new scalable HD-CNN architecture that integrates a category hierarchy with deep CNNs.

2.1. Convolutional Neural Networks

CNN-based models hold state-of-the-art performance in various computer vision tasks, including image classifcation [18], object detection [10, 13], and image parsing [7]. Recently, there has been considerable interest in enhancing CNN components, including pooling layers [36], activation units [11, 28], and nonlinear layers [21]. These enhancements either improve CNN training [36], or expand the network learning capacity. This work boosts CNN performance from an orthogonal angle and does not redesign a specific part within any existing CNN model. Instead, we design a novel generic hierarchical architecture that uses an existing CNN model as a building block. We embed multiple building blocks into a larger hierarchical deep CNN.

2.2. Category Hierarchy for Visual Recognition

In visual recognition, there is a vast literature exploiting category hierarchical structures [32]. For classification with a large number of classes using linear classifiers, a common strategy is to build a hierarchy or taxonomy of classifiers so that the number of classifiers evaluated given a testing image scales sub-linearly in the number of classes [2, 9]. The hierarchy can be either predefined[23, 33, 16] or learnt by top-down and bottom-up approaches [25, 12, 24, 20, 1, 6, 27]. In [5], the predefined category hierarchy of ImageNet dataset is utilized to achieve the trade-offs between classification accuracy and specificity. In [22], a hierarchical label tree is constructed to probabilistically combine predictions from leaf nodes. Such hierarchical classifier achieves significant speedup at the cost of certain accuracy loss.

One of the earliest attempts to introduce a category hierarchy in CNN-based methods is reported in [29] but their main goal is transferring knowledge between classes to improve the results for classes with insufficient training examples. In [3], various label relations are encoded in a hierarchy. Improved accuracy is achieved only when a subset of training images are relabeled with internal nodes in the hierarchical class tree. They are not able to improve the accuracy in the original setting where all training images are labeled with leaf nodes. In [34], a hierarchy of CNNs is introduced but they experimented with only two coarse categories mainly due to scalability constraints. HD-CNN exploits the category hierarchy in a novel way that we embed deep CNNs into the hierarchy in a scalable manner and achieves superior classification results over the standard CNN.

3. Overview of HD-CNN

3.1. Notations

The following notations are used below. A dataset consists of images $\{\mathbf{x}_i, y_i\}_i$. $\mathbf{x}_i$ and y_i denote the image data and label, respectively. There are C fine categories of images in the dataset $\{S_j^f\}_{j=1}^C$. We will learn a category hierarchy with K coarse categories $\{S_k^c\}_{k=1}^K$.

3.2. HD-CNN Architecture

HD-CNN is designed to mimic the structure of category hierarchy where fine categories are divided into coarse categories as in Fig 1(a). It performs end-to-end classification as illustrated in Fig 1(b). It mainly comprises four parts, namely shared layers, a single coarse category component B, multiple fine category components $\{F^k\}_{k=1}^K$ and a single probabilistic averaging layer. On the left side of Fig 1 (b) are the shared layers. They receive raw image pixel as input and extract low-level features. The configuration of shared layers are set to be the same as the preceding layers in the building block net.

On the top of Fig 1(b) are independent layers of coarse category component B which reuses the configuration of rear layers from the building block CNN and produces a fine prediction $\{B_{ij}^f\}_{j=1}^C$ for an image $\mathbf{x_i}$. To produce a prediction $\{B_{ik}\}_{k=1}^K$ over coarse categories, we append a fine-to-coarse aggregation layer which aggregates fine predictions into coarse ones when a mapping from fine categories to coarse ones $P : [1, C] \mapsto [1, K]$ is given. The coarse category probabilities serve two purposes. First, they are used as weights for combining the predictions made by fine category components. Second, when thresholded, they enable conditional executions of fine category components whose corresponding coarse probabilities are sufficiently large.

In the bottom-right of Fig 1 (b) are independent layers of a set of fine category classifiers $\{F^k\}_{k=1}^K$, each of which makes fine category predictions. As each fine component only excels in classifying a small set of categories, they produce a fine prediction over a partial set of categories. The probabilities of other fine categories absent in the partial set are implicitly set to zero. The layer configurations are mostly copied from the building block CNN except that in the final classification layer the number of filters is set to be the size of partial set instead of the full categories.

Both coarse category component and fine category components share common layers. The reason is three-fold. First, it is shown in [35] that preceding layers in deep networks response to class-agnostic low-level features such as corners and edges, while rear layers extract more class-specific features such as dog face and bird's legs. Since low-level features are useful for both coarse and fine classification tasks, we allow the preceding layers to be shared by both coarse and fine components. Second, it reduces both the total floating point operations and the memory footprint of network execution. Both are of practical significance to deploy HD-CNN in real applications. Last but not the least, it can decrease the number of HD-CNN parameters which is critical to the success of HD-CNN training.

On the right side of Fig 1 (b) is the probabilistic averaging layer which receives fine category predictions as well as coarse category prediction and produces a weighted average as the final prediction.

$$p(\mathbf{x}_i) = \frac{\sum_{k=1}^K B_{ik} p_k(\mathbf{x}_i)}{\sum_{k=1}^K B_{ik}} \tag{1}$$

where B_{ik} is the probability of coarse category k for the image $\mathbf{x}_i$ predicted by the coarse category component B. $p_k(\mathbf{x}_i)$ is the fine category prediction made by the fine category component F^k.

We stress that both coarse and fine category components reuse the layer configurations from the building block CNN. This flexible modular design allows us to choose the best module CNN as the building block, depending on the task at hand.

4. Learning a Category Hierarchy

Our goal of building a category hierarchy is grouping confusing fine categories into the same coarse category for which a dedicated fine category classifier will be trained. We employ a top-down approach to learn the hierarchy from the training data.

We randomly sample a held-out set of images with balanced class distribution from the training set. The rest of the training set is used to train a building block net. We obtain a confusion matrix $\mathbf{F}$ by evaluating the net on the held-out set. A distance matrix $\mathbf{D}$ is derived as $\mathbf{D} = 1 - \mathbf{F}$ and its diagonal entries are set to be zero. $\mathbf{D}$ is further transformed by $\mathbf{D} = 0.5 * (\mathbf{D} + \mathbf{D}^T)$ to be symmetric. The entry $\mathbf{D}_{ij}$ measures how easy it is to discriminate categories i and j. Spectral clustering is performed on $\mathbf{D}$ to cluster fine categories into K coarse categories. The result is a two-level category hierarchy representing a many-to-one mapping $P^d : [1, C] \mapsto [1, K]$ from fine to coarse categories. Here, the coarse categories are disjoint.

Overlapping Coarse Categories With disjoint coarse categories, the overall classification depends heavily on the coarse category classifier. If an image is routed to an incorrect fine category classifier, then the mistake can not be corrected as the probability of ground truth label is implicitly set to zero there. Removing the separability constraint between coarse categories can make the HD-CNN less dependent on the coarse category classifier.

Therefore, we add more fine categories to the coarse categories. For a certain fine classifier F^k, we prefer to add those fine categories $\{j\}$ that are likely to be misclassfied into the coarse category k. Therefore, we estimate the likelihood $u^k(j)$ that an image in fine category j is misclassified into a coarse category k on the held-out set.

$$u^k(j) = \frac{1}{\left|S_j^f\right|} \sum_{i \in S_j^f} B_{ik}^d \tag{2}$$

B_{ik}^d is the coarse category probability which is obtained by aggregating fine category probabilities $\{B_{ij}^f\}_j$ according to

the mapping P^d: $B^d_{ik} = \sum_{j|P^d(j)=k} B^f_{ij}$. We threshold the likelihood $u^k(j)$ using a parametric variable $u_t = (\gamma K)^{-1}$ and add to the partial set S^c_k all fine categories $\{j\}$ such that $u^k(j) \geq u_t$. Note that each branching component gives a full set prediction when $u_t = 0$ and a disjoint set prediction when $u_t = 1.0$. With overlapping coarse categories, the category hierarchy mapping P^d is extended to be a many-to-many mapping P^o and the coarse category predictions are updated accordingly $B^o_{ik} = \sum_{j|k\in P^o(j)} B_{ij}$. Note the sum of $\{B^o_{ik}\}^K_{k=1}$ exceeds 1 and hence we perform L_1 normalization. The use of overlapping coarse categories was also shown to be useful for hierarchical linear classifiers [24].

5. HD-CNN Training

As we embed fine category components into HD-CNN, the number of parameters in rear layers grows linearly in the number of coarse categories. Given the same amount of training data, this increases the training complexity and the risk of over-fitting. On the other hand, the training images within the stochastic gradient descent mini-batch are probabilistically routed to different fine category components. It requires larger mini-batch to ensure parameter gradients in the fine category components are estimated by a sufficiently large number of training samples. Large training mini-batch both increases the training memory footprint and slows down the training process. Therefore, we decompose the HD-CNN training into multiple steps instead of training the complete HD-CNN from scratch as outlined in Algorithm 1.

5.1. Pretraining HD-CNN

We sequentially pretrain the coarse category component and fine category components.

5.1.1 Initializing the Coarse Category Component

We first pretrain a building block CNN F^p using the training set. As both the preceding and rear layers in coarse category component resemble the layers in the building block CNN, we copy the weights of F^p into coarse category component for initialization purpose.

5.1.2 Pretraining the Rear Layers of Fine Category Components

Fine category components $\{F^k\}_k$ can be independently pretrained in parallel. Each F^k should specialize in classifying fine categories within the coarse category S^c_k. Therefore, the pretraining of each F^k only uses images $\{\mathbf{x_i}|i \in S^c_k\}$ from the coarse category S^c_k. The shared preceding layers are already initialized and kept fixed in this stage. For each F^k, we initialize all the rear layers except the last convolutional layer by copying the learned parameters from the pretrained model F^p.

Algorithm 1 HD-CNN training algorithm

1: **procedure** HD-CNN TRAINING
2: **Step 1:** Pretrain HD-CNN
3: **Step 1.1:** Initialize coarse category component
4: **Step 1.2:** Pretrain fine category components
5: **Step 2:** Fine-tune the complete HD-CNN

5.2. Fine-tuning HD-CNN

After both coarse category component and fine category components are properly pretrained, we fine-tune the complete HD-CNN. Once the category hierarchy as well as the associated mapping P^o is learnt, each fine category component focuses on classifying a fixed subset of fine categories. During fine-tuning, the semantics of coarse categories predicted by the coarse category component should be kept consistent with those associated with fine category components. Thus we add a coarse category consistency term to regularize the conventional multinomial logistic loss.

Coarse category consistency The learnt fine-to-coarse category mapping $P : [1, C] \mapsto [1, K]$ provides a way to specify the target coarse category distribution $\{t_k\}$. Specifically, t_k is set to be the fraction of all the training images within the coarse category S^c_k under the assumption the distribution over coarse categories across the training dataset is close to that within a training mini-batch.

$$t_k = \frac{\sum_{j|k\in P(j)} |S_j|}{\sum_{k'=1}^{K} \sum_{j|k'\in P(j)} |S_j|} \quad \forall k \in [1, K] \tag{3}$$

The final loss function we use for fine-tuning the HD-CNN is shown below.

$$E = -\frac{1}{n}\sum_{i=1}^{n} log(p_{y_i}) + \frac{\lambda}{2}\sum_{k=1}^{K}(t_k - \frac{1}{n}\sum_{i=1}^{n} B_{ik})^2 \tag{4}$$

where n is the size of training mini-batch. λ is a regularization constant and is set to $\lambda = 20$.

6. HD-CNN Testing

As we add fine category components into the HD-CNN, the number of parameters, memory footprint and execution time in rear layers, all scale linearly in the number of coarse categories. To ensure HD-CNN is scalable for large-scale visual recognition, we develop conditional execution and layer parameter compression techniques.

Conditional Execution. At test time, for a given image, it is not necessary to evaluate all fine category classifiers as most of them have insignificant weights B_{ik} as in Eqn 1.

Their contributions to the final prediction are negligible. Conditional executions of the top weighted fine components can accelerate the HD-CNN classification. Therefore, we threshold B_{ik} using a parametric variable $B_t = (\beta K)^{-1}$ and reset B_{ik} to zero when $B_{ik} < B_t$. Those fine category classifiers with $B_{ik} = 0$ are not evaluated.

Parameter Compression. In HD-CNN, the number of parameters in rear layers of fine category classifiers grows linearly in the number of coarse categories. Thus we compress the layer parameters at test time to reduce memory footprint. Specifically, we choose the Product Quantization approach [14] to compress the parameter matrix $W \in R^{m \times n}$ by first partitioning it horizontally into segments of width s. $W = [W^1, ..., W^{(n/s)}]$. Then k-means clustering is used to cluster the rows in $W^i, \forall i \in [1, (n/s)]$. By only storing the nearest cluster indices in a 8-bit integer matrix $I \in R^{m \times (n/s)}$ and cluster centers in a single-precision floating number matrix $C \in R^{k \times n}$, we can achieve a compression factor $(32mn)/(32kn + 8mn/s)$. The hyperparameters for parameter compression are (s, k).

7. Experiments

7.1. Overview

We evaluate HD-CNN on the benchmark datasets CIFAR100 [17] and ImageNet [4]. HD-CNN is implemented on the widely deployed *Caffe* [15] software. The network is trained by back propagation [18]. We run all the testing experiments on a single NVIDIA Tesla K40c card.

7.2. CIFAR100 Dataset

The CIFAR100 dataset consists of 100 classes of natural images. There are 50K training images and 10K testing images. We follow [11] to preprocess the datasets (e.g. global contrast normalization and ZCA whitening). Randomly cropped and flipped image patches of size 26×26 are used for training. We adopt a *NIN* network [1] with three stacked layers [21]. We denote it as CIFAR100-NIN which will be the HD-CNN building block. Fine category components share preceding layers from *conv1* to *pool1* which accounts for 6% of the total parameters and 29% of the total floating point operations. The remaining layers are used as independent layers.

For building the category hierarchy, we randomly choose 10K images from the training set as held-out set. Fine categories within the same coarse categories are visually more similar. We pretrain the rear layers of fine category components. The initial learning rate is 0.01 and it is decreased by a factor of 10 every 6K iterations. Fine-tuning is performed for 20K iterations with large mini-batches of size 256. The initial learning rate is 0.001 and is reduced by a factor of 10 once after 10K iterations.

[1] https://github.com/mavenlin/cuda-convnet/blob/master/NIN/cifar-100_def

Table 1: 10-view testing errors on CIFAR100 dataset. Notation **CCC**=coarse category consistency.

Method	Error
Model averaging (2 CIFAR100-NIN nets)	35.13
DSN [19]	34.68
CIFAR100-NIN-double	34.26
dasNet [30]	33.78
Base: CIFAR100-NIN	35.27
HD-CNN, no finetuning	33.33
HD-CNN, finetuning w/o CCC	33.21
HD-CNN, finetuning w/ CCC	**32.62**

For evaluation, we use 10-view testing [18]. We extract five 26×26 patches (the 4 corner patches and the center patch) as well as their horizontal reflections and average their predictions. The CIFAR100-NIN net obtains 35.27% testing error. Our HD-CNN achieves testing error of 32.62% which improves the building block net by 2.65%.

Category hierarchy. During the construction of the category hierarchy, the number of coarse categories can be adjusted by the clustering algorithm. We can also make the coarse categories either disjoint or overlapping by varying the hyperparameter γ. Thus we investigate their impacts on the classification error. We experiment with 5, 9, 14 and 19 coarse categories and vary the value of γ. The best results are obtained with 9 overlapping coarse categories and $\gamma = 5$ as shown in Fig 2 left. A histogram of fine category occurrences in 9 overlapping coarse categories is shown in Fig 2 right. The optimal value of coarse category number and hyperparameter γ are dataset dependent, mainly affected by the inherent hierarchy within the categories.

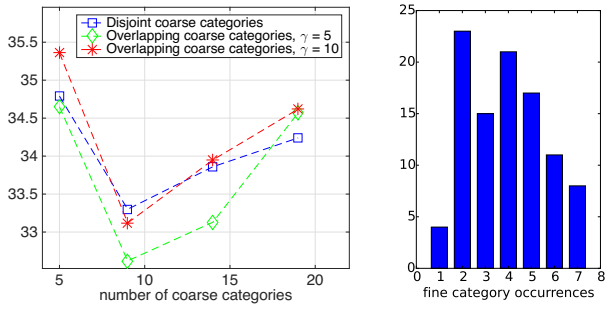


Figure 2: **Left**: HD-CNN 10-view testing error against the number of coarse categories on CIFAR100 dataset. **Right**: Histogram of fine category occurrences in 9 overlapping coarse categories.

Shared layers. The use of shared layers makes both com-

putational complexity and memory footprint of HD-CNN sublinear in the number of fine category classifiers when compared to the building block net. Our HD-CNN with 9 fine category classifiers based on CIFAR100-NIN consumes less than three times as much memory as the building block net without parameter compression. We also want to investigate the impact of the use of shared layers on the classification error, memory footprint and the net execution time (Table 2). We build another HD-CNN where coarse category component and all fine category components use independent preceding layers initialized from a pretrained building block net. Under the single-view testing where only a central cropping is used, we observe a minor error increase from 34.36% to 34.50%. But using shared layers dramatically reduces the memory footprint from 1356 MB to 459 MB and testing time from 2.43 seconds to 0.28 seconds.

Conditional executions. By varying the hyperparameter β, we can effectively affect the number of fine category components that will be executed. There is a trade-off between execution time and classification error. A larger value of β leads to higher accuracy at the cost of executing more components for fine categorization. By enabling conditional executions with hyperparameter $\beta = 6$, we obtain a substantial 2.8X speed up with merely a minor increase in error from 34.36% to 34.57% (Table 2). The testing time of HD-CNN is about 2.5 times as much as that of the building block net.

Parameter compression. As fine category CNNs have independent layers from *conv2* to *cccp6*, we compress them and reduce the memory footprint from 447MB to 286MB with a minor increase in error from 34.57% to 34.73%.

Comparison with a strong baseline. As our HD-CNN memory footprint is about two times as much as the building block model (Table 2), it is necessary to compare a stronger baseline of similar complexity with HD-CNN. We adapt CIFAR100-NIN and double the number of filters in all convolutional layers which accordingly increases the memory footprint by three times. We denote it as CIFAR100-NIN-double and obtain error 34.26% which is 1.01% lower than that of the building block net but is 1.64% higher than that of HD-CNN.

Comparison with model averaging. HD-CNN is fundamentally different from model averaging [18]. In model averaging, all models are capable of classifying the full set of the categories and each one is trained independently. The main sources of their prediction differences are different initializations. In HD-CNN, each fine category classifier only excels at classifying a partial set of categories. To compare HD-CNN with model averaging, we independently train two CIFAR100-NIN networks and take their averaged prediction as the final prediction. We obtain an error of 35.13%, which is about 2.51% higher than that of HD-CNN (Table 1). Note that HD-CNN is orthogonal to the model averaging and an ensemble of HD-CNN networks can further improve the performance.

Coarse category consistency. To verify the effectiveness of coarse category consistency term in our loss function (4), we fine-tune a HD-CNN using the traditional multinomial logistic loss function. The testing error is 33.21%, which is 0.59% higher than that of a HD-CNN fine-tuned with coarse category consistency (Table 1).

Comparison with state-of-the-art. Our HD-CNN improves on the current two best methods [19] and [30] by 2.06% and 1.16% respectively and sets new state-of-the-art results on CIFAR100 (Table 1).

7.3. ImageNet 1000-class Dataset

The ILSVRC-2012 ImageNet dataset consists of about 1.2 million training images, $50,000$ validation images. To demonstrate the generality of HD-CNN, we experiment with two different building block nets. In both cases, HD-CNNs achieve significantly lower testing errors than the building block nets.

7.3.1 Network-In-Network Building Block Net

We choose a public 4-layer NIN net [2] as our first building block as it has greatly reduced number of parameters compared to AlexNet [18] but similar error rates. It is denoted as ImageNet-NIN. In HD-CNN, various components share preceding layers from *conv1* to *pool3* which account for 26% of the total parameters and 82% of the total floating point operations.

We follow the training and testing protocols as in [18]. Original images are resized to 256×256. Randomly cropped and horizontally reflected 224×224 patches are used for training. At test time, the net makes a 10-view averaged prediction. We train ImageNet-NIN for 45 epochs. The top-1 and top-5 errors are 39.76% and 17.71%.

To build the category hierarchy, we take 100K training images as the held-out set and find 89 overlapping coarse categories. Each fine category CNN is fine-tuned for 40K iterations while the initial learning rate 0.01 is decreased by a factor of 10 every 15K iterations. Fine-tuning the complete HD-CNN is not performed as the required mini-batch size is significantly higher than that for the building block net. Nevertheless, we still achieve top-1 and top-5 errors of 36.66% and 15.80% and improve the building block net by 3.1% and 1.91%, respectively (Table 3). The class-wise top-5 error improvement over the building block net is shown in Fig 4 left.

Case studies We want to investigate how HD-CNN corrects the mistakes made by the building block net. In Fig 3, we

[2] https://gist.github.com/mavenlin/d802a5849de39225bcc6

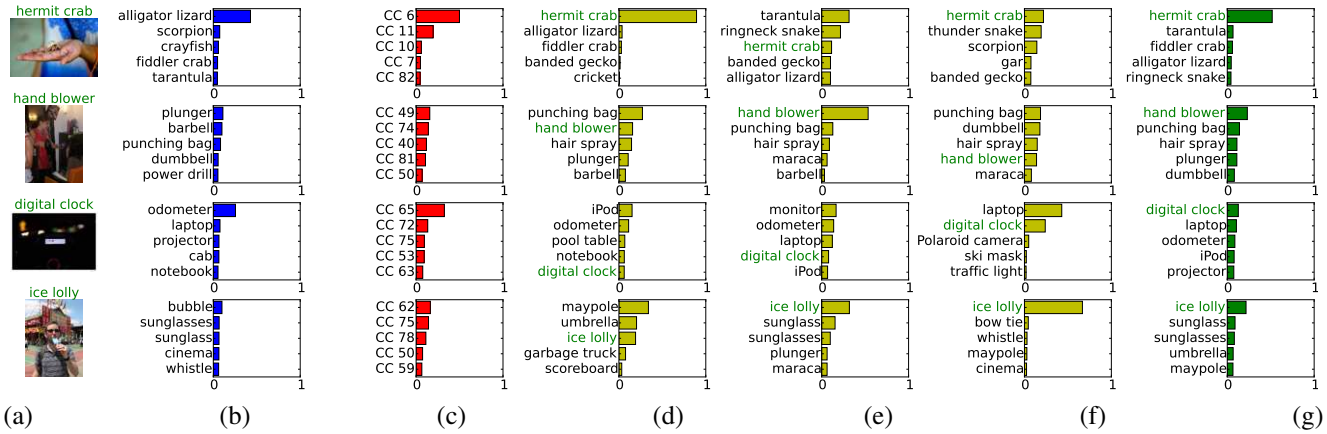


Figure 3: Case studies on ImageNet dataset. Each row represents a testing case. **Column (a)**: test image with ground truth label. **Column (b)**: top 5 guesses from the building block net ImageNet-NIN. **Column (c)**: top 5 Coarse Category (**CC**) probabilities. **Column (d)-(f)**: top 5 guesses made by the top 3 fine category CNN components. **Column (g)**: final top 5 guesses made by the HD-CNN. See text for details.

Table 2: Comparison of testing errors, memory footprint and testing time between building block nets and HD-CNNs on CIFAR100 and ImageNet datasets. Statistics are collected under *single-view* testing. Three building block nets CIFAR100-NIN, ImageNet-NIN and ImageNet-VGG-16-layer are used. The testing mini-batch size is 50. Notations: **SL**=Shared layers, **CE**=Conditional executions, **PC**=Parameter compression.

Model	top-1, top-5	Memory (MB)	Time (sec.)
Base:CIFAR100-NIN	37.29	188	0.04
HD-CNN w/o SL	34.50	1356	2.43
HD-CNN	**34.36**	459	0.28
HD-CNN+CE	34.57	447	0.10
HD-CNN+CE+PC	34.73	286	0.10
Base:ImageNet-NIN	41.52, 18.98	535	0.19
HD-CNN	**37.92, 16.62**	3544	3.25
HD-CNN+CE	38.16, 16.75	3508	0.52
HD-CNN+CE+PC	38.39, 16.89	1712	0.53
Base:ImageNet-VGG-16-layer	32.30, 12.74	4134	1.04
HD-CNN+CE+PC	**31.34, 12.26**	6863	5.28

collect four testing cases. In the first case, the building block net fails to predict the label of the tiny *hermit crab* in the top 5 guesses. In HD-CNN, two coarse categories #6 and #11 receive most of the coarse probability mass. The fine category component #6 specializes in classifying crab breeds and strongly suggests the ground truth label. By combining the predictions from the top fine category classifiers, the HD-CNN predicts *hermit crab* as the most probable label. In the second case, the ImageNet-NIN confuses the ground truth *hand blower* with other objects of close shapes and appearances, such as *plunger* and *barbell*. For HD-CNN, the coarse category component is also not confident about which coarse category the object belongs to and thus assigns even probability mass to the top coarse categories. For the top 3 fine category classifiers, #74 strongly predicts ground truth label while the other two #49 and #40 rank the ground truth label at the 2nd and 4th place respectively. Overall, the HD-CNN ranks the ground truth label at the 1st place. This demonstrates HD-CNN needs to rely on multiple fine category classifiers to make correct predictions for difficult cases.

Overlapping coarse categories.To investigate the impact of overlapping coarse categories on the classification, we train another HD-CNN with 89 fine category classifiers using disjoint coarse categories. It achieves top-1 and top-5 errors of 38.44% and 17.03% respectively, which is higher than those of the HD-CNN using overlapping coarse category hierarchy by 1.78% and 1.23% (Table 3).

Conditional executions. By varying the hyperparameter β, we can control the number of fine category components that will be executed. There is a trade-off between execution time and classification error as shown in Fig 4 right. A larger value of β leads to lower error at the cost of more executed fine category components. By enabling conditional executions with hyperparameter $\beta = 8$, we obtain a substantial 6.3X speed up with merely a minor increase of single-view testing top-5 error from 16.62% to 16.75% (Table 2). With such speedup, the HD-CNN testing time is less than 3 times as much as that of the building block net.

Parameter compression. We compress independent layers *conv4* and *cccp7* as they account for 60% of the parame-

Table 3: Comparisons of 10-view testing errors between ImageNet-NIN and HD-CNN. Notation **CC**=Coarse category.

Method	top-1, top-5
Base:ImageNet-NIN	39.76, 17.71
Model averaging (3 base nets)	38.54, 17.11
HD-CNN, disjoint CC	38.44, 17.03
HD-CNN	**36.66, 15.80**

ters in ImageNet-NIN. Their parameter matrices are of size 1024×3456 and 1024×1024 and we use compression hyperparameters $(s, k) = (3, 128)$ and $(s, k) = (2, 256)$. The compression factors are 4.8 and 2.7. The compression decreases the memory footprint from 3508MB to 1712MB and merely increases the top-5 error from 16.75% to 16.89% under single-view testing (Table 2).

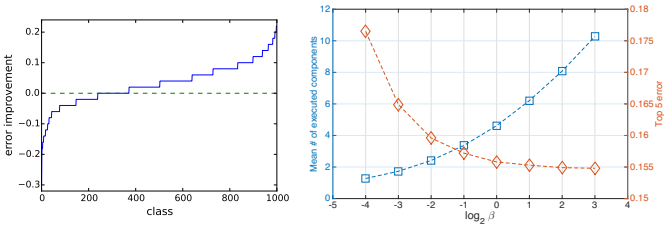


Figure 4: **Left**: Class-wise HD-CNN top-5 error improvement over the building block net. **Right**: Mean number of executed fine category classifiers and top-5 error against hyperparameter β on the ImageNet validation dataset.

Comparison with model averaging. As the HD-CNN memory footprint is about three times as much as the building block net, we independently train three ImageNet-NIN nets and average their predictions. We obtain top-5 error 17.11% which is 0.6% lower than the building block but is 1.31% higher than that of HD-CNN (Table 3).

7.3.2 VGG-16-layer Building Block Net

The second building block net we use is a 16-layer CNN from [26]. We denote it as ImageNet-VGG-16-layer[3]. The layers from *conv1_1* to *pool4* are shared and they account for 5.6% of the total parameters and 90% of the total floating number operations. The remaining layers are used as independent layers in coarse and fine category classifiers. We follow the training and testing protocols as in [26]. For training, we first sample a size S from the range $[256, 512]$ and resize the image so that the length of short edge is S. Then a randomly cropped and flipped patch of size 224×224 is used for training. For testing, dense evaluation is performed on three scales $\{256, 384, 512\}$ and the averaged prediction is used as the final prediction. Please refer to [26] for more training and testing details. On ImageNet validation set, ImageNet-VGG-16-layer achieves top-1 and top-5 errors 24.79% and 7.50% respectively.

We build a category hierarchy with 84 overlapping coarse categories. We implement multi-GPU training on Caffe by exploiting data parallelism [26] and train the fine category classifiers on two NVIDIA Tesla K40c cards. The initial learning rate is 0.001 and it is decreased by a factor of 10 every 4K iterations. HD-CNN fine-tuning is not performed. Due to large memory footprint of the building block net (Table 2), the HD-CNN with 84 fine category classifiers cannot fit into the memory directly. Therefore, we compress the parameters in layers *fc6* and *fc7* as they account for over 85% of the parameters. Parameter matrices in *fc6* and *fc7* are of size 4096×25088 and 4096×4096. Their compression hyperparameters are $(s, k) = (14, 64)$ and $(s, k) = (4, 256)$. The compression factors are 29.9 and 8 respectively. The HD-CNN obtains top-1 and top-5 errors 23.69% and 6.76% on ImageNet validation set and improves over ImageNet-VGG-16-layer by 1.1% and 0.74% respectively.

Table 4: Errors on ImageNet validation set.

Method	top-1, top-5
GoogLeNet,multi-crop [31]	N/A,7.9
VGG-19-layer, dense [26]	24.8,7.5
VGG-16-layer+VGG-19-layer,dense	24.0,7.1
Base:ImageNet-VGG-16-layer,dense	24.79,7.50
HD-CNN,dense	**23.69,6.76**

Comparison with state-of-the-art. Currently, the two best nets on ImageNet dataset are GoogLeNet [31] (Table 4) and VGG 19-layer network [26]. Using multi-scale multi-crop testing, a single GoogLeNet net achieves top-5 error 7.9%. With multi-scale dense evaluation, a single VGG 19-layer net obtains top-1 and top-5 errors 24.8% and 7.5% and improves top-5 error of GoogLeNet by 0.4%. Our HD-CNN decreases top-1 and top-5 errors of VGG 19-layer net by 1.11% and 0.74% respectively. Furthermore, HD-CNN slightly outperforms the results of averaging the predictions from VGG-16-layer and VGG-19-layer nets.

8. Conclusions and Future Work

We demonstrated that HD-CNN is a flexible deep CNN architecture to improve over existing deep CNN models. We showed this empirically on both CIFAR-100 and ImageNet datasets using three different building block nets. As part of future work, we plan to extend HD-CNN architectures to those with more than 2 hierarchical levels and also verify our empirical results in a theoretical framework.

[3] https://github.com/BVLC/caffe/wiki/Model-Zoo

References

[1] H. Bannour and C. Hudelot. Hierarchical image annotation using semantic hierarchies. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, 2012.

[2] S. Bengio, J. Weston, and D. Grangier. Label embedding trees for large multi-class tasks. In *NIPS*, 2010.

[3] J. Deng, N. Ding, Y. Jia, A. Frome, K. Murphy, S. Bengio, Y. Li, H. Neven, and H. Adam. Large-scale object classification using label relation graphs. In *ECCV*. 2014.

[4] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and F. Li. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[5] J. Deng, J. Krause, A. C. Berg, and L. Fei-Fei. Hedging your bets: Optimizing accuracy-specificity trade-offs in large scale visual recognition. In *CVPR*, 2012.

[6] J. Deng, S. Satheesh, A. C. Berg, and F. Li. Fast and balanced: Efficient label tree learning for large scale object recognition. In *Advances in Neural Information Processing Systems*, 2011.

[7] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013.

[8] R. Fergus, H. Bernal, Y. Weiss, and A. Torralba. Semantic label sharing for learning with many categories. In *ECCV*. 2010.

[9] T. Gao and D. Koller. Discriminative learning of relaxed hierarchy for large-scale visual recognition. In *ICCV*, 2011.

[10] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[11] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In *ICML*, 2013.

[12] G. Griffin and P. Perona. Learning and using taxonomies for fast visual categorization. In *CVPR*, 2008.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*. 2014.

[14] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2011.

[15] Y. Jia. Caffe: An open source convolutional architecture for fast feature embedding, 2013.

[16] Y. Jia, J. T. Abbott, J. Austerweil, T. Griffiths, and T. Darrell. Visual concept learning: Combining machine vision and bayesian generalization on concept hierarchies. In *NIPS*, 2013.

[17] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep*, 2009.

[18] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[19] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. *arXiv preprint arXiv:1409.5185*, 2014.

[20] L.-J. Li, C. Wang, Y. Lim, D. M. Blei, and L. Fei-Fei. Building and using a semantivisual image hierarchy. In *CVPR*, 2010.

[21] M. Lin, Q. Chen, and S. Yan. Network in network. *CoRR*, 2013.

[22] B. Liu, F. Sadeghi, M. Tappen, O. Shamir, and C. Liu. Probabilistic label trees for efficient large scale image classification. In *CVPR*, 2013.

[23] M. Marszalek and C. Schmid. Semantic hierarchies for visual object recognition. In *CVPR*, 2007.

[24] M. Marszałek and C. Schmid. Constructing category hierarchies for visual recognition. In *ECCV*. 2008.

[25] R. Salakhutdinov, A. Torralba, and J. Tenenbaum. Learning to share visual appearance for multiclass object detection. In *CVPR*, 2011.

[26] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[27] J. Sivic, B. C. Russell, A. Zisserman, W. T. Freeman, and A. A. Efros. Unsupervised discovery of visual object class hierarchies. In *CVPR*, 2008.

[28] J. T. Springenberg and M. Riedmiller. Improving deep neural networks with probabilistic maxout units. *arXiv preprint arXiv:1312.6116*, 2013.

[29] N. Srivastava and R. Salakhutdinov. Discriminative transfer learning with tree-based priors. In *NIPS*, 2013.

[30] M. F. Stollenga, J. Masci, F. Gomez, and J. Schmidhuber. Deep networks with internal selective attention through feedback connections. In *NIPS*, 2014.

[31] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

[32] A.-M. Tousch, S. Herbin, and J.-Y. Audibert. Semantic hierarchies for image annotation: A survey. *Pattern Recognition*, 2012.

[33] N. Verma, D. Mahajan, S. Sellamanickam, and V. Nair. Learning hierarchical similarity metrics. In *CVPR*, 2012.

[34] T. Xiao, J. Zhang, K. Yang, Y. Peng, and Z. Zhang. Error-driven incremental learning in deep convolutional neural network for large-scale image classification. In *Proceedings of the ACM International Conference on Multimedia*, 2014.

[35] M. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV*. 2014.

[36] M. D. Zeiler and R. Fergus. Stochastic pooling for regularization of deep convolutional neural networks. In *ICLR*, 2013.

[37] B. Zhao, F. Li, and E. P. Xing. Large-scale category structure aware image categorization. In *NIPS*, 2011.

[38] A. Zweig and D. Weinshall. Exploiting object hierarchy: Combining models from different category levels. In *ICCV*, 2007.

Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture

David Eigen[1] Rob Fergus[1,2]
[1] Dept. of Computer Science, Courant Institute, New York University
[2] Facebook AI Research
{deigen,fergus}@cs.nyu.edu

Abstract

In this paper we address three different computer vision tasks using a single multiscale convolutional network architecture: depth prediction, surface normal estimation, and semantic labeling. The network that we develop is able to adapt naturally to each task using only small modifications, regressing from the input image to the output map directly. Our method progressively refines predictions using a sequence of scales, and captures many image details without any superpixels or low-level segmentation. We achieve state-of-the-art performance on benchmarks for all three tasks.

1. Introduction

Scene understanding is a central problem in vision that has many different aspects. These include semantic labels describing the identity of different scene portions; surface normals or depth estimates describing the physical geometry; instance labels of the extent of individual objects; and affordances capturing possible interactions of people with the environment. Many of these are often represented with a pixel-map containing a value or label for each pixel, *e.g.* a map containing the semantic label of the object visible at each pixel, or the vector coordinates of the surface normal orientation.

In this paper, we address three of these tasks, depth prediction, surface normal estimation and semantic segmentation — all using a single common architecture. Our multi-scale approach generates pixel-maps directly from an input image, without the need for low-level superpixels or contours, and is able to align to many image details using a series of convolutional network stacks applied at increasing resolution. At test time, all three outputs can be generated in real time ($\sim$30Hz). We achieve state-of-the art results on all three tasks we investigate, demonstrating our model's versatility.

There are several advantages in developing a general model for pixel-map regression. First, applications to new tasks may be quickly developed, with much of the new work lying in defining an appropriate training set and loss function; in this light, our work is a step towards building off-the-shelf regressor models that can be used for many applications. In addition, use of a single architecture helps simplify the implementation of systems that require multiple modalities, e.g. robotics or augmented reality, which in turn can help enable research progress in these areas. Lastly, in the case of depth and normals, much of the computation can be shared between modalities, making the system more efficient.

2. Related Work

Convolutional networks have been applied with great success for object classification and detection [19, 12, 30, 32, 34]. Most such systems classify either a single object label for an entire input window, or bounding boxes for a few objects in each scene. However, ConvNets have recently been applied to a variety of other tasks, including pose estimation [36, 27], stereo depth [39, 25], and instance segmentation [14]. Most of these systems use ConvNets to find only local features, or generate descriptors of discrete proposal regions; by contrast, our network uses both local and global views to predict a variety of output types. In addition, while each of these methods tackle just one or two tasks at most, we are able to apply our network to three disparate tasks.

Our method builds upon the approach taken by Eigen *et al.* [8], who apply two convolutional networks in stages for single-image depth map prediction. We develop a more general network that uses a sequence of three scales to generate features and refine predictions to higher resolution, which we apply to multiple tasks, including surface normals estimation and per-pixel semantic labeling. Moreover, we improve performance in depth prediction as well, illustrating how our enhancements help improve all tasks.

Single-image surface normal estimation has been addressed by Fouhey *et al.* [10, 11], Ladicky *et al.* [21], Barron and Malik [3, 2], and most recently by Wang *et al.* [38], the latter in work concurrent with ours. Fouhey *et al.* match to discriminative local templates [10] followed by a global op-

timization on a grid drawn from vanishing point rays [11], while Ladicky *et al.* learn a regression from over-segmented regions to a discrete set of normals and mixture coefficients. Barron and Malik [3, 2] infer normals from RGB-D inputs using a set of handcrafted priors, along with illumination and reflectance. From RGB inputs, Wang *et al.* [38] use convolutional networks to combine normals estimates from local and global scales, while also employing cues from room layout, edge labels and vanishing points. Importantly, we achieve as good or superior results with a more general multiscale architecture that can naturally be used to perform many different tasks.

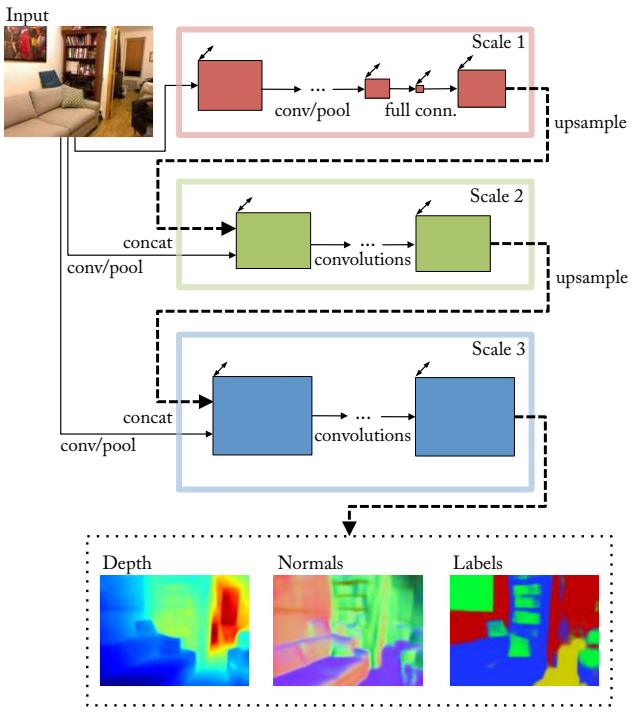
Prior work on semantic segmentation includes many different approaches, both using RGB-only data [35, 4, 9] as well as RGB-D [31, 29, 26, 6, 15, 17, 13]. Most of these use local features to classify over-segmented regions, followed by a global consistency optimization such as a CRF. By comparison, our method takes an essentially inverted approach: We make a consistent global prediction first, then follow it with iterative local refinements. In so doing, the local networks are made aware of their place within the global scene, and can can use this information in their refined predictions.

Gupta *et al.* [13, 14] create semantic segmentations first by generating contours, then classifying regions using either hand-generated features and SVM [13], or a convolutional network for object detection [14]. Notably, [13] also performs amodal completion, which transfers labels between disparate regions of the image by comparing planes from the depth.

Most related to our method in semantic segmentation are other approaches using convolutional networks. Farabet *et al.* [9] and Couprie *et al.* [6] each use a convolutional network applied to multiple scales in parallel generate features, then aggregate predictions using superpixels. Our method differs in several important ways. First, our model has a large, full-image field of view at the coarsest scale; as we demonstrate, this is of critical importance, particularly for depth and normals tasks. In addition, we do not use superpixels or post-process smoothing — instead, our network produces fairly smooth outputs on its own, allowing us to take a simple pixel-wise maximum.

Pinheiro *et al.* [28] use a recurrent convolutional network in which each iteration incorporates progressively more context, by combining a more coarsely-sampled image input along with the local prediction from the previous iteration. This direction is precisely the reverse of our approach, which makes a global prediction first, then iteratively refines it. In addition, whereas they apply the same network parameters at all scales, we learn distinct networks that can specialize in the edits appropriate to their stage.

Most recently, in concurrent work, Long *et al.* [24] adapt the recent VGG ImageNet model [32] to semantic segmentation by applying 1x1 convolutional label classifiers at feature maps from different layers, corresponding to different scales, and averaging the outputs. By contrast, we apply networks for different scales in series, which allows them to make more complex edits and refinements, starting from the full image field of view. Thus our architecture easily adapts to many tasks, whereas by considering relatively smaller context and summing predictions, theirs is specific to semantic labeling.



	Layer	1.1	1.2	1.3	1.4	1.5	1.6	1.7	upsamp
Scale 1 (AlexNet)	Size	37x27	18x13	18x13	18x13	8x6	1x1	19x14	74x55
	#convs	1	1	1	1	1	–	–	–
	#chan	96	256	384	384	256	4096	64	64
	ker. sz	11x11	5x5	3x3	3x3	3x3	–	–	–
	Ratio	/8	/16	/16	/16	/32	–	/16	/4
	l.rate	0.001	0.001	0.001	0.001	0.001	see text		
	Layer	1.1	1.2	1.3	1.4	1.5	1.6	1.7	upsamp
Scale 1 (VGG)	Size	150x112	75x56	37x28	18x14	9x7	1x1	19x14	74x55
	#convs	2	2	3	3	3	–	–	–
	#chan	64	128	256	512	512	4096	64	64
	ker. sz	3x3	3x3	3x3	3x3	3x3	–	–	–
	Ratio	/2	/4	/8	/16	/32	–	/16	/4
	l.rate	0.001	0.001	0.001	0.001	0.001	see text		
	Layer	2.1	2.2	2.3	2.4	2.5			upsamp
Scale 2	Size	74x55	74x55	74x55	74x55	74x55			147x109
	#chan	96+64	64	64	64	C			C
	ker. sz	9x9	5x5	5x5	5x5	5x5			–
	Ratio	/4	/4	/4	/4	/4			/2
	l.rate	0.001	0.01	0.01	0.01	0.001			
	Layer	3.1	3.2	3.3	3.4				final
Scale 3	Size	147x109	147x109	147x109	147x109				147x109
	#chan	96+C	64	64	C				C
	ker. sz	9x9	5x5	5x5	5x5				–
	Ratio	/2	/2	/2	/2				/2
	l.rate	0.001	0.01	0.01	0.001				

Figure 1. Model architecture. C is the number of output channels in the final prediction, which depends on the task. The input to the network is 320x240.

3. Model Architecture

Our model is a multi-scale deep network that first predicts a coarse global output based on the entire image area, then refines it using finer-scale local networks. This scheme is illustrated in Fig. 1. While our model was initially based upon the architecture proposed by [8], it offers several architectural improvements. First, we make the model deeper (more convolutional layers). Second, we add a third scale at higher resolution, bringing the final output resolution up to half the input, or 147×109 for NYUDepth. Third, instead of passing output *predictions* from scale 1 to scale 2, we pass multichannel *feature maps*; in so doing, we found we could also train the first two scales of the network jointly from the start, somewhat simplifying the training procedure and yielding performance gains.

Scale 1: Full-Image View The first scale in the network predicts a coarse but spatially-varying set of features for the entire image area, based on a large, full-image field of view, which we accomplish this through the use of two fully-connected layers. The output of the last full layer is reshaped to 1/16-scale in its spatial dimensions by 64 features, then upsampled by a factor of 4 to 1/4-scale. Note since the feature upsampling is linear, this corresponds to a decomposition of a big fully connected layer from layer 1.6 to the larger 74×55 map; since such a matrix would be prohibitively large and only capable of producing a blurry output given the more constrained input features, we constrain the resolution and upsample. Note, however, that the 1/16-scale output is still large enough to capture considerable spatial variation, and in fact is twice as large as the 1/32-scale final convolutional features of the coarse stack.

Since the top layers are fully connected, each spatial location in the output connects to the all the image features, incorporating a very large field of view. This stands in contrast to the multiscale approach of [6, 9], who produce maps where the field of view of each output location is a more local region centered on the output pixel. This full-view connection is especially important for depth and normals tasks, as we investigate in Section 7.1.

As shown in Fig. 1, we trained two different sizes of our model: One where this scale is based on an ImageNet-trained AlexNet [19], and one where it is initialized using the Oxford VGG network [32]. We report differences in performance between the models on all tasks, to measure the impact of model size in each.

Scale 2: Predictions The job of the second scale is to produce predictions at a mid-level resolution, by incorporating a more detailed but narrower view of the image along with the full-image information supplied by the coarse network. We accomplish this by concatenating the feature maps of the coarse network with those from a single layer of convolution and pooling, performed at finer stride (see Fig. 1). The output of the second scale is a 55x74 prediction (for NYUDepth), with the number of channels depending on the task. We train Scales 1 and 2 of the model together jointly, using SGD on the losses described in Section 4.

Scale 3: Higher Resolution The final scale of our model refines the predictions to higher resolution. We concatenate the Scale-2 outputs with feature maps generated from the original input at yet finer stride, thus incorporating a more detailed view of the image. The further refinement aligns the output to higher-resolution details, producing spatially coherent yet quite detailed outputs. The final output resolution is half the network input.

4. Tasks

We apply this same architecture structure to each of the three tasks we investigate: depths, normals and semantic labeling. Each makes use of a different loss function and target data defining the task.

4.1. Depth

For depth prediction, we use a loss function comparing the predicted and ground-truth log depth maps D and D^*. Letting $d = D - D^*$ be their difference, we set the loss to

$$L_{depth}(D, D^*) = \frac{1}{n}\sum_i d_i^2 - \frac{1}{2n^2}\left(\sum_i d_i\right)^2 + \frac{1}{n}\sum_i [(\nabla_x d_i)^2 + (\nabla_y d_i)^2] \quad (1)$$

where the sums are over valid pixels i and n is the number of valid pixels (we mask out pixels where the ground truth is missing). Here, $\nabla_x d_i$ and $\nabla_y d_i$ are the horizontal and vertical image gradients of the difference.

Our loss is similar to that of [8], who also use the l_2 and scale-invariant difference terms in the first line. However, we also include a first-order matching term $(\nabla_x d_i)^2 + (\nabla_y d_i)^2$, which compares image gradients of the prediction with the ground truth. This encourages predictions to have not only close-by values, but also similar local structure. We found it indeed produces outputs that better follow depth gradients, with no degradation in measured l_2 performance.

4.2. Surface Normals

To predict surface normals, we change the output from one channel to three, and predict the x, y and z components of the normal at each pixel. We also normalize the vector at each pixel to unit l_2 norm, and backpropagate through this normalization. We then employ a simple elementwise loss comparing the predicted normal at each pixel to the ground truth, using a dot product:

$$L_{normals}(N, N^*) = -\frac{1}{n}\sum_i N_i \cdot N_i^* = -\frac{1}{n} N \cdot N^* \quad (2)$$

where N and N^* are predicted and ground truth normal vector maps, and the sums again run over valid pixels (*i.e.* those with a ground truth normal).

For ground truth targets, we compute the normal map using the same method as in Silberman *et al.* [31], which estimates normals from depth by fitting least-squares planes to neighboring sets of points in the point cloud.

4.3. Semantic Labels

For semantic labeling, we use a pixelwise softmax classifier to predict a class label for each pixel. The final output then has as many channels as there are classes. We use a simple pixelwise cross-entropy loss,

$$L_{semantic}(C, C^*) = -\frac{1}{n}\sum_i C_i^* \log(C_i) \qquad (3)$$

where $C_i = e^{z_i} / \sum_c e^{z_{i,c}}$ is the class prediction at pixel i given the output z of the final convolutional linear layer 3.4.

When labeling the NYUDepth RGB-D dataset, we use the ground truth depth and normals as additional input channels. We convolve each of the three input types (RGB, depth and normals) with a different set of $32 \times 9 \times 9$ filters, then concatenate the resulting three feature sets along with the network output from the previous scale to form the input to the next. We also tried the "HHA" encoding proposed by [14], but did not see a benefit in our case, thus we opt for the simpler approach of using the depth and xyz-normals directly. Note the first scale is initialized using ImageNet, and we keep it RGB-only. Applying convolutions to each input type separately, rather than concatenating all the channels together in pixel space and filtering the joint input, enforces independence between the features at the lowest filter level, which we found helped performance.

5. Training

5.1. Training Procedure

We train our model in two phases using SGD: First, we jointly train both Scales 1 and 2. Second, we fix the parameters of these scales and train Scale 3. Since Scale 3 contains four times as many pixels as Scale 2, it is expensive to train using the entire image area for each gradient step. To speed up training, we instead use random crops of size 74x55: We first forward-propagate the entire image through scales 1 and 2, upsample, and crop the resulting Scale 3 input, as well as the original RGB input at the corresponding location. The cropped image and Scale 2 prediction are forward- and back-propagated through the Scale 3 network, and the weights updated. We find this speeds up training by about a factor of 3, including the overhead for inference of the first two scales, and results in about the same if not slightly better error from the increased stochasticity.

All three tasks use the same initialization and learning rates in nearly all layers, indicating that hyperparameter settings are in fact fairly robust to changes in task. Each were first tuned using the depth task, then verified to be an appropriate order of magnitude for each other task using a small validation set of 50 scenes. The only differences are: (*i*) The learning rate for the normals task is 10 times larger than depth or labels. (*ii*) Relative learning rates of layers 1.6 and 1.7 are 0.1 each for depth/normals, but 1.0 and 0.01 for semantic labeling. (*iii*) The dropout rate of layer 1.6 is 0.5 for depth/normals, but 0.8 for semantic labels, as there are fewer training images.

We initialize the convolutional layers in Scale 1 using ImageNet-trained weights, and randomly initialize the fully connected layers of Scale 1 and all layers in Scales 2 and 3. We train using batches of size 32 for the AlexNet-initialized model but batches of size 16 for the VGG-initialized model due to memory constraints. In each case we step down the global learning rate by a factor of 10 after approximately 2M gradient steps, and train for an additional 0.5M steps.

5.2. Data Augmentation

In all cases, we apply random data transforms to augment the training data. We use random scaling, in-plane rotation, translation, color, flips and contrast. When transforming an input and target, we apply corresponding transformations to RGB, depth, normals and labels. Note the normal vector transformation is the inverse-transpose of the worldspace transform: Flips and in-plane rotations require flipping or rotating the normals, while to scale the image by a factor s, we divide the depths by s but multiply the z coordinate of the normals and renormalize.

5.3. Combining Depth and Normals

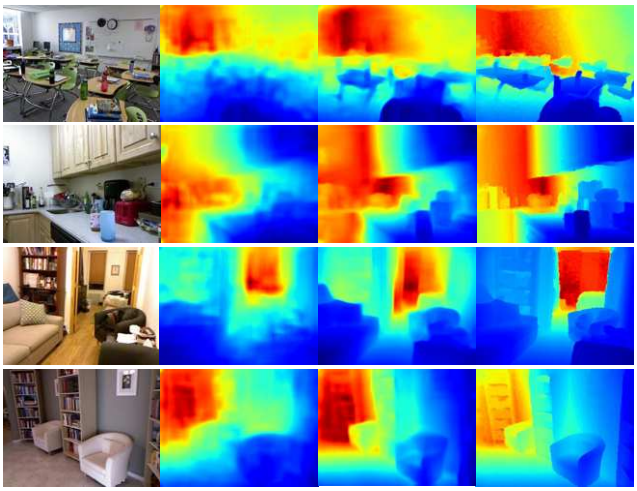
We combine both depths and normals networks together to share computation, creating a network using a single scale 1 stack, but separate scale 2 and 3 stacks. Thus we predict both depth and normals at the same time, given an RGB image. This produces a 1.6x speedup compared to using two separate models. [1]

6. Performance Experiments

6.1. Depth

We first apply our method to depth prediction on NYUDepth v2. We train using the entire NYUDepth v2 raw data distribution, using the scene split specified in the official train/test distribution. We then test on the common distribution depth maps, including filled-in areas, but constrained to the axis-aligned rectangle where there there is a valid depth map projection. Since the network output is a lower resolution than the original NYUDepth images, and excludes a small border, we bilinearly upsample our network outputs to the original 640x480 image scale, and extrapolate the missing border using a cross-bilateral filter. We compare our method to prior works Ladicky *et al.* [20],

[1] This shared model also enabled us to try enforcing compatibility between predicted normals and those obtained via finite difference of the predicted depth (predicting normals directly performs considerably better than using finite difference). However, while this constraint was able to improve the normals from finite difference, it failed to improve either task individually. Thus, while we make use of the shared model for computational efficiency, we do not use the extra compatibility constraint.



(a) (b) (c) (d)

Figure 2. Example depth results. (a) RGB input; (b) result of [8]; (c) our result; (d) ground truth. Note the color range of each image is individually scaled.

Depth Prediction							
	Ladicky[20]	Karsch[18]	Baig [1]	Liu [23]	Eigen[8]	Ours(A)	Ours(VGG)
$\delta < 1.25$	0.542	–	0.597	0.614	0.614	0.697	**0.769**
$\delta < 1.25^2$	0.829	–	–	0.883	0.888	0.912	**0.950**
$\delta < 1.25^3$	0.940	–	–	0.971	0.972	0.977	**0.988**
abs rel	–	0.350	0.259	0.230	0.214	0.198	**0.158**
sqr rel	–	–	–	–	0.204	0.180	**0.121**
RMS(lin)	–	1.2	0.839	0.824	0.877	0.753	**0.641**
RMS(log)	–	–	–	–	0.283	0.255	**0.214**
sc-inv.	–	–	0.242	–	0.219	0.202	**0.171**

Table 1. Depth estimation measurements. Note higher is better for top rows of the table, while lower is better for the bottom section.

Karsh *et al.* [18], Baig *et al.* [1], Liu *et al.* [23] and Eigen *et al.* [8].

The results are shown in Table 1. Our model obtains best performance in all metrics, due to our larger architecture and improved training. In addition, the VGG version of our model significantly outperforms the smaller AlexNet version, reenforcing the importance of model size; this is the case even though the depth task is seemingly far removed from the classification task with which the initial coarse weights were first trained. Qualitative results in Fig. 2 show substantial improvement in detail sharpness over [8].

6.2. Surface Normals

Next we apply our method to surface normals prediction. We compare against the 3D Primitives (3DP) and "Indoor Origami" works of Fouhey *et al.* [10, 11], Ladicky *et al.* [21], and Wang *et al.* [38]. As with the depth network, we used the full raw dataset for training, since ground-truth normal maps can be generated for all images. Since different systems have different ways of calculating ground truth normal maps, we compare using both the ground truth as constructed in [21] as well as the method used in [31]. The differences between ground truths are due primarily to the fact that [21] uses more aggressive smoothing; thus [21] tends to present flatter areas, while [31] is noisier but keeps more details present. We measure performance with the same metrics as in [10]: The mean and median angle from the ground truth across all unmasked pixels, as well as the percent of vectors whose angle falls within three thresholds.

Surface Normal Estimation (GT [21])					
	Angle Distance		Within t° Deg.		
	Mean	Median	11.25°	22.5°	30°
3DP [10]	35.3	31.2	16.4	36.6	48.2
Ladicky *&al.* [21]	33.5	23.1	27.5	49.0	58.7
Fouhey *&al.* [11]	35.2	17.9	40.5	54.1	58.9
Wang *&al.* [38]	26.9	14.8	42.0	61.2	68.2
Ours (AlexNet)	23.7	15.5	39.2	62.0	71.1
Ours (VGG)	**20.9**	**13.2**	**44.4**	**67.2**	**75.9**
Surface Normal Estimation (GT [31])					
	Angle Distance		Within t° Deg.		
	Mean	Median	11.25°	22.5°	30°
3DP [10]	37.7	34.1	14.0	32.7	44.1
Ladicky *&al.* [21]	35.5	25.5	24.0	45.6	55.9
Wang *&al.* [38]	28.8	17.9	35.2	57.1	65.5
Ours (AlexNet)	25.9	18.2	33.2	57.5	67.7
Ours (VGG)	**22.2**	**15.3**	**38.6**	**64.0**	**73.9**

Table 2. Surface normals prediction measured against the ground truth constructed by [21] (top) and [31] (bottom).

Results are shown in Table 2. The smaller version of our model performs similarly or slightly better than Wang *et al.*, while the larger version substantially outperforms all comparison methods. Figure 3 shows example predictions. Note the details captured by our method, such as the curvature of the blanket on the bed in the first row, sofas in the second row, and objects in the last row.

6.3. Semantic Labels

6.3.1 NYU Depth

We finally apply our method to semantic segmentation, first also on NYUDepth. Because this data provides a depth channel, we use the ground-truth depth and normals as input into the semantic segmentation network, as described in Section 4.3. We evaluate our method on semantic class sets with 4, 13 and 40 labels, described in [31], [6] and [13], respectively. The 4-class segmentation task uses high-level category labels "floor", "structure", "furniture" and "props", while the 13- and 40-class tasks use different sets of more fine-grained categories. We compare with several recent methods, using the metrics commonly used to evaluate each task: For the 4- and 13-class tasks we use pixel-wise and per-class accuracy; for the 40-class task, we also compare using the mean pixel-frequency weighted Jaccard index of each class, and the flat mean Jaccard index.

Results are shown in Table 3. We decisively outperform the comparison methods on the 4- and 14-class tasks. In the 40-class task, our model outperforms Gupta *et al.* '14 with both model sizes, and Long *et al.* with the larger size. Qualitative results are shown in Fig. 4. Even though our method does not use superpixels or any piecewise constant assumptions, it nevertheless tends to produce large constant regions most of the time.

4-Class Semantic Segmentation	Pixel	Class
Couprie *&al.* [6]	64.5	63.5
Khan *&al.* [15]	69.2	65.6
Stuckler *&al.* [33]	70.9	67.0
Mueller *&al.* [26]	72.3	71.9
Gupta *&al.* '13 [13]	78	–
Ours (AlexNet)	80.6	79.1
Ours (VGG)	**83.2**	**82.0**

13-Class Semantic	Pixel	Class
Couprie *&al.* [6]	52.4	36.2
Wang *&al.* [37]	–	42.2
Hermans *&al.* [17]	54.2	48.0
Khan *&al.* [15] *	58.3	45.1
Ours (AlexNet)	70.5	59.4
Ours (VGG)	**75.4**	**66.9**

40-Class Semantic Segmentation	Pix. Acc.	Per-Cls Acc.	Freq. Jaccard	Av. Jaccard
Gupta*&al.*'13 [13]	59.1	28.4	45.6	27.4
Gupta*&al.*'14 [14]	60.3	35.1	47.0	28.6
Long*&al.* [24]	65.4	**46.1**	49.5	34.0
Ours (AlexNet)	62.9	41.3	47.6	30.8
Ours (VGG)	**65.6**	45.1	**51.4**	**34.1**

Table 3. Semantic labeling on NYUDepth v2
* Khan*&al.* use a different overlapping label set.

Sift Flow Semantic Segmentation	Pix. Acc.	Per-Class Acc.	Freq. Jacc	Av. Jacc
Farabet *&al.* (1) [9]	78.5	29.6	–	–
Farabet *&al.* (2) [9]	74.2	46.0	–	–
Tighe *&al.* [35]	78.6	39.2	–	–
Pinheiro *&al.* [28]	77.7	29.8	–	–
Long *&al.* [24]	85.1	51.7	76.1	**39.5**
Ours (AlexNet) (1)	84.0	42.0	73.7	33.1
Ours (AlexNet) (2)	81.6	48.2	71.3	32.6
Ours (VGG) (1)	**86.8**	46.4	**77.9**	38.8
Ours (VGG) (2)	83.8	**55.7**	74.7	37.6

Table 4. Semantic labeling on the Sift Flow dataset. (1) and (2) correspond to non-reweighted and class-reweighted versions of our model (see text).

6.3.2 Sift Flow

We confirm our method can be applied to additional scene types by evaluating on the Sift Flow dataset [22], which contains images of outdoor cityscapes and landscapes segmented into 33 categories. We found no need to adjust convolutional kernel sizes or learning rates for this dataset, and simply transfer the values used for NYUDepth directly; however, we do adjust the output sizes of the layers to match the new image sizes.

We compare against Tighe *et al.* [35], Farabet *et al.* [9], Pinheiro [28] and Long *et al.* [24]. Note that Farabet *et al.* train two models, using empirical or rebalanced class distributions by resampling superpixels. We train a more class-balanced version of our model by reweighting each class in the cross-entropy loss; we weight each pixel by $\alpha_c = median\_freq/freq(c)$ where $freq(c)$ is the number of pixels of class c divided by the total number of pixels in images where c is present, and $median\_freq$ is the median of these frequencies.

Results are in Table 4; we compare regular (1) and reweighted (2) versions of our model against comparison methods. Our smaller model substantially outperforms all but Long *et al.* , while our larger model performs similarly to Long *et al.* This demonstrates our model's adaptability not just to different tasks but also different data.

6.3.3 Pascal VOC

In addition, we also verify our method using Pascal VOC. Similarly to Long *et al.* [24], we train using the 2011 training set augmented with 8498 training images collected by Hariharan *et al.* [16], and evaluate using the 736 images from the 2011 validation set not also in the Hariharan extra set, as well as on the 2011 and 2012 test sets. We perform online data augmentations as in our NYUDepth and Sift Flow models, and use the same learning rates. Because these images have arbitrary aspect ratio, we train our model on square inputs, and scale the smaller side of each image to 256; at test time we apply the model with a stride of 128 to cover the image (two applications are usually sufficient).

Pascal VOC Semantic Segmentation	2011 Validation				2011 Test	2012 Test
	Pix. Acc.	Per-Cls Acc.	Freq.Jacc	Av.Jacc	Av.Jacc	Av.Jacc
Dai*&al.*[7]	–	–	–	–	–	61.8
Long*&al.*[24]	90.3	75.9	83.2	62.7	**62.7**	62.2
Chen*&al.*[5]	–	–	–	–	–	**71.6**
Ours (VGG)	90.3	72.4	82.9	62.2	62.5	62.6

Table 5. Semantic labeling on Pascal VOC 2011 and 2012.

Contributions of Scales	Depth	Normals	4-Class		13-Class	
			RGB+D+N	RGB	RGB+D+N	RGB
	Pixelwise Error lower is better		Pixelwise Accuracy higher is better			
Scale 1 only	0.218	29.7	71.5	71.5	58.1	58.1
Scale 2 only	0.290	31.8	77.4	67.2	65.1	53.1
Scales 1 + 2	0.216	26.1	80.1	74.4	69.8	63.2
Scales 1 + 2 + 3	0.198	25.9	80.6	75.3	70.5	64.0

Table 6. Comparison of networks for different scales for depth, normals and semantic labeling tasks with 4 and 13 categories. Largest single contributing scale is underlined.

Results are shown in Table 5 and Fig. 5. We compare with Dai *et al.* [7], Long *et al.* [24] and Chen *et al.* [5]; the latter is a more recent work that augments a convolutional network with large top-layer field of and fully-connected CRF. Our model performs comparably to Long *et al.*, even as it generalizes to multiple tasks, demonstrated by its adeptness at depth and normals prediction.

7. Probe Experiments

7.1. Contributions of Scales

We compare performance broken down according to the different scales in our model in Table 6. For depth, normals and 4- and 13-class semantic labeling tasks, we train and evaluate the model using just scale 1, just scale 2, both, or all three scales 1, 2 and 3. For the coarse scale-1-only prediction, we replace the last fully connected layer of the coarse stack with a fully connected layer that outputs directly to target size, *i.e.* a pixel map of either 1, 3, 4 or 13 channels depending on the task. The spatial resolution is the same as is used for the coarse features in our model, and is upsampled in the same way.

We report the "abs relative difference" measure (*i.e.* $|D - D^*|/D^*$) to compare depth, mean angle distance for normals, and pixelwise accuracy for semantic segmentation.

First, we note there is progressive improvement in all tasks as scales are added (rows 1, 3, and 4). In addition, we find the largest single contribution to performance is the

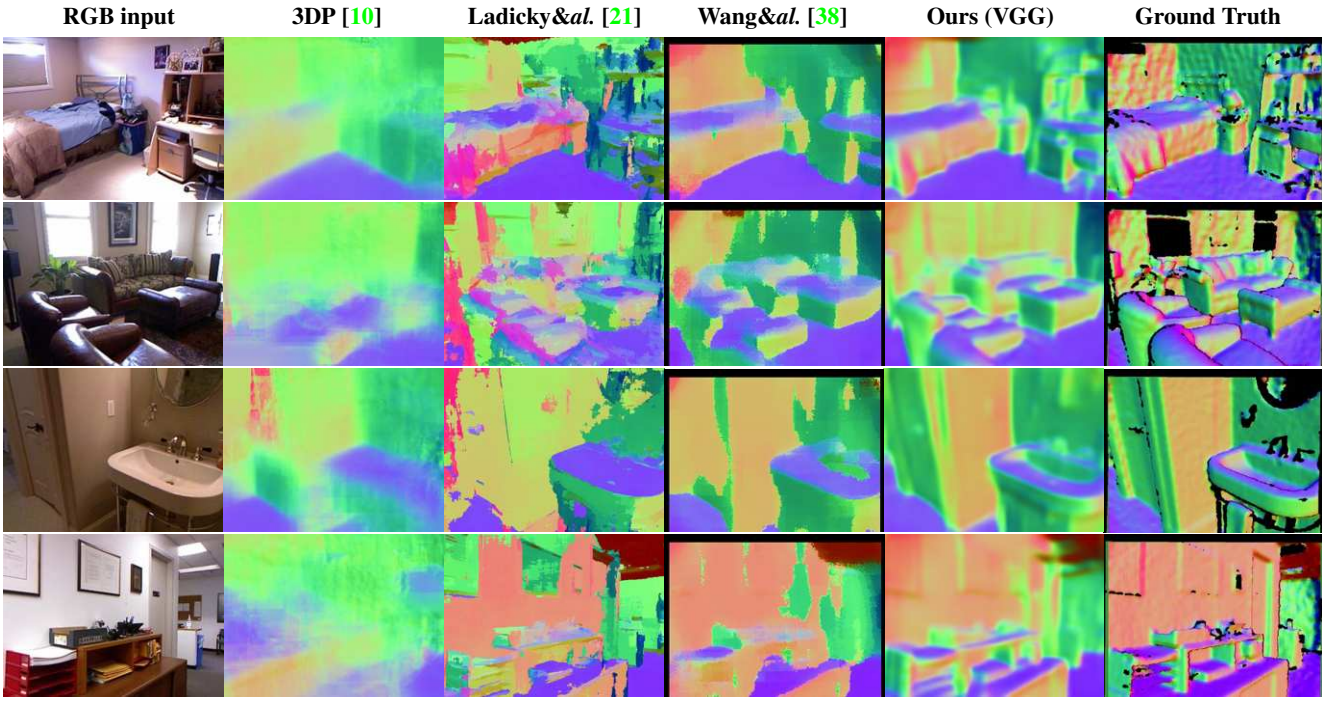


Figure 3. Comparison of surface normal maps.

Effect of Depth/Normals Inputs				
	Scale 2 only		Scales 1 + 2	
	Pix. Acc.	Per-class	Pix. Acc.	Per-class
RGB only	53.1	38.3	63.2	50.6
RGB + pred. D&N	58.7	43.8	65.0	49.5
RGB + g.t. D&N	65.1	52.3	69.8	58.9

Table 7. Comparison of RGB-only, predicted depth/normals, and ground-truth depth/normals as input to the 13-class semantic task.

coarse Scale 1 for depth and normals, but the more local Scale 2 for the semantic tasks — however, this is only due to the fact that the depth and normals channels are introduced at Scale 2 for the semantic labeling task. Looking at the labeling network with RGB-only inputs, we find that the coarse scale is again the larger contributer, indicating the importance of the global view. (Of course, this scale was also initialized with ImageNet convolution weights that are much related to the semantic task; however, even initializing randomly achieves 54.5% for 13-class scale 1 only, still the largest contribution, albeit by a smaller amount).

7.2. Effect of Depth and Normals Inputs

The fact that we can recover much of the depth and normals information from the RGB image naturally leads to two questions: (*i*) How important are the depth and normals inputs relative to RGB in the semantic labeling task? (*ii*) What might happen if we were to replace the true depth and normals inputs with the predictions made by our network?

To study this, we trained and tested our network using either Scale 2 alone or both Scales 1 and 2 for the 13-class semantic labeling task under three input conditions: (*a*) the RGB image only, (*b*) the RGB image along with predicted depth and normals, or (*c*) RGB plus true depth and normals. Results are in Table 7. Using ground truth depth/normals shows substantial improvements over RGB alone. Predicted depth/normals appear to have little effect when using both scales, but a tangible improvement when using only Scale 2. We believe this is because any relevant information provided by predicted depths/normals for labeling can also be extracted from the input; thus the labeling network can learn this same information itself, just from the label targets. However, this supposes that the network structure is capable of learning these relations: If this is not the case, *e.g.* when using only Scale 2, we do see improvement. This is also consistent with Section 7.1, where we found the coarse network was important for prediction in all tasks — indeed, supplying the predicted depth/normals to scale 2 is able to recover much of the performance obtained by the RGB-only scales 1+2 model.

8. Discussion

Together, depth, surface normals and semantic labels provide a rich account of a scene. We have proposed a simple and fast multiscale architecture using convolutional networks that gives excellent performance on all three modalities. The models beat existing methods on the vast majority of benchmarks we explored. This is impressive given that many of these methods are specific to a single modality and often slower and more complex algorithms than ours. As such, our model provides a convenient new baseline for the three tasks. To this end, code and trained models can be found at http://cs.nyu.edu/~deigen/dnl/.

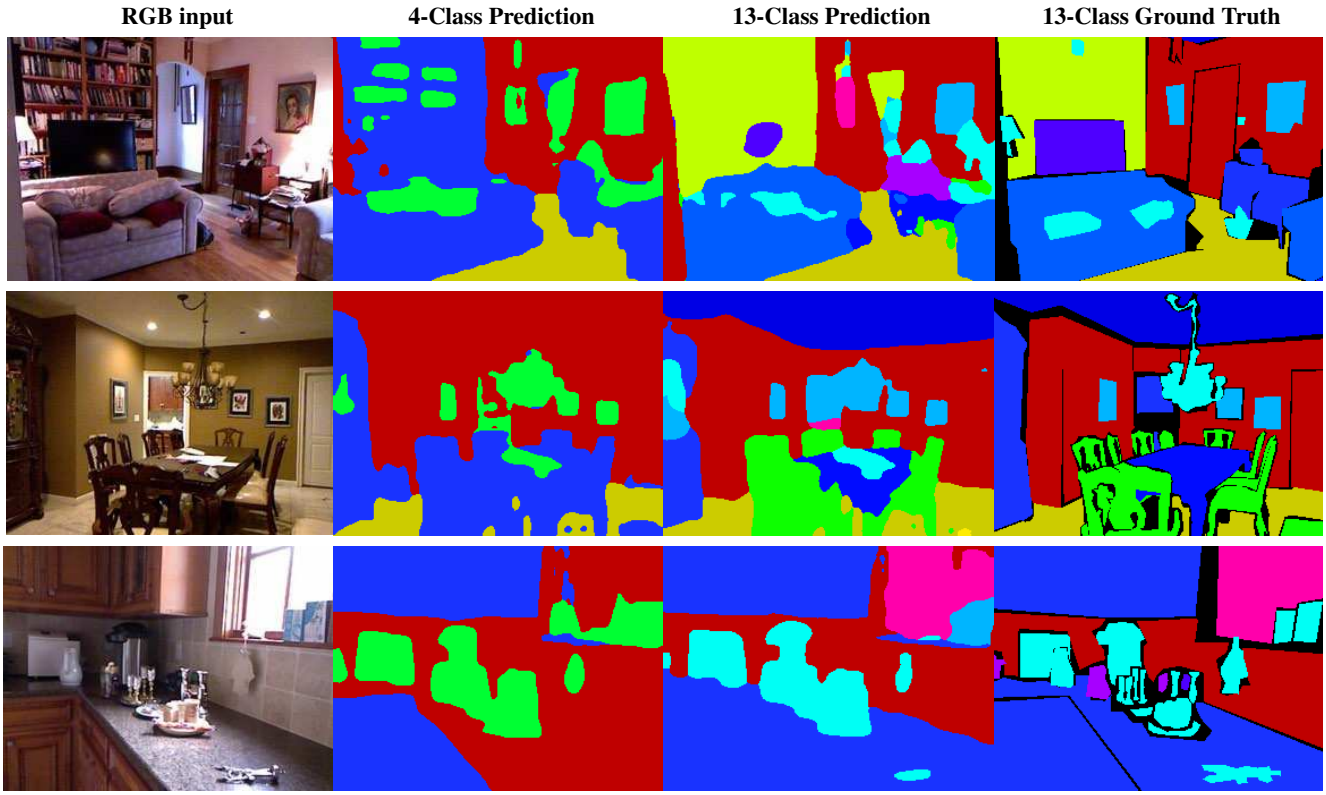


Figure 4. Example semantic labeling results for NYUDepth: (a) input image; (b) 4-class labeling result; (c) 13-class result; (d) 13-class ground truth.

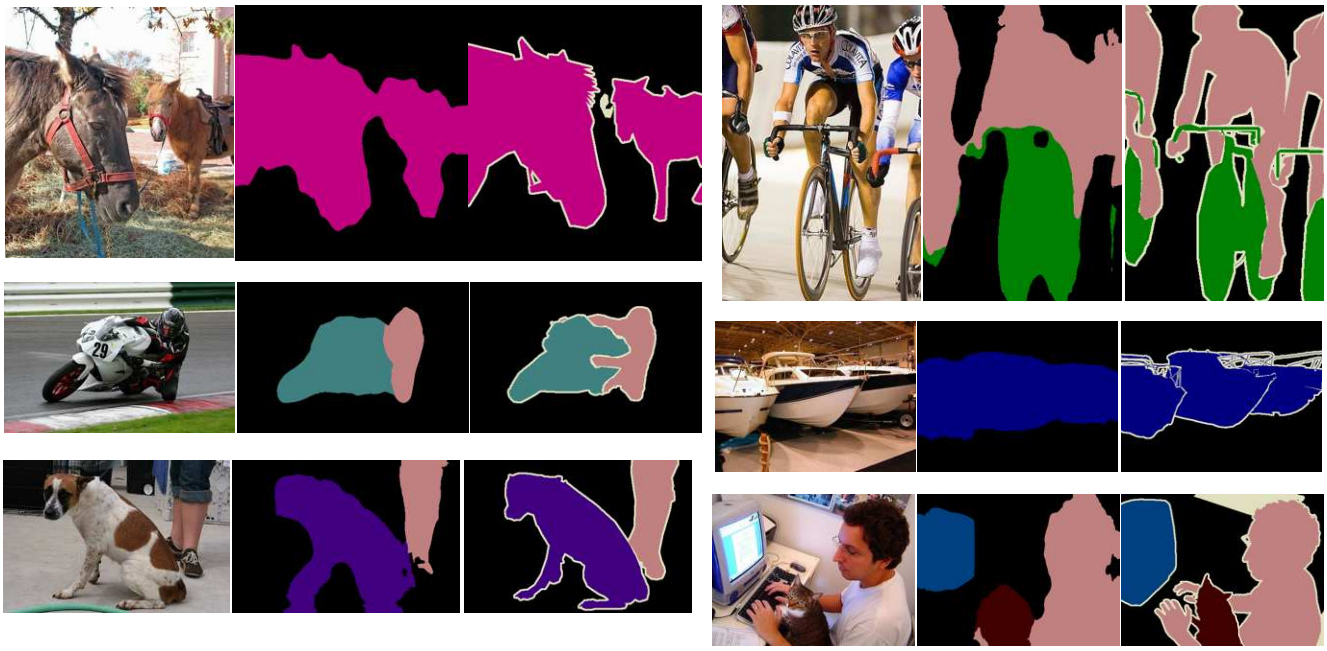


Figure 5. Example semantic labeling results for Pascal VOC 2011. For each image, we show RGB input, our prediction, and ground truth.

Acknowledgements

This work was supported by an ONR #N00014-13-1-0646 and an NSF CAREER grant.

References

[1] M. H. Baig and L. Torresani. Coarse-to-fine depth estimation from a single image via coupled regression and dictionary learning. *arXiv:1501.04537*, 2015. 5

[2] J. T. Barron and J. Malik. Intrinsic scene properties from a single rgb-d image. *CVPR*, 2013. 1, 2

[3] J. T. Barron and J. Malik. Shape, illumination, and reflectance from shading. *TPAMI*, 2015. 1, 2

[4] J. Carreira and C. Sminchisescu. Cpmc: Automatic object segmentation using constrained parametric min-cuts. *PAMI*, 2012. 2

[5] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *ICLR*, 2015. 6

[6] C. Couprie, C. Farabet, L. Najman, and Y. LeCun. Indoor semantic segmentation using depth information. *ICLR*, 2013. 2, 3, 5, 6

[7] J. Dai, K. He, and J. Sun. Convolutional feature masking for joint object and stuff segmentation. *arXiv 1412.1283*, 2014. 6

[8] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. *NIPS*, 2014. 1, 3, 5

[9] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Scene parsing with multiscale feature learning, purity trees, and optimal covers. *arXiv:1202.2160*, 2012. 2, 3, 6

[10] D. F. Fouhey, A. Gupta, and M. Hebert. Data-driven 3d primitives for single image understanding. In *ICCV*, 2013. 1, 5, 7

[11] D. F. Fouhey, A. Gupta, and M. Hebert. Unfolding an indoor origami world. In *ECCV*, 2014. 1, 5

[12] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CVPR*, 2014. 1

[13] S. Gupta, P. Arbelaez, and J. Malik. Perceptual organization and recognition of indoor scenes from rgb-d images. In *CVPR*, 2013. 2, 5, 6

[14] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik. Learning rich features from rgb-d images for object detection and segmentation. In *ECCV*. 2014. 1, 2, 4, 6

[15] S. K. Hameed, M. Bennamoun, F. Sohel, and R. Togneri. Geometry driven semantic labeling of indoor scenes. In *ECCV*. 2014. 2, 6

[16] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Simultaneous detection and segmentation. In *ECCV*. 2014. 6

[17] A. Hermans, G. Floros, and B. Leibe. Dense 3d semantic mapping of indoor scenes from rgb-d images. *ICRA*, 2014. 2, 6

[18] K. Karsch, C. Liu, S. B. Kang, and N. England. Depth extraction from video using non-parametric sampling. In *TPAMI*, 2014. 5

[19] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1, 3

[20] L. Ladicky, J. Shi, and M. Pollefeys. Pulling things out of perspective. In *CVPR*, 2014. 4, 5

[21] L. Ladicky, B. Zeisl, and M. Pollefeys. Discriminatively trained dense surface normal estimation. In *ECCV*, 2014. 1, 5, 7

[22] C. Liu, J. Yuen, A. Torralba, J. Sivic, and W. Freeman. Sift flow: dense correspondence across difference scenes. 2008. 6

[23] F. Liu, C. Shen, and G. Lin. Deep convolutional neural fields for depth estimation from a single image. *arXiv:1411.6387*, 2014. 5

[24] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014. 2, 6

[25] R. Memisevic and C. Conrad. Stereopsis via deep learning. In *NIPS Workshop on Deep Learning*, 2011. 1

[26] A. C. Muller and S. Behnke. Learning depth-sensitive conditional random fields for semantic segmentation of rgb-d images. *ICRA*, 2014. 2, 6

[27] M. Osadchy, Y. Le Cun, and M. L. Miller. Synergistic face detection and pose estimation with energy-based models. In *Toward Category-Level Object Recognition*, pages 196–206. Springer, 2006. 1

[28] P. Pinheiro and R. Collobert. Recurrent convolutional neural networks for scene labeling. In *ICML*, 2014. 2, 6

[29] X. Ren, L. Bo, and D. Fox. Rgb-(d) scene labeling: Features and algorithms. In *CVPR*, 2012. 2

[30] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *ICLR*, 2013. 1

[31] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012. 2, 4, 5

[32] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 1, 2, 3

[33] J. Stuckler, B. Waldvogel, H. Schulz, and S. Behnke. Dense real-time mapping of object-class semantics from rgb-d video. *J. Real-Time Image Processing*, 2014. 6

[34] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. 1

[35] J. Tighe and S. Lazebnik. Finding things: Image parsing with regions and per-exemplar detectors. In *CVPR*, 2013. 2, 6

[36] J. Tompson, A. Jain, Y. LeCun, and C. Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. *NIPS*, 2014. 1

[37] A. Wang, J. Lu, G. Wang, J. Cai, and T.-J. Cham. Multi-modal unsupervised feature learning for rgb-d scene labeling. In *ECCV*, 2014. 6

[38] X. Wang, D. F. Fouhey and A. Gupta. Designing deep networks for surface normal estimation. In *CVPR*, 2015. 1, 2, 5, 7

[39] J. Zbontar and Y. LeCun. Computing the stereo matching cost with a convolutional neural network. *CoRR*, abs/1409.4326, 2014. 1

High-for-Low and Low-for-High: Efficient Boundary Detection from Deep Object Features and its Applications to High-Level Vision

Gedas Bertasius
University of Pennsylvania
gberta@seas.upenn.edu

Jianbo Shi
University of Pennsylvania
jshi@seas.upenn.edu

Lorenzo Torresani
Dartmouth College
lt@dartmouth.edu

Abstract

Most of the current boundary detection systems rely exclusively on low-level features, such as color and texture. However, perception studies suggest that humans employ object-level reasoning when judging if a particular pixel is a boundary. Inspired by this observation, in this work we show how to predict boundaries by exploiting object-level features from a pretrained object-classification network. Our method can be viewed as a High-for-Low approach where high-level object features inform the low-level boundary detection process. Our model achieves state-of-the-art performance on an established boundary detection benchmark and it is efficient to run.

Additionally, we show that due to the semantic nature of our boundaries we can use them to aid a number of high-level vision tasks. We demonstrate that by using our boundaries we improve the performance of state-of-the-art methods on the problems of semantic boundary labeling, semantic segmentation and object proposal generation. We can view this process as a Low-for-High scheme, where low-level boundaries aid high-level vision tasks.

Thus, our contributions include a boundary detection system that is accurate, efficient, generalizes well to multiple datasets, and is also shown to improve existing state-of-the-art high-level vision methods on three distinct tasks.

1. Introduction

In the vision community, boundary detection has always been considered a low-level problem. However, psychological studies suggest that when a human observer perceives boundaries, object level reasoning is used [13, 25, 17]. Despite these findings, most of the boundary detection methods rely exclusively on low-level color and gradient features. In this work, we present a method that uses object-level features to detect boundaries. We argue that using object-level information to predict boundaries is more similar to how humans reason. Our boundary detection scheme can be viewed as a *High-for-Low* approach where we use high-level object features as cues for a low-level boundary detection process. Throughout the rest of the paper, we refer to our proposed boundaries as *High-for-Low* boundaries (HFL).

	Low-Level Task	High-Level Tasks			
	BD	SBL		SS	OP
	ODS	MF	AP	PI-IOU	MR
SotA	0.76 [27]	28.0 [11]	19.9 [11]	45.8 [5]	0.88 [31]
HFL	**0.77**	**62.5**	**54.6**	**48.8**	**0.90**

Table 1: Summary of results achieved by our proposed method (HFL) and state-of-the-art methods (SotA). We provide results on four vision tasks: Boundary Detection (BD), Semantic Boundary Labeling (SBL), Semantic Segmentation (SS), and Object Proposal (OP). The evaluation metrics include ODS F-score for BD task, max F-score (MF) and average precision (AP) for SBL task, per image intersection over union (PI-IOU) for SS task, and max recall (MR) for OP task. Our method produces better results in each of these tasks according to these metrics.

We present an efficient deep network that uses object-level information to predict the boundaries. Our proposed architecture reuses features from the sixteen convolutional layers of the network of Simonyan et al. [29], which we refer to as VGG net. The VGG net has been trained for object classification, and therefore, reusing its features allows our method to utilize high-level object information to predict HFL boundaries. In the experimental section, we demonstrate that using object-level features produces semantically meaningful boundaries and also achieves above state-of-the-art boundary detection accuracy.

Additionally, we demonstrate that we can successfully apply our HFL boundaries to a number of high-level vision tasks. We show that by using HFL boundaries we improve the results of three existing state-of-the-art methods on the tasks of semantic boundary labeling, semantic segmenta-

tion and object proposal generation. Therefore, using HFL boundaries to boost the results in high level vision tasks can be viewed as a *Low-for-High* scheme, where boundaries serve as low-level cues to aid high-level vision tasks.

We present the summarized results for the boundary detection and the three mentioned high-level vision tasks in Table 1. Specifically, we compare our proposed method and an appropriate state-of-the-art method for that task. As the results indicate, we achieve better results in each of the tasks for each presented evaluation metric. We present more detailed results for each of these tasks in the later sections.

In summary, our contributions are as follows. First, we show that using object-level features for boundary detection produces perceptually informative boundaries that outperform prior state-of-the-art boundary detection methods. Second, we demonstrate that we can use HFL boundaries to enhance the performance on the high-level vision tasks of semantic boundary labeling, semantic segmentation and object proposal. Finally, our method can detect boundaries in near-real time. Thus, we present a boundary detection system that is accurate, efficient, and is also applicable to high level vision tasks.

2. Related Work

Most of the contour detection methods predict boundaries based purely on color, text, or other low-level features. We can divide these methods into three broad categories: spectral methods, supervised discriminative methods and deep learning based methods.

Spectral methods formulate contour detection problem as an eigenvalue problem. The solution to this problem is then used to reason about the boundaries. The most successful approaches in this genre are the MCG detector [2], gPb detector [1], PMI detector [14], and Normalized Cuts [28].

Some of the notable discriminative boundary detection methods include sketch tokens (ST) [18], structured edges (SE) [6] and sparse code gradients (SCG) [23]. While SCG use supervised SVM learning [4], the latter two methods rely on a random forest classifier and models boundary detection as a classification task.

Recently there have been attempts to apply deep learning to the task of boundary detection. SCT [20] is a sparse coding approach that reconstructs an image using a learned dictionary and then detect boundaries. Both N^4 fields [10] and DeepNet [16] approaches use Convolutional Neural Networks (CNNs) to predict edges. N^4 fields rely on dictionary learning and the use of the Nearest Neighbor algorithm within a CNN framework while DeepNet uses a traditional CNN architecture to predict contours.

The most similar to our approach is DeepEdge [3], which uses a multi-scale bifurcated network to perform contour detection using object-level features. However, we show that our method achieves better results even without the complicated multi-scale and bifurcated architecture of DeepEdge. Additionally, unlike DeepEdge, our system can run in near-real time.

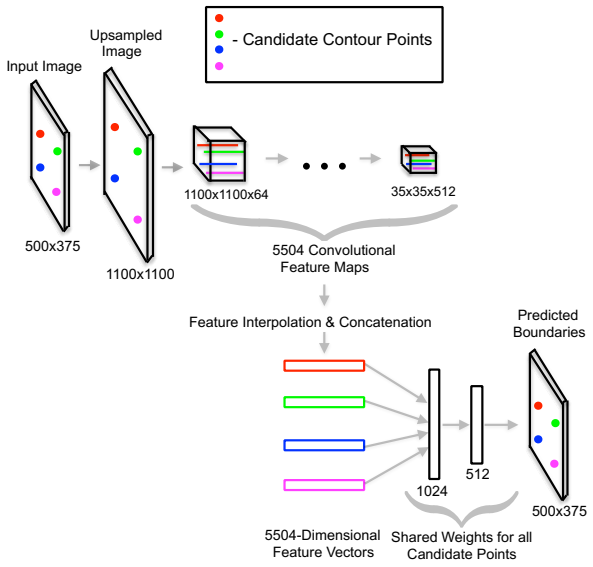


Figure 1: An illustration of our architecture (best viewed in color). First we extract a set of candidate contour points. Then we upsample the image and feed it through 16 convolutional layers pretrained for object classification. For each candidate point, we find its correspondence in each of the feature maps and perform feature interpolation. This yields a 5504-dimensional feature vector for each candidate point. We feed each of these vectors to two fully connected layers and store the predictions to produce a final boundary map.

In comparison to prior approaches, we offer several contributions. First, we propose to use object-level information to predict boundaries. We argue that such an approach leads to semantic boundaries, which are more consistent with humans reasoning. Second, we avoid feature engineering by learning boundaries from human-annotated data. Finally, we demonstrate excellent results for both low-level and high-level vision tasks. For the boundary detection task, our proposed HFL boundaries outperform all of the prior methods according to both F-score metrics. Additionally, we show that because HFL boundaries are based on object-level features, they can be used to improve performance in the high-level vision tasks of semantic boundary labeling, semantic segmentation, and object proposal generation.

3. Boundary Detection

In this section, we describe our proposed architecture and the specific details on how we predict HFL boundaries using our method. The detailed illustration of our architecture is presented in Figure 1.

Figure 2: A visualization of selected convolutional feature maps from VGG network (resized to the input image dimension). Because VGG was optimized for an object classification task, it produces high activation values on objects and their parts.

3.1. Selection of Candidate Contour Points

We first extract a set of candidate contour points with a high recall. Due to its efficiency and high recall performance, we use the SE edge detector [6]. In practice, we could eliminate this step and simply try to predict boundaries at every pixel. However, selecting a set of initial candidate contour points, greatly reduces the computational cost. Since our goal is to build a boundary detector that is both accurate and efficient, we use these candidate points to speed up the computation of our method.

3.2. Object-Level Features

After selecting candidate contour points, we up-sample the original input image to a larger dimension (for example 1100×1100). The up-sampling is done to minimize the loss of information due to the input shrinkage caused by pooling at the different layers. Afterwards, we feed the up-sampled image through 16 convolutional layers of the VGG net.

We use the VGG net as our model because it has been trained to recognize a large number of object classes (the 1000 categories of the ImageNet dataset [24]) and thus encodes object-level features that apply to many classes. To preserve specific location information we utilize only the 16 convolutional layers of the VGG net. We don't use fully connected layers because they don't preserve spatial information, which is crucial for accurate boundary detection.

We visualize some of the selected convolutional maps in Figure 2. Note the high activation values around the various objects in the images, which confirms our hypothesis that the VGG net encodes object specific information in its convolutional feature maps.

3.3. Feature Interpolation

Similarly to [26, 12, 19], we perform feature interpolation in deep layers. After the up-sampled image passes through all 16 convolutional layers, for each selected candidate contour point we find its corresponding point in the feature maps. Due to the dimension differences in convolutional maps these correspondences are not exact. Thus we perform feature interpolation by finding the four nearest points and averaging their activation values. This is done in each of the 5504 feature maps. Thus, this results in a 5504-dimensional vector for each candidate point.

We note that the interpolation of convolutional feature maps is the crucial component that enables our system to predict the boundaries efficiently. Without feature interpolation, our method would need to independently process the candidate edge points by analyzing a small image patch around each point, as for example done in DeepEdge [3] which feeds one patch at a time through a deep network. However, when the number of candidate points is large (e.g., DeepEdge considers about 15K points at each of 4 different scales), their patches overlap significantly and thus a large amount of computation is wasted by recalculating filter response values over the same pixels. Instead, we can compute the features for all candidate points with a single pass through the network by performing deep convolution over the *entire* image (i.e., feeding the entire image rather than one patch at a time) and then by interpolating the convolutional feature maps at the location of each candidate edge point so as to produce its feature descriptor. Thanks to this speedup, our method has a runtime of 1.2 seconds (using a K40 GPU), which is better than the runtimes of prior deep-learning based edge detection methods [27, 10, 16, 3].

3.4. Learning to Predict Boundaries

After performing feature interpolation, we feed the 5504-dimensional feature vectors corresponding to each of the candidate contour points to two fully connected layers that are optimized to the human agreement criterion. To be more precise, we define our prediction objective as a fraction of human annotators agreeing on the presence of the boundary at a particular pixel. Therefore, a learning objective aims at mimicking the judgement of the human labelers.

Finally, to detect HFL boundaries, we accumulate the predictions from the fully connected layers for each of the

candidate points and produce a boundary probability map as illustrated in Figure 1.

3.5. Implementation Details

In this section, we describe the details behind the training procedure of our model. We use the Caffe library [15] to implement our network architecture.

In the training stage, we freeze the weights in all of the convolutional layers. To learn the weights in the two fully connected layers we train our model to optimize the least squares error of the regression criterion that we described in the previous subsection. To enforce regularization we set a dropout rate of 0.5 in the fully connected layers.

Our training dataset includes $80K$ points from the BSDS500 dataset [22]. As described in the previous subsection, the labels represent the fraction of human annotators agreeing on the boundary presence. We divide the label space into four quartiles, and select an equal number of samples for each quartile to balance the training dataset. In addition to the training dataset, we also sample a hold-out dataset of size $40,000$. We use this for the hard-positive mining [21] in order to reduce the number of false-negative predictions.

For the first 25 epochs we train the network on the original $80,000$ training samples. After the first 25 epochs, we test the network on the hold-out dataset and detect false negative predictions made by our network. We then augment the original $80,000$ training samples with the false negatives and the same number of randomly selected true negatives. For the remaining 25 epochs, we train the network on this augmented dataset.

3.6. Boundary Detection Results

In this section, we present our results on the BSDS500 dataset [22], which is the most established benchmark for boundary detection. The quality of the predicted boundaries is evaluated using three standard measures: fixed contour threshold (ODS), per-image best threshold (OIS), and average precision (AP).

We compare our approach to the state-of-the-art based on two different sets of BSDS500 ground truth boundaries. First, we evaluate the accuracy by matching each of the predicted boundary pixels with the ground truth boundaries that were annotated by *any* of the human annotators. This set of ground truth boundaries is referred to as "any". We present the results for "any" ground truth boundaries in the lower half of Table 2. As indicated by the results, HFL boundaries outperform all the prior methods according to both F-score measures.

Recently, there has been some criticism raised about the procedure for boundary detection evaluation on the BSDS500 dataset. One issue with the BSDS500 dataset involves the so called "orphan" boundaries: the boundaries that are marked by only one or two human annotators. These "orphan" boundaries comprise around 30% of BSDS500 dataset but most of them are considered uninformative. However, the standard evaluation benchmark rewards the methods that predict these boundaries. To resolve

Consensus GT	*ODS*	*OIS*	*AP*	*FPS*
SCG [23]	0.6	0.64	0.56	1/280
DeepNet [16]	0.61	0.64	0.61	$1/5^{\ddagger}$
PMI [14]	0.61	**0.68**	0.56	1/900
DeepEdge [3]	0.62	0.64	0.64	$1/1000^{\ddagger}$
N^4-fields [10]	0.64	0.67	0.64	$1/6^{\ddagger}$
HFL	**0.65**	**0.68**	**0.67**	$5/6^{\ddagger}$
Any GT	*ODS*	*OIS*	*AP*	*FPS*
SE [6]	0.75	0.77	0.80	**2.5**
MCG [2]	0.75	0.78	0.76	1/24
N^4-fields [10]	0.75	0.77	0.78	$1/6^{\ddagger}$
DeepEdge [3]	0.75	0.77	**0.81**	$1/1000^{\ddagger}$
MSC [30]	0.76	0.78	0.79	-
DeepContour [27]	0.76	0.77	0.8	$1/30^{\ddagger}$
HFL	**0.77**	**0.79**	0.8	$5/6^{\ddagger}$

Table 2: Boundary detection results on BSDS500 benchmark. Upper half of the table illustrates the results for "consensus" ground-truth criterion while the lower half of the table depicts the results for "any" ground-truth criterion. In both cases, our method outperforms all prior methods according to both ODS (optimal dataset scale) and OIS (optimal image scale) metrics. We also report the run-time of our method (‡ GPU time) in the FPS column (frames per second), which shows that our algorithm is faster than prior approaches based on deep learning [27, 10, 16, 3].

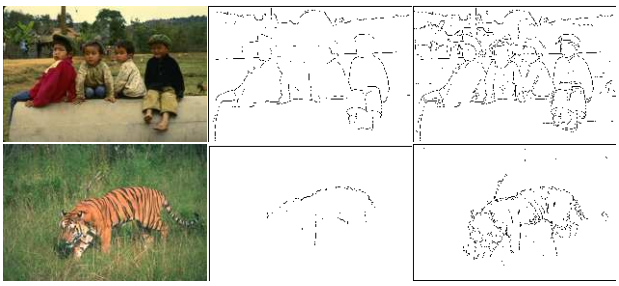


Figure 5: Qualitative results on the BSDS benchmark. The first column of images represent input images. The second column illustrates SE [6], while the third column depicts HFL boundaries. Notice that SE boundaries are predicted with low confidence if there is no significant change in color between the object and the background. Instead, because our model is defined in terms of object-level features, it can predict object boundaries with high confidence even if there is no significant color variation in the scene.

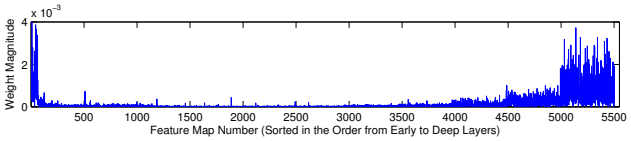


Figure 6: We train a linear regression model and visualize its weight magnitudes in order to understand which features are used most heavily in the boundary prediction (this linear regression is used only for the visualization purposes and not for the accuracy analysis). Note how heavily weighted features lie in the deepest layers of the network, i.e., the layers that are most closely associated with object information.

this issue we also evaluate our HFL boundaries on the so called "consensus" set of ground truth boundaries. These "consensus" boundaries involve only boundaries that are marked by *all* of the human annotators and hence, are considered perceptually meaningful. In the upper half of Table 2, we present the results achieved by our method on the "consensus" set of the ground truth boundaries. Our HFL boundaries outperform or tie all the prior methods in each of the three evaluation metrics, thus suggesting that HFL boundaries are similar to the boundaries that humans annotated. We also report the runtimes in Table 2 and note that our method runs faster than previous deep-learning based edge detection systems [27, 10, 16, 3].

Our proposed model computes a highly nonlinear function of the 5504-dimensional feature vector of each candidate point. Thus, it is difficult to assess which features are used most heavily by our edge predictor. However, we can gain a better insight by replacing the nonlinear function with a simple linear model. In Fig. 6 we show the weight magnitudes of a simple linear regression model (we stress that this linear model is used only for feature visualization purposes). From this Figure, we observe that many important features are located in the deepest layers of the VGG network. As shown in [7], these layers encode high-level object information, which confirms our hypothesis that high-level information is useful for boundary detection.

Finally, we present some qualitative results achieved by our method in Figure 5. These examples illustrate the effective advantage that HFL boundaries provide over another state-of-the-art edge detection system, the SE system [6]. Specifically, observe the parts of the image where there is a boundary that separates an object from the background but where the color change is pretty small. Notice that because the SE boundary detection is based on low-level color and texture features, it captures these boundaries with very low confidence. In comparison, because HFL boundaries rely on object-level features, it detects these boundaries with high confidence.

4. High-Level Vision Applications

In this section, we describe our proposed *Low-for-High* pipeline: using low-level boundaries to aid a number of high-level vision tasks. We focus on the tasks of semantic boundary labeling, semantic segmentation and object proposal generation. We show that using HFL boundaries improves the performance of state-of-the-art methods in each of these high-level vision tasks.

4.1. Semantic Boundary Labeling

The task of semantic boundary labeling requires not only to predict the boundaries but also to associate a specific object class to each of the boundaries. This implies that given our predicted boundaries we also need to label them with object class information. We approach this problem by adopting the ideas from the recent work on Fully Convolutional Networks (FCN) [19]. Given an input image, we concurrently feed it to our boundary-predicting network (described in Section 3), and also through the FCN that was pretrained for 20 Pascal VOC classes and the background class. While our proposed network produces HFL boundaries, the FCN model predicts class probabilities for each of the pixels. We can then merge the two output maps as follows. For a given boundary point we consider a 9×9 grid around that point from each of the 21 FCN object-class probability maps. We calculate the maximum value inside each grid, and then label the boundary at a given pixel with the object-class that corresponds to the maximum probability across these 21 maps. We apply this procedure for each of the boundary points, in order to associate object-class labels to the boundaries. Note that we consider the grids around the boundary pixel because the output of the FCN has a poor localization, and considering the grids rather than individual pixels leads to higher accuracy.

We can also merge HFL boundaries with the state-of-the-art DeepLab-CRF segmentation [5] to obtain higher accuracy. We do this in a similar fashion as just described. First, around a given boundary point we extract a 9×9 grid from the DeepLab-CRF segmentation. We then compute the mode value in the grid (excluding the background class), and use the object-class corresponding to the mode value as a label for the given boundary point. We do this for each of the boundary points. By merging HFL boundaries and the output of FCN or DeepLab-CRF, we get semantic boundaries that are highly localized and also contain object-specific information.

4.1.1 Semantic Boundary Labeling Results

In this section, we present semantic boundary labeling results on the SBD dataset [11], which includes ground truth boundaries that are also labeled with one of 20 Pascal VOC classes. The boundary detection accuracy for each class is

Method (Metric)	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mean
InvDet (MF)	42.6	49.5	15.7	16.8	36.7	43.0	40.8	22.6	18.1	26.6	10.2	18.0	35.2	29.4	48.2	14.3	26.8	11.2	22.2	32.0	28.0
HFL-FC8 (MF)	71.6	59.6	68.0	54.1	57.2	68.0	58.8	69.3	43.3	65.8	33.3	67.9	67.5	62.2	69.0	43.8	68.5	33.9	57.7	54.8	58.7
HFL-CRF (MF)	**73.9**	**61.4**	**74.6**	**57.2**	**58.8**	**70.4**	**61.6**	**71.9**	**46.5**	**72.3**	**36.2**	**71.1**	**73.0**	**68.1**	**70.3**	**44.4**	**73.2**	**42.6**	**62.4**	**60.1**	**62.5**
InvDet (AP)	38.4	29.6	9.6	9.9	24.2	33.6	31.3	17.3	10.7	16.4	3.7	12.1	28.5	20.4	45.7	7.6	16.1	5.7	14.6	22.7	19.9
HFL-FC8 (AP)	66.0	50.7	58.9	40.6	47.1	62.9	51.0	59.0	25.6	54.6	15.3	57.8	57.3	55.9	62.2	27.5	55.6	18.0	50.1	40.6	47.8
HFL-CRF (AP)	**71.2**	**55.2**	**69.3**	**45.7**	**48.9**	**71.1**	**56.8**	**65.7**	**29.1**	**65.9**	**17.7**	**64.5**	**68.3**	**64.7**	**65.9**	**29.1**	**66.5**	**25.7**	**60.0**	**49.8**	**54.6**

Table 3: Results of semantic boundary labeling on the SBD benchmark using the Max F-score (MF) and Average Precision (AP) metrics. Our method (HFL) outperforms Inverse Detectors [11] for all 20 categories according to both metrics. Note that using the CRF output to label the boundaries produces better results than using the outputs from the FC8 layer of FCN.

evaluated using the maximum F-score (MF), and average precision (AP) measures.

Labeling boundaries with the semantic object information is a novel and still relatively unexplored problem. Therefore, we found only one other approach (Inverse Detectors) that tried to tackle this problem [11]. The basic idea behind Inverse Detectors consists of several steps. First, generic boundaries in the image are detected. Then, a number of object proposal boxes are generated. These two sources of information are then used to construct the features. Finally, a separate classifier is used to label the boundaries with the object-specific information.

Table 3 shows that our approach significantly outperforms Inverse Detectors according to both the maximum F-score and the average precision metrics for all twenty categories. As described in Section 4.1 we evaluate the two variants of our method. Denoted by HFL-FC8 is the variant for which we label HFL boundaries with the outputs from the last layer (FC8) of the pretrained FCN. We denote with HFL-CRF the result of labeling our boundaries with the output from the DeepLab-CRF [5]. Among these two variants, we show that the latter one produces better results. This is expected since the CRF framework enforces spatial coherence in the semantic segments.

In Figure 7, we present some of the qualitative results produced by our method. We note that even with multiple objects in the image, our method successfully recognizes and localizes boundaries of each of the classes.

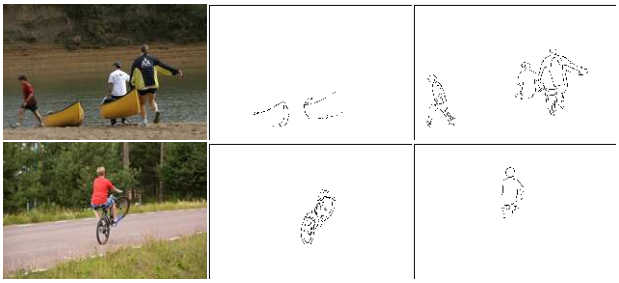


Figure 7: A visualization of the predicted semantic boundary labels. Images in the first column are input examples. Columns two and three show semantic HFL boundaries of different object classes. Note that even with multiple objects appearing simultaneously, our method outputs precise semantic boundaries.

4.2. Semantic Segmentation

For the semantic segmentation task, we propose to enhance the DeepLab-CRF [5] with our predicted HFL boundaries. DeepLab-CRF is a system comprised of a Fully Convolutional Network (described in Section 4.1) and a dense CRF applied on top of FCN predictions.

Specifically, in the CRF, the authors propose to use a Gaussian kernel and a bilateral term including position and color terms as the CRF features (see [5]). While in most cases the proposed scheme works well, DeepLab-CRF sometimes produces segmentations that are not spatially coherent, particularly for images containing small object regions.

We propose to address this issue by adding features based on our predicted HFL boundaries in the CRF framework. Note that we use predicted boundaries from Section 3 and not the boundaries labeled with the object information that we obtained in Section 4.1. We use the Normalized Cut [28] framework to generate our features.

First, we construct a pixel-wise affinity matrix $\mathbf{W}$ using our HFL boundaries. We measure the similarity between two pixels as:

$$W_{ij} = \exp\left(-\max_{p \in \overline{ij}} \{\frac{M(p)^2}{\sigma^2}\}\right)$$

where W_{ij} represents the similarity between pixels i and j, p denotes the boundary point along the line segment $\overline{ij}$ connecting pixels i and j, M depicts the magnitude of the boundary at pixel p, and σ denotes the smoothness parameter, which is usually set to 14% of the maximum boundary value in the image.

The intuitive idea is that two pixels are similar (i.e. $W_{ij} = 1$) if there is no boundary crossing the line connecting these two pixels (i.e. $M(p) = 0 \quad \forall p \in \overline{ij}$) or if the boundary strength is low. We note that it is not necessary to build a full affinity matrix W. We build a sparse affin-

Metric	Method (Dataset)	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mean
PP-IOU	DL-CRF (VOC)	**78.6**	41.1	**83.5**	**75.3**	72.9	83.1	**76.6**	**80.8**	**37.8**	**72.1**	66.5	**64.7**	65.8	**75.7**	**80.5**	**34.4**	**75.9**	**47.4**	86.6	**77.9**	**68.9**
	DL-CRF+HFL (VOC)	77.9	**41.2**	83.1	74.4	**73.2**	**85.5**	76.1	80.6	35.7	71.0	**66.6**	64.3	**65.9**	75.2	80.2	32.8	75.2	47.0	**87.1**	**77.9**	68.5
	DL-CRF (SBD)	74.2	68.0	**81.9**	64.6	**71.8**	86.3	**78.3**	**84.3**	**41.6**	**78.0**	49.9	**82.0**	**78.5**	77.1	80.1	**54.3**	**75.6**	**49.8**	**79.5**	70.1	**71.4**
	DL-CRF+HFL (SBD)	**75.1**	**69.2**	81.6	**64.8**	71.3	**86.4**	78.1	84.1	41.2	77.8	**50.4**	81.6	78.2	**78.5**	**80.7**	53.8	74.9	49.1	**79.5**	**70.4**	**71.4**
PI-IOU	DL-CRF (VOC)	46.1	**28.0**	48.5	54.5	45.5	57.6	34.1	**47.3**	19.5	**61.4**	**41.6**	42.5	34.4	61.8	**62.1**	**22.1**	50.5	41.0	61.2	31.9	44.6
	DL-CRF+HFL (VOC)	**47.5**	27.6	**50.4**	**63.5**	**47.7**	**57.9**	**38.7**	47.2	**21.1**	57.3	41.2	**43.7**	**36.0**	**66.4**	61.1	21.3	**53.9**	**42.1**	**70.9**	**34.6**	**46.5**
	DL-CRF (SBD)	59.4	36.5	58.0	38.6	32.0	58.1	44.7	59.6	25.8	51.8	28.1	59.0	46.9	50.3	61.8	22.2	45.9	33.4	62.1	41.0	45.8
	DL-CRF+HFL (SBD)	**63.4**	**42.5**	**58.4**	**41.3**	**32.5**	**61.2**	**45.7**	**61.4**	**28.4**	**55.5**	**31.5**	**61.4**	**51.8**	**54.6**	**62.1**	**24.9**	**52.6**	**34.2**	**67.1**	**45.1**	**48.8**

Table 4: Semantic segmentation results on the SBD and VOC 2007 datasets. We measure the results according to PP-IOU (per pixel) and PI-IOU (per image) evaluation metrics. We denote the original DeepLab-CRF system and our proposed modification as DL-CRF and DL-CRF+HFL, respectively. According to the PP-IOU metric, our proposed features (DL-CRF+HFL) yield almost equivalent results as the original DeepLab-CRF system. However, based on PI-IOU metric, our proposed features improve the mean accuracy by 3% and 1.9% on SBD and VOC 2007 datasets respectively.

ity matrix connecting every pair of pixels i and j that have distance 5 or less from each other.

After building a boundary-based affinity matrix $\mathbf{W}$ we set $D_{ii} = \sum_{i \neq j} W_{ij}$ and compute eigenvectors $\mathbf{v}$ of the generalized eigenvalue system:

$$(\mathbf{D} - \mathbf{W})\mathbf{v} = \lambda \mathbf{D}\mathbf{v}$$

We then resize the eigenvectors v to the original image dimensions, and use them as additional features to the CRF part of DeepLab-CRF system. In our experiments, we use the 16 eigenvectors corresponding to the smallest eigenvalues, which results in 16 extra feature channels.

Note that the eigenvectors contain soft segmentation information. Because HFL boundaries predict object-level contours with high confidence, the eigenvectors often capture regions corresponding to objects. We visualize a few selected eigenvectors in Figure 8. In the experimental section, we demonstrate that our proposed features make the output produced by DeepLab-CRF more spatially coherent and improve the segmentation accuracy according to one of the metrics.

We also note that our proposed features are applicable to any generic method that incorporates CRF. For instance, even if DeepLab-CRF used an improved DeepLab network architecture, our features would still be beneficial because they contribute directly to the CRF part and not the DeepLab network part of the system.

4.2.1 Semantic Segmentation Results

In this section, we present semantic segmentation results on the SBD [11] and also Pascal VOC 2007 [8] datasets, which both provide ground truth segmentations for 20 Pascal VOC classes. We evaluate the results in terms of two metrics. The first metric measures the accuracy in terms of pixel intersection-over-union averaged per pixel (PP-IOU) across the 20 classes. According to this metric, the accuracy is computed on a per pixel basis. As a result, the images that contain large object regions are given more importance.

Figure 8: In this figure, the first column depicts an input image while the second and third columns illustrate two selected eigenvectors for that image. The eigenvectors contain soft segmentation information. Because HFL boundaries capture object-level boundaries, the resulting eigenvectors primarily segment regions corresponding to the objects.

We observe that while DeepLab-CRF works well on the images containing large object regions, it produces spatially disjoint outputs for the images with smaller and object regions (see Figure 9). This issue is often being overlooked, because according to the PP-IOU metric, the images with large object regions are given more importance and thus contribute more to the accuracy. However, certain applications may require accurate segmentation of small objects. Therefore, in addition to PP-IOU, we also consider the PI-IOU metric (pixel intersection-over-union averaged per image across the 20 classes), which gives equal weight to each of the images.

For both of the metrics we compare the semantic segmentation results of a pure DeepLab-CRF [5] and also a modification of DeepLab-CRF with our proposed features added to the CRF framework. We present the results for both of the metrics in Table 4.

Based on these results, we observe that according to the first metric (PP-IOU), our proposed features yield almost equivalent results as the original DeepLab-CRF system. However, according to the second metric (PI-IOU) our features yield an average improvement of 3% and 1.9% in

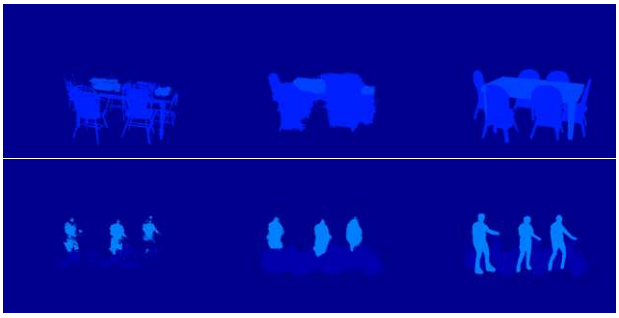


Figure 9: An illustration of the more challenging semantic segmentation examples. The first column depicts the predictions achieved by DeepLab-CRF, while the second column illustrates the results after adding our proposed features to the CRF framework. The last column represents ground truth segmentations. Notice how our proposed features render the predicted semantic segments more spatially coherent and overall more accurate.

SBD and VOC 2007 datasets respectively.

We also visualize the qualitative results produced by both approaches in Figure 9. Notice how our proposed features make the segmentations look smoother relative to the segmentations produced by the original DeepLab-CRF system.

Once again, we want to stress that our HFL features are applicable to any method that uses the CRF. Therefore, based on the results presented in this section, we believe that our proposed features could be beneficial in a wide array of problems that involve the use of the CRF framework.

4.3. Object Proposals

Finally, we show that our method produces object-level boundaries that can be successfully exploited in an object proposal scheme. Specifically we adopt the EdgeBoxes approach [31], which can be applied to any generic boundaries to produce a list of object proposal boxes. The original EdgeBoxes method uses SE boundaries to generate the boxes. However, SE boundaries are predicted using low-level color and texture features, rather than object-level features. Thus, here we validate the hypothesis that the EdgeBoxes proposals can be improved by replacing the SE boundaries with our HFL boundaries.

4.3.1 Object Proposal Results

In this section, we present object proposal results on the Pascal VOC 2012 dataset [9]. We evaluate the quality of bounding-box proposals according to three metrics: area under the curve (AUC), the number of proposals needed to reach recall of 75%, and the maximum recall over 5000 object bounding-boxes. Additionally, we compute the accuracy for each of the metrics for three different intersection over union (IOU) values: $0.65, 0.7$, and 0.75. We present these results in Table 5. As described in Section 4.3, we use EdgeBoxes [31], a package that uses generic boundaries, to generate object proposals. We compare the quality of the generated object proposals when using SE boundaries and HFL boundaries. We demonstrate that for each IOU value and for each of the three evaluation metrics, HFL boundaries produce better or equivalent results. This confirms our hypothesis that HFL boundaries can be used effectively for high-level vision tasks such as generating object proposals.

Method	IoU 0.65			IoU 0.7			IoU 0.75		
	AUC	N@75%	Recall	AUC	N@75%	Recall	AUC	N@75%	Recall
SE	0.52	413	0.93	0.47	658	0.88	**0.41**	inf	0.75
HFL	**0.53**	**365**	**0.95**	**0.48**	**583**	**0.9**	**0.41**	**2685**	**0.77**

Table 5: Comparison of object proposal results. We compare the quality of object proposals using Structured Edges [6] and HFL boundaries. We evaluate the performance for three different IOU values and demonstrate that using HFL boundaries produces better results for each evaluation metric and for each IOU value.

5. Conclusions

In this work, we presented an efficient architecture that uses object-level information to predict semantically meaningful boundaries. Most prior edge detection methods rely exclusively on low-level features, such as color or texture, to detect the boundaries. However, perception studies suggest that humans employ object-level reasoning when deciding whether a given pixel is a boundary [13, 25, 17]. Thus, we propose a system that focuses on the semantic object-level cues rather than low level image information to detect the boundaries. For this reason we refer to our boundary detection scheme as a *High-for-Low* approach, where high-level object features inform the low-level boundary detection process. In this paper we demonstrated that our proposed method produces boundaries that accurately separate objects and the background in the image and also achieve higher F-score compared to any prior work.

Additionally, we showed that, because HFL boundaries are based on object-level features, they can be employed to aid a number of high level vision tasks in a *Low-for-High* fashion. We use our boundaries to boost the accuracy of state-of-the-art methods on the high-level vision tasks of semantic boundary labeling, semantic segmentation, and object proposals generation. We show that using HFL boundaries leads to better results in each of these tasks.

To conclude, our boundary detection method is accurate, efficient, applicable to a variety of datasets, and also useful for multiple high-level vision tasks. We plan to release the source code for HFL upon the publication of the paper .

6. Acknowledgements

We thank Mohammad Haris Baig for the suggestions and help with the software. This research was funded in part by NSF award CNS-1205521.

References

[1] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, May 2011. 2

[2] P. Arbelaez, J. Pont-Tuset, J. Barron, F. Marqués, and J. Malik. Multiscale combinatorial grouping. In *Computer Vision and Pattern Recognition (CVPR)*, 2014. 2, 4

[3] G. Bertasius, J. Shi, and L. Torresani. Deepedge: A multi-scale bifurcated deep network for top-down contour detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. 2, 3, 4, 5

[4] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998. 2

[5] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *CoRR*, abs/1412.7062, 2014. 1, 5, 6, 7

[6] P. Dollár and C. L. Zitnick. Fast edge detection using structured forests. *PAMI*, 2015. 2, 3, 4, 5, 8

[7] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *CoRR*, abs/1310.1531, 2013. 5

[8] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html. 7

[9] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html. 8

[10] Y. Ganin and V. S. Lempitsky. N^4-fields: Neural network nearest neighbor fields for image transforms. *ACCV*, 2014. 2, 3, 4, 5

[11] B. Hariharan, P. Arbelaez, L. Bourdev, S. Maji, and J. Malik. Semantic contours from inverse detectors. In *International Conference on Computer Vision (ICCV)*, 2011. 1, 5, 6, 7

[12] B. Hariharan, P. A. Arbeláez, R. B. Girshick, and J. Malik. Hypercolumns for object segmentation and fine-grained localization. *CoRR*, abs/1411.5752, 2014. 3

[13] P.-J. Hsieh, E. Vul, and N. Kanwisher. Recognition alters the spatial pattern of fmri activation in early retinotopic cortex. *Journal of Neurophysiology*, 103(3):1501–1507, 2010. 1, 8

[14] P. Isola, D. Zoran, D. Krishnan, and E. H. Adelson. Crisp boundary detection using pointwise mutual information. In *ECCV*, 2014. 2, 4

[15] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. 4

[16] J. J. Kivinen, C. K. Williams, N. Heess, and D. Technologies. Visual boundary prediction: A deep neural prediction network and quality dissection. *AISTATS*, 1(2):9, 2014. 2, 3, 4, 5

[17] Z. Kourtzi and N. Kanwisher. Representation of perceived object shape by the human lateral occipital complex. *Science*, 293:1506–1509, 2001. 1, 8

[18] J. Lim, C. L. Zitnick, and P. Dollár. Sketch tokens: A learned mid-level representation for contour and object detection. In *CVPR*, 2013. 2

[19] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *CVPR (to appear)*, Nov. 2015. 3, 5

[20] M. Maire, S. X. Yu, and P. Perona. Reconstructive sparse code transfer for contour detection and semantic labeling. In *Asian Conference on Computer Vision (ACCV)*, 2014. 2

[21] T. Malisiewicz, A. Gupta, and A. A. Efros. Ensemble of exemplar-svms for object detection and beyond. In *ICCV*, 2011. 4

[22] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001. 4

[23] X. Ren and L. Bo. Discriminatively Trained Sparse Code Gradients for Contour Detection. In *Advances in Neural Information Processing Systems*, December 2012. 2, 4

[24] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge, 2014. 3

[25] J. L. Sanguinetti, J. J. Allen, and M. A. Peterson. The ground side of an object perceived as shapeless yet processed for semantics. *Psychological science*, page 0956797613502814, 2013. 1, 8

[26] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229, 2013. 3

[27] W. Shen, X. Wang, Y. Wang, X. Bai, and Z. Zhang. Deepcontour: A deep convolutional feature learned by positive-sharing loss for contour detection. June 2015. 1, 3, 4, 5

[28] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905, 1997. 2, 6

[29] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 1

[30] A. Sironi, V. Lepetit, and P. Fua. Multiscale centerline detection by learning a scale-space distance transform. June 2014. 4

[31] C. L. Zitnick and P. Dollár. Edge boxes: Locating object proposals from edges. In *ECCV*, 2014. 1, 8

Dense Optical Flow Prediction from a Static Image

Jacob Walker, Abhinav Gupta, and Martial Hebert
Robotics Institute, Carnegie Mellon University
{jcwalker, abhinavg, hebert}@cs.cmu.edu

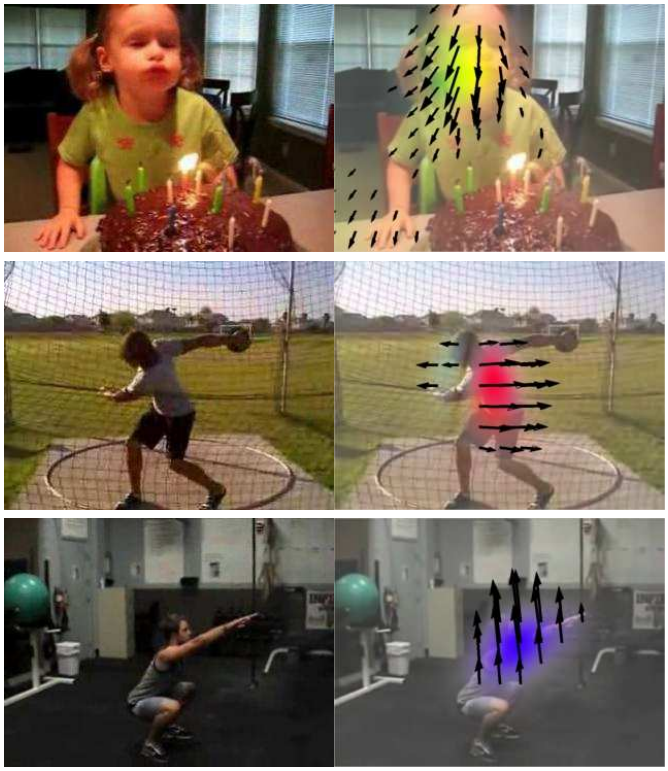
Abstract

Given a scene, what is going to move, and in what direction will it move? Such a question could be considered a non-semantic form of action prediction. In this work, we present predictive convolutional neural networks (P-CNN). Given a static image, P-CNN predicts the future motion of each and every pixel in the image in terms of optical flow. Our P-CNN model leverages the data in tens of thousands of realistic videos to train our model. Our method relies on absolutely no human labeling and is able to predict motion based on the context of the scene. Since P-CNNs make no assumptions about the underlying scene they can predict future optical flow on a diverse set of scenarios. In terms of quantitative performance, P-CNN outperforms all previous approaches by large margins.

1. Introduction

Consider the images shown in Figure 1. Given the girl in front of the cake, we humans can easily predict that her head will move downward to extinguish the candle. The man with the discus is in a position to twist his body strongly to the right, and the squatting man on the bottom has nowhere to move but up. Humans have an amazing ability to not only recognize what is present in the image but also predict what is going to happen next. Prediction is an important component of visual understanding and cognition. In order for computers to react to their environment, simple activity detection is not always sufficient. For successful interactions, robots need to predict the future and plan accordingly.

While prediction is still a relatively new problem, there has been some recent work that has focused on this task. The most common approach to this prediction problem is to use a planning-based agent-centric approach: an object [10] or a patch [24] is modeled as an agent that performs actions based on its current state and the goal state. Each action is decided based on compatibility with the environment and how this actions helps the agent move closer to the goal state. The priors on actions are modeled via transition matrices. Such an approach has been shown to produce impressive results: predicting trajectories of humans in parking lots [10] or hallucinating car movements on streets [24]. There are two main problems with this approach. First, the predictions are still sparse, and the motion is still modeled as a trajectory. Second, and more importantly, these approaches have always been shown to perform in restrictive domains such as parking lots or streets.



(a) Input Image (b) Prediction

Figure 1: **Motion Prediction.** Consider single, static input images (a). Our method can first identify what these actions are and predict (b) correct motion based on the pose and stage of the action without any video information. We use the color coding from [1] shown on the right.

In this paper, we take the next step towards generalized prediction — a framework that can be learned from tens of thousands of realistic videos. The framework can work in indoor and outdoor environments if the agent is an animal, a human, or even a car. It can account for one or multiple agents. Specifically, we look at the task of motion prediction — given a static image we predict the dense expected optical flow as if this image was part of a video. This optical flow represents how and where each and every pixel in the image is going to move in the future. Of course, we can see that motion prediction is a highly-context dependent problem. The future motion not only depends on what is active in the scene but also its context. For example, someone's entire body may move up or down if they are jumproping, but most of the body will be stationary if they are playing the flute. Instead of modeling agents and its context separately under restrictive assumptions, we use a learning based approach for motion prediction. Specifically, we train a deep network that can incorporate all of this contextual information to make accurate predictions of future motion in a wide variety of scenes. We train our model from thousands of realistic video datasets, namely the UCF-101 [21] and the HMDB-51 [13].

Contributions: Our paper makes threefold contributions. First, we present a predictive-CNN (P-CNN) model for motion prediction. Given a static image, our P-CNN model predicts expected motion in terms of optical flow. Our CNN-based model is agent-free and makes almost no assumptions about the underlying scene. Therefore, we show experimental results on diverse set of scenes. Second, P-CNN gives state of the art performance on prediction compared to contemporary approaches. Finally, we also present an proof of concept extension of P-CNN which makes long-range prediction about future motion. Our preliminary results indicate the P-CNN might indeed be promising even on the task of long-range prediction.

2. Background

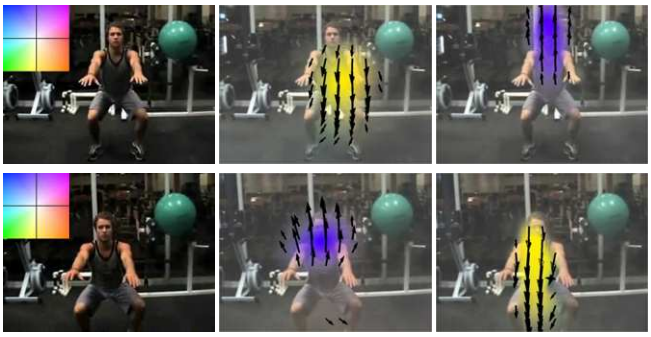
Prediction has caught the interest of the vision community in recent years. Most of research in this area has looked at different aspects of the problem. The first aspect of interest is what should be predicted: what is the output space of prediction. Some of the initial work in this area focused on predicting the optical flow for the given input image [28]. Others have looked at more semantic forms of prediction: that is, predict the action class of what is going to happen next [5, 14]. But one of the issues with semantic prediction is that it tells us nothing about the future action beyond the category. One of our goals in prediction is to go beyond classification and predict the spatial layout of future actions. For example, in case of agents such as humans, the output space of prediction can be trajectories themselves [10]. But recent approaches have argued for much richer form of predictions even in terms of pixels [24] or the features of the next frame [7, 19].

The other aspect of research in visual prediction looks at the question: what is the right approach for prediction? There have been two classes of approaches for the temporal prediction. The first is data-driven, non-parametric approach. In case of non-parameteric approaches, they do not make any assumptions about the underlying scene. For example, [28] simply retrieves videos visually similar to the static scene, allowing a warping [16] of the matched action into the scene. The other end of the spectrum is parametric and domain-specific approaches. Here, we make assumptions of what are the active elements in the scene whether they may be cars or people. Once the assumption is made, then a model is developed to predict agent behavior. This includes forecasting pedestrian trajectories [10], human-human interactions [7], [14], human expressions through SOSVM [5], and human-object interaction through graphical models [11], [3].

Some of the recent work in this area has looked at a more hybrid approach. For example, Walker et al. [24] builds a data-derived dictionary of rigid objects given a video domain and then makes long-term motion and appearance predictions using a transition and context model. Recent approaches such as [19] and [22] have even looked at training convolutional neural networks for predicting one future frame in a clip [19] or motion of handwritten characters [22].

We make multiple advances over previous work in this paper. First, our unsupervised method can generalize across a large number of diverse domains. While [24] does not explicitly require video labels, it is still domain dependent, requiring a human-given distinction between videos in and outside the domain. In addition, [24] focused only on birds-eye domains where scene depth was limited or non existent, while our method is able to generalize to scenes with perspective. [18] also uses unsupervised methods to train a Structured Random Forest for motion prediction. However, the authors only learn a model from the simple KTH [15] dataset. We show that our method is able to learn from a set of videos that is far more diverse across scenes and actions. In addition, we demonstrate much better generalization can be obtained as compared to the nearest-neighbor approach of Yuen et al. [28].

Convolutional Neural Networks: We show in this paper that a convolutional neural network can be trained for the task of motion prediction in terms of optical flow. Current work on CNNs have largely focused on recognition tasks both in images and video [12, 9, 4, 23, 29, 20, 22]. There has been some initial work where CNNs have been combined with recurrent models for prediction. For example, [19] uses a LSTM [6] to predict the immediate next frame given a video input. [22] uses a recurrent architecture to



(a) Input Image (b) Prediction (c) Ground Truth

Figure 3: Consider the images on the left. Is the man squatting up or down? The bottom is near completion (or just starting), and the top image is right in the middle of the action. Our dataset contains a large number of ambiguous images such as these. In our evaluation we consider the underlying distribution of movements predicted by our network. It is highly likely that this man is going to move up or down, but unlikely that he will veer off to the left or right.

predict motions of handwritten characters from a video. On the other hand, our approach predicts motion for each and every pixel from a static image for any generic scene.

3. Methods

Our goal is to learn a mapping between the input RGB image and the output space which corresponds to the predicted motion of each and every pixel in terms of optical flow. We propose to use CNNs as the underlying learning algorithm for this task. However, there are few questions that need to be answered: what is a good output space and what is a good loss function? Should we model optical flow prediction as a regression or a classification problem? What is a good architecture to solve this problem? We now discuss these issues below.

3.1. Regression as Classification

Intuitively, motion estimation can be posed as a regression problem since the space is continuous. Indeed, this is exactly the approach used in [18], where the authors used structured random forests to regress the magnitude and direction of the optical flow. However, such an approach has one drawback: such an output space tends to smoothen results as the ambiguity is handled by averaging out the flow. Interestingly, in a related problem of surface normal prediction, researchers have proposed reformulating structured regression as a classification problem [26, 17]. Specifically, they quantize the surface normal vectors into a codebook of clusters and then output space becomes predicting the cluster membership. In our work, we take a similar approach. We quantize optical flow vectors into 40 clusters by k-means. We can then treat the problem in a manner similar to semantic segmentation, where we classify each region as the image as a particular cluster of optical flow. We use a soft-max loss layer at the output for computing gradients.

However, at test time, we create a soft output by considering the underlying distribution of all the clusters, taking a weighted-probability sum over all the classes in a given pixel for the final output. Transforming the problem into classification also leads directly to a discrete probability distribution over vector directions and magnitudes. As the problem of motion prediction can be ambiguous depending on the image (see Figure 3), we can utilize this probability distribution over directions to measure how informative our predictions are. We may be unsure if the man in Figure 3 is sitting down or standing up given only the image, but we can be quite sure he will not turn right or left. In the same way, our network can rank upward and downward facing clusters much higher than other directions. Even if the ground truth is upward, and the highest ranked cluster is downward, it may be that the second-highest cluster is also upward. A discrete probability distribution, through classification, allows an easier understanding of how well our network may be performing. In addition, we can simply compute the entropy of the distribution, allowing us to compute the confidence of our motion prediction and retrieve images that are more likely to be correct.

3.2. Network Design

Our model is similar to the standard seven-layer architecture from [12]. To simplify the description, we denote the convolutional layer as $C(k, s)$, which indicates the there are k kernels, each having the size of $s \times s$. During convolution, we set all the strides to 1 except for the first layer, which is 4. We also denote the local response normalization layer as LRN, and the max-pooling layer as MP. The stride for pooling is 2 and we set the pooling operator size as 3×3. Finally, $F(n)$ denotes fully connected layer with n neurons. Our network architecture can be described as:

This can be described as: $C(96, 11) \rightarrow LRN \rightarrow P \rightarrow C(256, 5) \rightarrow LRN \rightarrow P \rightarrow C(384, 3) \rightarrow C(384, 3) \rightarrow C(256, 3) \rightarrow P \rightarrow F(4096) \rightarrow F(4096)$. We used a modified version of the popular Caffe toolbox [8] for our implementation. For computational simplicity, we use 200x200 windows as input. We used a learning rate of 0.0001 and a stepsize of 50000 iterations. Other network parameters were set to default. The only exception is that we used Xavier initialization of parameters. Instead of using the default softmax output, we used a spatial softmax loss function from [26] to classify every region in the image. This leads to a $M \times N \times K$ softmax layer, where M is the number of rows, N is the number of columns, and C is the number of clusters in our codebook. We used $M = 20$, $N = 20$, and $K = 40$ for a softmax layer of 16,000 neurons. Our

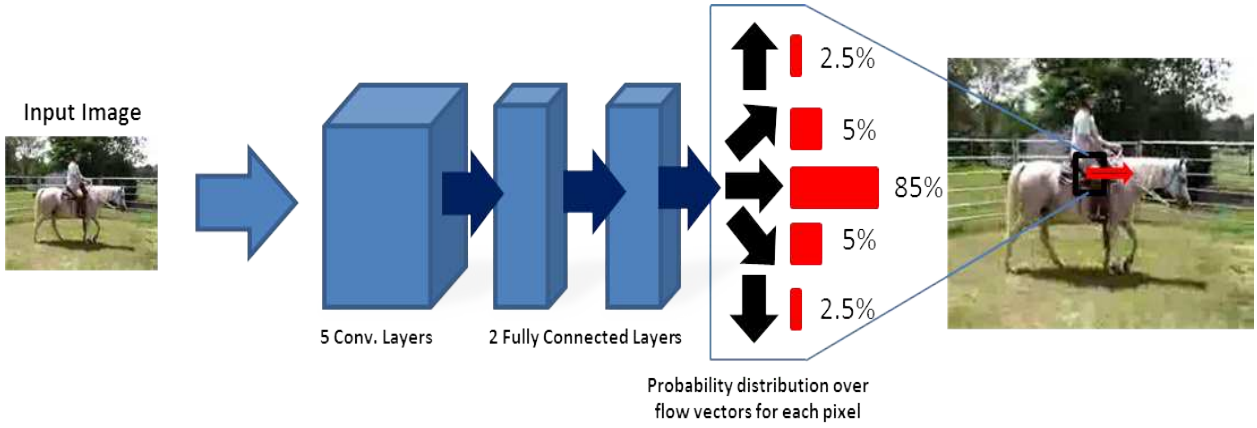


Figure 2: **Overview**. Our network is similar to the standard 7-layer architecture [12] used for many recognition tasks. We take a 200x200 image as input. However, we use a spatial softmax as the final output. For every pixel in the image we predict a distribution of various motions with various directions and magnitudes. We can combine a weighted average of these vectors to produce the final output for each pixel. For computational reasons, we predict a coarse 20x20 output.

softmax loss is spatial, summing over all the individual region losses. Let I represent the image and Y be the ground truth optical flow labels represented as quantized clusters. Then our spatial loss function $L(I, Y)$ is following:

$$L(I,Y) = -\sum_{i=1}^{M\times N}\sum_{r=1}^{K}(\mathbb{1}(y_i = r)\log F_{i,r}(I)) \quad (1)$$

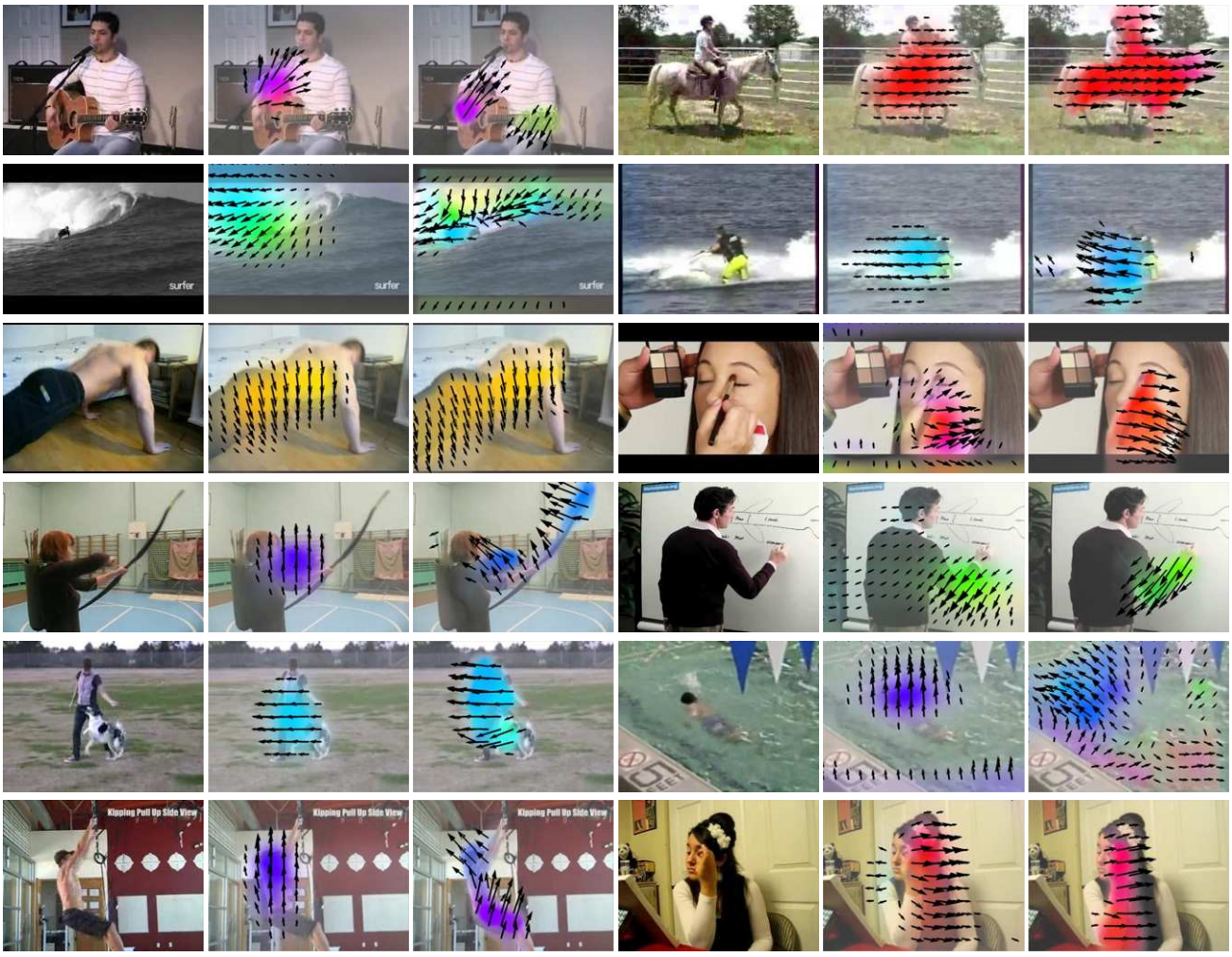
$F_{i,r}(I)$ represents the probability that the ith pixel will move according to cluster r. $\mathbb{1}(y_i = r)$ is an indicator function.

Data Augmentation: For many deep networks, datasets which are insufficiently diverse or too small will lead directly to overfitting. [20] and [9] show that training directly on datasets such as the UCF-101 for action classification leads to overfitting, as there are only data on the order of tens of thousands of videos. However, our problem of single-frame prediction is different from this task. We find that we are able to build a generalizable representation for prediction by training our model over 350,000 frames from the UCF-101 dataset as well as over 150,000 frames from the HMDB-51 dataset. We benefit additionally from data augmentation techniques. We randomly flip each image as well as use randomly cropped windows. For each input, we also change our labels according to our spatial transformation. In this way we are able to avoid spatial biases (such as humans always appearing in the middle of the image) and train a general model on a far smaller set of videos than for recognition tasks.

Labeling: We automatically label our training dataset with an optical flow algorithm. With a publically available implementation, we chose DeepFlow [27] to compute optical flow. The UCF-101 and the HMDB-51 dataset use realistic, sometimes low-quality videos from a wide variety of sources. They often suffer from compression artifacts. Thus, we aim to make our labels somewhat less noisy by taking the average optical flow of five future frames for each image. The videos in these datasets are also unstabilized. [25] showed that action recognition can be greatly improved with camera stabilization. In order to further denoise our labels, we wish to focus on the motion of objects inside the image, not the camera motion. We thus use the stabilization portion of the implementation of [25] to automatically stabilize videos using an estimated homography.

4. Experiments

For our experiments, we focused on two datasets, the UCF-101 and HMDB-51, which have been popular for action recognition. For both of these datasets, we compared against baselines using 3-fold cross validation with the splits specified by the dataset organizers. For training, we subsampled frames by a factor of 5. For testing, we sampled 26,000 frames per split. For our comparison with AlexNet finetuning, we used a split which incorporated a larger portion of the training data. We will release this split publicly. We used three baselines for evaluation. First we used the technique of [18], a SRF approach to motion prediction. We took their publicly available implementation and trained a model according to their default parameters. Because of the much larger size of our datasets, we had to sample SIFT-patches less densely. We also use a Nearest-Neighbor baseline using both fc7 features from the pretrained AlexNet network as well as pooled-5 features. Finally, we compare unsupervised training from scratch with finetuning on the supervised AlexNet network.



(a) Input Image (b) Prediction (c) Ground Truth (a) Input Image (b) Prediction (c) Ground Truth

Figure 4: Qualitative results from our method for the single frame model. Our network can find the active elements in the scene and correctly predict future motion based on the context in a wide variety and scenes and actions. The color coding is on the right.

4.1. Evaluation Metrics

Because of the complexity and sometimes high level of label ambiguity in motion prediction, we use a variety of metrics to evaluate our method and baselines. Following from [18], we use traditional End-Point-Error, measuring the Euclidean distance of the estimated optical flow vector from the ground truth vector. In addition, given vectors $\mathbf{x}_1$ and $\mathbf{x}_2$, we also measure direction similarity using the cosine similarity distance: $\frac{\mathbf{x}_1^T \mathbf{x}_2}{\|\mathbf{x}_1\|\|\mathbf{x}_2\|}$ and orientation similarity (angle taken on half-circle): $\frac{|\mathbf{x}_1^T \mathbf{x}_2|}{\|\mathbf{x}_1\|\|\mathbf{x}_2\|}$

The orientation similarity measures how parallel is predicted optical flow vector with respect to given ground truth optical flow vector. Some motions may be strictly left-right or up-down, but the exact direction may be ambiguous. This measure accounts for this situation.

We choose these metrics established by earlier work. However, we also add some additional metrics to account for the level of ambiguity in many of the test images. As [18] notes, EPE is a poor metric in the case where motion is small and may reasonably proceed in more than one possible direction. We thus additionally look at the underlying distribution of the predicted classes to understand how well the algorithm accounts for this ambiguity. For instance, if we are shown an image as in Figure 3, it is unknown if the man will move up or down. It is certainly the case, however, that he will not move right or left. Given the probability distribution over the quantized flow clusters, we check to see if the ground truth is within the top probable clusters. For the implementation of [18], we create an estimated probability distribution by quantizing the regression output from all the trees and then, for each pixel, we bin count the clus-

ImageNet-Pretrained vs. Trained from Scratch

Method	EPE	EPE-Canny	EPE-NZ
Pretrained	1.19	1.12	3.12
From Scratch	1.28	1.21	3.21

—	Orient	Orient-Canny	Orient-NZ
Pretrained	.661	.659	.692
From Scratch	.659	.658	.691

—	Top-5	Top-5-Canny	Top-5-NZ
Pretrained	91.0%	91.1%	65.8%
From Scratch	89.9%	90.3%	65.1%

Table 1: Here we compare our unsupervised network, trained from scratch, to the same network fine-tuned from supervised AlexNet features on UCF-101. The Canny suffix represents pixels on the Canny edges, and the NZ suffix represents moving pixels according to the ground-truth. NN represents a nearest-neighbor approach. Dir and Orient represent direction and orientation metrics respectively. For EPE, less is better, and for other metrics, higher is better.

ters over the trees. For Nearest-Neighbor we take the top-N matched frames and use the matched clusters in each pixel as our top-N ranking. We evaluate over the mean rank of all pixels in the image.

Following [18], we also evaluate over the Canny edges. Because of the simplicity of the datasets in [18], Canny edges were a good approximation for measuring the error of pixels of moving objects in the scene. However, our data includes highly cluttered scenes that incorporate multiple non-moving objects. In addition, we find that our network is very effective at identifying moving vs non-moving elements in the scene. We find that the difference between overall pixel mean and Canny edges is very small across all metrics and baselines. Thus, we also evaluate over the moving pixels according to the ground-truth. Moving pixels in this case includes all clusters in our codebook except for the vector of smallest magnitude. While unfortunately this metric depends on the choice of codebook, we find that the greatest variation in performance and ambiguity lies in predicting the direction and magnitude of the active elements in the scene.

4.2. Qualitative Results

Figure 4 shows some of our qualitative results. For single frame prediction, our network is able to predict motion in many different contexts. Although most of our datasets consist of human actions, our model can generalize beyond simply detecting general motion on humans. Our method is able to successfully predict the falling of the ocean wave in the second row, and it predicts the motion of the entire horse in the first row. Furthermore, our network can specify motion depending on the action being performed. For the man playing guitar and the man writing on the wall, the arm is

UCF-101

Method	EPE	EPE-Canny	EPE-NZ
SRF [18]	1.30	1.23	3.24
NN pooled-5	2.31	2.20	4.40
NN fc7	2.24	2.16	4.27
Ours	**1.27**	**1.17**	**3.19**

—	Dir	Dir-Canny	Dir-NZ
SRF [18]	.004	.000	-.013
NN pooled-5	-.001	-.001	-.067
NN fc7	-.005	-.006	-.060
Ours	**.045**	**.025**	**.092**

—	Orient	Orient-Canny	Orient-NZ
SRF [18]	.492	.600	.515
NN pooled-5	.650	.650	.677
NN fc7	.649	.649	.651
Ours	**.659**	**.657**	**.688**

—	Top-5	Top-5-Canny	Top-5-NZ
SRF [18]	79.4%	81.7%	10.0%
NN pooled-5	77.8%	79.5%	20.0%
NN fc7	78.3%	79.9%	18.8%
Ours	**89.7%**	**90.5%**	**65.0%**

—	Top-10	Top-10-Canny	Top-10-NZ
SRF [18]	82.2%	84.4%	17.2%
NN pooled-5	83.2%	85.3%	32.9%
NN fc7	84.0%	85.4%	32.3%
Ours	**96.5%**	**96.7%**	**90.9%**

Table 2: Single-image evaluation using the 3-fold split on UCF-101. The Canny suffix represents pixels on the Canny edges, and the NZ suffix represents moving pixels according to the ground-truth. NN represents a nearest-neighbor approach. Dir and Orient represent direction and orientation metrics respectively. For EPE, less is better, and for other metrics, higher is better.

the most salient part to be moved. For the man walking the dog and the man doing a pushup, the entire body will move according to the action.

4.3. Quantitative Results

We show in tables 2 and 3 that our method strongly outperforms both the Nearest-Neighbor and SRF-based baselines by a large margin by most metrics. This holds true for both datasets. Interestingly, the SRF-based approach seems to come close to ours based on End-Point-Error, but is heavily outperformed on all other metrics. This is largely a product of the End-Point-Error metric, as we find that the SRF tends to output the mean (optical flow with very small magnitude). This is consistent with the results found in [18], where actions with low, bidirectional motion can result in higher EPE than predicting no motion at all. When we account for this ambiguity in motion in the top-N metric, however, the difference in performance is large. Compared to unsupervised training from scratch, we find that finetuning

HMDB-51

Method	EPE	EPE-Canny	EPE-NZ
SRF [18]	1.23	1.20	3.46
NN pooled-5	2.51	2.49	4.89
NN fc7	2.43	2.43	4.69
Ours	**1.21**	**1.17**	**3.45**

—	Dir	Dir-Canny	Dir-NZ
SRF [18]	.000	.000	-.010
NN pooled-5	-.008	-.007	-.061
NN fc7	-.007	-.005	-.061
Ours	**.019**	**.012**	**.030**

—	Orient	Orient-Canny	Orient-NZ
SRF [18]	.461	.557	.495
NN pooled-5	.631	.631	.644
NN fc7	.630	.631	.655
Ours	**.636**	**.636**	**.667**

—	Top-5	Top-5-Canny	Top-5-NZ
SRF [18]	81.9%	83.6%	13.5%
NN pooled-5	76.3%	77.8%	14.0%
NN fc7	77.3%	78.7%	13.5%
Ours	**90.2%**	**90.5%**	**61.0%**

—	Top-10	Top-10-Canny	Top-10-NZ
SRF [18]	84.4%	86.1%	22.1%
NN pooled-5	82.9%	84.0%	23.9%
NN fc7	83.6%	84.4%	23.2%
Ours	**95.9%**	**95.9%**	**87.5%**

Table 3: Single-image evaluation using the 3-fold split on HMDB-51. The Canny suffix represents pixels on the Canny edges, and the NZ suffix represents moving pixels according to the ground-truth. NN represents a nearest-neighbor approach. Dir and Orient represent direction and orientation metrics respectively. For EPE, less is better, and for other metrics, higher is better.

from supervised, pretrained features yield only a very small improvement in performance. Looking over all pixels, the difference in performance between approaches is small. On absolute levels, the orientation and top-N metrics also tend to be high. This is due to the fact that most pixels in the image are not going to move. Outputting low or zero-motion over the entire image can thus lead to good performance for many metrics. Canny edge pixels yield similar results, as our natural images often include background clutter with objects that do not move. The most dramatic differences appear over the non-zero pixels. The direction metric is for our method is low at .09 because of direction ambiguity, but orientation similarity is much larger. The largest performance gains come at the top-N rankings. For 40 clusters, random chance for top-5 is 12.5%, and for top-10 it is 25%. Nearest-neighbor does slightly better than chance, but SRF actually performs slightly worse. This is most likely due to the SRF tendency to output the overall mean flow, which is of low magnitude. Our method performs much better, with the ground truth direction and magnitude vector coming in 65% of the time in the top-5 ranking, and to a very high 90.9% of the time in the top ten. In spite of exposure to more data and use of semantic labels, we see in Table 1 that finetuning using the ImageNet network only leads to a small increase in performance compared to training from scratch.

5. Multi-Frame Prediction

Until now we have described an architecture for predicting optical flow given a static image as input. However, it would be interesting to predict not just the next frame but a few seconds into future. How should we design such a network?

We present a proof-of-concept network to predict 6 future frames. In order to predict multiple frames into the future, we take our pre-trained single frame network and output the seventh feature layer into a "temporally deep" network, using the implementation of [2]. This network architecture is the same as an unrolled recurrent neural network with some important differences. On a high level, our network is similar to the unfactored architecture in [2], with each sequence having access to the image features and the previous hidden state in order to predict the next state. We replace the LSTM module with a fully connected layer as in a RNN. However, we also do not use a true recurrent network. The weights for each sequence layer are not shared, and each sequence has access to all the past hidden states. We used 2000 hidden states in our network, but we predict at most six future sequences. We attempted to use recurrent architectures with the publicly available LSTM implementation from [2]. However, in our experiments they always regressed to a mean trajectory across the data. Our fully connected network has much higher number of parameters than a RNN and therefore highlights the inherent difficulty of this task. Due to the much larger size of the state space, we do not predict optical flow for each and every pixel. Instead, we use kmeans to created a codebook of 1000 possible optical flow frames, and we predict one of 1000 class as output as each time step. This can be thought of as analogous to a sequential prediction problem similar to caption generation. Instead of a sequence of words, our "words" are clusters of optical flow frames, and our "sentence" is an entire trajectory. We used a set number of sequences, six, in our experiments with each frame representing the average optical flow of one-sixth of a second.

6. Conclusion

In this paper we have presented an approach to generalized event prediction in static scenes. Namely, our framework focuses on motion prediction as a non-semantic form of action prediction. By using an optical flow algorithm

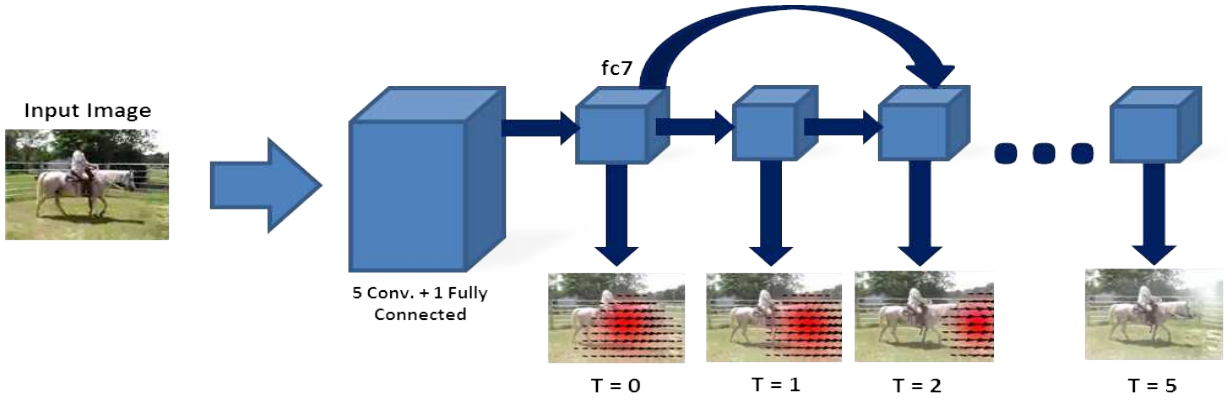
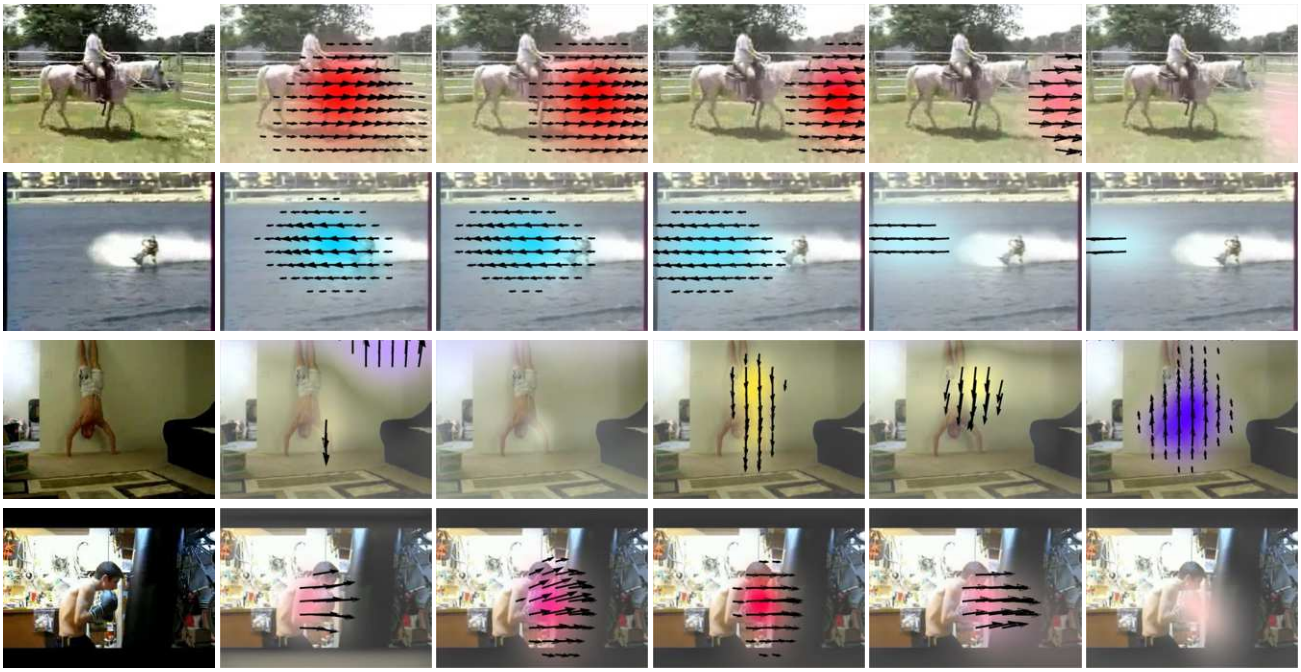


Figure 5: **Overview**. For our multiframe prediction, we predict entire clustered frames of optical flow as a sequence of frames. We take the learned features for our single frame model as our input, and we input them to a series of six fully connected layers, with each layer having access to the states of the past layers.



(a) Input Image (b) Frame 1 (c) Frame 2 (c) Frame 3 (d) Frame 4 (e) Frame 5

Figure 6: Qualitative results for multi-frame prediction. The four rows represent predictions from our multi-frame model for future frames. Our extension can predict optical flow over multiple frames.

to label the data, we can train this model on a large number of unsupervised videos. Furthermore, our framework utilizes the success of deep networks to outperform contemporary approaches to motion prediction. We find that our network successfully predicts motion based on the context of the scene and the stage of the action taking place. Possible work includes incorporating this motion model to predict semantic action labels in images and video. Another possible direction is to utilize the predicted optical flow to predict in raw pixel space, synthesizing a video from a single image. **Acknowedgements:** We thank Xiaolong Wang for many helpful discussions. We thank the NVIDIA Corporation for the donation of Tesla K40 GPUs for this research. In addition, this work was supported by NSF grant IIS1227495.

References

[1] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. *IJCV*, 92(1):1–31, 2011. 1

[2] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. *arXiv preprint arXiv:1411.4389*, 2014. 7

[3] D. Fouhey and C. L. Zitnick. Predicting object dynamics in scenes. In *CVPR*, 2014. 2

[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 2

[5] M. Hoai and F. De la Torre. Max-margin early event detectors. *IJCV*, 107(2):191–202, 2014. 2

[6] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 2

[7] D.-A. Huang and K. M. Kitani. Action-reaction: Forecasting the dynamics of human interaction. In *ECCV*. 2014. 2

[8] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. 3

[9] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014. 2, 4

[10] K. Kitani, B. Ziebart, D. Bagnell, and M. Hebert. Activity forecasting. In *ECCV*, 2012. 1, 2

[11] H. S. Koppula and A. Saxena. Anticipating human activities using object affordances for reactive robotic response. In *RSS*, 2013. 2

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012. 2, 3, 4

[13] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. HMDB: a large video database for human motion recognition. In *ICCV*, 2011. 2

[14] T. Lan, T.-C. Chen, and S. Savarese. A hierarchical representation for future action prediction. In *ECCV*. 2014. 2

[15] I. Laptev, B. Caputo, et al. Recognizing human actions: A local svm approach. In *ICPR*, 2004. 2

[16] C. Liu, J. Yuen, and A. Torralba. Sift flow: Dense correspondence across scenes and its applications. *PAMI*, 2011. 2

[17] B. Z. Lubor Ladickỳ and M. Pollefeys. Discriminatively trained dense surface normal estimation. 3

[18] S. L. Pintea, J. C. van Gemert, and A. W. Smeulders. Déjà vu: Motion prediction in static images. In *ECCV*. 2014. 2, 3, 4, 5, 6, 7

[19] M. Ranzato, A. Szlam, J. Bruna, M. Mathieu, R. Collobert, and S. Chopra. Video (language) modeling: a baseline for generative models of natural videos. *arXiv preprint arXiv:1412.6604*, 2014. 2

[20] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *NIPS*, 2014. 2, 4

[21] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012. 2

[22] N. Srivastava, E. Mansimov, and R. Salakhutdinov. Unsupervised learning of video representations using lstms. *arXiv preprint arXiv:1502.04681*, 2015. 2

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014. 2

[24] J. Walker, A. Gupta, and M. Hebert. Patch to the future: Unsupervised visual prediction. In *CVPR*, 2014. 1, 2

[25] H. Wang and C. Schmid. Action recognition with improved trajectories. In *ICCV*, Sydney, Australia, 2013. 4

[26] X. Wang, D. F. Fouhey, and A. Gupta. Designing deep networks for surface normal estimation. *arXiv preprint arXiv:1411.4958*, 2014. 3

[27] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid. DeepFlow: Large displacement optical flow with deep matching. In *ICCV*, 2013. 4

[28] J. Yuen and A. Torralba. A data-driven approach for event prediction. In *ECCV*, 2010. 2

[29] N. Zhang, M. Paluri, M. Ranzato, T. Darrell, and L. Bourdev. Panda: Pose aligned networks for deep attribute modeling. In *CVPR*, 2014. 2

Ask Your Neurons: A Neural-based Approach to Answering Questions about Images

Mateusz Malinowski[1] Marcus Rohrbach[2] Mario Fritz[1]
[1]Max Planck Institute for Informatics, Saarbrücken, Germany
[2]UC Berkeley EECS and ICSI, Berkeley, CA, United States

Abstract

We address a question answering task on real-world images that is set up as a Visual Turing Test. By combining latest advances in image representation and natural language processing, we propose Neural-Image-QA, an end-to-end formulation to this problem for which all parts are trained jointly. In contrast to previous efforts, we are facing a multi-modal problem where the language output (answer) is conditioned on visual and natural language input (image and question). Our approach Neural-Image-QA doubles the performance of the previous best approach on this problem. We provide additional insights into the problem by analyzing how much information is contained only in the language part for which we provide a new human baseline. To study human consensus, which is related to the ambiguities inherent in this challenging task, we propose two novel metrics and collect additional answers which extends the original DAQUAR dataset to DAQUAR-Consensus.

1. Introduction

With the advances of natural language processing and image understanding, more complex and demanding tasks have become within reach. Our aim is to take advantage of the most recent developments to push the state-of-the-art for answering natural language questions on real-world images. This task unites inference of question intends and visual scene understanding with a word sequence prediction task.

Most recently, architectures based on the idea of layered, end-to-end trainable artificial neural networks have improved the state of the art across a wide range of diverse tasks. Most prominently Convolutional Neural Networks have raised the bar on image classification tasks [16] and Long Short Term Memory Networks are dominating performance on a range of sequence prediction tasks such as machine translation [28].

Very recently these two trends of employing neural architectures have been combined fruitfully with methods that can generate image [12] and video descriptions [30]. Both are conditioning on the visual features that stem from deep learning architectures and employ recurrent neural network approaches to produce descriptions.

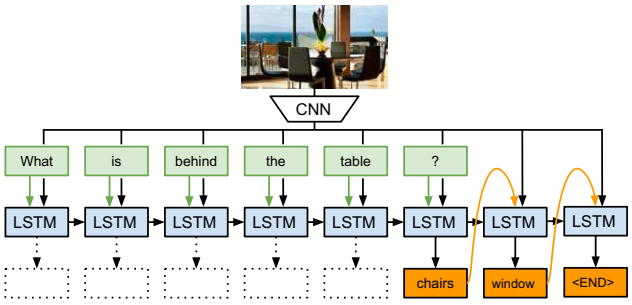


Figure 1. Our approach Neural-Image-QA to question answering with a Recurrent Neural Network using Long Short Term Memory (LSTM). To answer a question about an image, we feed in both, the image (CNN features) and the question (green boxes) into the LSTM. After the (variable length) question is encoded, we generate the answers (multiple words, orange boxes). During the answer generation phase the previously predicted answers are fed into the LSTM until the ⟨END⟩ symbol is predicted.

To further push the boundaries and explore the limits of deep learning architectures, we propose an architecture for answering questions about images. In contrast to prior work, this task needs conditioning on language as well visual input. Both modalities have to be interpreted and jointly represented as an answer depends on inferred meaning of the question and image content.

While there is a rich body of work on natural language understanding that has addressed textual question answering tasks based on semantic parsing, symbolic representation and deduction systems, which also has seen applications to question answering on images [20], there is initial evidence that deep architectures can indeed achieve a similar goal [33]. This motivates our work to seek end-to-end architectures that learn to answer questions in a single holistic and monolithic model.

We propose Neural-Image-QA, an approach to question

answering with a recurrent neural network. An overview is given in Figure 1. The image is analyzed via a Convolutional Neural Network (CNN) and the question together with the visual representation is fed into a Long Short Term Memory (LSTM) network. The system is trained to produce the correct answer to the question on the image. CNN and LSTM are trained jointly and end-to-end starting from words and pixels.

Contributions: We proposes a novel approach based on recurrent neural networks for the challenging task of answering of questions about images. It combines a CNN with a LSTM into an end-to-end architecture that predict answers conditioning on a question and an image. Our approach significantly outperforms prior work on this task – doubling the performance. We collect additional data to study human consensus on this task, propose two new metrics sensitive to these effects, and provide a new baseline, by asking humans to answer the questions without observing the image. We demonstrate a variant of our system that also answers question without accessing any visual information, which beats the human baseline.

2. Related Work

As our method touches upon different areas in machine learning, computer vision and natural language processing, we have organized related work in the following way:

Convolutional Neural Networks for visual recognition. We are building on the recent success of Convolutional Neural Networks (CNN) for visual recognition [16, 17, 25], that are directly learnt from the raw image data and pre-trained on large image corpora. Due to the rapid progress in this area within the last two years, a rich set of models [27, 29] is at our disposal.

Recurrent Neural Networks (RNN) for sequence modeling. Recurrent Neural Networks allow Neural Networks to handle sequences of flexible length. A particular variant called Long Short Term Memory (LSTM) [9] has shown recent success on natural language tasks such as machine translation [3, 28].

Combining RNNs and CNNs for description of visual content. The task of describing visual content like still images as well as videos has been successfully addressed with a combination of the previous two ideas [5, 12, 31, 32, 37]. This is achieved by using the RNN-type model that first gets to observe the visual content and is trained to afterwards predict a sequence of words that is a description of the visual content. Our work extends this idea to question answering, where we formulate a model trained to generate an answer based on visual as well as natural language input.

Grounding of natural language and visual concepts. Dealing with natural language input does involve the association of words with meaning. This is often referred to as grounding problem - in particular if the "meaning" is associated with a sensory input. While such problems have been historically addressed by symbolic semantic parsing techniques [15, 22], there is a recent trend of machine learning-based approaches [12, 13, 14] to find the associations. Our approach follows the idea that we do not enforce or evaluate any particular representation of "meaning" on the language or image modality. We treat this as latent and leave this to the joint training approach to establish an appropriate internal representation for the question answering task.

Textual question answering. Answering on purely textual questions has been studied in the NLP community [2, 18] and state of the art techniques typically employ semantic parsing to arrive at a logical form capturing the intended meaning and infer relevant answers. Only very recently, the success of the previously mentioned neural sequence models as RNNs has carried over to this task [10, 33]. More specifically [10] uses dependency-tree Recursive NN instead of LSTM, and reduce the question-answering problem to a classification task. Moreover, according to [10] their method cannot be easily applied to vision. [33] propose different kind of network - memory networks - and it is unclear how to apply [33] to take advantage of the visual content. However, neither [10] nor [33] show an end-to-end, monolithic approaches that produce multiple words answers for question on images.

Visual Turing Test. Most recently several approaches have been proposed to approach Visual Turing Test [21], i.e. answering question about visual content. For instance [8] have proposed a binary (yes/no) version of Visual Turing Test on synthetic data. In [20], we present a question answering system based on a semantic parser on a more varied set of human question-answer pairs. In contrast, in this work, our method is based on a neural architecture, which is trained end-to-end and therefore liberates the approach from any ontological commitment that would otherwise be introduced by a semantic parser.

We like to note that shortly after this work, several neural-based models [24, 19, 7] have also been suggested. Also several new datasets for Visual Turing Tests have just been proposed [1, 35] that are worth further investigations.

3. Approach

Answering questions on images is the problem of predicting an answer $\boldsymbol{a}$ given an image $\boldsymbol{x}$ and a question $\boldsymbol{q}$ according to a parametric probability measure:

$$\hat{\boldsymbol{a}} = \arg\max_{\boldsymbol{a} \in \mathcal{A}} p(\boldsymbol{a}|\boldsymbol{x}, \boldsymbol{q}; \boldsymbol{\theta}) \tag{1}$$

where $\boldsymbol{\theta}$ represent a vector of all parameters to learn and $\mathcal{A}$ is a set of all answers. Later we describe how we represent $\boldsymbol{x}$, $\boldsymbol{a}$, $\boldsymbol{q}$, and $p(\cdot|\boldsymbol{x}, \boldsymbol{q}; \boldsymbol{\theta})$ in more details.

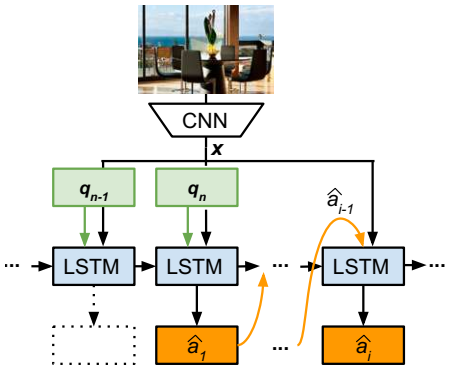


Figure 2. Our approach Neural-Image-QA, see Section 3 for details.

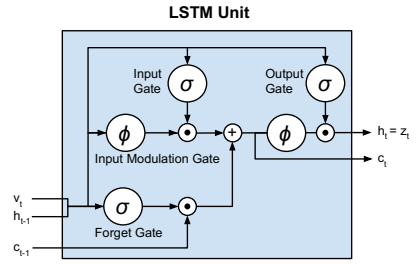


Figure 3. LSTM unit. See Section 3, Equations (3)-(8) for details.

In our scenario questions can have multiple word answers and we consequently decompose the problem to predicting a set of answer words $\boldsymbol{a}_{\boldsymbol{q},\boldsymbol{x}} = \{\boldsymbol{a}_1, \boldsymbol{a}_2, ..., \boldsymbol{a}_{\mathcal{N}(q,x)}\}$, where $\boldsymbol{a}_t$ are words from a finite vocabulary $\mathcal{V}'$, and $\mathcal{N}(q,x)$ is the number of answer words for the given question and image. In our approach, named Neural-Image-QA, we propose to tackle the problem as follows. To predict multiple words we formulate the problem as predicting a sequence of words from the vocabulary $\mathcal{V} := \mathcal{V}' \cup \{\$\}$ where the extra token \$ indicates the end of the answer sequence, and points out that the question has been fully answered. We thus formulate the prediction procedure recursively:

$$\hat{\boldsymbol{a}}_t = \arg\max_{\boldsymbol{a}\in\mathcal{V}} p(\boldsymbol{a}|\boldsymbol{x}, \boldsymbol{q}, \hat{A}_{t-1}; \boldsymbol{\theta}) \tag{2}$$

where $\hat{A}_{t-1} = \{\hat{\boldsymbol{a}}_1, \ldots, \hat{\boldsymbol{a}}_{\boldsymbol{t-1}}\}$ is the set of previous words, with $\hat{A}_0 = \{\}$ at the beginning, when our approach has not given any answer so far. The approach is terminated when $\hat{a}_t = \$$. We evaluate the method solely based on the predicted answer words ignoring the extra token \$. To ensure uniqueness of the predicted answer words, as we want to predict the *set* of answer words, the prediction procedure can be be trivially changed by maximizing over $\mathcal{V} \setminus \hat{A}_{t-1}$. However, in practice, our algorithm learns to not predict any previously predicted words.

As shown in Figure 1 and Figure 2, we feed Neural-Image-QA with a question as a sequence of words, i.e. $\boldsymbol{q} = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{n-1}, [\![?]\!]]$, where each $\boldsymbol{q}_t$ is the t-th word question and $[\![?]\!] := \boldsymbol{q}_n$ encodes the question mark - the end of the question. Since our problem is formulated as a variable-length input/output sequence, we model the parametric distribution $p(\cdot|\boldsymbol{x}, \boldsymbol{q}; \boldsymbol{\theta})$ of Neural-Image-QA with a recurrent neural network and a softmax prediction layer. More precisely, Neural-Image-QA is a deep network built of CNN [17] and Long-Short Term Memory (LSTM) [9]. LSTM has been recently shown to be effective in learning a variable-length sequence-to-sequence mapping [5, 28].

Both question and answer words are represented with one-hot vector encoding (a binary vector with exactly one non-zero entry at the position indicating the index of the word in the vocabulary) and embedded in a lower dimensional space, using a jointly learnt latent linear embedding. In the training phase, we augment the question words sequence $\boldsymbol{q}$ with the corresponding ground truth answer words sequence $\boldsymbol{a}$, i.e. $\hat{\boldsymbol{q}} := [\boldsymbol{q}, \boldsymbol{a}]$. During the test time, in the prediction phase, at time step t, we augment $\boldsymbol{q}$ with previously predicted answer words $\hat{\boldsymbol{a}}_{1..t} := [\hat{\boldsymbol{a}}_1, \ldots, \hat{\boldsymbol{a}}_{t-1}]$, i.e. $\hat{\boldsymbol{q}}_t := [\boldsymbol{q}, \hat{\boldsymbol{a}}_{1..t}]$. This means the question $\boldsymbol{q}$ and the previous answers are encoded implicitly in the hidden states of the LSTM, while the latent hidden representation is learnt. We encode the image $\boldsymbol{x}$ using a CNN and provide it at every time step as input to the LSTM. We set the input $\boldsymbol{v}_t$ as a concatenation of $[\boldsymbol{x}, \hat{\boldsymbol{q}}_t]$.

As visualized in detail in Figure 3, the LSTM unit takes an input vector $\boldsymbol{v}_t$ at each time step t and predicts an output word $\boldsymbol{z_t}$ which is equal to its latent hidden state $\boldsymbol{h}_t$. As discussed above $\boldsymbol{z_t}$ is a linear embedding of the corresponding answer word $\boldsymbol{a_t}$. In contrast to a simple RNN unit the LSTM unit additionally maintains a memory cell $\boldsymbol{c}$. This allows to learn long-term dynamics more easily and significantly reduces the vanishing and exploding gradients problem [9]. More precisely, we use the LSTM unit as described in [36] and the *Caffe* implementation from [5]. With the *sigmoid* nonlinearity $\sigma : \mathbb{R} \mapsto [0,1]$, $\sigma(v) = (1+e^{-v})^{-1}$ and the *hyperbolic tangent* nonlinearity $\phi : \mathbb{R} \mapsto [-1,1]$, $\phi(v) = \frac{e^v - e^{-v}}{e^v + e^{-v}} = 2\sigma(2v) - 1$, the LSTM updates for time step t given inputs $\boldsymbol{v}_t$, $\boldsymbol{h}_{t-1}$, and the memory cell $\boldsymbol{c}_{t-1}$ as follows:

$$\boldsymbol{i}_t = \sigma(W_{vi}\boldsymbol{v}_t + W_{hi}\boldsymbol{h}_{t-1} + \boldsymbol{b}_i) \tag{3}$$
$$\boldsymbol{f}_t = \sigma(W_{vf}\boldsymbol{v}_t + W_{hf}\boldsymbol{h}_{t-1} + \boldsymbol{b}_f) \tag{4}$$
$$\boldsymbol{o}_t = \sigma(W_{vo}\boldsymbol{v}_t + W_{ho}\boldsymbol{h}_{t-1} + \boldsymbol{b}_o) \tag{5}$$
$$\boldsymbol{g}_t = \phi(W_{vg}\boldsymbol{v}_t + W_{hg}\boldsymbol{h}_{t-1} + \boldsymbol{b}_g) \tag{6}$$
$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{g}_t \tag{7}$$
$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \phi(\boldsymbol{c}_t) \tag{8}$$

where $\odot$ denotes element-wise multiplication. All the weights W and biases b of the network are learnt jointly with the cross-entropy loss. Conceptually, as shown in

Figure 3, Equation 3 corresponds to the input gate, Equation 6 the input modulation gate, and Equation 4 the forget gate, which determines how much to keep from the previous memory c_{t-1} state. As Figures 1 and 2 suggest, all the output predictions that occur before the question mark are excluded from the loss computation, so that the model is penalized solely based on the predicted answer words.

Implementation We use default hyper-parameters of LSTM [5] and CNN [11]. All CNN models are first pre-trained on the ImageNet dataset [25], and next we randomly initialize and train the last layer together with the LSTM network on the task. We find this step crucial in obtaining good results. We have explored the use of a 2 layered LSTM model, but have consistently obtained worse performance. In a pilot study, we have found that *GoogleNet* architecture [11, 29] consistently outperforms the *AlexNet* architecture [11, 16] as a CNN model for our task and model.

4. Experiments

In this section we benchmark our method on a task of answering questions about images. We compare different variants of our proposed model to prior work in Section 4.1. In addition, in Section 4.2, we analyze how well questions can be answered without using the image in order to gain an understanding of biases in form of prior knowledge and common sense. We provide a new human baseline for this task. In Section 4.3 we discuss ambiguities in the question answering tasks and analyze them further by introducing metrics that are sensitive to these phenomena. In particular, the WUPS score [20] is extended to a consensus metric that considers multiple human answers. Additional results are available in the supplementary material and on the project webpage [1].

Experimental protocol We evaluate our approach on the DAQUAR dataset [20] which provides $12,468$ human question answer pairs on images of indoor scenes [26] and follow the same evaluation protocol by providing results on accuracy and the WUPS score at $\{0.9, 0.0\}$. We run experiments for the full dataset as well as their proposed reduced set that restricts the output space to only 37 object categories and uses 25 test images. In addition, we also evaluate the methods on different subsets of DAQUAR where only 1, 2, 3 or 4 word answers are present.

WUPS scores We base our experiments as well as the consensus metrics on WUPS scores [20]. The metric is a generalization of the accuracy measure that accounts for word-level ambiguities in the answer words. For instance 'carton' and 'box' can be associated with a similar concept,

[1] https://www.d2.mpi-inf.mpg.de/visual-turing-challenge

	Accuracy	WUPS @0.9	WUPS @0.0
Malinowski et al. [20]	7.86	11.86	38.79
Neural-Image-QA (ours)			
- multiple words	17.49	23.28	57.76
- single word	**19.43**	**25.28**	**62.00**
Human answers [20]	50.20	50.82	67.27
Language only (ours)			
- multiple words	17.06	22.30	56.53
- single word	17.15	22.80	58.42
Human answers, no images	7.34	13.17	35.56

Table 1. Results on DAQUAR, all classes, single reference, in %.

and hence models should not be strongly penalized for this type of mistakes. Formally:

$$\text{WUPS}(A,T) = \frac{1}{N}\sum_{i=1}^{N} \min\{ \prod_{a\in A^i} \max_{t\in T^i} \mu(a,t), \prod_{t\in T^i} \max_{a\in A^i} \mu(a,t)\}$$

To embrace the aforementioned ambiguities, [20] suggest using a thresholded taxonomy-based Wu-Palmer similarity [34] for μ. The smaller the threshold the more forgiving metric. As in [20], we report WUPS at two extremes, 0.0 and 0.9.

4.1. Evaluation of Neural-Image-QA

We start with the evaluation of our Neural-Image-QA on the full DAQUAR dataset in order to study different variants and training conditions. Afterwards we evaluate on the reduced DAQUAR for additional points of comparison to prior work.

Results on full DAQUAR Table 1 shows the results of our Neural-Image-QA method on the full set ("multiple words") with 653 images and 5673 question-answer pairs available at test time. In addition, we evaluate a variant that is trained to predict only a single word ("single word") as well as a variant that does not use visual features ("Language only"). In comparison to the prior work [20] (shown in the first row in Table 1), we observe strong improvements of over 9% points in accuracy and over 11% in the WUPS scores [second row in Table 1 that corresponds to "multiple words"]. Note that, we achieve this improvement despite the fact that the only published number available for the comparison on the full set uses ground truth object annotations [20] – which puts our method at a disadvantage. Further improvements are observed when we train only on a single word answer, which doubles the accuracy obtained

	Accuracy	WUPS @0.9	WUPS @0.0
Neural-Image-QA (ours)	21.67	27.99	65.11
Language only (ours)	19.13	25.16	61.51

Table 2. Results of the single word model on the one-word answers subset of DAQUAR, all classes, single reference, in %.

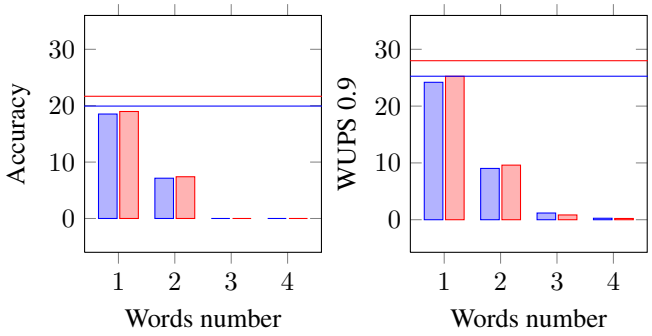


Figure 4. Language only (blue bar) and Neural-Image-QA (red bar) "multi word" models evaluated on different subsets of DAQUAR. We consider 1, 2, 3, 4 word subsets. The blue and red horizontal lines represent "single word" variants evaluated on the answers with exactly 1 word.

in prior work. We attribute this to a joint training of the language and visual representations and the dataset bias, where about 90% of the answers contain only a single word.

We further analyze this effect in Figure 4, where we show performance of our approach ("multiple words") in dependence on the number of words in the answer (truncated at 4 words due to the diminishing performance). The performance of the "single word" variants on the one-word subset are shown as horizontal lines. Although accuracy drops rapidly for longer answers, our model is capable of producing a significant number of correct two words answers. The "single word" variants have an edge on the single answers and benefit from the dataset bias towards these type of answers. Quantitative results of the "single word" model on the one-word answers subset of DAQUAR are shown in Table 2. While we have made substantial progress compared to prior work, there is still a 30% points margin to human accuracy and 25 in WUPS score ["Human answers" in Table 1].

Results on reduced DAQUAR In order to provide performance numbers that are comparable to the proposed Multi-World approach in [20], we also run our method on the reduced set with 37 object classes and only 25 images with 297 question-answer pairs at test time.

Table 3 shows that Neural-Image-QA also improves on the reduced DAQUAR set, achieving 34.68% Accuracy and 40.76% WUPS at 0.9 substantially outperforming [20] by

	Accuracy	WUPS @0.9	WUPS @0.0
Malinowski et al. [20]	12.73	18.10	51.47
Neural-Image-QA (ours)			
- multiple words	29.27	36.50	79.47
- single word	**34.68**	**40.76**	79.54
Language only (ours)			
- multiple words	32.32	38.39	80.05
- single word	31.65	38.35	**80.08**

Table 3. Results on reduced DAQUAR, single reference, with a reduced set of 37 object classes and 25 test images with 297 question-answer pairs, in %

21.95% Accuracy and 22.6 WUPS. Similarly to previous experiments, we achieve the best performance using the "single word" variant.

4.2. Answering questions without looking at images

In order to study how much information is already contained in questions, we train a version of our model that ignores the visual input. The results are shown in Table 1 and Table 3 under "Language only (ours)". The best "Language only" models with 17.15% and 32.32% compare very well in terms of accuracy to the best models that include vision. The latter achieve 19.43% and 34.68% on the full and reduced set respectively.

In order to further analyze this finding, we have collected a new human baseline "Human answer, no image", where we have asked participants to answer on the DAQUAR questions without looking at the images. It turns out that humans can guess the correct answer in 7.86% of the cases by exploiting prior knowledge and common sense. Interestingly, our best "language only" model outperforms the human baseline by over 9%. A substantial number of answers are plausible and resemble a form of common sense knowledge employed by humans to infer answers without having seen the image.

4.3. Human Consensus

We observe that in many cases there is an inter human agreement in the answers for a given image and question and this is also reflected by the human baseline performance on the question answering task of 50.20% ["Human answers" in Table 1]. We study and analyze this effect further by extending our dataset to multiple human reference answers in Section 4.3.1, and proposing a new measure – inspired by the work in psychology [4, 6, 23] – that handles disagreement in Section 4.3.2, as well as conducting additional experiments in Section 4.3.3.

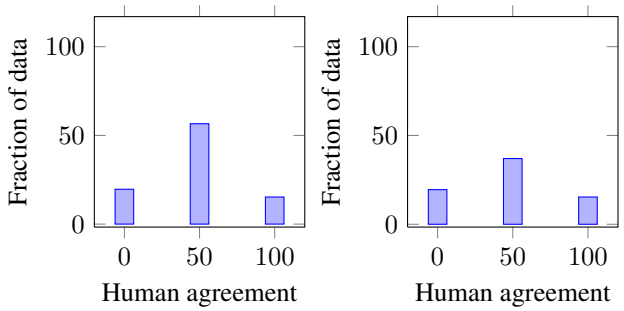


Figure 5. Study of inter human agreement. At x-axis: no consensus (0%), at least half consensus (50%), full consensus (100%). Results in %. Left: consensus on the whole data, right: consensus on the test data.

4.3.1 DAQUAR-Consensus

In order to study the effects of consensus in the question answering task, we have asked multiple participants to answer the same question of the DAQUAR dataset given the respective image. We follow the same scheme as in the original data collection effort, where the answer is a set of words or numbers. We do not impose any further restrictions on the answers. This extends the original data [20] to an average of 5 test answers per image and question. We refer to this dataset as DAQUAR-Consensus.

4.3.2 Consensus Measures

While we have to acknowledge inherent ambiguities in our task, we seek a metric that prefers an answer that is commonly seen as preferred. We make two proposals:

Average Consensus: We use our new annotation set that contains multiple answers per question in order to compute an expected score in the evaluation:

$$\frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}\min\{\prod_{a\in A^i}\max_{t\in T_k^i}\mu(a,t),\ \prod_{t\in T_k^i}\max_{a\in A^i}\mu(a,t)\} \tag{9}$$

where for the i-th question A^i is the answer generated by the architecture and T_k^i is the k-th possible human answer corresponding to the k-th interpretation of the question. Both answers A^i and T_k^i are sets of the words, and μ is a membership measure, for instance WUP [34]. We call this metric "Average Consensus Metric (ACM)" since, in the limits, as K approaches the total number of humans, we truly measure the inter human agreement of every question.

Min Consensus: The Average Consensus Metric puts more weights on more "mainstream" answers due to the summation over possible answers given by humans. In order to measure if the result was at least with one human in agreement, we propose a "Min Consensus Metric (MCM)" by replacing the averaging in Equation 9 with a max operator. We call such metric Min Consensus and suggest using both metrics in the benchmarks. We will make the implementation of both metrics publicly available.

$$\frac{1}{N}\sum_{i=1}^{N}\max_{k=1}^{K}\left(\min\{\prod_{a\in A^i}\max_{t\in T_k^i}\mu(a,t),\ \prod_{t\in T_k^i}\max_{a\in A^i}\mu(a,t)\}\right) \tag{10}$$

Intuitively, the max operator uses in evaluation a human answer that is the closest to the predicted one – which represents a minimal form of consensus.

	Accuracy	WUPS @0.9	WUPS @0.0
Subset: No agreement			
Language only (ours)			
- multiple words	8.86	12.46	38.89
- single word	8.50	12.05	40.94
Neural-Image-QA (ours)			
- multiple words	**10.31**	**13.39**	40.05
- single word	9.13	13.06	**43.48**
Subset: $\geq 50\%$ agreement			
Language only (ours)			
- multiple words	21.17	27.43	66.68
- single word	20.73	27.38	67.69
Neural-Image-QA (ours)			
- multiple words	20.45	27.71	67.30
- single word	**24.10**	**30.94**	**71.95**
Subset: Full Agreement			
Language only (ours)			
- multiple words	27.86	35.26	78.83
- single word	25.26	32.89	79.08
Neural-Image-QA (ours)			
- multiple words	22.85	33.29	78.56
- single word	**29.62**	**37.71**	**82.31**

Table 4. Results on DAQUAR, all classes, single reference in % (the subsets are chosen based on DAQUAR-Consensus).

4.3.3 Consensus results

Using the multiple reference answers in DAQUAR-Consensus we can show a more detailed analysis of inter human agreement. Figure 5 shows the fraction of the data where the answers agree between all available questions ("100"), at least 50% of the available questions and do not agree at all (no agreement - "0"). We observe that for the majority of the data, there is a partial agreement, but even full disagreement is possible. We split the dataset

	Accuracy	WUPS @0.9	WUPS @0.0
Average Consensus Metric			
Language only (ours)			
- multiple words	11.60	18.24	52.68
- single word	11.57	18.97	54.39
Neural-Image-QA (ours)			
- multiple words	11.31	18.62	53.21
- single word	**13.51**	**21.36**	**58.03**
Min Consensus Metric			
Language only (ours)			
- multiple words	22.14	29.43	66.88
- single word	22.56	30.93	69.82
Neural-Image-QA (ours)			
- multiple words	22.74	30.54	68.17
- single word	**26.53**	**34.87**	**74.51**

Table 5. Results on DAQUAR-Consensus, all classes, consensus in %.

into three parts according to the above criteria "No agreement", "$\geq 50\%$ agreement" and "Full agreement" and evaluate our models on these splits (Table 4 summarizes the results). On subsets with stronger agreement, we achieve substantial gains of up to 10% and 20% points in accuracy over the full set (Table 1) and the **Subset: No agreement** (Table 4), respectively. These splits can be seen as curated versions of DAQUAR, which allows studies with factored out ambiguities.

The aforementioned "Average Consensus Metric" generalizes the notion of the agreement, and encourages predictions of the most agreeable answers. On the other hand "Min Consensus Metric" has a desired effect of providing a more optimistic evaluation. Table 5 shows the application of both measures to our data and models.

Moreover, Table 6 shows that "MCM" applied to human answers at test time captures ambiguities in interpreting questions by improving the score of the human baseline from [20] (here, as opposed to Table 5, we exclude the original human answers from the measure). It also cooperates well with WUPS at 0.9, which takes word ambiguities into account, gaining an about 20% higher score.

4.4. Qualitative results

We show predicted answers of different variants of our architecture in Table 7, 8, and 9. We have chosen the examples to highlight differences between Neural-Image-QA and the "Language only". We use a "multiple words" approach only in Table 8, otherwise the "single word" model is shown. Despite some failure cases, "Language only" makes "reasonable guesses" like predicting that the largest object could be table or an object that could be found on the bed is either a pillow or doll.

	Accuracy	WUPS @0.9	WUPS @0.0
WUPS [20]	50.20	50.82	67.27
ACM (ours)	36.78	45.68	64.10
MCM (ours)	60.50	69.65	82.40

Table 6. Min and Average Consensus on human answers from DAQUAR, as reference sentence we use all answers in DAQUAR-Consensus which are not in DAQUAR, in %

4.5. Failure cases

While our method answers correctly on a large part of the challenge (e.g. $\approx$ 35 WUPS at 0.9 on "what color" and "how many" question subsets), spatial relations ($\approx$ 21 WUPS at 0.9) which account for a substantial part of DAQUAR remain challenging. Other errors involve questions with small objects, negations, and shapes (below 12 WUPS at 0.9). Too few training data points for the aforementioned cases may contribute to these mistakes.

Table 9 shows examples of failure cases that include (in order) strong occlusion, a possible answer not captured by our ground truth answers, and unusual instances (red toaster).

5. Conclusions

We have presented a neural architecture for answering natural language questions about images that contrasts with prior efforts based on semantic parsing and outperforms prior work by doubling performance on this challenging task. A variant of our model that does not use the image to answer the question performs only slightly worse and even outperforms a new human baseline that we have collected under the same condition. We conclude that our model has learnt biases and patterns that can be seen as forms of common sense and prior knowledge that humans use to accomplish this task. We observe that indoor scene statistics, spatial reasoning, and small objects are not well captured by the global CNN representation, but the true limitations of this representation can only be explored on larger datasets. We extended our existing DAQUAR dataset to DAQUAR-Consensus, which now provides multiple reference answers which allows to study inter-human agreement and consensus on the question answer task. We propose two new metrics: "Average Consensus", which takes into account human disagreement, and "Min Consensus" that captures disagreement in human question answering.

Acknowledgements. Marcus Rohrbach was supported by a fellowship within the FITweltweit-Program of the German Academic Exchange Service (DAAD).

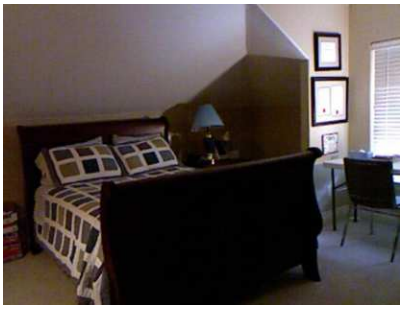
			
What is on the right side of the cabinet?		How many drawers are there?	What is the largest object?
Neural-Image-QA:	bed	3	bed
Language only:	bed	6	table

Table 7. Examples of questions and answers. Correct predictions are colored in green, incorrect in red.

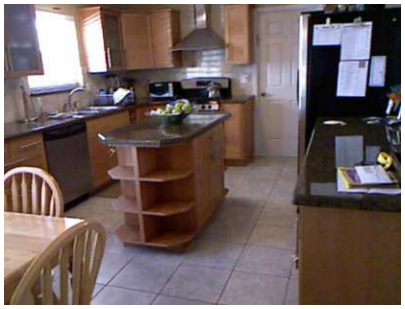
			
What is on the refrigerator?		What is the colour of the comforter?	What objects are found on the bed?
Neural-Image-QA:	magnet, paper	blue, white	bed sheets, pillow
Language only:	magnet, paper	blue, green, red, yellow	doll, pillow

Table 8. Examples of questions and answers with multiple words. Correct predictions are colored in green, incorrect in red.

			
How many chairs are there?		What is the object fixed on the window?	Which item is red in colour?
Neural-Image-QA:	1	curtain	remote control
Language only:	4	curtain	clock
Ground truth answers:	2	handle	toaster

Table 9. Examples of questions and answers - failure cases.

References

[1] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh. Vqa: Visual question answering. *arXiv:1505.00468*, 2015.

[2] J. Berant and P. Liang. Semantic parsing via paraphrasing. In *ACL*, 2014.

[3] K. Cho, B. van Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, D. Bahdanau, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*, 2014.

[4] J. Cohen et al. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 1960.

[5] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *CVPR*, 2015.

[6] J. L. Fleiss and J. Cohen. The equivalence of weighted kappa and the intraclass correlation coefficient as measures of reliability. *Educational and psychological measurement*, 1973.

[7] H. Gao, J. Mao, J. Zhou, Z. Huang, L. Wang, and W. Xu. Are you talking to a machine? dataset and methods for multilingual image question answering. In *NIPS*, 2015.

[8] D. Geman, S. Geman, N. Hallonquist, and L. Younes. Visual turing test for computer vision systems. *Proceedings of the National Academy of Sciences*, 2015.

[9] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[10] M. Iyyer, J. Boyd-Graber, L. Claudino, R. Socher, and H. D. III. A neural network for factoid question answering over paragraphs. In *EMNLP*, 2014.

[11] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093*, 2014.

[12] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *CVPR*, 2015.

[13] A. Karpathy, A. Joulin, and L. Fei-Fei. Deep fragment embeddings for bidirectional image sentence mapping. In *NIPS*, 2014.

[14] C. Kong, D. Lin, M. Bansal, R. Urtasun, and S. Fidler. What are you talking about? text-to-image coreference. In *CVPR*, 2014.

[15] J. Krishnamurthy and T. Kollar. Jointly learning to parse and perceive: Connecting natural language to the physical world. *TACL*, 2013.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[18] P. Liang, M. I. Jordan, and D. Klein. Learning dependency-based compositional semantics. *Computational Linguistics*, 2013.

[19] L. Ma, Z. Lu, and H. Li. Learning to answer questions from image using convolutional neural network. *arXiv:1506.00333*, 2015.

[20] M. Malinowski and M. Fritz. A multi-world approach to question answering about real-world scenes based on uncertain input. In *NIPS*, 2014.

[21] M. Malinowski and M. Fritz. Towards a visual turing challenge. In *Learning Semantics (NIPS workshop)*, 2014.

[22] C. Matuszek, N. Fitzgerald, L. Zettlemoyer, L. Bo, and D. Fox. A joint model of language and perception for grounded attribute learning. In *ICML*, 2012.

[23] N. Nakashole, T. Tylenda, and G. Weikum. Fine-grained semantic typing of emerging entities. In *ACL*, 2013.

[24] M. Ren, R. Kiros, and R. Zemel. Image question answering: A visual semantic embedding model and a new dataset. In *NIPS*, 2015.

[25] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *arXiv:1409.0575*, 2014.

[26] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[27] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.

[28] I. Sutskever, O. Vinyals, and Q. V. V. Le. Sequence to sequence learning with neural networks. In *NIPS*. 2014.

[29] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv:1409.4842*, 2014.

[30] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko. Sequence to sequence – video to text. In *ICCV*, 2015.

[31] S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. Mooney, and K. Saenko. Translating videos to natural language using deep recurrent neural networks. In *NAACL*, 2015.

[32] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. *arXiv:1411.4555*, 2014.

[33] J. Weston, S. Chopra, and A. Bordes. Memory networks. *arXiv:1410.3916*, 2014.

[34] Z. Wu and M. Palmer. Verbs semantics and lexical selection. In *ACL*, 1994.

[35] L. Yu, E. Park, A. C. Berg, and T. L. Berg. Visual madlibs: Fill in the blank image generation and question answering. *arXiv:1506.00278*, 2015.

[36] W. Zaremba and I. Sutskever. Learning to execute. *arXiv preprint arXiv:1410.4615*, 2014.

[37] C. L. Zitnick, D. Parikh, and L. Vanderwende. Learning the visual interpretation of sentences. In *ICCV*, 2013.

Ask Your Neurons: A Neural-based Approach to Answering Questions about Images - A Supplementary Material

Mateusz Malinowski[1] Marcus Rohrbach[2] Mario Fritz[1]
[1]Max Planck Institute for Informatics, Saarbrücken, Germany
[2]UC Berkeley EECS and ICSI, Berkeley, CA, United States

	Accuracy	WUPS @0.9	WUPS @0.0
Malinowski et al. [1]	7.86	11.86	38.79
Neural-Image-QA (ours)			
- multiple words	17.49	23.28	57.76
- single word	**19.43**	**25.28**	**62.00**
Human answers [1]	50.20	50.82	67.27
Language only (ours)			
- multiple words	17.06	22.30	56.53
- single word	17.15	22.80	58.42
Human answers, no images	7.34	13.17	35.56

Table 1. Results on DAQUAR, all classes, single reference, in %. Replication of Table 1 from the main paper for convenience.

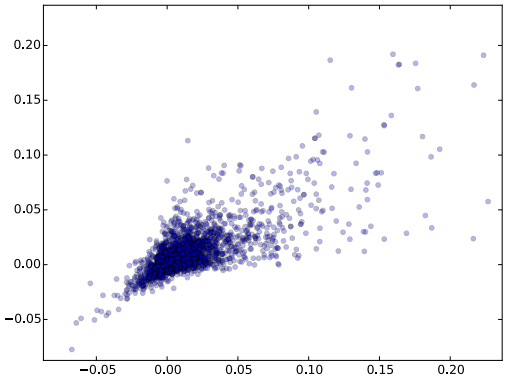


Figure 1. Figure showing correlation between question and answer words of the "Language only" model (at x-axis), and a similar correlation of the "Human-baseline" [1] (at y-axis).

In this supplemental material, we additionally provide qualitative examples of different variants of our architecture and show the correlations of predicted answer words and question words with human answer and question words.

The examples are chosen to highlight challenges as well as differences between "Neural-Image-QA" and "Language only" architectures. Table 9 also shows a few failure cases. In all cases but "multiple words answer", we use the best "single word" variants. Although "Language only" ignores the image, it is still able to make "reasonable guesses" by exploiting biases captured by the dataset that can be viewed as a type of common sense knowledge. For instance, "tea kettle" often sits on the oven, cabinets are usually "brown", "chair" is typically placed in front of a table, and we commonly keep a "photo" on a cabinet (Table 2, 4, 5, 8). This effect is analysed in Figure 1. Each data point in the plot represents the correlation between a question and a predicted answer words for our "Language only" model (x-axis) versus the correlation in the human answers (y-axis).

Despite the reasonable guesses of the "Language only" architecture, the "Neural-Image-QA" predicts in average better answers (shown in Table 1 that we have replicated from the main paper for the convenience of the reader) by exploiting the visual content of images. For instance in Table 6 the "Language only" model incorrectly answers "6" on the question "How many burner knobs are there ?" because it has seen only this answer during the training with exactly the same question but on different image.

Both models, "Language only" and "Neural-Image-QA", have difficulties to answer correctly on long questions or such questions that expect a larger number of answer words (Table 9). On the other hand both models are doing well on predicting a type of the question (e.g. "what color ..." result in a color name in the answer, or "how many ..." questions result in a number), there are a few rare cases with an incorrect type of the predicted answer (the last example in Table 9).

References

[1] M. Malinowski and M. Fritz. A multi-world approach to question answering about real-world scenes based on uncertain input. In *NIPS*, 2014.

			
	What are the objects close to the wall?	What is on the stove?	What is left of sink?
Neural-Image-QA:	wall decoration	tea kettle	tissue roll
Language only:	books	tea kettle	towel
Ground truth answers:	wall decoration	tea kettle	tissue roll

Table 2. Examples of compound answer words.

			
	How many lamps are there?	How many pillows are there on the bed?	How many pillows are there on the sofa?
Neural-Image-QA:	2	2	3
Language only:	2	3	3
Ground truth answers:	2	2	3

Table 3. Counting questions.

			
	What color is the towel?	What color are the cabinets?	What is the colour of the pillows?
Neural-Image-QA:	brown	brown	black, white
Language only:	white	brown	blue, green, red
Ground truth answers:	white	brown	black, red, white

Table 4. Questions about color.



	What is hanged on the chair?	What is the object close to the sink?	What is the object on the table in the corner?
Neural-Image-QA:	clothes	faucet	lamp
Language only:	jacket	faucet	plant
Ground truth answers:	clothes	faucet	lamp

Table 5. Correct answers by our "Neural-Image-QA" architecture.



	What are the things on the cabinet?	What is in front of the shelf?	How many burner knobs are there?
Neural-Image-QA:	photo	chair	4
Language only:	photo	basket	6
Ground truth answers:	photo	chair	4

Table 6. Correct answers by our "Neural-Image-QA" architecture.



	What is the object close to the counter?	What is the colour of the table and chair?	How many towels are hanged?
Neural-Image-QA:	sink	brown	3
Language only:	stove	brown	4
Ground truth answers:	sink	brown	3

Table 7. Correct answers by our "Neural-Image-QA" architecture.

			
	What is on the right most side on the table?	What are the things on the coffee table?	What is in front of the table?
Neural-Image-QA:	lamp	books	chair
Language only:	machine	jacket	chair
Ground truth answers:	lamp	books	chair

Table 8. Correct answers by our "Neural-Image-QA" architecture.

			
	What is on the left side of the white oven on the floor and on right side of the blue armchair?	What are the things on the cabinet?	What color is the frame of the mirror close to the wardrobe?
Neural-Image-QA:	oven	chair, lamp, photo	pink
Language only:	exercise equipment	candelabra	curtain
Ground truth answers:	garbage bin	lamp, photo, telephone	white

Table 9. Failure cases.

Learning to See by Moving

Pulkit Agrawal
UC Berkeley
pulkitag@eecs.berkeley.edu

João Carreira
UC Berkeley
carreira@eecs.berkeley.edu

Jitendra Malik
UC Berkeley
malik@eecs.berkeley.edu

Abstract

The current dominant paradigm for feature learning in computer vision relies on training neural networks for the task of object recognition using millions of hand labelled images. Is it also possible to learn useful features for a diverse set of visual tasks using any other form of supervision ? In biology, living organisms developed the ability of visual perception for the purpose of moving and acting in the world. Drawing inspiration from this observation, in this work we investigate if the awareness of egomotion can be used as a supervisory signal for feature learning. As opposed to the knowledge of class labels, information about egomotion is freely available to mobile agents. We show that using the same number of training images, features learnt using egomotion as supervision compare favourably to features learnt using class-label as supervision on the tasks of scene recognition, object recognition, visual odometry and keypoint matching.

"We move in order to see and we see in order to move"

J.J Gibson

1. Introduction

Recent advances in computer vision have shown that visual features learnt by neural networks trained for the task of object recognition using more than a million labelled images are useful for many computer vision tasks like semantic segmentation, object detection and action classification [18, 10, 1, 33]. However, object recognition is one among many tasks for which vision is used. For example, humans use visual perception for recognizing objects, understanding spatial layouts of scenes and performing actions such as moving around in the world. Is there something special about the task of object recognition or is it the case that useful visual representations can be learnt through other modes of supervision? Clearly, biological agents perform complex visual tasks and it is unlikely that they require external supervision in form of millions of labelled examples. Unlabelled visual data is freely available and in theory this data can be used to learn useful visual representations. However, until now unsupervised learning approaches [4, 22, 29, 32] have not yet delivered on their promise and are nowhere to be seen in current applications on complex real world imagery.

Biological agents use perceptual systems for obtaining sensory information about their environment that enables them to act and accomplish their goals [13, 9]. Both biological and robotic agents employ their motor system for executing actions in their environment. Is it also possible that these agents can use their own motor system as a source of supervision for learning useful perceptual representations? Motor theories of perception have a long history [13, 9], but there has been little work in formulating computational models of perception that make use of motor information. In this work we focus on visual perception and present a model based on egomotion (i.e. self motion) for learning useful visual representations. When we say useful visual representations [34], we mean representations that possess the following two characteristics - (1) ability to perform multiple visual tasks and (2) ability of performing new visual tasks by learning from only a few labeled examples provided by an extrinsic teacher.

Mobile agents are naturally aware of their egomotion (i.e. self-motion) through their own motor system. In other words, knowledge of egomotion is "freely" available. For example, the vestibular system provides the sense of orientation in many mammals. In humans and other animals, the brain has access to information about eye movements and the actions performed by the animal [9]. A mobile robotic agent can estimate its egomotion either from the motor commands it issues to move or from odometry sensors such as gyroscopes and accelerometers mounted on the agent itself.

We propose that useful visual representations can be learnt by performing the simple task of correlating visual stimuli with egomotion. A mobile agent can be treated like a camera moving in the world and thus the knowledge of egomotion is the same as the knowledge of camera motion. Using this insight, we pose the problem of correlating visual stimuli with egomotion as the problem of predicting the camera transformation from the consequent pairs of im-

ages that the agent receives while it moves. Intuitively, the task of predicting camera transformation between two images should force the agent to learn features that are adept at identifying visual elements that are present in both the images (i.e. visual correspondence). In the past, features such as SIFT, that were hand engineered for finding correspondences were also found to be very useful for tasks such as object recognition [23, 19]. This suggests that egomotion based learning can also result in features that are useful for such tasks.

In order to test our hypothesis of feature learning using egomotion, we trained multilayer neural networks to predict the camera transformation between pairs of images. As a proof of concept, we first demonstrate the usefulness of our approach on the MNIST dataset [21]. We show that features learnt using our method outperform previous approaches of unsupervised feature learning when class-label supervision is available only for a limited number of examples (section 3.4) Next, we evaluated the efficacy of our approach on real world imagery. For this purpose, we used image and odometry data recorded from a car moving through urban scenes, made available as part of the KITTI [12] and the San Francisco (SF) city [7] datasets. This data mimics the scenario of a robotic agent moving around in the world. The quality of features learnt from this data were evaluated on four tasks (1) Scene recognition on SUN [38] (section 5.1), (2) Visual odometery (section 5.4), (3) Keypoint matching (section 5.3) and (4) Object recognition on Imagenet [31] (section 5.2). Our results show that for the same amount of training data, features learnt using egomotion as supervision compare favorably to features learnt using class-label as supervision. We also show that egomotion based pretraining outperforms a previous approach based on slow feature analysis for unsupervised learning from videos [37, 14, 25]. To the best of our knowledge, this work provides the first effective demonstration of learning visual representations from non-visual access to egomotion information in real world setting.

The rest of this paper is organized as following: In section 2 we discuss the related work, in section 3, 4, 5 we present the method, dataset details and we conclude with the discussion in section 6.

2. Related Work

Past work in unsupervised learning has been dominated by approaches that pose feature learning as the problem of discovering compact and rich representations of images that are also sufficient to reconstruct the images [6, 3, 22, 32, 27, 30]. Another line of work has focused on learning features that are invariant to transformations either from video [37, 14, 25] or from images [11, 29]. [24] perform feature learning by modeling spatial transformations using boltzmann machines, but donot evaluate the quality of learnt features.

Despite a lot of work in unsupervised learning (see [4] for a review), a method that works on complex real world imagery is yet to be developed. An alternative to unsupervised learning is to learn features using intrinsic reward signals that are freely available to a system (i.e self-supervised learning). For instance, [15] used intrinsic reward signals available to a robot for learning features that predict path traversability, while [28] trained neural networks for driving vehicles directly from visual input.

In this work we propose to use non-visual access to egomotion information as a form of self-supervision for visual feature learning. Unlike any other previous work, we show that our method works on real world imagery. Closest to our method is the the work of transforming auto-encoders [16] that used egomotion to reconstruct the transformed image from an input source image. This work was purely conceptual in nature and the quality of learned features was not evaluated. In contrast, our method uses egomotion as supervision by predicting the transformation between two images using a siamese-like network model [8].

Our method can also be seen as an instance of feature learning from videos. [37, 14, 25] perform feature learning from videos by imposing the constraint that temporally close frames should have similar feature representations (i.e. slow feature analysis) without accounting for either the camera motion or the motion of objects in the scene. In many settings the camera motion dominates the motion content of the video. Our key observation is that knowledge of camera motion (i.e. egomotion) is *freely* available to mobile agents and can be used as a powerful source of self-supervision.

3. A Simple Model of Motion-based Learning

We model the visual system of the agent with a Convolutional Neural Network (CNN, [20]). The agent optimizes its visual representations (i.e. updating the weights of the CNN) by minimizing the error between the egomotion information (i.e. camera transformation) obtained from its motor system and egomotion predicted using its visual inputs only. Performing this task is equivalent to training a CNN with two streams (i.e. Siamese Style CNN or SCNN[8]) that takes two images as inputs and predicts the egomotion that the agent underwent as it moved between the two spatial locations from which the two images were obtained. In order to learn useful visual representations, the agent continuously performs this task as it moves around in its environment.

In this work we use the pretraining-finetuning paradigm for evaluating the utility of learnt features. Pretraining is the process of optimizing the weights of a randomly initialized CNN for an auxiliary task that is not the same as the target task. Finetuning is the process of modifying the weights of

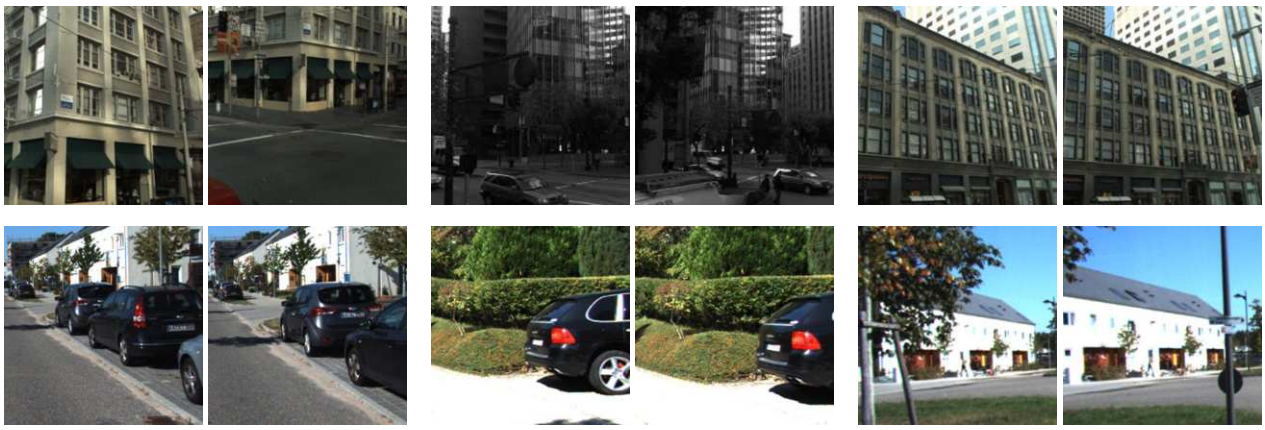


Figure 1: Exploring the utility of egomotion as supervision for learning useful visual features. A mobile agent equipped with visual sensors receives a sequence of images as inputs while it moves in its environment. The movement of the agent is equivalent to the movement of a camera. In this work, egomotion based learning is posed as the problem of predicting camera transformation from image pairs. The top and bottom rows of the figure show some sample image pairs from the SF and KITTI datasets that were used for feature learning.

a pretrained CNN for the given target task. Our experiments compare the utility of features learnt using egomotion based pretraining against class-label based and slow-feature based pretraining on multiple target tasks.

3.1. Two Stream Architecture

Each stream of the CNN independently computes features for one image. Both streams share the same architecture and the same set of weights and consequently perform the same set of operations for computing features. The individual streams have been called as Base-CNN (BCNN). Features from two BCNNs are concatenated and passed downstream into another CNN called as the Top-CNN (TCNN) (see figure 2). TCNN is responsible for using the BCNN features to predict the camera transformation between the input pair of images. After pretraining, the TCNN is removed and a single BCNN is used as a standard CNN for feature computation for the target task.

3.2. Shorthand for CNN architectures

The abbreviations Ck, Fk, P, D, Op represent a convolutional(C) layer with k filters, a fully-connected(F) layer with k filters, pooling(P), dropout(D) and the output(Op) layers respectively. We used ReLU non-linearity after every convolutional/fully-connected layer, except for the output layer. The dropout layer was always used with dropout of 0.5. The output layer was a fully connected layer with number of units equal to the number of desired outputs. As an example of our notation, C96-P-F500-D refers to a network with 96 filters in the convolution layer followed by ReLU non-linearity, a pooling layer, a fully-connected layer with 500 unit, ReLU non-linearity and a dropout layer. We used

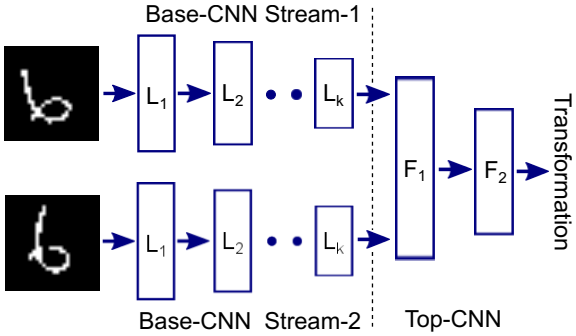


Figure 2: Description of the method for feature learning. Visual features are learnt by training a Siamese style Convolutional Neural Network (SCNN, [8]) that takes as inputs two images and predicts the transformation between the images (i.e. egomotion). Each stream of the SCNN (called as Base-CNN or BCNN) computes features for one image. The outputs of two BCNNs are concatenated and passed as inputs to a second multilayer CNN called as the Top-CNN (TCNN) (shown as layers F_1, F_2). The two BCNNs have the same architecture and share weights. After feature learning, TCNN is discarded and a single BCNN stream is used as a standard CNN for extracting features for performing target tasks like scene recognition.

[17] for training all our models.

3.3. Slow Feature Analysis (SFA) Baseline

Slow Feature Analysis (SFA) is a method for feature learning based on the principle that useful features change

slowly in time. We used the following contrastive loss formulation of SFA [8, 25],

$$L(x_{t_1}, x_{t_2}, W) =$$

$$\begin{cases} D(x_{t_1}, x_{t_2}) & \text{if } |t_1 - t_2| \leq T \\ 1 - max\big(0, m - D(x_{t_1}, x_{t_2})\big) & \text{if } |t_1 - t_2| > T \end{cases} \quad (1)$$

where, L is the loss, x_{t_1}, x_{t_2} refer to feature representations of frames observed at times t_1, t_2 respectively, W are the parameters that specify the feature extraction process, D is a measure of distance with parameter, m is a predefined margin and T is a predefined time threshold for determining whether the two frames are temporally close or not. In this work, x_t are features computed using a CNN with weights W and D was chosen to be the L2 distance. SFA pretraining was performed using two stream architectures that took pairs of images as inputs and produced outputs x_{t_1}, x_{t_2} as outputs from the two streams respectively.

3.4. Proof of Concept using MNIST

On MNIST, egomotion was emulated by generating synthetic data consisting of random transformation (translations and rotations) of digit images. From the training set of 60K images, digits were randomly sampled and then transformed using two different sets of random transformations to generate image pairs. CNNs were trained for predicting the transformations between these image pairs.

3.4.1 Data

For egomotion based pretraining, relative translation between the digits was constrained to be an integer value in the range [-3, 3] and relative rotation θ was constrained to lie within the range [-30°, 30°]. The prediction of transformation was posed as a classification task with three separate soft-max losses (one each for translation along X, Y axes and the rotation about Z-axis). SCNN was trained to minimize the sum of these three losses. Translations along X, Y were separately binned into seven uniformly spaced bins each. The rotations were binned into bins of size 3°each resulting into a total of 20 bins (or classes). For SFA based pretraining, image pairs with relative translation in the range [-1, 1] and relative rotation within [-3°, 3°] were considered to be temporally close to each other (see equation 1). A total of 5 million image pairs were used for both pretraining procedures.

3.4.2 Network Architectures

We experimented with multiple BCNN architectures and chose the optimal architecture for each pretraining method separately. For egmotion based pretraining, the two BCNN streams were concatenated using the TCNN: *F1000-D-Op*.

Table 1: Comparison of various pretraining methods on MNIST reveals that egomotion based pretraining outperforms many previous approaches for unsupervised learning. The performance is reported as the **error rate**.

Method	# examples for finetuning			
	100	**300**	**1000**	**10000**
Autoencoder [17]	24.1	12.2	7.7	4.8
Ranzato et al. [29]	-	7.18	3.21	**0.85**
Lee et al. [22]	-	-	2.62	-
Train from Scratch	20.1	8.3	4.5	1.6
SFA (m=10)	11.2	6.4	3.5	2.1
SFA (m=100)	11.9	6.4	4.8	4.7
Egomotion (ours)	**8.7**	**3.6**	**2.0**	0.9

Pretraining was performed for 40K iterations (i.e. 5M examples) using an initial learning rate of 0.01 which was reduced by a factor of 2 after every 10K iterations.

The following architecture was used for finetuning: *BCNN-F500-D-Op*. In order to evaluate the quality of BCNN features, the learning rate of all layers in the BCNN were set to 0 during finetuning for digit classification. Finetuning was performed for 4K iterations (which is equivalent to training for 50 epochs for the 10K labelled training examples) with a constant learning rate of 0.01.

3.4.3 Results

The BCNN features were evaluated by computing the error rates on the task of digit classification using 100, 300, 1K and 10K class-labelled examples for training. These sets were constructed by randomly sampling digits from the standard training set of 60K digits. For this part of the experiment, the original digit images were used (i.e. without any transformations or data augmentation). The standard test set of 10K digits was used for evaluation and error rates averaged across 3 runs are reported in table 1.

The BCNN architecture: C96-P-C256-P, was found to be optimal for egomotion and SFA based pretraining and also for training from scratch (i.e. random weight initialization). Results for other architectures are provided in the supplementary material. For SFA based pretraining, we experimented with multiple values of the margin m and found that $m = 10, 100$ led to the best performance. Our method outperforms convolutional deep belief networks [22], a previous approach based on learning features invariant to transformations [29] and SFA based pretraining.

4. Learning Visual Features From Egomotion in Natural Environments

We used two main sources of real world data for feature learning: the KITTI and SF datasets, which were collected using cameras and odometry sensors mounted on a car driving through urban scenes. Details about the data, the experimental procedure, the network architectures and the results are provided in sections 4.1, 4.2, 4.3 and 5 respectively.

4.1. KITTI Dataset

The KITTI dataset provided odometry and image data recorded during 11 short trips of variable length made by a car moving through urban landscapes. The total number of frames in the entire dataset was 23,201. Out of 11, 9 sequences were used for training and 2 for validation. The total number of images in the training set was 20,501.

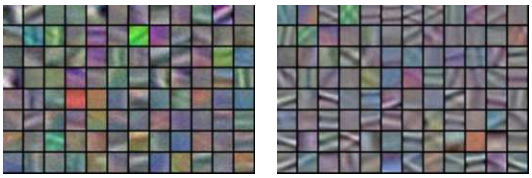
The odometry data was used to compute the camera transformation between pairs of images recorded from the car. The direction in which the camera pointed was assumed to be the Z axis and the image plane was taken to be the XY plane. X-axis and Y-axis refer to horizontal and vertical directions in the image plane. As significant camera transformations in the KITTI data were either due to translations along the Z/X axis or rotation about the Y axis, only these three dimensions were used to express the camera transformation. The rotation was represented as the euler angle about the Y-axis. The task of predicting the transformation between pair of images was posed as a classification problem. The three dimensions of camera transformation were individually binned into 20 uniformly spaced bins each. The training image pairs were selected from frames that were at most ± 7 frames apart to ensure that images in any given pair would have a reasonable overlap. For SFA based pretraining, pairs of frames that were separated by atmost ± 7 frames were considered to be temporally close to each other.

The SCNN was trained to predict camera transformation from pairs of 227×227 pixel sized image regions extracted from images of overall size 370×1226 pixels. For each image pair, the coordinates for cropping image regions were randomly chosen. Figure 1 illustrates typical image crops.

4.2. SF Dataset

SF dataset provides camera transformation between $\approx$ 136K pairs of images (constructed from a set of 17,357 unique images). This dataset was constructed using Google StreetView [7]. $\approx 130K$ image pairs were used for training and $\approx 6K$ pairs for validation.

Just like KITTI, the task of predicting camera transformation was posed as a classification problem. Unlike KITTI, significant camera transformation was found along all six dimensions of transformation (i.e. the 3 euler angles and the 3 translations). Since, it is unreasonable to expect that visual features can be used to infer big camera transformations, rotations between [-30°, 30°] were binned into 10 uniformly spaced bins and two extra bins were used for rotations larger and smaller than 30°and -30°respectively. The three translations were individually binned into 10 uniformly spaced bins each. Images were resized to a size of 360×480 and image regions of size 227×227 were used for training the SCNN.



(a) KITTI-Net (b) SF-Net

Figure 3: Visualization of layer 1 filters learnt by egomotion based pretraining on (a) KITTI and (b) SF datasets. A large majority of layer-1 filters are color detectors and some of them are edge detectors. This is expected as color is a useful cue for determining correspondences between image pairs.

4.3. Network Architecture

BCNN closely followed the architecture of first five AlexNet layers [18]: *C96-P-C256-P-C384-C384-C256-P*. TCNN architecture was: *C256-C128-F500-D-Op*. The convolutional filters in the TCNN were of spatial size 3×3. The networks were trained for 60K iterations with a batch size of 128. The initial learning rate was set to 0.001 and was reduced by a factor of two after every 20K iterations.

We term the networks pretrained using egomotion on KITTI and SF datasets as KITTI-Net and SF-Net respectively. The net pretrained on KITTI with SFA is called KITTI-SFA-Net. Figure 3 shows the layer-1 filters of KITTI-Net and SF-Net. A large majority of layer-1 filters are color detectors, while some of them are edge detectors. As color is a useful cue for determining correspondences between closeby frames of a video sequence, learning of color detectors as layer-1 filters is not surprising. The fraction of filters that detect edges is higher for the SF-Net. This is not surprising either, because higher fraction of images in the SF dataset contain structured objects like buildings and cars.

5. Evaluating Motion-based Learning

For evaluating the merits of the proposed approach, features learned using egomotion based supervision were compared against features learned using class-label and SFA based supervision on the challenging tasks of scene recogni-

tion, intra-class keypoint matching and visual odometry and object recognition. The ultimate goal of feature learning is to find features that can generalize from only a few supervised examples on a new task. Therefore it makes sense to evaluate the quality of features when only a few labelled examples for the target task are provided. Consequently, the scene and object recognition experiments were performed in the setting when only 1-20 labelled examples per class were available for finetuning.

The KITTI-Net and SF-Net (examples of models trained using egomotion based supervision) were trained using only only $\approx$ 20K unique images. To make a fair comparison with class-label based supervision, a model with AlexNet architecture was trained using only 20K images taken from the training set of ILSVRC12 challenge (i.e. 20 examples per class). This model has been referred to as AlexNet-20K. In addition, some experiments presented in this work also make comparison with AlexNet models trained with 100K and 1M images that have been named as AlexNet-100K and AlexNet-1M respectively.

5.1. Scene Recognition

SUN dataset consisting of 397 indoor/outdoor scene categories was used for evaluating scene recognition performance. This dataset provides 10 standard splits of 5 and 20 training images per class and a standard test set of 50 images per class. Due to time limitation of running 10 runs of the experiment, we evaluated the performance using only 3 train/test splits.

For evaluating the utility of CNN features produced by different layers, separate linear (SoftMax) classifiers were trained on features produced by individual CNN layers (i.e. BCNN layers of KITTI-Net, KITTI-SFA-Net and SF-Net). Table 2 reports recognition accuracy (averaged over 3 train/test splits) for various networks considered in this study. KITTI-Net outperforms SF-Net and is comparable to AlexNet-20K. This indicates that given a fixed budget of pretraining images, egomotion based supervision learns features that are almost as good as the features using class-based supervision on the task of scene recognition. The performance of features computed by layers 1-3 (abbreviated as L1, L2, L3 in table 2) of the KITTI-SFA-Net and KITTI-Net is comparable, whereas layer 4, 5 features of KITTI-Net significantly outperform layer 4, 5 features of KITTI-SFA-Net. This indicates that egomotion based pretraining results into learning of higher-level features, while SFA based pretraining results into learning of lower-level features only.

The KITTI-Net outperforms GIST[26], which was specifically developed for scene classification, but is outperformed by Dense SIFT with spatial pyramid matching (SPM) kernel [19]. The KITTI-Net was trained using limited visual data ($\approx 20Kframes$) containing visual imagery of limited diversity. The KITTI data mainly contains images of roads, buildings, cars, few pedestrians, trees and some vegetation. It is in fact surprising that a network trained on data with such little diversity is competitive on classifying indoor and outdoor scenes with the AlexNet-20K that was trained on a much more diverse set of images. We believe that with more diverse training data for egomotion based learning, the performance of learnt features will be better than currently reported numbers.

The KITTI-Net outperformed the SF-Net except for the performance of layer 1 (L1). As it was possible to extract a larger number of image region pairs from the KITTI dataset as compared to the SF dataset (see section 4.1, 4.2), the result that KITTI-Net outperforms SF-Net is not surprising. Because KITTI-Net was found to be superior to the SF-Net in this experiment, the KITTI-Net was used for all other experiments described in this paper.

5.2. Object Recognition

If egomotion based pretraining learns useful features for object recognition, then a net initialized with KITTI-Net weights should outperform a net initialized with random weights on the task of object recognition. For testing this, we trained CNNs using 1, 5, 10 and 20 images per class from the ILSVRC-2012 challenge. As this dataset contains 1000 classes, the total number of training examples available for training for these networks were 1K, 5K, 10K and 20K respectively. All layers of KITTI-Net, KITTI-SFA-Net and AlexNet-Scratch (i.e. CNN with random weight initialization) were finetuned for image classification.

The results of the experiment presented in table 3 show that egomotion based supervision (KITTI-Net) clearly outperforms SFA based supervision(KITTI-SFA-Net) and AlexNet-Scratch. As expected, the improvement offered by motion-based pretraining is larger when the number of examples provided for the target task are fewer. These result show that egomotion based pretraining learns features useful for object recognition.

5.3. Intra-Class Keypoint Matching

Identifying the same keypoint of an object across different instances of the same object class is an important visual task. Visual features learned using egomotion, SFA and class-label based supervision were evaluated for this task using keypoint annotations on the PASCAL dataset [5].

Keypoint matching was computed in the following way: First, ground-truth object bounding boxes (GT-BBOX) from PASCAL-VOC2012 dataset were extracted and resized (while preserving the aspect ratio) to ensure that the smaller side of the boxes was of length 227 pixels. Next, feature maps from layers 2-5 of various CNNs were computed for every GT-BBOX. The keypoint matching score was computed between all pairs of GT-BBOX belonging to the same object class. For given pair of GT-BBOX, the fea-

Table 2: Comparing the **accuracy** of neural networks pre-trained using motion-based and class-label based supervision for the task of scene recognition on the SUN dataset. The performance of layers 1-6 (labelled as L1-L6) of these networks was evaluated after finetuning the network using 5/20 images per class from the SUN dataset. The performance of the KITTI-Net (i.e. motion-based pretraining) fares favorably with a network pretrained on Imagenet (i.e. class-based pretraining) with the same number of pretraining images (i.e. 20K).

Method	**Pretrain Supervision**	**#Pretrain**	**#Finetune**	**L1**	**L2**	**L3**	**L4**	**L5**	**L6**	**#Finetune**	**L1**	**L2**	**L3**	**L4**	**L5**	**L6**
AlexNet-1M	Class-Label	1M	5	5.3	10.5	12.1	12.5	18.0	23.6	20	11.8	22.2	25.0	26.8	33.3	37.6
AlexNet-20K		20K	5	4.9	6.3	6.6	6.3	6.6	6.7	20	8.7	12.6	12.4	11.9	12.5	12.4
KITTI-SFA-Net	Slowness	20.5K	5	4.5	5.7	6.2	3.4	0.5	-	20	8.2	11.2	12.0	7.3	1.1	-
SF-Net	Egomotion	18K	5	4.4	5.2	4.9	5.1	4.7	-	20	8.6	11.6	10.9	10.4	9.1	-
KITTI-Net		20.5K	5	4.3	6.0	5.9	5.8	6.4	-	20	7.9	12.2	12.1	11.7	12.4	-
GIST [38]	Human	-	5	6.2						20	11.6					
SPM [38]	Human	-	5	8.4						20	16.0					

Table 3: Top-5 **accuracy** on the task of object recognition on the ILSVRC-12 validation set. AlexNet-Scratch refers to a net with AlexNet architecture initialized with randomly weights. The weights of KITTI-Net and KITTI-SFA-Net were learned using egomotion based and SFA based supervision on the KITTI dataset respectively. All the networks were finetuned using $1, 5, 10, 20$ examples per class. The KITTI-Net clearly outperforms AlexNet-Scratch and KITTI-SFA-Net.

Method	1	5	10	20
AlexNet-Scratch	1.1	3.1	5.9	14.1
KITTI-SFA-Net (Slowness)	1.5	3.9	6.1	14.9
KITTI-Net (Egomotion)	**2.3**	**5.1**	**8.6**	**15.8**

tures associated with keypoints in the first image were used to predict the location of the same keypoints in the second image. The normalized pixel distance between the actual and predicted keypoint locations was taken as the error in keypoint matching. More details about this procedure have been provided in the supp. materials.

It is natural to expect that accuracy of keypoint matching would depend on the camera transformation between the two viewpoints of the object(i.e. viewpoint distance). In order to make a holistic evaluation of the utility of features learnt by different pretraining methods on this task, matching error was computed as a function of viewpoint distance [36]. Figure 4 reports the matching error averaged across all keypoints, all pairs of GT-BBOX and all classes using features extracted from layers conv-3 and conv-4.

KITTI-Net trained only with 20K unique frames was superior to AlexNet-20K and AlexNet-100K and inferior only to AlexNet-1M. A net with AlexNet architecture initialized with random weights (AlexNet-Rand), surprisingly performed better than AlexNet-20K. One possible explanation for this observation is that with only 20K examples, features learnt by AlexNet-20K only capture coarse global appearance of objects and are therefore poor at keypoint matching. SIFT has been hand engineered for finding correspondences across images and performs as well as the best AlexNet-1M features for this task (i.e. conv-4 features). KITTI-Net also significantly outperforms KITTI-SFA-Net. These results indicate that features learnt by egomotion based pretraining are superior to SFA and class-label based pretraining for the task of keypoint matching.

Table 4: Comparing the **accuracy** of various pretraining methods on the task of visual odometry.

Method	**Translation Acc.**			**Rotation Acc.**		
	δX	δY	δZ	$\delta\theta_1$	$\delta\theta_2$	$\delta\theta_3$
SF-Net	40.2	**58.2**	38.4	45.0	**44.8**	40.5
KITTI-Net	**43.4**	57.9	**40.2**	**48.4**	44.0	**41.0**
AlexNet-1M	41.8	58.0	39.0	46.0	44.5	40.5

5.4. Visual Odometry

Visual odometry is the task of estimating the camera transformation between image pairs. All layers of KITTI-Net and AlexNet-1M were finetuned for 25K iterations using the training set of SF dataset on the task of visual odometry (see section 4.2 for task description). The performance of various CNNs was evaluated on the validation set of SF dataset and the results are reported in table 4.

Performance of KITTI-Net was either superior or comparable to AlexNet-1M on this task. As the evaluation was made on the SF dataset itself, it was not surprising that on some metrics SF-Net outperformed KITTI-Net. The results of this experiment indicate that egomotion based feature learning is superior to class-label based feature learning on the task of visual odometry.

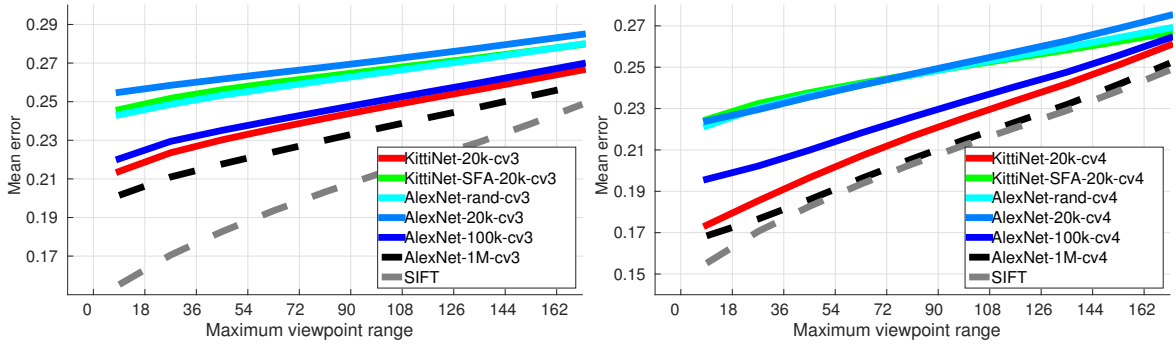


Figure 4: Intra-class keypoint matching error as a function of viewpoint distance averaged over 20 PASCAL objects using features from layers conv3 (left) and conv4 (right) of various CNNs used in this work. Please see the text for more details.

6. Discussion

In this work, we have shown that egomotion is a useful source of intrinsic supervision for visual feature learning in mobile agents. In contrast to class labels, knowledge of egomotion is "freely" available. On MNIST, egomotion-based feature learning outperforms many previous unsupervised methods of feature learning. Given the same budget of pretraining images, on task of scene recognition, egomotion-based learning performs almost as well as class-label-based learning. Further, egomotion based features outperform features learnt by a CNN trained using class-label supervision on two orders of magnitude more data (AlexNet-1M) on the task of visual odometry and one order of magnitude more data on the task of intra-class keypoint matching. In addition to demonstrating the utility of egomotion based supervision, these results also suggest that features learnt by class-label based supervision are not optimal for all visual tasks. This means that future work should look at what kinds of pretraining are useful for what tasks.

One potential criticism of our work is that we have trained and evaluated high capacity deep models on relatively little data (e.g. only 20K unique images available on the KITTI dataset). In theory, we could have learnt better features by downsizing the networks. For example, in our experiments with MNIST we found that pretraining a 2-layer network instead of 3-layer results in better performance (table 1). In this work, we have made a conscious choice of using standard deep models because the main goal of this work was not to explore novel feature extraction architectures but to investigate the value of egmotion for learning visual representations on architectures known to perform well on practical applications. Future research focused on exploring architectures that are better suited for egomotion based learning can only make a stronger case for this line of work. While egomotion is freely available to mobile agents, there are currently no publicly available datasets as large as Imagenet. Consequently, we were unable to evaluate the utility of motion-based supervision across the full spectrum of training set sizes.

In this work, we chose to first pretrain our models using a base task (i.e. egomotion) and then finetune these models for target tasks. An equally interesting setting is that of online learning where the agent has continuous access to intrinsic supervision (such as egomotion) and occasional explicit access to extrinsic teacher signal (such as the class labels). We believe that such a training procedure is likely to result in learning of better features. Our intuition behind this is that seeing different views of the same instance of an object (say) car, may not be sufficient to learn that different instances of the car class should be grouped together. The occasional extrinsic signal about object labels may prove useful for the agent to learn such concepts. Also, current work makes use of passively collected egomotion data and it would be interesting to investigate if it is possible to learn better visual representations if the agent can actively decide on how to explores its environment (i.e. active learning [2]).

Acknowledgements

This work was supported in part by ONR MURI-N00014-14-1-0671. Pulkit Agrawal was partially supported by Fulbright Science and Technology Fellowship. João Carreira was supported by the Portuguese Science Foundation, FCT, under grant SFRH/BPD/84194/2012. We gratefully acknowledge NVIDIA corporation for the donation of Tesla GPUs for this research.

Appendix

A. Keypoint Matching Score

Consider images of two instances of the same object class (for example airplane images as shown in first row of figure 5) for which keypoint matching score needs to be computed.

The images are pre-processed in the following way:

- Crop the groundtruth bounding box from the image.
- Pad the images by 30 pixels along each dimension.
- Resize each image so that the smallest side is 227 pixels. The aspect ratio of the image is preserved.

A.1. Keypoint Matching using CNN

Assume that the l^{th} layer of the CNN is used for feature computation. The feature map produced by the l^{th} layer is of dimensionality $I \times J \times M$, where (I, J) are the spatial dimensions and M is the number of filters in the l^{th} layer. Thus, the l^{th} layer produces a M dimensional feature vector for each of the $I \times J$ grid position in the feature map.

The coordinates of the keypoints are provided in the image coordinate system [5]. For the keypoints in the first image, we first determine their grid position in the $I \times J$ feature map. Each grid position has an associated receptive field in the image. The keypoints are assigned to the grid positions for which the center of receptive field is closest to the keypoints. This means that each keypoint is assigned one location in the feature map.

Let the M dimensional feature vector associated with the k^{th} keypoint in the first image be F_1^k. Let the M dimensional feature vector at grid location C_{ij} for the second image be $F_2(C_{ij})$. The location of matching keypoint in the second image is determined by solving:

$$C_* = argmin_{C_{ij}} \|F_1^k - F_2(C_{ij})\|_2 \tag{2}$$

C_* is transformed into the image coordinate system by computing the center of receptive field (in the image) associated with this grid position. Let this transformed coordinates be C_*^{im} and the coordinates of the corresponding keypoint (in the second image) be C_{gt}^{im}. The matching error for the k^{th} keypoint (E_k) is defined as:

$$E_k = \frac{\|C_*^{im} - C_{gt}^{im}\|_2}{L_D^2} \tag{3}$$

where, L_D^2 is the length of diagonal (in pixels) of the second image. As different images have different sizes, dividing by L_D^2 normalizes for the difference in sizes. The matching error for a pair of images of instances belonging to the same class is calculated as:

$$E_{instance} = \frac{\sum_{k=1}^{K} E_k}{K} \tag{4}$$

The average matching error across all pairs of the instance of the same class is given by E_{class}:

$$E_{class} = \frac{\sum_{instance} E_{instance}}{\#pairs} \tag{5}$$

where, $\#pairs$ is the number of pairs of object instances belonging to the same class. In Figure 4 of the main paper we report the matching error averaged across all the 20 classes.

A.2. Keypoint Matching using SIFT

SIFT features are extracted using a square window of size 72 pixels and a stride of 8 pixels using the open source code from [35]. The stride of 8 pixels was chosen to have a fair comparison with the CNN features. The CNN features were computed with a stride of 8 for layer conv-2 and stride of 16 for layers conv-3, conv-4 and conv-5 respectively. The matching error using SIFT was calculated in the same way as for the CNNs.

A.3. Effect of Viewpoint on Keypoint Matching

Intuitively, matching instances of the same object that are related by a large transformation (i.e. viewpoint distance) should be harder than matching instances with a small viewpoint distance. Therefore, in order to obtain a holistic understanding of the accuracy of features in performing keypoint matching it is instructive to study the accuracy of matching as a function of viewpoint distance.

[36] aligned instances of the same class (from PASCAL-VOC-2012) in a global coordinate system and provide a rotation matrix (R) for each instance in the class. To measure the viewpoint distance, we computed the riemannian metric on the manifold of rotation matrices $||log(R_i R_j^T)||_F$, where log is the matrix logarithm, $||.||_F$ is the Frobenius norm of the matrix and R_i, R_j are the rotation matrices for the i^{th}, j^{th} instances respectively. We binned the distances into 10 uniform bins (of 18°each). In Figure 4 of the main paper we show the mean error in keypoint matching in each of these viewpoints bin. The matching error in the k^{th} bin is calculated by considering all the instances with a viewpoint distance $\leq k \times 18°$, for $k \in [1, 10]$. As expected we find that keypoint matching is worse for larger viewpoint distances.

A.4. Matching Error for layers 2 and 5

References

[1] P. Agrawal, R. Girshick, and J. Malik. Analyzing the performance of multilayer neural networks for object recognition. In *Computer Vision–ECCV 2014*, pages 329–344. Springer, 2014.

[2] R. Bajcsy. Active perception. *Proceedings of the IEEE*, 76(8):966–1005, 1988.

[3] H. Barlow. Unsupervised learning. *Neural computation*, 1(3):295–311, 1989.

[4] Y. Bengio, A. C. Courville, and P. Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR, abs/1206.5538*, 1, 2012.

[5] L. Bourdev, S. Maji, T. Brox, and J. Malik. Detecting people using mutually consistent poselet activations. In *Computer Vision–ECCV 2010*, pages 168–181. Springer, 2010.

[6] H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4-5):291–294, 1988.

Figure 5: Example matchings between pairs of objects (randomly chosen) with viewpoints within 60 degrees of each other, for classes "aeroplane", "bottle", "dog", "person" and "tvmonitor" from PASCAL VOC. The matchings have been shown for features from layer conv-4 of AlexNet-20K, AlexNet-100K, AlexNet-1M, KittiNet-20K and SIFT. The left image shows the ground truth keypoints that were matched with the keypoints in the right image. Right images shows the location of the ground truth keypoint (shown by solid dot) and lines joining the predicted keypoint location (tip of the line) with the ground keypoint location. Please see section A for details of keypoint matching procedure and figure 4 in the main paper for numerical results. This figure is best seen in color and with zoom.

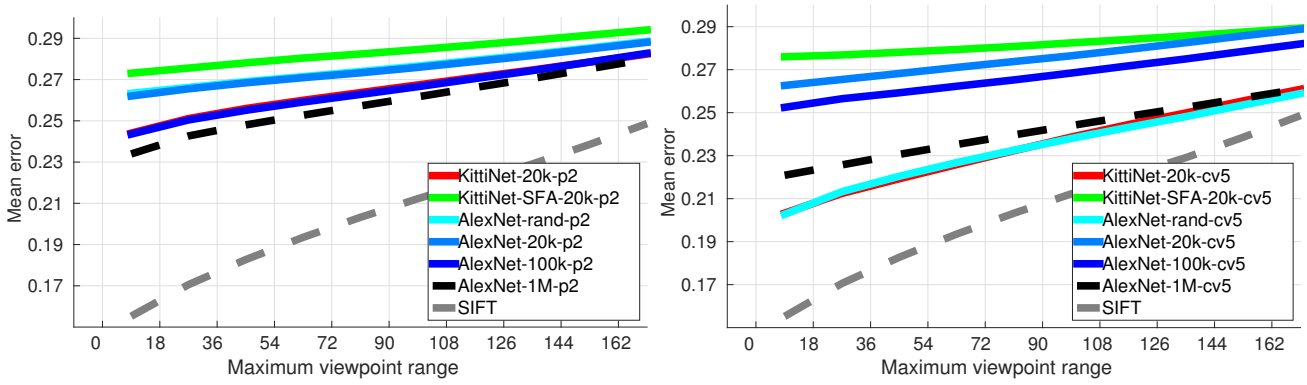


Figure 6: Intra-class keypoint matching error as a function of viewpoint distance averaged over 20 PASCAL objects using features extracted from layers pool-2 (left) and conv-5 (right) of various networks used in this work. Please see section 5.3 for more details.

[7] D. M. Chen, G. Baatz, K. Koser, S. S. Tsai, R. Vedantham, T. Pylvanainen, K. Roimela, X. Chen, J. Bach, M. Pollefeys, et al. City-scale landmark identification on mobile devices. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 737–744, 2011.

[8] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 539–546. IEEE, 2005.

[9] J. E. Cutting. *Perception with an eye for motion*, volume 177.

[10] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *arXiv preprint arXiv:1310.1531*, 2013.

[11] P. Fischer, A. Dosovitskiy, and T. Brox. Descriptor matching with convolutional neural networks: a comparison to sift. *arXiv preprint arXiv:1405.5769*, 2014.

[12] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[13] J. J. Gibson. *The Ecological Approach to Visual Perception*. Houghton Mifflin, 1979.

[14] R. Goroshin, J. Bruna, J. Tompson, D. Eigen, and Y. LeCun. Unsupervised feature learning from temporal data. *arXiv preprint arXiv:1504.02518*, 2015.

[15] R. Hadsell, P. Sermanet, J. Ben, A. Erkan, J. Han, B. Flepp, U. Muller, and Y. LeCun. Online learning for offroad robots: Using spatial label propagation to learn long-range traversability. In *Proc. of Robotics: Science and Systems (RSS)*, volume 11, page 32, 2007.

[16] G. E. Hinton, A. Krizhevsky, and S. D. Wang. Transforming auto-encoders. In *Artificial Neural Networks and Machine Learning–ICANN*, pages 44–51. Springer, 2011.

[17] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[19] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2169–2178. IEEE, 2006.

[20] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[22] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 609–616. ACM, 2009.

[23] D. G. Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[24] R. Memisevic and G. E. Hinton. Learning to represent spatial transformations with factored higher-order boltzmann machines. *Neural Computation*, 22(6):1473–1492, 2010.

[25] H. Mobahi, R. Collobert, and J. Weston. Deep learning from temporal coherence in video. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 737–744. ACM, 2009.

[26] A. Oliva and A. Torralba. Building the gist of a scene: The role of global image features in recognition. *Progress in brain research*, 155:23–36, 2006.

[27] B. A. Olshausen et al. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[28] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. Technical report, DTIC Document, 1989.

[29] M. Ranzato, F. J. Huang, Y.-L. Boureau, and Y. LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[30] M. Ranzato, A. Szlam, J. Bruna, M. Mathieu, R. Collobert, and S. Chopra. Video (language) modeling: a baseline for generative models of natural videos. *arXiv preprint arXiv:1412.6604*, 2014.

[31] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 2015.

[32] R. Salakhutdinov and G. E. Hinton. Deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, pages 448–455, 2009.

[33] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems*, pages 568–576, 2014.

[34] S. Soatto. Visual scene representations: Sufficiency, minimality, invariance and approximations. *arXiv preprint arXiv:1411.7676*, 2014.

[35] A. Vedaldi and B. Fulkerson. Vlfeat: An open and portable library of computer vision algorithms. In *Proceedings of the international conference on Multimedia*, pages 1469–1472. ACM, 2010.

[36] S. Vicente, J. Carreira, L. Agapito, and J. Batista. Reconstructing pascal voc. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 41–48. IEEE, 2014.

[37] L. Wiskott and T. J. Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural computation*, 14(4):715–770, 2002.

[38] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 3485–3492. IEEE, 2010.

Unsupervised Visual Representation Learning by Context Prediction

Carl Doersch[1,2] Abhinav Gupta[1] Alexei A. Efros[2]

[1] School of Computer Science
Carnegie Mellon University

[2] Dept. of Electrical Engineering and Computer Science
University of California, Berkeley

Abstract

This work explores the use of spatial context as a source of free and plentiful supervisory signal for training a rich visual representation. Given only a large, unlabeled image collection, we extract random pairs of patches from each image and train a convolutional neural net to predict the position of the second patch relative to the first. We argue that doing well on this task requires the model to learn to recognize objects and their parts. We demonstrate that the feature representation learned using this within-image context indeed captures visual similarity across images. For example, this representation allows us to perform unsupervised visual discovery of objects like cats, people, and even birds from the Pascal VOC 2011 detection dataset. Furthermore, we show that the learned ConvNet can be used in the R-CNN framework [19] and provides a significant boost over a randomly-initialized ConvNet, resulting in state-of-the-art performance among algorithms which use only Pascal-provided training set annotations.

1. Introduction

Recently, new computer vision methods have leveraged large datasets of millions of labeled examples to learn rich, high-performance visual representations [29]. Yet efforts to scale these methods to truly Internet-scale datasets (i.e. hundreds of **b**illions of images) are hampered by the sheer expense of the human annotation required. A natural way to address this difficulty would be to employ unsupervised learning, which aims to use data without any annotation. Unfortunately, despite several decades of sustained effort, unsupervised methods have not yet been shown to extract useful information from large collections of full-sized, real images. After all, without labels, it is not even clear *what* should be represented. How can one write an objective function to encourage a representation to capture, for example, objects, if none of the objects are labeled?

Interestingly, in the text domain, *context* has proven to be a powerful source of automatic supervisory signal for learning representations [3, 38, 9, 37]. Given a large text corpus, the idea is to train a model that maps each word to a feature vector, such that it is easy to predict the words

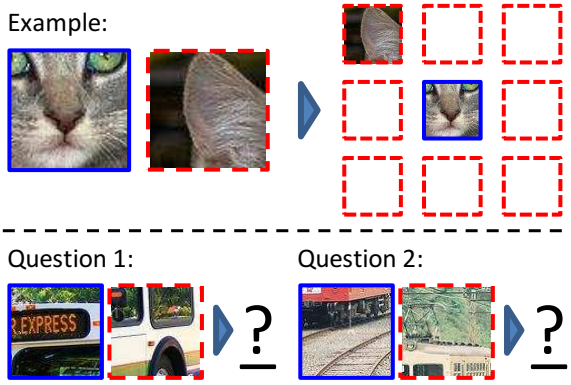


Figure 1. Our task for learning patch representations involves randomly sampling a patch (blue) and then one of eight possible neighbors (red). Can you guess the spatial configuration for the two pairs of patches? Note that the task is much easier once you have recognized the object!

Answer key: Q1: Bottom right Q2: Top center

in the context (i.e., a few words before and/or after) given the vector. This converts an apparently unsupervised problem (finding a good similarity metric between words) into a "self-supervised" one: learning a function from a given word to the words surrounding it. Here the context prediction task is just a "pretext" to force the model to learn a good word embedding, which, in turn, has been shown to be useful in a number of real tasks, such as semantic word similarity [37].

Our paper aims to provide a similar "self-supervised" formulation for image data: a supervised task involving predicting the context for a patch. Our task is illustrated in Figures 1 and 2. We sample random pairs of patches in one of eight spatial configurations, and present each pair to a machine learner, providing no information about the patches' original position within the image. The algorithm must then guess the position of one patch relative to the other. Our underlying hypothesis is that doing well on this task requires understanding scenes and objects, *i.e.* a good visual representation for this task will need to extract objects and their parts in order to reason about their relative spatial location. "Objects," after all, consist of multiple parts that can be detected independently of one another, and which

occur in a specific spatial configuration (if there is no specific configuration of the parts, then it is "stuff" [1]). We present a ConvNet-based approach to learn a visual representation from this task. We demonstrate that the resulting visual representation is good for both object detection, providing a significant boost on PASCAL VOC 2007 compared to learning from scratch, as well as for unsupervised object discovery / visual data mining. This means, surprisingly, that our representation generalizes *across* images, despite being trained using an objective function that operates on a single image at a time. That is, instance-level supervision appears to improve performance on category-level tasks.

2. Related Work

One way to think of a good image representation is as the latent variables of an appropriate generative model. An ideal generative model of natural images would both generate images according to their natural distribution, and be concise in the sense that it would seek common causes for different images and share information between them. However, inferring the latent structure given an image is intractable for even relatively simple models. To deal with these computational issues, a number of works, such as the wake-sleep algorithm [23], contrastive divergence [22], deep Boltzmann machines [45], and variational Bayesian methods [28, 43] use sampling to perform approximate inference. Generative models have shown promising performance on smaller datasets such as handwritten digits [23, 22, 45, 28, 43], but none have proven effective for high-resolution natural images.

Unsupervised representation learning can also be formulated as learning an embedding (i.e. a feature vector for each image) where images that are semantically similar are close, while semantically different ones are far apart. One way to build such a representation is to create a supervised "pretext" task such that an embedding which solves the task will also be useful for other real-world tasks. For example, denoising autoencoders [53, 4] use reconstruction from noisy data as a pretext task: the algorithm must connect images to other images with similar objects to tell the difference between noise and signal. Sparse autoencoders also use reconstruction as a pretext task, along with a sparsity penalty [39], and such autoencoders may be stacked to form a deep representation [32, 31]. (however, only [31] was successfully applied to full-sized images, requiring a million CPU hours to discover just three objects). We believe that current reconstruction-based algorithms struggle with low-level phenomena, like stochastic textures, making it hard to even measure whether a model is generating well.

Another pretext task is "context prediction." A strong tradition for this kind of task already exists in the text domain, where "skip-gram" [37] models have been shown to generate useful word representations. The idea is to train a

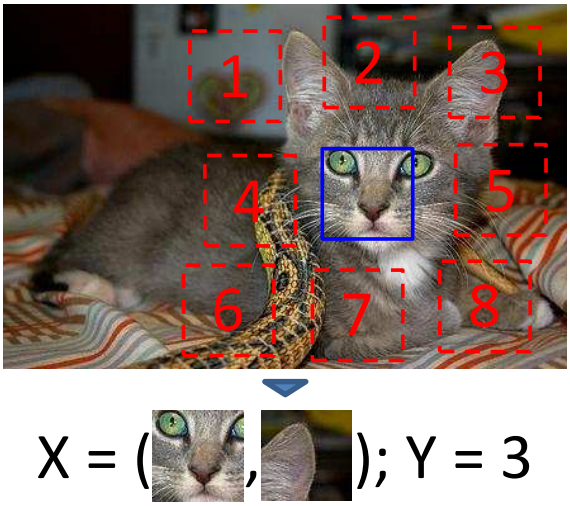


Figure 2. The algorithm receives two patches in one of these eight possible spatial arrangements, without any context, and must then classify which configuration was sampled.

model (e.g. a deep network) to predict, from a single word, the n preceding and n succeeding words. In principle, similar reasoning could be applied in the image domain, a kind of visual "fill in the blank" task, but, again, one runs into the problem of determining whether the predictions themselves are correct [12], unless one cares about predicting only very low-level features [14, 30, 50]. To address this, [36] predicts the appearance of an image region by consensus voting of the transitive nearest neighbors of its surrounding regions. Our previous work [12] explicitly formulates a statistical test to determine whether the data is better explained by a prediction or by a low-level null hypothesis model.

The key problem that these approaches must address is that predicting pixels is much harder than predicting words, due to the huge variety of pixels that can arise from the same semantic object. In the text domain, one interesting idea is to switch from a pure prediction task to a discrimination task [38, 9]. In this case, the pretext task is to discriminate true snippets of text from the same snippets where a word has been replaced at random. A direct extension of this to 2D might be to discriminate between real images vs. images where one patch has been replaced by a random patch from elsewhere in the dataset. However, such a task would be trivial, since discriminating low-level color statistics and lighting would be enough. To make the task harder and more high-level, in this paper, we instead classify between multiple possible configurations of patches sampled from *the same image*, which means they will share lighting and color statistics, as shown on Figure 2.

Another line of work in unsupervised learning from images aims to discover object categories using hand-crafted features and various forms of clustering (e.g. [48, 44] learned a generative model over bags of visual words). Such representations lose shape information, and will readily dis-

cover clusters of, say, foliage. A few subsequent works have attempted to use representations more closely tied to shape [33, 40], but relied on contour extraction, which is difficult in complex images. Many other approaches [20, 27, 16] focus on defining similarity metrics which can be used in more standard clustering algorithms; [42], for instance, re-casts the problem as frequent itemset mining. Geometry may also be used to for verifying links between images [41, 6, 21], although this can fail for deformable objects.

Video can provide another cue for representation learning. For most scenes, the identity of objects remains unchanged even as appearance changes with time. This kind of temporal coherence has a long history in visual learning literature [18, 56], and contemporaneous work shows strong improvements on modern detection datasets [54].

Finally, our work is related to a line of research on discriminative patch mining [13, 47, 26, 34, 49, 11], which has emphasized weak supervision as a means of object discovery. Like the current work, they emphasize the utility of learning representations of patches (i.e. object parts) before learning full objects and scenes, and argue that scene-level labels can serve as a pretext task. For example, [13] trains detectors to be sensitive to different geographic locales, but the actual goal is to discover specific elements of architectural style.

3. Learning Visual Context Prediction

We aim to learn an image representation for our pretext task, i.e., predicting the relative position of patches within an image. We employ Convolutional Neural Networks (ConvNets), which are well known to learn complex image representations with minimal human feature design. Building a ConvNet that can predict a relative offset for a pair of patches is, in principle, straightforward: the network must feed the two input patches through several convolution layers, and produce an output that assigns a probability to each of the eight spatial configurations (Figure 2) that might have been sampled (i.e. a softmax output). Note, however, that we ultimately wish to learn a feature embedding for *individual* patches, such that patches which are visually similar (across different images) would be close in the embedding space.

To achieve this, we use a late-fusion architecture shown in Figure 3: a pair of AlexNet-style architectures [29] that process each patch separately, until a depth analogous to fc6 in AlexNet, after which point the representations are fused. For the layers that process only one of the patches, weights are tied between both sides of the network, such that the same fc6-level embedding function is computed for both patches. Because there is limited capacity for joint reasoning—i.e., only two layers receive input from both patches—we expect the network to perform the bulk of the

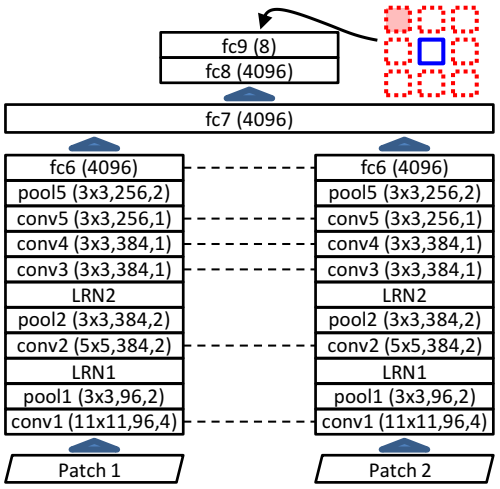


Figure 3. Our architecture for pair classification. Dotted lines indicate shared weights. 'conv' stands for a convolution layer, 'fc' stands for a fully-connected one, 'pool' is a max-pooling layer, and 'LRN' is a local response normalization layer. Numbers in parentheses are kernel size, number of outputs, and stride (fc layers have only a number of outputs). The LRN parameters follow [29]. All conv and fc layers are followed by ReLU nonlinearities, except fc9 which feeds into a softmax classifier.

semantic reasoning for each patch separately. When designing the network, we followed AlexNet where possible.

To obtain training examples given an image, we sample the first patch uniformly, without any reference to image content. Given the position of the first patch, we sample the second patch randomly from the eight possible neighboring locations as in Figure 2.

3.1. Avoiding "trivial" solutions

When designing a pretext task, care must be taken to ensure that the task forces the network to extract the desired information (high-level semantics, in our case), without taking "trivial" shortcuts. In our case, low-level cues like boundary patterns or textures continuing between patches could potentially serve as such a shortcut. Hence, for the relative prediction task, it was important to include a gap between patches (in our case, approximately half the patch width). Even with the gap, it is possible that long lines spanning neighboring patches could could give away the correct answer. Therefore, we also randomly jitter each patch location by up to 7 pixels (see Figure 2).

However, even these precautions are not enough: we were surprised to find that, for some images, another trivial solution exists. We traced the problem to an unexpected culprit: chromatic aberration. Chromatic aberration arises from differences in the way the lens focuses light at different wavelengths. In some cameras, one color channel (commonly green) is shrunk toward the image center relative to the others [5, p. 76]. A ConvNet, it turns out, can learn to localize a patch relative to the lens itself (see Section 4.2) sim-

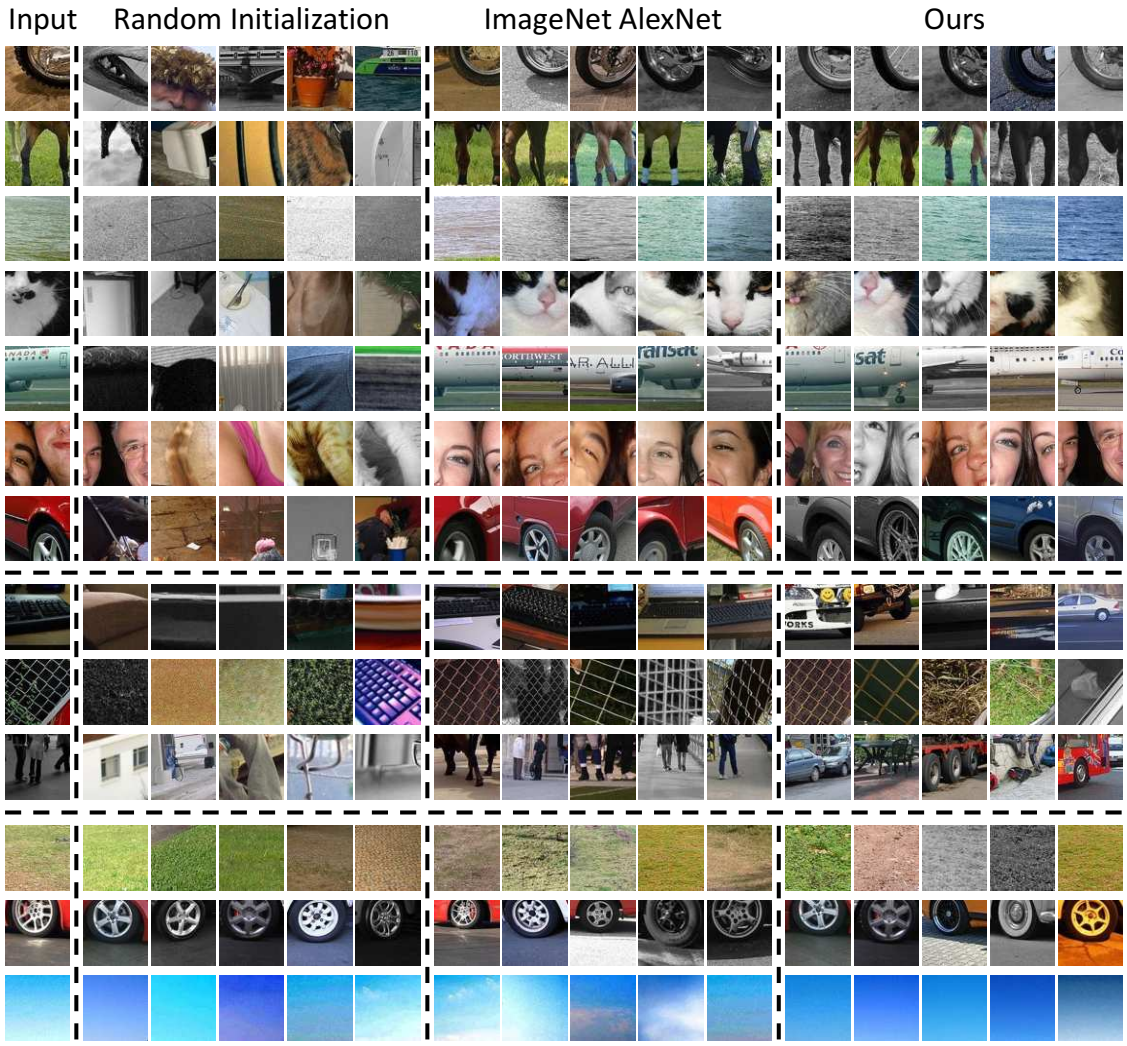


Figure 4. Examples of patch clusters obtained by nearest neighbors. The query patch is shown on the far left. Matches are for three different features: fc6 features from a random initialization of our architecture, AlexNet fc7 after training on labeled ImageNet, and the fc6 features learned from our method. Queries were chosen from 1000 randomly-sampled patches. The top group is examples where our algorithm performs well; for the middle AlexNet outperforms our approach; and for the bottom all three features work well.

ply by detecting the separation between green and magenta (red + blue). Once the network learns the absolute location on the lens, solving the relative location task becomes trivial. To deal with this problem, we experimented with two types of pre-processing. One is to shift green and magenta toward gray ('projection'). Specifically, let $a = [-1, 2, -1]$ (the 'green-magenta color axis' in RGB space). We then define $B = I - a^T a/(aa^T)$, which is a matrix that subtracts the projection of a color onto the green-magenta color axis. We multiply every pixel value by B. An alternative approach is to randomly drop 2 of the 3 color channels from each patch ('color dropping'), replacing the dropped colors with Gaussian noise (standard deviation $\sim 1/100$ the standard deviation of the remaining channel). For qualitative results, we show the 'color-dropping' approach, but found both performed similarly; for the object detection results, we show both results.

Implementation Details: We use Caffe [25], and train on the ImageNet [10] 2012 training set (1.3M images), using only the images and discarding the labels. First, we resize each image to between 150K and 450K total pixels, preserving the aspect-ratio. From these images, we sample patches at resolution 96-by-96. For computational efficiency, we only sample the patches from a grid like pattern, such that each sampled patch can participate in as many as 8 separate pairings. We allow a gap of 48 pixels between the sampled patches in the grid, but also jitter the location of each patch in the grid by -7 to 7 pixels in each direction. We preprocess patches by (1) mean subtraction (2) projecting or dropping colors (see above), and (3) randomly downsampling some patches to as little as 100 total pixels, and then upsampling it, to build robustness to pixelation. When ap-

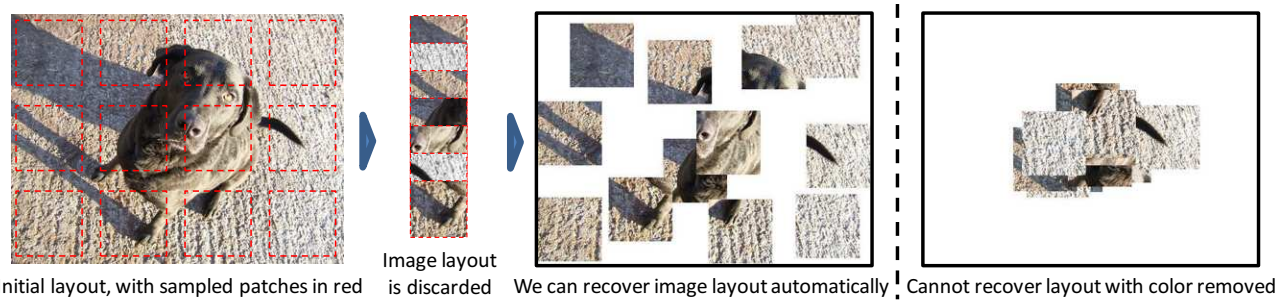


Figure 5. We trained a network to predict the absolute (x, y) coordinates of randomly sampled patches. Far left: input image. Center left: extracted patches. Center right: the location the trained network predicts for each patch shown on the left. Far right: the same result after our color projection scheme. Note that the far right patches are shown *after* color projection; the operation's effect is almost unnoticeable.

plying simple SGD to train the network, we found that the network predictions would degenerate to a uniform prediction over the 8 categories, with all activations for fc6 and fc7 collapsing to 0. This meant that the optimization became permanently stuck in a saddle point where it ignored the input from the lower layers (which helped minimize the variance of the final output), and therefore that the net could not tune the lower-level features and escape the saddle point. Hence, Our final implementation employs batch normalization [24], which forces the network activations to vary across examples. We also find that high momentum values (e.g. .999) accelerated learning. For experiments, we use a ConvNet trained on a K40 GPU for approximately four weeks.

4. Experiments

We first demonstrate the network has learned to associate semantically similar patches, using simple nearest-neighbor matching. We then apply the trained network in two domains. First, we use the model as "pre-training" for a standard vision task with only limited training data: specifically, we use the VOC 2007 object detection. Second, we evaluate visual data mining, where the goal is to start with an unlabeled image collection and discover object classes. Finally, we analyze the performance on the layout prediction "pretext task" to see how much is left to learn from this supervisory signal.

4.1. Nearest Neighbors

Recall our intuition that training should assign similar representations to semantically similar patches. In this section, our goal is to understand which patches our network considers similar. We begin by sampling random 96x96 patches, which we represent using fc6 features (i.e. we remove fc7 and higher shown in Figure 3, and use only one of the two stacks). We find nearest neighbors using normalized correlation of these features. Results for some patches (selected out of 1000 random queries) are shown in Figure 4. For comparison, we repeated the experiment using fc7 features from AlexNet trained on ImageNet (obtained by upsampling the patches), and using fc6 features from our architecture but without any training (random weights initialization). As shown in Figure 4, the matches returned by our feature often capture the semantic information that we are after, matching AlexNet in terms of semantic content (in some cases, e.g. the car wheel, our matches capture pose better). Interestingly, in a few cases, random (untrained) ConvNet also does reasonably well.

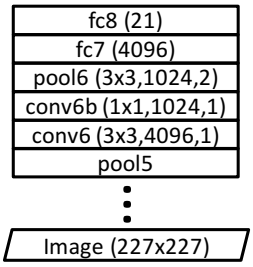


Figure 6. Our architecture for Pascal VOC detection. Layers from conv1 through pool5 are copied from our patch-based network (Figure 3). The new 'conv6' layer is created by converting the fc6 layer into a convolution layer. Kernel sizes, output units, and stride are given in parentheses, as in Figure 3.

4.2. Aside: Learnability of Chromatic Aberration

We noticed in early nearest-neighbor experiments that some patches retrieved match patches from the same absolute location in the image, regardless of content, because those patches displayed similar aberration. To further demonstrate this phenomenon, we trained a network to predict the absolute (x, y) coordinates of patches sampled from ImageNet. While the overall accuracy of this regressor is not very high, it does surprisingly well for some images: for the top 10% of images, the average (root-mean-square) error is .255, while chance performance (always predicting the image center) yields a RMSE of .371. Figure 5 shows one such result. Applying the proposed "projection" scheme increases the error on the top 10% of images to .321.

4.3. Object Detection

Previous work on the Pascal VOC challenge [15] has shown that pre-training on ImageNet (i.e., training a ConvNet to solve the ImageNet challenge) and then "fine-tuning" the network (i.e. re-training the ImageNet model for PASCAL data) provides a substantial boost over training on the Pascal training set alone [19, 2]. However, as far as we are aware, no works have shown that *unsupervised* pre-training on images can provide such a performance boost, no matter how much data is used.

Since we are already using a ConvNet, we adopt the cur-

VOC-2007 Test	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
DPM-v5[17]	33.2	60.3	10.2	16.1	**27.3**	54.3	58.2	23.0	20.0	24.1	26.7	12.7	58.1	48.2	43.2	12.0	21.1	36.1	46.0	43.5	33.7
[8] w/o context	52.6	52.6	19.2	25.4	18.7	47.3	56.9	42.1	16.6	41.4	41.9	27.7	47.9	51.5	29.9	20.0	41.1	36.4	48.6	53.2	38.5
Regionlets[55]	54.2	52.0	20.3	24.0	20.1	55.5	68.7	42.6	19.2	44.2	**49.1**	26.6	57.0	54.5	43.4	16.4	36.6	37.7	**59.4**	52.3	41.7
Scratch-R-CNN[2]	49.9	60.6	24.7	23.7	20.3	52.5	64.8	32.9	20.4	43.5	34.2	29.9	49.0	60.4	47.5	28.0	42.3	28.6	51.2	50.0	40.7
Scratch-Ours	52.6	60.5	23.8	24.3	18.1	50.6	65.9	29.2	19.5	43.5	35.2	27.6	46.5	59.4	46.5	25.6	42.4	23.5	50.0	50.6	39.8
Ours-projection	58.4	62.8	**33.5**	27.7	24.4	**58.5**	68.5	41.2	**26.3**	**49.5**	42.6	**37.3**	**55.7**	**62.5**	49.4	**29.0**	**47.5**	28.4	54.7	56.8	45.7
Ours-color-dropping	**60.5**	**66.5**	29.6	**28.5**	26.3	56.1	**70.4**	**44.8**	24.6	45.5	45.4	35.1	52.2	60.2	**50.0**	28.1	46.7	**42.6**	54.8	**58.6**	**46.3**
Ours-Yahoo100m	56.2	63.9	29.8	27.8	23.9	57.4	69.8	35.6	23.7	47.4	43.0	29.5	52.9	62.0	48.7	28.4	45.1	33.6	49.0	55.5	44.2
Ours-VGG	63.6	64.4	42.0	42.9	18.9	67.9	69.5	65.9	28.2	48.1	58.4	58.5	66.2	64.9	54.1	26.1	43.9	55.9	69.8	50.9	53.0
ImageNet-R-CNN[19]	64.2	69.7	50	41.9	32.0	62.6	71.0	60.7	32.7	58.5	46.5	56.1	60.6	66.8	54.2	31.5	52.8	48.9	57.9	64.7	54.2

Table 1. Results on VOC-2007. R-CNN performance with our unsupervised pre-training is 5% MAP better than training from scratch, but still 8% below pre-training with ImageNet label supervision.

rent state-of-the-art R-CNN pipeline [19]. R-CNN works on object proposals that have been resized to 227x227. Our algorithm, however, is aimed at 96x96 patches. We find that downsampling the proposals to 96x96 loses too much detail. Instead, we adopt the architecture shown in Figure 6. As above, we use only one stack from Figure 3. Second, we resize the convolution layers to operate on inputs of 227x227. This results in a pool5 that is 7x7 spatially, so we must convert the previous fc6 layer into a convolution layer (which we call conv6) following [35]. Note our conv6 layer has 4096 channels, where each unit connects to a 3x3 region of pool5. A conv layer with 4096 channels would be quite expensive to connect directly to a 4096-dimensional fully-connected layer. Hence, we add another layer after conv6 (called conv6b), using a 1x1 kernel, which reduces the dimensionality to 1024 channels (and adds a nonlinearity). Finally, we feed the outputs through a pooling layer to a fully connected layer (fc7) which in turn connects to a final fc8 layer which feeds into the softmax. We fine-tune this network according to the procedure described in [19] (conv6b, fc7, and fc8 start with random weights), and use fc7 as the final representation. We do not use bounding-box regression, and take the appropriate results from [19] and [2].

Table 1 shows our results. Our architecture trained from scratch (random initialization) performs slightly worse than AlexNet trained from scratch. However, our pre-training makes up for this, boosting the from-scratch number by 6% MAP, and outperforms an AlexNet-style model trained from scratch on Pascal by over 5%. This puts us about 8% behind the performance of R-CNN pre-trained with ImageNet labels [19]. This is the best result we are aware of on VOC 2007 without using labels outside the dataset. We ran additional baselines initialized with batch normalization, but found they performed worse than the ones shown.

To understand the effect of various dataset biases [52], we also performed a preliminary experiment pre-training on a randomly-selected 2M subset of the Yahoo/Flickr 100-million Dataset [51], which was collected entirely automatically. The performance after fine-tuning is slightly worse than Imagenet, but there is still a considerable boost over the from-scratch model. We also performed a preliminary experiment with a VGG-style [46] (16-layer) network, shown as "Ours-VGG" in Table 1.

4.4. Visual Data Mining

Visual data mining [41, 13, 47, 42], or unsupervised object discovery [48, 44, 20], aims to use a large image collection to discover image fragments which happen to depict the same semantic objects. Applications include dataset visualization, content-based retrieval, and tasks that require relating visual data to other unstructured information (e.g. GPS coordinates [13]). For automatic data mining, our approach from section 4.1 is inadequate: although object patches match to similar objects, textures match just as readily to similar textures. Suppose, however, that we sampled two non-overlapping patches from the same object. Not only would the nearest neighbor lists for both patches share many images, but within those images, the nearest neighbors would be in roughly the same spatial configuration. For texture regions, on the other hand, the spatial configurations of the neighbors would be random, because the texture has no global layout.

To implement this, we first sample a constellation of four adjacent patches from an image (we use four to reduce the likelihood of a matching spatial arrangement happening by chance). We find the top 100 images which have the strongest matches for all four patches, ignoring spatial layout. We then use a type of geometric verification [7] to filter away the images where the four matches are not geometrically consistent. Because our features are more semantically-tuned, we can use a much weaker type of geometric verification than [7]. Finally, we rank the different constellations by counting the number of times the top 100 matches geometrically verify.

Implementation Details: To compute whether a set of four matched patches geometrically verifies, we first compute the best-fitting square S to the patch centers (via least-squares), while constraining that side of S be between $2/3$ and $4/3$ of the average side of the patches. We then compute the squared error of the patch centers relative to S (normalized by dividing the sum-of-squared-errors by the square of the side of S). The patch is geometrically verified if this normalized squared error is less than 1. When sampling

Figure 7. Object clusters discovered by our algorithm. The number beside each cluster indicates its ranking, determined by the fraction of the top matches that geometrically verified. For all clusters, we show the raw top 7 matches that verified geometrically. The full ranking is available on our project webpage.

patches do not use any of the data augmentation preprocessing steps (e.g. downsampling). We use the color-dropping version of our network.

We applied the described mining algorithm to Pascal VOC 2011, with no pre-filtering of images and no additional labels. We show some of the resulting patch clusters in Figure 7. The results are visually comparable to our previous work [12], although we discover a few objects that were not found in [12], such as monitors, birds, torsos, and plates of food. The discovery of birds and torsos—which are notoriously deformable—provides further evidence for the invariances our algorithm has learned. We believe we have covered all objects discovered in [12], with the exception of (1) trusses and (2) railroad tracks without trains (though we do discover them with trains). For some objects like dogs, we discover more variety and rank the best ones higher. Furthermore, many of the clusters shown in [12] depict gratings (14 out of the top 100), whereas none of ours do (though two of our top hundred depict diffuse gradients). As in [12], we often re-discover the same object multiple times with different viewpoints, which accounts for most of the gaps between ranks in Figure 7. The main disadvantages of our algorithm relative to [12] are 1) some loss of purity, and 2) that we cannot currently determine an object mask automatically (although one could imagine dynamically adding more sub-patches to each proposed object).

To ensure that our algorithm has not simply learned an object-centric representation due to the various biases [52] in ImageNet, we also applied our algorithm to 15,000 Street View images from Paris (following [13]). The results in Figure 8 show that our representation captures scene layout and architectural elements. For this experiment, to rank clusters, we use the de-duplication procedure originally proposed in [13].

4.4.1 Quantitative Results

As part of the qualitative evaluation, we applied our algorithm to the subset of Pascal VOC 2007 selected in [47]: specifically, those containing at least one instance of *bus*,

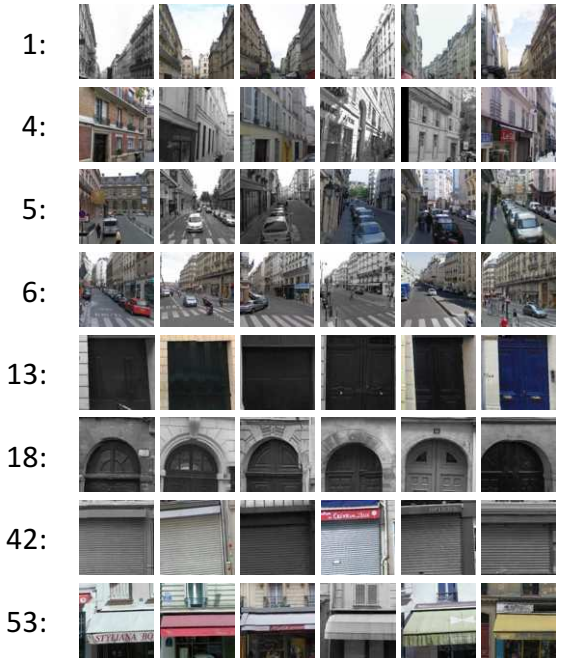


Figure 8. Clusters discovered and automatically ranked via our algorithm (§ 4.4) from the Paris Street View dataset.

dining table, *motorbike*, *horse*, *sofa*, or *train*, and evaluate via a purity coverage curve following [12]. We select 1000 sets of 10 images each for evaluation. The evaluation then sorts the sets by *purity*: the fraction of images in the cluster containing the same category. We generate the curve by walking down the ranking. For each point on the curve, we plot average purity of all sets up to a given point in the ranking against *coverage*: the fraction of images in the dataset that are contained in at least one of the sets up to that point. As shown in Figure 9, we have gained substantially in terms of coverage, suggesting increased invariance for our learned feature. However, we have also lost some highly-pure clusters compared to [12]—which is not very surprising considering that our validation procedure is considerably simpler.

Implementation Details: We initialize 16,384 clusters by sampling patches, mining nearest neighbors, and geometric verification ranking as described above. The resulting clusters are highly redundant. The cluster selection procedure of [12] relies on a likelihood ratio score that is calibrated across clusters, which is not available to us. To select clusters, we first select the top 10 geometrically-verified neighbors for each cluster. Then we iteratively select the highest-ranked cluster that contributes at least one image to our coverage score. When we run out of images that aren't included in the coverage score, we choose clusters to cover each image at least twice, and then three times, and so on.

4.5. Accuracy on the Relative Prediction Task Task

Can we improve the representation by further training on our relative prediction pretext task? To find out, we briefly analyze classification performance on pretext task itself. We sampled 500 random images from Pascal VOC 2007, sampled 256 pairs of patches from each, and classified them into the eight relative-position categories from Figure 2. This gave an accuracy of 38.4%, where chance performance is 12.5%, suggesting that the pretext task is quite hard (indeed, human performance on the task is similar). To measure possible overfitting, we also ran the same experiment on ImageNet, which is the dataset we used for training. The network was 39.5% accurate on the training set, and 40.3% accurate on the validation set (which the network never saw during training), suggesting that little overfitting has occurred.

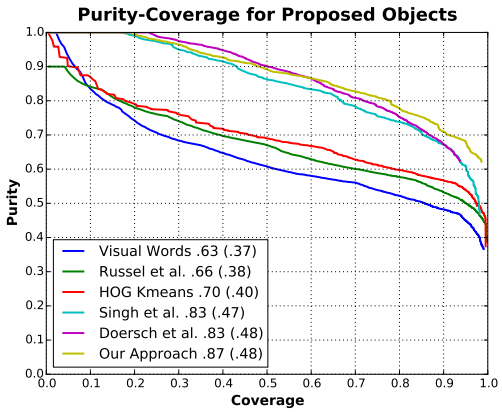


Figure 9. Purity vs coverage for objects discovered on a subset of Pascal VOC 2007. The numbers in the legend indicate area under the curve (AUC). In parentheses is the AUC up to a coverage of .5.

One possible reason why the pretext task is so difficult is because, for a large fraction of patches within each image, the task is almost impossible. Might the task be easiest for image regions corresponding to objects? To test this hypothesis, we repeated our experiment using only patches sampled from within Pascal object ground-truth bounding boxes. We select only those boxes that are at least 240 pixels on each side, and which are not labeled as truncated, occluded, or difficult. Surprisingly, this gave essentially the same accuracy of 39.2%, and a similar experiment only on cars yielded 45.6% accuracy. So, while our algorithm is sensitive to objects, it is almost as sensitive to the layout of the rest of the image.

Acknowledgements We thank Xiaolong Wang and Pulkit Agrawal for help with baselines, Berkeley and CMU vision group members for many fruitful discussions, and Jitendra Malik for putting gelato on the line. This work was partially supported by Google Graduate Fellowship to CD, ONR MURI N000141010934, Intel research grant, an NVidia hardware grant, and an Amazon Web Services grant.

References

[1] E. H. Adelson. On seeing stuff: the perception of materials by humans and machines. In *Photonics West 2001-Electronic Imaging*, 2001. 2

[2] P. Agrawal, R. Girshick, and J. Malik. Analyzing the performance of multilayer neural networks for object recognition. In *ECCV*. 2014.

5, 6

[3] R. K. Ando and T. Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *JMLR*, 2005. 1

[4] Y. Bengio, E. Thibodeau-Laufer, G. Alain, and J. Yosinski. Deep generative stochastic networks trainable by backprop. *ICML*, 2014. 2

[5] D. Brewster and A. D. Bache. *Treatise on optics*. Blanchard and Lea, 1854. 3

[6] O. Chum, M. Perdoch, and J. Matas. Geometric min-hashing: Finding a (thick) needle in a haystack. In *CVPR*, 2009. 3

[7] O. Chum, J. Philbin, J. Sivic, M. Isard, and A. Zisserman. Total recall: Automatic query expansion with a generative feature model for object retrieval. In *ICCV*, 2007. 6

[8] R. G. Cinbis, J. Verbeek, and C. Schmid. Segmentation driven object detection with Fisher vectors. In *ICCV*, 2013. 6

[9] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *ICML*, 2008. 1, 2

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 4

[11] C. Doersch, A. Gupta, and A. A. Efros. Mid-level visual element discovery as discriminative mode seeking. In *NIPS*, 2013. 3

[12] C. Doersch, A. Gupta, and A. A. Efros. Context as supervisory signal: Discovering objects with predictable context. In *ECCV*. 2014. 2, 7, 8

[13] C. Doersch, S. Singh, A. Gupta, J. Sivic, and A. A. Efros. What makes Paris look like Paris? *SIGGRAPH*, 2012. 3, 6, 7

[14] J. Domke, A. Karapurkar, and Y. Aloimonos. Who killed the directed model? In *CVPR*, 2008. 2

[15] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *IJCV*, 2010. 5

[16] A. Faktor and M. Irani. clustering by composition–unsupervised discovery of image categories. In *ECCV*. 2012. 3

[17] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *PAMI*, 2010. 6

[18] P. Földiák. Learning invariance from transformation sequences. *Neural Computation*, 1991. 3

[19] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 1, 5, 6

[20] K. Grauman and T. Darrell. Unsupervised learning of categories from sets of partially matching image features. In *CVPR*, 2006. 3, 6

[21] K. Heath, N. Gelfand, M. Ovsjanikov, M. Aanjaneya, and L. J. Guibas. Image webs: Computing and exploiting connectivity in image collections. In *CVPR*, 2010. 3

[22] G. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 2006. 2

[23] G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal. The "wake-sleep" algorithm for unsupervised neural networks. *Proceedings. IEEE*, 1995. 2

[24] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 5

[25] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM-MM*, 2014. 4

[26] M. Juneja, A. Vedaldi, C. V. Jawahar, and A. Zisserman. Blocks that shout: Distinctive parts for scene classification. In *CVPR*, 2013. 3

[27] G. Kim, C. Faloutsos, and M. Hebert. Unsupervised modeling of object categories using link analysis techniques. In *CVPR*, 2008. 3

[28] D. P. Kingma and M. Welling. Auto-encoding variational bayes. 2014. 2

[29] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1, 3

[30] H. Larochelle and I. Murray. The neural autoregressive distribution estimator. In *AISTATS*, 2011. 2

[31] Q. V. Le. Building high-level features using large scale unsupervised learning. In *ICASSP*, 2013. 2

[32] H. Lee, A. Battle, R. Raina, and A. Y. Ng. Efficient sparse coding algorithms. In *NIPS*, 2006. 2

[33] Y. J. Lee and K. Grauman. Foreground focus: Unsupervised learning from partially matching images. *IJCV*, 2009. 3

[34] Q. Li, J. Wu, and Z. Tu. Harvesting mid-level visual concepts from large-scale internet images. In *CVPR*, 2013. 3

[35] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *arXiv preprint arXiv:1411.4038*, 2014. 6

[36] T. Malisiewicz and A. Efros. Beyond categories: The visual memex model for reasoning about object relationships. In *NIPS*, 2009. 2

[37] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013. 1, 2

[38] D. Okanohara and J. Tsujii. A discriminative language model with pseudo-negative samples. In *ACL*, 2007. 1, 2

[39] B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 1996. 2

[40] N. Payet and S. Todorovic. From a set of shapes to object discovery. In *ECCV*. 2010. 3

[41] T. Quack, B. Leibe, and L. Van Gool. World-scale mining of objects and events from community photo collections. In *CIVR*, 2008. 3, 6

[42] K. Rematas, B. Fernando, F. Dellaert, and T. Tuytelaars. Dataset fingerprints: Exploring image collections through data mining. In *CVPR*, 2015. 3, 6

[43] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *ICML*, 2014. 2

[44] B. C. Russell, W. T. Freeman, A. A. Efros, J. Sivic, and A. Zisserman. Using multiple segmentations to discover objects and their extent in image collections. In *CVPR*, 2006. 2, 6

[45] R. Salakhutdinov and G. E. Hinton. Deep boltzmann machines. In *ICAIS*, 2009. 2

[46] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, 2014. 6

[47] S. Singh, A. Gupta, and A. A. Efros. Unsupervised discovery of mid-level discriminative patches. In *ECCV*, 2012. 3, 6, 7

[48] J. Sivic, B. C. Russell, A. A. Efros, A. Zisserman, and W. T. Freeman. Discovering objects and their location in images. In *ICCV*, 2005. 2, 6

[49] J. Sun and J. Ponce. Learning discriminative part detectors for image classification and cosegmentation. In *ICCV*, 2013. 3

[50] L. Theis and M. Bethge. Generative image modeling using spatial lstms. In *NIPS*, 2015. 2

[51] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li. The new data and new challenges in multimedia research. *arXiv preprint arXiv:1503.01817*, 2015. 6

[52] A. Torralba and A. A. Efros. Unbiased look at dataset bias. In *CVPR*, 2011. 6, 7

[53] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, 2008. 2

[54] X. Wang and A. Gupta. Unsupervised learning of visual representations using videos. In *ICCV*, 2015. 3

[55] X. Wang, M. Yang, S. Zhu, and Y. Lin. Regionlets for generic object detection. In *ICCV*, 2013. 6

[56] L. Wiskott and T. J. Sejnowski. Slow feature analysis:unsupervised learning of invariances. *Neural Computation*, 2002. 3

PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization

Alex Kendall Matthew Grimes Roberto Cipolla
University of Cambridge
agk34, mkg30, rc10001 @cam.ac.uk

King's College *Old Hospital* *Shop Façade* *St Mary's Church*

Figure 1: **PoseNet: Convolutional neural network monocular camera relocalization.** Relocalization results for an input image (top), the predicted camera pose of a visual reconstruction (middle), shown again overlaid in red on the original image (bottom). Our system relocalizes to within approximately $2m$ and 3° for large outdoor scenes spanning $50,000m^2$. For an online demonstration, please see our project webpage: mi.eng.cam.ac.uk/projects/relocalisation/

Abstract

We present a robust and real-time monocular six degree of freedom relocalization system. Our system trains a convolutional neural network to regress the 6-DOF camera pose from a single RGB image in an end-to-end manner with no need of additional engineering or graph optimisation. The algorithm can operate indoors and outdoors in real time, taking 5ms per frame to compute. It obtains approximately 2m and 3°accuracy for large scale outdoor scenes and 0.5m and 5°accuracy indoors. This is achieved using an efficient 23 layer deep convnet, demonstrating that convnets can be used to solve complicated out of image plane regression problems. This was made possible by leveraging transfer learning from large scale classification data. We show that the PoseNet localizes from high level features and is robust to difficult lighting, motion blur and different camera intrinsics where point based SIFT registration fails. Furthermore we show how the pose feature that is produced generalizes to other scenes allowing us to regress pose with only a few dozen training examples.

1. Introduction

Inferring where you are, or localization, is crucial for mobile robotics, navigation and augmented reality. This paper addresses the lost or kidnapped robot problem by introducing a novel relocalization algorithm. Our proposed system, PoseNet, takes a single 224x224 RGB image and regresses the camera's 6-DoF pose relative to a scene. Fig. 1 demonstrates some examples. The algorithm is simple in the fact that it consists of a convolutional neural network (convnet) trained end-to-end to regress the camera's orientation and position. It operates in real time, taking 5ms to run, and obtains approximately 2m and 3 degrees accuracy for large scale outdoor scenes (covering a ground area of up to $50,000m^2$).

Our main contribution is the deep convolutional neural network camera pose regressor. We introduce two novel techniques to achieve this. We leverage transfer learning from recognition to relocalization with very large scale classification datasets. Additionally we use structure from motion to automatically generate training labels (camera poses) from a video of the scene. This reduces the human labor in creating labeled video datasets to just recording the

video.

Our second main contribution is towards understanding the representations that this convnet generates. We show that the system learns to compute feature vectors which are easily mapped to pose, and which also generalize to unseen scenes with a few additional training samples.

Appearance-based relocalization has had success [4, 23] in coarsely locating the camera among a limited, discretized set of place labels, leaving the pose estimation to a separate system. This paper presents a means of computing continuous pose directly from appearance. The scene may include multiple objects and need not be viewed under consistent conditions. For example the scene may include dynamic objects like people and cars or experience changing weather conditions.

Simultaneous localization and mapping (SLAM) is a traditional solution to this problem. We introduce a new framework for localization which removes several issues faced by typical SLAM pipelines, such as the need to store densely spaced keyframes, the need to maintain separate mechanisms for appearance-based localization and landmark-based pose estimation, and a need to establish frame-to-frame feature correspondence. We do this by mapping monocular images to a high-dimensional representation that is robust to nuisance variables. We empirically show that this representation is a smoothly varying injective (one-to-one) function of pose, allowing us to regress pose directly from the image without need of tracking.

Training convolutional networks is usually dependent on very large labeled image datasets, which are costly to assemble. Examples include the *ImageNet* [5] and *Places* [29] datasets, with 14 million and 7 million hand-labeled images, respectively. We employ two techniques to overcome this limitation:

- an automated method of labeling data using structure from motion to generate large regression datasets of camera pose
- transfer learning which trains a pose regressor, pretrained as a classifier, on immense image recognition datasets. This converges to a lower error in less time, even with a very sparse training set, as compared to training from scratch.

2. Related work

There are generally two approaches to localization: metric and appearance-based. Metric SLAM localizes a mobile robot by focusing on creating a sparse [13, 11] or dense [16, 7] map of the environment. Metric SLAM estimates the camera's continuous pose, given a good initial pose estimate. Appearance-based localization provides this coarse estimate by classifying the scene among a limited number of discrete locations. Scalable appearance-based localizers have been proposed such as [4] which uses SIFT features [15] in a bag of words approach to probabilistically recognize previously viewed scenery. Convnets have also been used to classify a scene into one of several location labels [23]. Our approach combines the strengths of these approaches: it does not need an initial pose estimate, and produces a continuous pose. Note we do not build a map, rather we train a neural network, whose size, unlike a map, does not require memory linearly proportional to the size of the scene (see fig. 13).

Our work most closely follows from the Scene Coordinate Regression Forests for relocalization proposed in [20]. This algorithm uses depth images to create scene coordinate labels which map each pixel from camera coordinates to global scene coordinates. This was then used to train a regression forest to regress these labels and localize the camera. However, unlike our approach, this algorithm is limited to RGB-D images to generate the scene coordinate label, in practice constraining its use to indoor scenes.

Previous research such as [27, 14, 9, 3] has also used SIFT-like point based features to match and localize from landmarks. However these methods require a large database of features and efficient retrieval methods. A method which uses these point features is structure from motion (SfM) [28, 1, 22] which we use here as an offline tool to automatically label video frames with camera pose. We use [8] to generate a dense visualisation of our relocalization results.

Despite their ability in classifying spatio-temporal data, convolutional neural networks are only just beginning to be used for regression. They have advanced the state of the art in object detection [24] and human pose regression [25]. However these have limited their regression targets to lie in the 2-D image plane. Here we demonstrate regressing the full 6-DOF camera pose transform including depth and out-of-plane rotation. Furthermore, we show we are able to learn regression as opposed to being a very fine resolution classifier.

It has been shown that convnet representations trained on classification problems generalize well to other tasks [18, 17, 2, 6]. We show that you can apply these representations of classification to 6-DOF regression problems. Using these pre-learned representations allows convnets to be used on smaller datasets without overfitting.

3. Model for deep regression of camera pose

In this section we describe the convolutional neural network (convnet) we train to estimate camera pose directly from a monocular image, I. Our network outputs a pose vector $\mathbf{p}$, given by a 3D camera position $\mathbf{x}$ and orientation represented by quaternion $\mathbf{q}$:

$$\mathbf{p} = [\mathbf{x}, \mathbf{q}] \tag{1}$$

Pose $\mathbf{p}$ is defined relative to an arbitrary global reference frame. We chose quaternions as our orientation representation, because arbitrary 4-D values are easily mapped to legitimate rotations by normalizing them to unit length. This is a simpler process than the orthonormalization required of rotation matrices.

3.1. Simultaneously learning location and orientation

To regress pose, we train the convnet on Euclidean loss using stochastic gradient descent with the following objective loss function:

$$loss(I) = \|\hat{\mathbf{x}} - \mathbf{x}\|_2 + \beta \left\| \hat{\mathbf{q}} - \frac{\mathbf{q}}{\|\mathbf{q}\|} \right\|_2 \quad (2)$$

Where β is a scale factor chosen to keep the expected value of position and orientation errors to be approximately equal.

The set of rotations lives on the unit sphere in quaternion space. However the Euclidean loss function makes no effort to keep $\mathbf{q}$ on the unit sphere. We find, however, that during training, $\mathbf{q}$ becomes close enough to $\hat{\mathbf{q}}$ such that the distinction between spherical distance and Euclidean distance becomes insignificant. For simplicity, and to avoid hampering the optimization with unnecessary constraints, we chose to omit the spherical constraint.

We found that training individual networks to regress position and orientation separately performed poorly compared to when they were trained with full 6-DOF pose labels (fig. 2). With just position, or just orientation information, the convnet was not as effectively able to determine the function representing camera pose. We also experimented with branching the network lower down into two separate components to regress position and orientation. However, we found that it too was less effective, for similar reasons: separating into distinct position and orientation regressors denies each the information necessary to factor out orientation from position, or vice versa.

In our loss function (2) a balance β must be struck between the orientation and translation penalties (fig. 2). They are highly coupled as they are regressed from the same model weights. We observed that the optimal β was given by the ratio between expected error of position and orientation at the end of training, not the beginning. We found β to be greater for outdoor scenes as position errors tended to be relatively greater. Following this intuition we fine tuned β using grid search. For the indoor scenes it was between 120 to 750 and outdoor scenes between 250 to 2000.

We found it was important to randomly initialize the final position regressor layer so that the norm of the weights corresponding to each position dimension was proportional to that dimension's spatial extent.

Classification problems have a training example for every category. This is not possible for regression as the output is continuous and infinite. Furthermore, other convnets that have been used for regression operate off very large datasets [25, 19]. For localization regression to work off limited data we leverage the powerful representations learned off these large classification datasets by pretraining the weights on these datasets.

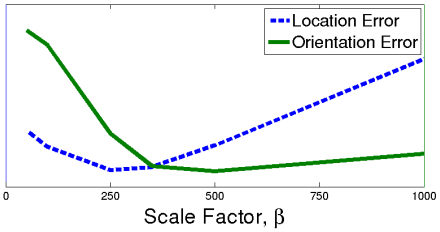


Figure 2: Relative performance of position and orientation regression on **a single convnet with a range of scale factors** for an indoor scene, Chess. This demonstrates that learning with the optimum scale factor leads to the convnet uncovering a more accurate pose function.

3.2. Architecture

For the experiments in this paper we use a state of the art deep neural network architecture for classification, GoogLeNet [24], as a basis for developing our pose regression network. GoogLeNet is a 22 layer convolutional network with six 'inception modules' and two additional intermediate classifiers which are discarded at test time. Our model is a slightly modified version of GoogLeNet with 23 layers (counting only the layers with trainable parameters). We modified GoogLeNet as follows:

- Replace all three softmax classifiers with affine regressors. The softmax layers were removed and each final fully connected layer was modified to output a pose vector of 7-dimensions representing position (3) and orientation (4).
- Insert another fully connected layer before the final regressor of feature size 2048. This was to form a localization feature vector which may then be explored for generalisation.
- At test time we also normalize the quaternion orientation vector to unit length.

We rescaled the input image so that the smallest dimension was 256 pixels before cropping to the 224x224 pixel input to the GoogLeNet convnet. The convnet was trained on random crops (which do not affect the camera pose). At test time we evaluate it with both a single center crop and also densely with 128 uniformly spaced crops of the input image, averaging the resulting pose vectors. With parallel GPU processing, this results in a computational time increase from 5ms to 95ms per image.

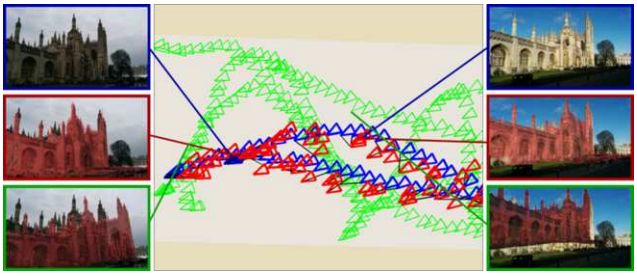


Figure 3: Magnified view of a sequence of **training (green)** and **testing (blue)** cameras for King's College. We show the **predicted camera pose in red** for each testing frame. The images show the test image (top), the predicted view from our convnet overlaid in red on the input image (middle) and the nearest neighbour training image overlaid in red on the input image (bottom). This shows our system can interpolate camera pose effectively in space between training frames.

We experimented with rescaling the original image to different sizes before cropping for training and testing. Scaling up the input is equivalent to cropping the input before downsampling to 256 pixels on one side. This increases the spatial resolution of the input pixels. We found that this does not increase the localization performance, indicating that context and field of view is more important than resolution for relocalization.

The PoseNet model was implemented using the Caffe library [10]. It was trained using stochastic gradient descent with a base learning rate of 10^-5, reduced by 90% every 80 epochs and with momentum of 0.9. Using one half of a dual-GPU card (NVidia Titan Black), training took an hour using a batch size of 75. For reasons of time, we did not explore multi-GPU training, although it is reasonable to expect better results from using double the throughput and memory. We subtracted a separate image mean for each scene as we found this to improve experimental performance.

4. Dataset

Deep learning performs extremely well on large datasets, however producing these datasets is often expensive or very labour intensive. We overcome this by leveraging structure from motion to autonomously generate training labels (camera poses). This reduces the human labour to just recording the video of each scene.

For this paper we release an outdoor urban localization dataset, *Cambridge Landmarks*[1], with 5 scenes. This novel dataset provides data to train and test pose regression algorithms in a large scale outdoor urban setting. A bird's eye view of the camera poses is shown in fig. 4 and further details can be found in table 6. Significant urban clutter such as pedestrians and vehicles were present and data was collected from many different points in time representing different lighting and weather conditions. Train and test images are taken from distinct walking paths and not sampled from the same trajectory making the regression challenging (see fig. 3). We release this dataset for public use and hope to add scenes to this dataset as this project progresses.

The dataset was generated using structure from motion techniques [28] which we use as ground truth measurements for this paper. A Google LG Nexus 5 smartphone was used by a pedestrian to take high definition video around each scene. This video was subsampled in time at 2Hz to generate images to input to the SfM pipeline. There is a spacing of about 1m between each camera position.

To test on indoor scenes we use the publically available *7 Scenes* dataset [20], with scenes shown in fig. 5. This dataset contains significant variation in camera height and was designed for RGB-D relocalization. It is extremely challenging for purely visual relocalization using SIFT-like features, as it contains many ambiguous textureless features.

5. Experiments

We show that PoseNet is able to effectively localize across both the indoor *7 Scenes* dataset and outdoor *Cambridge Landmarks* dataset in table 6. To validate that the convnet is regressing pose beyond that of the training examples we show the performance for finding the nearest neighbour representation in the training data from the feature vector produced by the localization convnet. As our performance exceeds this we conclude that the convnet is successfully able to regress pose beyond training examples (see fig. 3). We also compare our algorithm to the RGB-D SCoRe Forest algorithm [20].

Fig. 7 shows cumulative histograms of localization error for two indoor and two outdoor scenes. We note that although the SCoRe forest is generally more accurate, it requires depth information, and uses higher-resolution imagery. The indoor dataset contains many ambiguous and textureless features which make relocalization without this depth modality extremely difficult. We note our method often localizes the most difficult testing frames, above the 95th percentile, more accurately than SCoRe across all the scenes. We also observe that dense cropping only gives a modest improvement in performance. It is most important in scenes with significant clutter like pedestrians and cars, for example King's College, Shop Façade and St Mary's Church.

We explored the robustness of this method beyond what was tested in the dataset with additional images from dusk, rain, fog, night and with motion blur and different cameras with unknown intrinsics. Fig. 8 shows the convnet gener-

[1]To download the dataset please see our project webpage:
`mi.eng.cam.ac.uk/projects/relocalisation/`

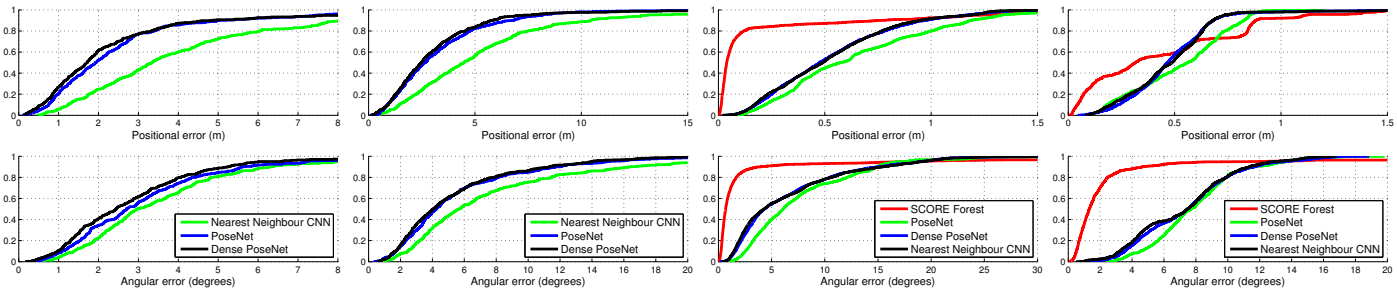
King's College *Street* *Old Hospital* *Shop Façade* *St Mary's Church*

Figure 4: **Map of dataset** showing training frames (green), testing frames (blue) and their predicted camera pose (red). The testing sequences are distinct trajectories from the training sequences and each scene covers a very large spatial extent.

Figure 5: **7 Scenes dataset** example images from left to right; Chess, Fire, Heads, Office, Pumpkin, Red Kitchen and Stairs.

Scene	# Frames Train	# Frames Test	Spatial Extent (m)	SCoRe Forest (Uses RGB-D)	Dist. to Conv. Nearest Neighbour	PoseNet	Dense PoseNet
King's College	1220	343	140 x 40m	N/A	3.34m, 2.96°	1.92m, 2.70°	1.66m, 2.43°
Street	3015	2923	500 x 100m	N/A	1.95m, 4.51°	3.67m, 3.25°	2.96m, 3.00°
Old Hospital	895	182	50 x 40m	N/A	5.38m, 4.51°	2.31m, 2.69°	2.62m, 2.45°
Shop Façade	231	103	35 x 25m	N/A	2.10m, 5.20°	1.46m, 4.04°	1.41m, 3.59°
St Mary's Church	1487	530	80 x 60m	N/A	4.48m, 5.65°	2.65m, 4.24°	2.45m, 3.98°
Chess	4000	2000	3 x 2 x 1m	0.03m, 0.66°	0.41m, 5.60°	0.32m, 4.06°	0.32m, 3.30°
Fire	2000	2000	2.5 x 1 x 1m	0.05m, 1.50°	0.54m, 7.77°	0.47m, 7.33°	0.47m, 7.02°
Heads	1000	1000	2 x 0.5 x 1m	0.06m, 5.50°	0.28m, 7.00°	0.29m, 6.00°	0.30m, 6.09°
Office	6000	4000	2.5 x 2 x 1.5m	0.04m, 0.78°	0.49m, 6.02°	0.48m, 3.84°	0.48m, 3.62°
Pumpkin	4000	2000	2.5 x 2 x 1m	0.04m, 0.68°	0.58m, 6.08°	0.47m, 4.21°	0.49m, 4.06°
Red Kitchen	7000	5000	4 x 3 x 1.5m	0.04m, 0.76°	0.58m, 5.65°	0.59m, 4.32°	0.58m, 4.17°
Stairs	2000	1000	2.5 x 2 x 1.5m	0.32m, 1.32°	0.56m, 7.71°	0.47m, 6.93°	0.48m, 6.54°

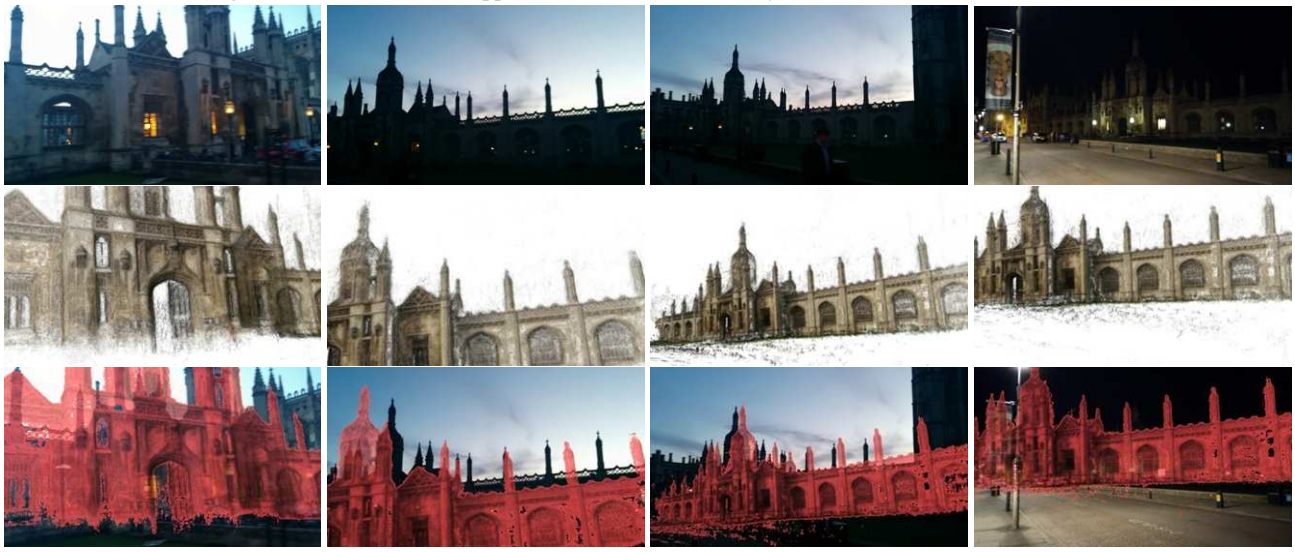
Figure 6: **Dataset details and results.** We show median performance for PoseNet on all scenes, evaluated on a single 224x224 center crop and 128 uniformly separated dense crops. For comparison we plot the results from SCoRe Forest [20] which uses depth, therefore fails on outdoor scenes. This system regresses pixel-wise world coordinates of the input image at much larger resolution. This requires a dense depth map for training and an extra RANSAC step to determine the camera's pose. Additionally, we compare to matching the nearest neighbour feature vector representation from PoseNet. This demonstrates our regression PoseNet performs better than a classifier.



(a) King's College (b) St Mary's Church (c) Pumpkin (d) Stairs

Figure 7: **Localization performance.** These figures show our localization accuracy for both position and orientation as a cumulative histogram of errors for the entire testing set. The regression convnet outperforms the nearest neighbour feature matching which demonstrates we regress finer resolution results than given by training. Comparing to the RGB-D SCoRe Forest approach shows that our method is competitive, but outperformed by a more expensive depth approach. Our method does perform better on the hardest few frames, above the 95th percentile, with our worst error lower than the worst error from the SCoRe approach.

(a) Relocalization with increasing levels of motion blur. The system is able to recognize the pose as high level features such as the contour outline still exist. Blurring the landmark increases apparent contour size and the system believes it is closer.



(b) Relocalization under difficult dusk and night lighting conditions. In the dusk sequences, the landmark is silhouetted against the backdrop however again the convnet seems to recognize the contours and estimate pose.

(c) Relocalization with different weather conditions. PoseNet is able to effectively estimate pose in fog and rain.

(d) Relocalization with significant people, vehicles and other dynamic objects.

(e) Relocalization with unknown camera intrinsics: SLR with focal length 45mm (left), and iPhone 4S with focal length 35mm (right) compared to the dataset's camera which had a focal length of 30mm.

Figure 8: **Robustness to challenging real life situations.** Registration with point based techniques such as SIFT fails in examples (a-c), therefore ground truth measurements are not available. None of these types of challenges were seen during training. As convnets are able to understand objects and contours they are still successful at estimating pose from the building's contour in the silhouetted examples (b) or even under extreme motion blur (a). Many of these quasi invariances were enhanced by pretraining from the scenes dataset.

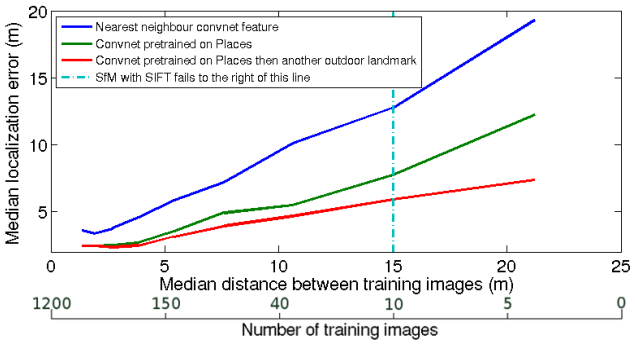


Figure 9: **Robustness to a decreasing training baseline** for the King's College scene. Our system exhibits graceful decline in performance as fewer training samples are used.

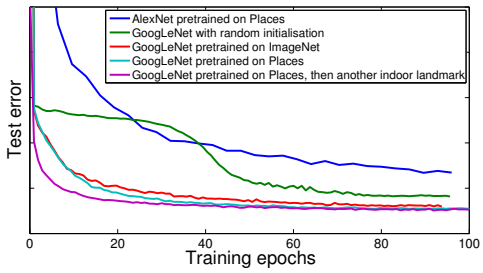


Figure 10: **Importance of transfer learning.** Shows how pretraining on large datasets gives an increase in both performance and training speed.

ally handles these challenges well. SfM with SIFT fails in all these cases so we were not able to generate a ground truth camera pose, however we infer the accuracy by viewing the 3D reconstruction from the predicted camera pose, and overlaying this onto the input image.

5.1. Robustness against training image spacing

We demonstrate in fig. 9 that, for an outdoor scale scene, we gain little by spacing the training images more closely than 4m. The system is robust to very large spatial separation between training images, achieving reasonable performance even with only a few dozen training samples. The pose accuracy deteriorates gracefully with increased training image spacing, whereas SIFT-based SfM sharply fails after a certain threshold as it requires a small baseline [15].

5.2. Importance of transfer learning

In general convnets require large amounts of training data. We sidestep this problem by starting our pose training from a network pretrained on giant datasets such as *ImageNet* and *Places*. Similar to what has been demonstrated for classification tasks, fig. 10 shows how transfer learning can be utilised effectively between classification and complicated regression tasks. Such 'transfer learning' has been demonstrated elsewhere for training classifiers [18, 17, 2], but here we demonstrate transfer learning from classification to the qualitatively different task of pose regression. It is not immediately obvious that a network trained to output pose-invariant classification labels would be suitable as a starting point for a pose regressor. We find, however, that this is not a problem in practice. A possible explanation is that, in order for its output to be invariant to pose, the classifier network must keep track of pose, to better factor its effects away from identity cues. This would agree with our own findings that a network trained to output position and orientation outperforms a network trained to output only position. By preserving orientation information in the intermediate representations, it is better able to factor the effects of orientation out of the final position estimation. Transfer learning gives not only a large improvement in training speed, but also end performance.

The relevance of data is also important. In fig. 10 the *Places* and *ImageNet* curves initially have the same performance. However, ultimately the *Places* pretraining performs better due to being a more relevant dataset to this localization task.

5.3. Visualising features relevant to pose

Fig. 11 shows example saliency maps produced by PoseNet. The saliency map, as used in [21], is the magnitude of the gradient of the loss function with respect to the pixel intensities. This uses the sensitivity of the pose with respect to the pixels as an indicator of how important the convnet considers different parts of the image.

These results show that the strongest response is observed from higher-level features such as windows and spires. However a more surprising result is that PoseNet is also very sensitive to large textureless patches such as road, grass and sky. These textureless patches may be more informative than the highest responding points because the effect of a group of pixels on the pose variable is the sum of the saliency map values over that group of pixels. This evidence points to the net being able to localize off information from these textureless surfaces, something which interest-point based features such as SIFT or SURF fail to do.

The last observation is that PoseNet has an attenuated response to people and other noisy objects, effectively masking them. These objects are dynamic, and the convnet has identified them as not appropriate for localization.

5.4. Viewing the internal representation

t-SNE [26] is an algorithm for embedding high-dimensional data in a low dimensional space, in a way that tries to preserve Euclidean distances. It is often used, as we do here, to visualize high-dimensional feature vectors in

Figure 11: **Saliency maps.** This figure shows the saliency map superimposed on the input image. The saliency maps suggest that the convnet exploits not only distinctive point features (à la SIFT), but also large textureless patches, which can be as informative, if not more so, to the pose. This, combined with a tendency to disregard dynamic objects such as pedestrians, enables it to perform well under challenging circumstances. (Best viewed electronically.)

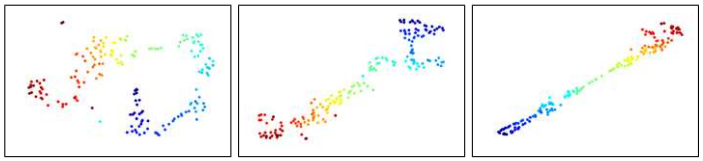


Figure 12: **Feature vector visualisation.** t-SNE visualisation of the feature vectors from a video sequence traversing an outdoor scene (King's College) in a straight line. Colour represents time. The feature representations are generated from the convnet with weights trained on *Places* (a), *Places* then another outdoor scene, St Mary's Church (b), *Places* then this outdoor scene, King's College (c). Despite (a,b) not being trained on this scene, these visualizations suggest that it is possible to compute the pose as a simple, if non-linear, function of these representations.

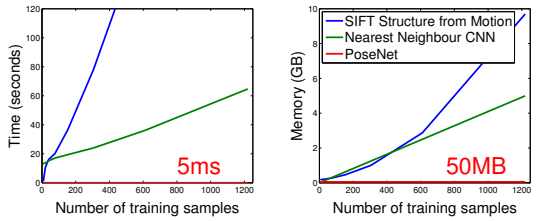


Figure 13: **Implementation efficiency.** Experimental speed and memory use of the convnet regression, nearest neighbour convnet feature vector and SIFT relocalization methods.

two dimensions. In fig. 12 we apply t-SNE to the feature vectors computed from a sequence of video frames taken by a pedestrian. As these figures show, the feature vectors are a function that smoothly varies with, and is largely one-to-one with, pose. This 'pose manifold' can be observed not only on networks trained on other scenes, but also networks trained on classification image sets without pose labels. This further suggests that classification convnets preserve pose information up to the final layer, regardless of whether it's expressed in the output. However, the mapping from feature vector to pose becomes more complicated for networks not trained on pose data. Furthermore, as this manifold exists on scenes that the convnet was not trained on, the convnet must learn some generic representation of the relationship between landmarks, geometry and camera motion. This demonstrates that the feature vector that is produced from regression is able to generalize to other tasks in the same way as classification convnets.

5.5. System efficiency

Fig. 13 compares system performance of PoseNet on a modern desktop computer. Our network is very scalable, as it only takes 50 MB to store the weights, and $5ms$ to compute each pose, compared to the gigabytes and minutes for metric localization with SIFT. These values are independent of the number of training samples in the system while metric localization scales $\mathcal{O}(n^2)$ with training data size [28]. For comparison matching to the convnet nearest neighbour is also shown. This requires storing feature vectors for each training frame, then perform a linear search to find the nearest neighbour for a given test frame.

6. Conclusions

We present, to our knowledge, the first application of deep convolutional neural networks to end-to-end 6-DOF camera pose localization. We have demonstrated that one can sidestep the need for millions of training images by use of transfer learning from networks trained as classifiers. We showed that such networks preserve ample pose information in their feature vectors, despite being trained to produce pose-invariant outputs. Our method tolerates large baselines that cause SIFT-based localizers to fail sharply.

In future work, we aim to pursue further uses of multiview geometry as a source of training data for deep pose regressors, and explore probabilistic extensions to this algorithm [12]. It is obvious that a finite neural network has an upper bound on the physical area that it can learn to localize within. We leave finding this limit to future work.

References

[1] S. Agarwal, Y. Furukawa, N. Snavely, I. Simon, B. Curless, S. M. Seitz, and R. Szeliski. Building rome in a day. *Communications of the ACM*, 54(10):105–112, 2011.

[2] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1798–1828, 2013.

[3] A. Bergamo, S. N. Sinha, and L. Torresani. Leveraging structure from motion to learn discriminative codebooks for scalable landmark classification. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 763–770. IEEE, 2013.

[4] M. Cummins and P. Newman. FAB-MAP: Probabilistic localization and mapping in the space of appearance. *The International Journal of Robotics Research*, 27(6):647–665, 2008.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[6] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *arXiv preprint arXiv:1310.1531*, 2013.

[7] J. Engel, T. Schöps, and D. Cremers. LSD-SLAM: Large-scale direct monocular slam. In *Computer Vision–ECCV 2014*, pages 834–849. Springer, 2014.

[8] Y. Furukawa, B. Curless, S. M. Seitz, and R. Szeliski. Towards internet-scale multi-view stereo. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1434–1441. IEEE, 2010.

[9] Q. Hao, R. Cai, Z. Li, L. Zhang, Y. Pang, and F. Wu. 3d visual phrases for landmark recognition. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3594–3601. IEEE, 2012.

[10] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[11] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert. iSAM2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, page 0278364911430419, 2011.

[12] A. Kendall and R. Cipolla. Modelling uncertainty in deep learning for camera relocalization. *arXiv preprint arXiv:1509.05909*, 2015.

[13] G. Klein and D. Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007.

[14] Y. Li, N. Snavely, D. Huttenlocher, and P. Fua. Worldwide pose estimation using 3d point clouds. In *Computer Vision–ECCV 2012*, pages 15–29. Springer, 2012.

[15] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[16] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. DTAM: Dense tracking and mapping in real-time. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2320–2327. IEEE, 2011.

[17] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1717–1724. IEEE, 2014.

[18] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pages 512–519. IEEE, 2014.

[19] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[20] J. Shotton, B. Glocker, C. Zach, S. Izadi, A. Criminisi, and A. Fitzgibbon. Scene coordinate regression forests for camera relocalization in RGB-D images. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 2930–2937. IEEE, 2013.

[21] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[22] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3d. In *ACM transactions on graphics (TOG)*, volume 25, pages 835–846. ACM, 2006.

[23] N. Sünderhauf, F. Dayoub, S. Shirazi, B. Upcroft, and M. Milford. On the performance of convnet features for place recognition. *arXiv preprint arXiv:1501.04158*, 2015.

[24] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

[25] A. Toshev and C. Szegedy. Deeppose: Human pose estimation via deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1653–1660. IEEE, 2014.

[26] L. Van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

[27] J. Wang, H. Zha, and R. Cipolla. Coarse-to-fine vision-based localization by indexing scale-invariant features. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 36(2):413–422, 2006.

[28] C. Wu. Towards linear-time incremental structure from motion. In *3D Vision-3DV 2013, 2013 International Conference on*, pages 127–134. IEEE, 2013.

[29] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. In *Advances in Neural Information Processing Systems*, pages 487–495, 2014.

Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books

Yukun Zhu [*,1] Ryan Kiros[*,1] Richard Zemel[1] Ruslan Salakhutdinov[1]
Raquel Urtasun[1] Antonio Torralba[2] Sanja Fidler[1]
[1]University of Toronto [2]Massachusetts Institute of Technology
{yukun,rkiros,zemel,rsalakhu,urtasun,fidler}@cs.toronto.edu, torralba@csail.mit.edu

Abstract

Books are a rich source of both fine-grained information, how a character, an object or a scene looks like, as well as high-level semantics, what someone is thinking, feeling and how these states evolve through a story. This paper aims to align books to their movie releases in order to provide rich descriptive explanations for visual content that go semantically far beyond the captions available in current datasets. To align movies and books we exploit a neural sentence embedding that is trained in an unsupervised way from a large corpus of books, as well as a video-text neural embedding for computing similarities between movie clips and sentences in the book. We propose a context-aware CNN to combine information from multiple sources. We demonstrate good quantitative performance for movie/book alignment and show several qualitative examples that showcase the diversity of tasks our model can be used for.

1. Introduction

A truly intelligent machine needs to not only parse the surrounding 3D environment, but also understand why people take certain actions, what they will do next, what they could possibly be thinking, and even try to empathize with them. In this quest, language will play a crucial role in grounding visual information to high-level semantic concepts. Only a few words in a sentence may convey really rich semantic information. Language also represents a natural means of interaction between a naive user and our vision algorithms, which is particularly important for applications such as social robotics or assistive driving.

Combining images or videos with language has gotten significant attention in the past year, partly due to the creation of CoCo [18], Microsoft's large-scale captioned image dataset. The field has tackled a diverse set of tasks such as captioning [13, 11, 36, 35, 21], alignment [11, 15, 34], Q&A [20, 19], visual model learning from textual descriptions [8, 26], and semantic visual search with natural multi-sentence queries [17].

[*]Denotes equal contribution

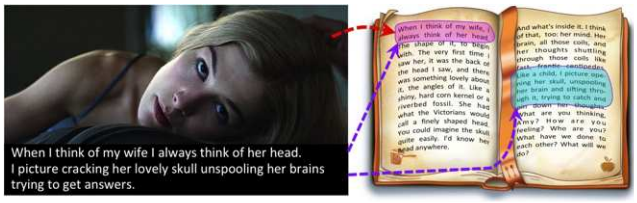


Figure 1: Shot from the movie *Gone Girl*, along with the subtitle, aligned with the book. We reason about the visual and dialog (text) alignment between the movie and a book.

Books provide us with very rich, descriptive text that conveys both fine-grained visual details (how people or scenes look like) as well as high-level semantics (what people think and feel, and how their states evolve through a story). This source of knowledge, however, does not come with associated visual information that would enable us to ground it with descriptions. Grounding descriptions in books to vision would allow us to get textual *explanations* or stories behind visual information rather than simplistic *captions* available in current datasets. It can also provide us with extremely large amount of data (with tens of thousands books available online).

In this paper, we exploit the fact that many books have been turned into movies. Books and their movie releases have a lot of common knowledge as well as they are complementary in many ways. For instance, books provide detailed descriptions about the intentions and mental states of the characters, while movies are better at capturing visual aspects of the settings.

The first challenge we need to address, and the focus of this paper, is to align books with their movie releases in order to obtain rich descriptions for the visual content. We aim to align the two sources with two types of information: *visual*, where the goal is to link a movie shot to a book paragraph, and *dialog*, where we want to find correspondences between sentences in the movie's subtitle and sentences in the book. We formulate the problem of movie/book alignment as finding correspondences between shots in the movie as well as dialog sentences in the subtitles and sentences in the book (Fig. 1). We introduce a novel sentence similarity measure based on a neural sen-

tence embedding trained on millions of sentences from a large corpus of books. On the visual side, we extend the neural image-sentence embeddings to the video domain and train the model on DVS descriptions of movie clips. Our approach combines different similarity measures and takes into account contextual information contained in the nearby shots and book sentences. Our final alignment model is formulated as an energy minimization problem that encourages the alignment to follow a similar timeline. To evaluate the book-movie alignment model we collected a dataset with 11 movie/book pairs annotated with 2,070 shot-to-sentence correspondences. We demonstrate good quantitative performance and show several qualitative examples that showcase the diversity of tasks our model can be used for.

The alignment model can have multiple applications. Imagine an app which allows the user to browse the book as the scenes unroll in the movie: perhaps its ending or acting are ambiguous, and one would like to query the book for answers. Vice-versa, while reading the book one might want to switch from text to video, particularly for the juicy scenes. We also show other applications of learning from movies and books such as book retrieval (finding the book that goes with a movie and finding other similar books), and captioning CoCo images with story-like descriptions.

2. Related Work

Most effort in the domain of vision and language has been devoted to the problem of image captioning. Older work made use of fixed visual representations and translated them into textual descriptions [6, 16]. Recently, several approaches based on RNNs emerged, generating captions via a learned joint image-text embedding [13, 11, 36, 21]. These approaches have also been extended to generate descriptions of short video clips [35]. In [24], the authors go beyond describing *what* is happening in an image and provide explanations about *why* something is happening.

For text-to-image alignment, [15, 7] find correspondences between nouns and pronouns in a caption and visual objects using several visual and textual potentials. Lin *et al*. [17] does so for videos. In [11], the authors use RNN embeddings to find the correspondences. [37] combines neural embeddings with soft attention in order to align the words to image regions.

Early work on movie-to-text alignment include dynamic time warping for aligning movies to scripts with the help of subtitles [5, 4]. Sankar *et al*. [28] further developed a system which identified sets of visual and audio features to align movies and scripts without making use of the subtitles. Such alignment has been exploited to provide weak labels for person naming tasks [5, 30, 25].

Closest to our work is [34], which aligns plot synopses to shots in the TV series for story-based content retrieval. This work adopts a similarity function between sentences in plot synopses and shots based on person identities and keywords in subtitles. Our work differs with theirs in several important aspects. First, we tackle a more challenging problem of movie/book alignment. Unlike plot synopsis, which closely follow the storyline of movies, books are more verbose and might vary in the storyline from their movie release. Furthermore, we use learned neural embeddings to compute the similarities rather than hand-designed similarity functions.

Parallel to our work, [33] aims to align scenes in movies to chapters in the book. However, their approach operates on a very coarse level (chapters), while ours does so on the sentence/paragraph level. Their dataset thus evaluates on 90 scene-chapter correspondences, while our dataset draws 2,070 shot-to-sentences alignments. Furthermore, the approaches are inherently different. [33] matches the presence of characters in a scene to those in a chapter, as well as uses hand-crafted similarity measures between sentences in the subtitles and dialogs in the books, similarly to [34].

Rohrbach *et al*. [27] recently released the Movie Description dataset which contains clips from movies, each time-stamped with a sentence from DVS (Descriptive Video Service). The dataset contains clips from over a 100 movies, and provides a great resource for the captioning techniques. Our effort here is to align movies with books in order to obtain longer, richer and more high-level video descriptions.

We start by describing our new dataset, and then explain our proposed approach.

3. The MovieBook and BookCorpus Datasets

We collected two large datasets, one for movie/book alignment and one with a large number of books.

The MovieBook Dataset. Since no prior work or data exist on the problem of movie/book alignment, we collected a new dataset with 11 movies along with the books on which they were based on. For each movie we also have a subtitle file, which we parse into a set of time-stamped sentences. Note that no speaker information is provided in the subtitles. We automatically parse each book into sentences, paragraphs (based on indentation in the book), and chapters (we assume a chapter title has indentation, starts on a new page, and does not end with an end symbol).

Our annotators had the movie and a book opened side by side. They were asked to iterate between browsing the book and watching a few shots/scenes of the movie, and trying to find correspondences between them. In particular, they marked the exact time (in seconds) of correspondence in the movie and the matching line number in the book file, indicating the beginning of the matched sentence. On the video side, we assume that the match spans across a *shot* (a video unit with smooth camera motion). If the match was longer in duration, the annotator also indicated the ending time of the match. Similarly for the book, if more sentences

	BOOK						MOVIE		ANNOTATION	
Title	# sent.	# words	# unique words	avg. # words per sent.	max # words per sent.	# paragraphs	# shots	# sent. in subtitles	# dialog align.	# visual align.
Gone Girl	12,603	148,340	3,849	15	153	3,927	2,604	2,555	76	106
Fight Club	4,229	48,946	1,833	14	90	2,082	2,365	1,864	104	42
No Country for Old Men	8,050	69,824	1,704	10	68	3,189	1,348	889	223	47
Harry Potter and the Sorcerers Stone	6,458	78,596	2,363	15	227	2,925	2,647	1,227	164	73
Shawshank Redemption	2,562	40,140	1,360	18	115	637	1,252	1,879	44	12
The Green Mile	9,467	133,241	3,043	17	119	2,760	2,350	1,846	208	102
American Psycho	11,992	143,631	4,632	16	422	3,945	1,012	1,311	278	85
One Flew Over the Cuckoo Nest	7,103	112,978	2,949	19	192	2,236	1,671	1,553	64	25
The Firm	15,498	135,529	3,685	11	85	5,223	2,423	1,775	82	60
Brokeback Mountain	638	10,640	470	20	173	167	1,205	1,228	80	20
The Road	6,638	58,793	1,580	10	74	2,345	1,108	782	126	49
All	85,238	980,658	9,032	15	156	29,436	19,985	16,909	1,449	621

Table 1: Statistics for our **MovieBook Dataset** with ground-truth for alignment between books and their movie releases.

# of books	# of sentences	# of words	# of unique words	mean # of words per sentence	median # of words per sentence
11,038	74,004,228	984,846,357	1,316,420	13	11

Table 2: Summary statistics of our **BookCorpus** dataset. We use this corpus to train the sentence embedding model.

matched, the annotator indicated from which to which line a match occurred. Each alignment was also tagged, indicating whether it was a *visual*, *dialogue*, or an *audio match*. Note that even for dialogs, the movie and book versions are semantically similar but not exactly the same. Thus deciding on what defines a match or not is also somewhat subjective and may slightly vary across our annotators. Altogether, the annotators spent 90 hours labeling 11 movie/book pairs, locating 2,070 correspondences.

Table 1 presents our dataset, while Fig. 8 shows a few ground-truth alignments. One can see the complexity and diversity of the data: the number of sentences per book vary from 638 to 15,498, even though the movies are similar in duration. This indicates a huge diversity in descriptiveness across literature, and presents a challenge for matching. The sentences also vary in length, with the sentences in Brokeback Mountain being twice as long as those in The Road. The longest sentence in American Psycho has 422 words and spans over a page in the book.

Aligning movies with books is challenging even for humans, mostly due to the scale of the data. Each movie is on average 2h long and has 1,800 shots, while a book has on average 7,750 sentences. Books also have different styles of writing, formatting, different and challenging language, slang (*going* vs *goin'*, or even *was* vs *'us*), etc. As one can see from Table 1, finding visual matches turned out to be particularly challenging. This is because the visual descriptions in books can be either very short and hidden within longer paragraphs or even within a longer sentence, or very verbose – in which case they get obscured with the surrounding text – and are hard to spot. Of course, how close the movie follows the book is also up to the director, which can be seen through the number of alignments that our annotators found across different movie/books.

The BookCorpus Dataset. In order to train our sentence similarity model we collected a corpus of 11,038 books from the web. These are free books written by yet unpublished authors. We only included books that had more than 20K words in order to filter out perhaps noisier shorter stories. The dataset has books in 16 different genres, e.g., *Romance* (2,865 books), *Fantasy* (1,479), *Science fiction* (786), *Teen* (430), etc. Table 2 highlights the summary statistics of our book corpus.

4. Aligning Books and Movies

Our approach aims to align a movie with a book by exploiting visual information as well as dialogs. We take shots as video units and sentences from subtitles to represent dialogs. Our goal is to match these to the sentences in the book. We propose several measures to compute similarities between pairs of sentences as well as shots and sentences. We use our novel deep neural embedding trained on our large corpus of books to predict similarities between sentences. Note that an extended version of the sentence embedding is described in detail in [14] showing how to deal with million-word vocabularies, and demonstrating its performance on a large variety of NLP benchmarks. For comparing shots with sentences we extend the neural embedding of images and text [13] to operate in the video domain. We next develop a novel contextual alignment model that combines information from various similarity measures and a larger time-scale in order to make better local alignment predictions. Finally, we propose a simple pairwise Conditional Random Field (CRF) that smooths the alignments by encouraging them to follow a linear timeline, both in the video and book domain.

We first explain our sentence, followed by our joint video to text embedding. We next propose our contextual model that combines similarities and discuss CRF in more detail.

4.1. Skip-Thought Vectors

In order to score the similarity between two sentences, we exploit our architecture for learning unsupervised representations of text [14]. The model is loosely inspired by

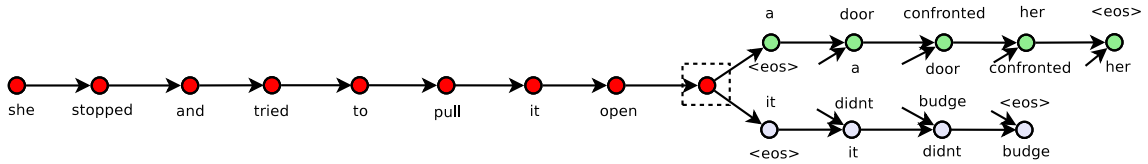


Figure 2: Sentence neural embedding [14]. Given a tuple (s_{i-1}, s_i, s_{i+1}) of consecutive sentences in text, where s_i is the i-th sentence, we encode s_i and aim to reconstruct the previous s_{i-1} and the following sentence s_{i+1}. Unattached arrows are connected to the encoder output. Colors depict which components share parameters. $\langle$eos$\rangle$ is the end of sentence token.

he drove down the street off into the distance .	he started the car , left the parking lot and merged onto the highway a few miles down the road . he shut the door and watched the taxi drive off . she watched the lights flicker through the trees as the men drove toward the road . he jogged down the stairs , through the small lobby , through the door and into the street .
the most effective way to end the battle .	a messy business to be sure , but necessary to achieve a fine and noble end . they saw their only goal as survival and logically planned a strategy to achieve it . there would be far fewer casualties and far less destruction . the outcome was the lisbon treaty .

Table 3: Qualitative results from the sentence embedding model. For each query sentence on the left, we retrieve the 4 nearest neighbor sentences (by inner product) chosen from books the model has not seen before.

the skip-gram [22] architecture for learning representations of words. In the word skip-gram model, a word w_i is chosen and must predict its surrounding context (e.g. w_{i+1} and w_{i-1} for a context window of size 1). Our model works in a similar way but at the sentence level. That is, given a sentence tuple (s_{i-1}, s_i, s_{i+1}) our model first encodes the sentence s_i into a fixed vector, then conditioned on this vector tries to reconstruct the sentences s_{i-1} and s_{i+1}, as shown in Fig. 2. The motivation for this architecture is inspired by the distributional hypothesis: sentences that have similar surrounding context are likely to be both semantically and syntactically similar. Thus, two sentences that have similar syntax and semantics are likely to be encoded to a similar vector. Once the model is trained, we can map any sentence through the encoder to obtain vector representations, then score their similarity through an inner product.

The learning signal of the model depends on having contiguous text, where sentences follow one another in sequence. A natural corpus for training our model is thus a large collection of books. Given the size and diversity of genres, our BookCorpus allows us to learn very general representations of text. For instance, Table 3 illustrates the nearest neighbours of query sentences, taken from held out books that the model was not trained on. These qualitative results demonstrate that our intuition is correct, with resulting nearest neighbors corresponds largely to syntactically and semantically similar sentences. Note that the sentence embedding is general and can be applied to other domains not considered in this paper, which is explored in [14].

To construct an encoder, we use a recurrent neural network, inspired by the success of encoder-decoder models for neural machine translation [10, 2, 1, 31]. Two kinds of activation functions have recently gained traction: long short-term memory (LSTM) [9] and the gated recurrent unit (GRU) [3]. Both types of activation successfully solve the vanishing gradient problem, through the use of gates to control the flow of information. The LSTM unit explicity employs a cell that acts as a carousel with an identity weight. The flow of information through a cell is controlled by input, output and forget gates which control what goes into a cell, what leaves a cell and whether to reset the contents of the cell. The GRU does not use a cell but employs two gates: an update and a reset gate. In a GRU, the hidden state is a linear combination of the previous hidden state and the proposed hidden state, where the combination weights are controlled by the update gate. GRUs have been shown to perform just as well as LSTM on several sequence prediction tasks [3] while being simpler. Thus, we use GRU as the activation function for our encoder and decoder RNNs.

Suppose that we are given a sentence tuple (s_{i-1}, s_i, s_{i+1}), and let w_i^t denote the t-th word for s_i and let $\mathbf{x}_i^t$ be its word embedding. We break the model description into three parts: the encoder, decoder and objective function.

Encoder. Let $w_i^1, \ldots, w_i^N$ denote words in sentence s_i with N the number of words in the sentence. The encoder produces a hidden state $\mathbf{h}_i^t$ at each time step which forms the representation of the sequence $w_i^1, \ldots, w_i^t$. Thus, the hidden state $\mathbf{h}_i^N$ is the representation of the whole sentence. The GRU produces the next hidden state as a linear combination of the previous hidden state and the proposed state update (we drop subscript i):

$$\mathbf{h}^t = (1 - \mathbf{z}^t) \odot \mathbf{h}^{t-1} + \mathbf{z}^t \odot \bar{\mathbf{h}}^t \tag{1}$$

where $\bar{\mathbf{h}}^t$ is the proposed state update at time t, $\mathbf{z}^t$ is the update gate and $(\odot)$ denotes a component-wise product. The update gate takes values between zero and one. In the extreme cases, if the update gate is the vector of ones, the previous hidden state is completely forgotten and $\mathbf{h}^t = \bar{\mathbf{h}}^t$. Alternatively, if the update gate is the zero vector, than the

hidden state from the previous time step is simply copied over, that is $\mathbf{h}^t = \mathbf{h}^{t-1}$. The update gate is computed as

$$\mathbf{z}^t = \sigma(\mathbf{W}_z\mathbf{x}^t + \mathbf{U}_z\mathbf{h}^{t-1}) \quad (2)$$

where $\mathbf{W}_z$ and $\mathbf{U}_z$ are the update gate parameters. The proposed state update is given by

$$\bar{\mathbf{h}}^t = \tanh(\mathbf{W}\mathbf{x}^t + \mathbf{U}(\mathbf{r}^t \odot \mathbf{h}^{t-1})) \quad (3)$$

where $\mathbf{r}_t$ is the reset gate, which is computed as

$$\mathbf{r}^t = \sigma(\mathbf{W}_r\mathbf{x}^t + \mathbf{U}_r\mathbf{h}^{t-1}) \quad (4)$$

If the reset gate is the zero vector, than the proposed state update is computed only as a function of the current word. Thus after iterating this equation sequence for each word, we obtain a sentence vector $\mathbf{h}_i^N = \mathbf{h}_i$ for sentence s_i.

Decoder. The decoder computation is analogous to the encoder, except that the computation is conditioned on the sentence vector $\mathbf{h}_i$. Two separate decoders are used, one for the previous sentence s_{i-1} and one for the next sentence s_{i+1}. These decoders use different parameters to compute their hidden states but both share the same vocabulary matrix $\mathbf{V}$ that takes a hidden state and computes a distribution over words. Thus, the decoders are analogous to an RNN language model but conditioned on the encoder sequence. Alternatively, in the context of image caption generation, the encoded sentence $\mathbf{h}_i$ plays a similar role as the image.

We describe the decoder for the next sentence s_{i+1} (computation for s_{i-1} is identical). Let $\mathbf{h}_{i+1}^t$ denote the hidden state of the decoder at time t. The update and reset gates for the decoder are given as follows (we drop $i+1$):

$$\begin{aligned} \mathbf{z}^t &= \sigma(\mathbf{W}_z^d\mathbf{x}^{t-1} + \mathbf{U}_z^d\mathbf{h}^{t-1} + \mathbf{C}_z\mathbf{h}_i) \quad (5)\\ \mathbf{r}^t &= \sigma(\mathbf{W}_r^d\mathbf{x}^{t-1} + \mathbf{U}_r^d\mathbf{h}^{t-1} + \mathbf{C}_r\mathbf{h}_i) \quad (6)\end{aligned}$$

the hidden state $\mathbf{h}_{i+1}^t$ is then computed as:

$$\bar{\mathbf{h}}^t = \tanh(\mathbf{W}^d\mathbf{x}^{t-1} + \mathbf{U}^d(\mathbf{r}^t \odot \mathbf{h}^{t-1}) + \mathbf{C}\mathbf{h}_i) \quad (7)$$

$$\mathbf{h}_{i+1}^t = (1 - \mathbf{z}^t) \odot \mathbf{h}^{t-1} + \mathbf{z}^t \odot \bar{\mathbf{h}}^t \quad (8)$$

Given $\mathbf{h}_{i+1}^t$, the probability of word w_{i+1}^t given the previous $t-1$ words and the encoder vector is

$$P(w_{i+1}^t | w_{i+1}^{<t}, \mathbf{h}_i) \propto \exp(\mathbf{v}_{w_{i+1}^t}\mathbf{h}_{i+1}^t) \quad (9)$$

where $\mathbf{v}_{w_{i+1}^t}$ denotes the row of $\mathbf{V}$ corresponding to the word of w_{i+1}^t. An analogous computation is performed for the previous sentence s_{i-1}.

Objective. Given (s_{i-1}, s_i, s_{i+1}), the objective optimized is the sum of log-probabilities for the next and previous sentences conditioned on the representation of the encoder:

$$\sum_t \log P(w_{i+1}^t | w_{i+1}^{<t}, \mathbf{h}_i) + \sum_t \log P(w_{i-1}^t | w_{i-1}^{<t}, \mathbf{h}_i) \quad (10)$$

The total objective is the above summed over all such training tuples. Adam algorithm [12] is used for optimization.

4.2. Visual-semantic embeddings of clips and DVS

The model above describes how to obtain a similarity score between two sentences, whose representations are learned from millions of sentences in books. We now discuss how to obtain similarities between shots and sentences.

Our approach closely follows the image-sentence ranking model proposed by [13]. In their model, an LSTM is used for encoding a sentence into a fixed vector. A linear mapping is applied to image features from a convolutional network. A score is computed based on the inner product between the normalized sentence and image vectors. Correct image-sentence pairs are trained to have high score, while incorrect pairs are assigned low scores.

In our case, we learn a visual-semantic embedding between movie clips and their DVS description. DVS ("Descriptive Video Service") is a service that inserts audio descriptions of the movie between the dialogs in order to enable the visually impaired to follow the movie like anyone else. We used the movie description dataset of [27] for learning our embedding. This dataset has 94 movies, and 54,000 described clips. We represent each movie clip as a vector corresponding to mean-pooled features across each frame in the clip. We used the GoogLeNet architecture [32] as well as hybrid-CNN [38] for extracting frame features. For DVS, we pre-processed the descriptions by removing names and replacing these with a *someone* token.

The LSTM architecture in this work is implemented using the following equations. As before, we represent a word embedding at time t of a sentence as $\mathbf{x}^t$:

$$\begin{aligned} \mathbf{i}^t &= \sigma(\mathbf{W}_{xi}\mathbf{x}^t + \mathbf{W}_{hi}\mathbf{m}^{t-1} + \mathbf{W}_{ci}\mathbf{c}^{t-1}) \quad (11)\\ \mathbf{f}^t &= \sigma(\mathbf{W}_{xf}\mathbf{x}^t + \mathbf{W}_{hf}\mathbf{m}^{t-1} + \mathbf{W}_{cf}\mathbf{c}^{t-1}) \quad (12)\\ \mathbf{a}^t &= \tanh(\mathbf{W}_{xc}\mathbf{x}^t + \mathbf{W}_{hc}\mathbf{m}^{t-1}) \quad (13)\\ \mathbf{c}^t &= \mathbf{f}^t \odot \mathbf{c}^{t-1} + \mathbf{i}^t \odot \mathbf{a}^t \quad (14)\\ \mathbf{o}^t &= \sigma(\mathbf{W}_{xo}\mathbf{x}^t + \mathbf{W}_{ho}\mathbf{m}^{t-1} + \mathbf{W}_{co}\mathbf{c}^t) \quad (15)\\ \mathbf{m}^t &= \mathbf{o}^t \odot \tanh(\mathbf{c}^t) \quad (16)\end{aligned}$$

where (σ) denotes the sigmoid activation function and $(\odot)$ indicates component-wise multiplication. The states $(\mathbf{i}^t, \mathbf{f}^t, \mathbf{c}^t, \mathbf{o}^t, \mathbf{m}^t)$ correspond to the input, forget, cell, output and memory vectors, respectively. If the sentence is of length N, then the vector $\mathbf{m}^N = \mathbf{m}$ is the vector representation of the sentence.

Let $\mathbf{q}$ denote a movie clip vector, and let $\mathbf{v} = \mathbf{W}_I\mathbf{q}$ be the embedding of the movie clip. We define a scoring function $s(\mathbf{m}, \mathbf{v}) = \mathbf{m} \cdot \mathbf{v}$, where $\mathbf{m}$ and $\mathbf{v}$ are first scaled to have unit norm (making s equivalent to cosine similarity). We then optimize the following pairwise ranking loss:

$$\begin{aligned} \min_{\boldsymbol{\theta}} &\sum_{\mathbf{m}}\sum_k \max\{0, \alpha - s(\mathbf{m}, \mathbf{v}) + s(\mathbf{m}, \mathbf{v}_k)\} \quad (17)\\ &+ \sum_{\mathbf{v}}\sum_k \max\{0, \alpha - s(\mathbf{v}, \mathbf{m}) + s(\mathbf{v}, \mathbf{m}_k)\},\end{aligned}$$

with $\mathbf{m}_k$ a contrastive (non-descriptive) sentence vector for a clip embedding $\mathbf{v}$, and vice-versa with $\mathbf{v}_k$. We train our model with stochastic gradient descent without momentum.

4.3. Context aware similarity

We employ the clip-sentence embedding to compute similarities between each shot in the movie and each sentence in the book. For dialogs, we use several similarity measures each capturing a different level of semantic similarity. We compute BLEU [23] between each subtitle and book sentence to identify nearly identical matches. Similarly to [34], we use a tf-idf measure to find near duplicates but weighing down the influence of the less frequent words. Finally, we use our sentence embedding learned from books to score pairs of sentences that are semantically similar but may have a very different wording (i.e., paraphrasing).

These similarity measures indicate the alignment between the two modalities. However, at the local, sentence level, alignment can be rather ambiguous. For example, despite being a rather dark book, *Gone Girl* contains 15 occurrences of the sentence "I love you". We exploit the fact that a match is not completely isolated but that the sentences (or shots) around it are also to some extent similar.

We design a context aware similarity measure that takes into account all individual similarity measures as well as a fixed context window in both, the movie and book domain, and predicts a new similarity score. We stack a set of M similarity measures into a tensor $S(i, j, m)$, where i, j, and m are the indices of sentences in the subtitle, in the book, and individual similarity measures, respectively. In particular, we use $M = 9$ similarities: visual and sentence embedding, BLEU1-5, tf-idf, and a uniform prior. We want to predict a combined score $\text{score}(i, j) = f(S(\mathbf{I}, \mathbf{J}, \mathbf{M}))$ at each location (i, j) based on all measurements in a fixed volume defined by $\mathbf{I}$ around i, $\mathbf{J}$ around j, and $1, \ldots, M$. Evaluating the function $f(\cdot)$ at each location (i, j) on a 3-D tensor S is very similar to applying a convolution using a kernel of appropriate size. This motivates us to formulate the function $f(\cdot)$ as a deep convolutional neural network (CNN). In this paper, we adopt a 3-layer CNN as illustrated in Figure 3. We adopt the ReLU non-linearity with dropout to regularize our model. We optimize the cross-entropy loss over the training set using Adam algorithm.

4.4. Global Movie/Book Alignment

So far, each shot/sentence was matched independently. However, most shots in movies and passages in the books follow a similar timeline. We would like to incorporate this prior into our alignment. In [34], the authors use dynamic time warping by enforcing that the shots in the movie can only match forward in time (to plot synopses in their case). However, the storyline of the movie and book can have crossings in time (Fig. 8), and the alignment might contain

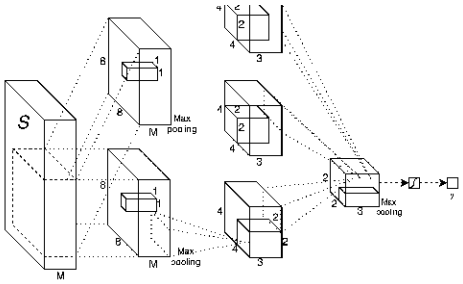


Figure 3: Our CNN for context-aware similarity computation. It has 3 conv. layers and a sigmoid layer on top.

giant leaps forwards or backwards. Therefore, we formulate a movie/book alignment problem as inference in a Conditional Random Field that encourages nearby shots/dialog alignments to be consistent. Each node y_i in our CRF represents an alignment of the shot in the movie with its corresponding subtitle sentence to a sentence in the book. Its state space is thus the set of all sentences in the book. The CRF energy of a configuration $\mathbf{y}$ is formulated as:

$$-\log p(\mathbf{x}, \mathbf{y}; \omega) = \sum_{i=1}^{K} \omega_u \phi_u(y_i) + \sum_{i=1}^{K} \sum_{j \in \mathcal{N}(i)} \omega_p \psi_p(y_i, y_j)$$

where K is the number of nodes (shots), and $\mathcal{N}(i)$ the left and right neighbor of y_i. Here, $\phi_u(\cdot)$ and $\psi_p(\cdot)$ are unary and pairwise potentials, respectively, and $\omega = (\omega_u, \omega_p)$. We directly use the output of the CNN from 4.3 as the unary potential $\phi_u(\cdot)$. For the pairwise potential, we measure the time span $d_s(y_i, y_j)$ between two neighbouring sentences in the subtitle and the distance $d_b(y_i, y_j)$ of their state space in the book. One pairwise potential is defined as:

$$\psi_p(y_i, y_j) = \frac{\left(d_s(y_i, y_j) - d_b(y_i, y_j)\right)^2}{\left(d_s(y_i, y_j) - d_b(y_i, y_j)\right)^2 + \sigma^2} \qquad (18)$$

Here σ^2 is a robustness parameter to avoid punishing giant leaps too harsh. Both d_s and d_b are normalized to $[0, 1]$. In addition, we also employ another pairwise potential $\psi_q(y_i, y_j) = \frac{(d_b(y_i, y_j))^2}{(d_b(y_i, y_j))^2 + \sigma^2}$ to encourage state consistency between nearby nodes. This potential is helpful when there is a long silence (no dialog) in the movie.

Inference. Our CRF is a chain, thus exact inference is possible using dynamic programming. We also prune some states that are very far from the uniform alignment (over $1/3$ length of the book) to further speed up computation.

Learning. Since ground-truth is only available for a sparse set of shots, we regard the states of unobserved nodes as hidden variables and learn the CRF weights with [29].

5. Experimental Evaluation

We evaluate our model on our dataset of 11 movie/book pairs. We train the parameters in our model (CNN and CRF)

on *Gone Girl*, and test our performance on the remaining 10 movies. In terms of training speed, our video-text model "watches" 1,440 movies per day and our sentence model reads 870 books per day. We also show various qualitative results demonstrating the power of our approach. We provide more results in the Appendix of the paper.

5.1. Movie/Book Alignment

Evaluating the performance of movie/book alignment is an interesting problem on its own. This is because our ground-truth is far from exhaustive – around 200 correspondences were typically found between a movie and its book, and likely a number of them got missed. Thus, evaluating the precision is rather tricky. We thus focus our evaluation on recall, similar to existing work on retrieval. For each shot that has a GT correspondence in book, we check whether our prediction is close to the annotated one. We evaluate recall at the paragraph level, i.e., we say that the GT paragraph was recalled, if our match was at most 3 paragraphs away, and the shot was at most 5 subtitle sentences away. As a noisier measure, we also compute recall and precision at multiple alignment thresholds and report AP (avg. prec.).

The results are presented in Table 4. Columns show different instantiations of our model: we show the leave-one-feature-out setting ($\emptyset$ indicates that all features were used), compare how different depths of the context-aware CNN influence the performance, and compare it to our full model (CRF) in the last column. We get the highest boost by adding more layers to the CNN – recall improves by 14%, and AP doubles. Generally, each feature helps performance. Our sentence embedding (BOOK) helps by 4%, while noisier video-text embedding helps by 2% in recall. CRF which encourages temporal smoothness generally helps (but not for all movies), bringing additional 2%. We also show how a uniform timeline performs on its own. That is, for each shot (measured in seconds) in the movie, we find the sentence at the same location (measured in lines) in the book.

We add another baseline to evaluate the role of context in our model. Instead of using our CNN that considers contextual information, we build a linear SVM to combine different similarity measures in a single node (shot) – the final similarity is used as a unary potential in our CRF alignment model. The Table shows that our CNN contextual model outperforms the SVM baseline by 30% in recall, and doubles the AP. We plot alignment for a few movies in Fig. 8.

Running Times. We show the typical running time of each component in our model in Table 5. For each movie-book pair, calculating BLEU score takes most of the time. Note that BLEU does not contribute significantly to the performance and is of optional use. With respect to the rest, extracting visual features VIS (mean pooling GoogleNet features over the shot frames) and SCENE features (mean pooling hybrid-CNN features [38] over the shot frames), takes most of the time (about 80% of the total time).

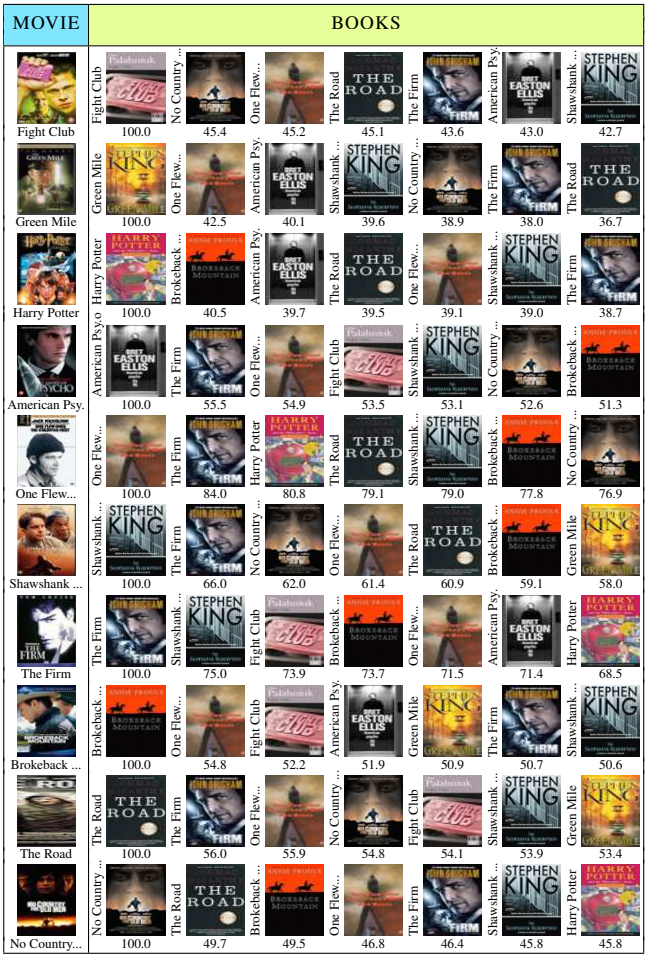


Table 6: **Book "retrieval"**. For a movie (left), we rank books wrt to their alignment similarity with the movie. We normalize similarity to be 100 for the highest scoring book.

We also report training times for our contextual model (CNN) and the CRF alignment model. Note that the times are reported for one movie/book pair since we used only one such pair to train all our CNN and CRF parameters. We chose *Gone Girl* for training since it had the best balance between the dialog and visual correspondences.

5.2. Describing Movies via the Book

We next show qualitative results of our alignment. In particular, we run our model on each movie/book pair, and visualize the passage in the book that a particular shot in the movie aligns to. We show best matching paragraphs as well as a paragraph before and after. The results are shown in Fig. 8. One can see that our model is able to retrieve a semantically meaningful match despite large dialog deviations from those in the book, and the challenge of matching a visual representation to the verbose text in the book.

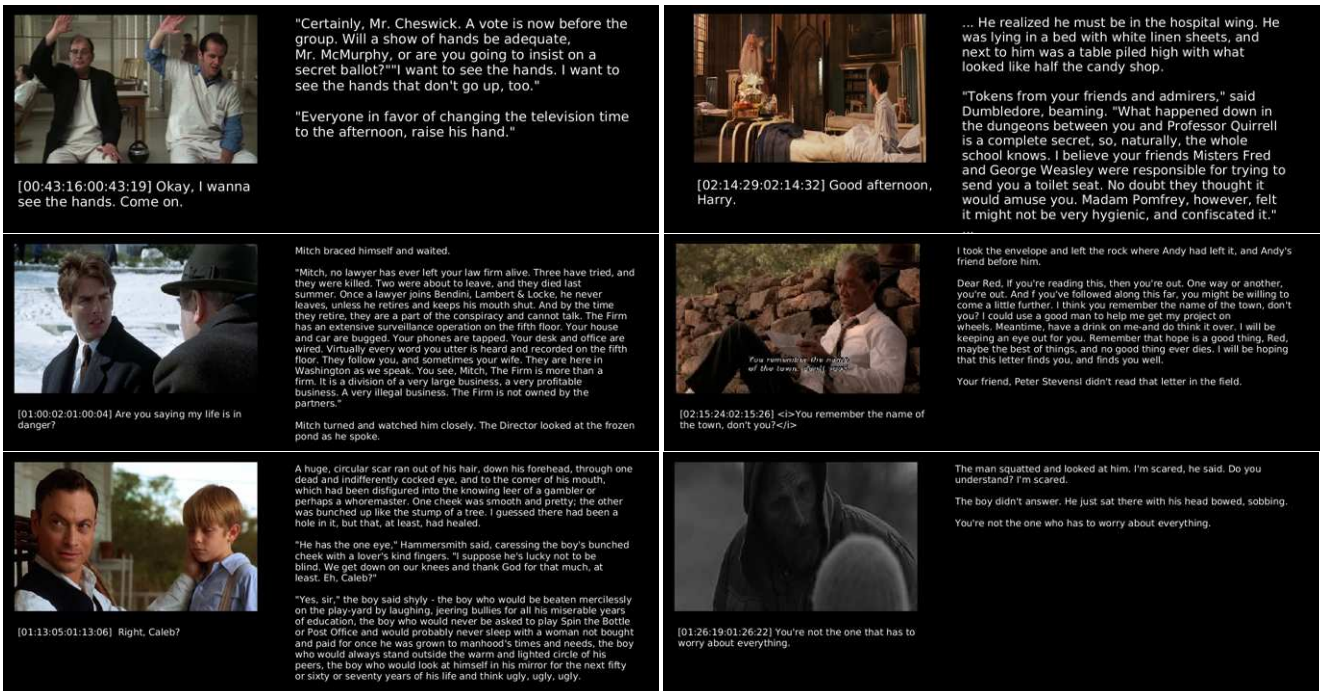


Figure 4: Describing movie clips via the book: we align the movie to the book, and show a shot from the movie and its corresponding paragraph (plus one before and after) from the book.

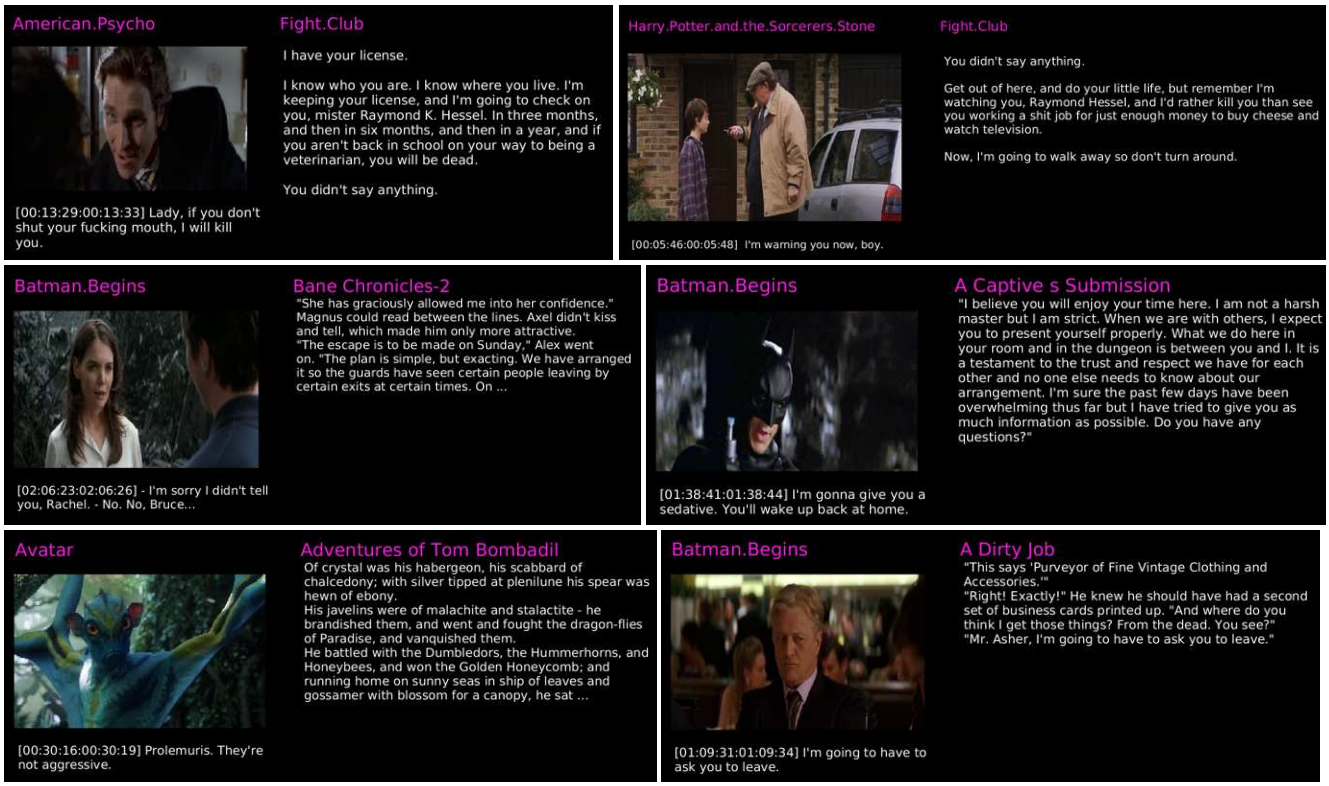


Figure 5: We can use our model to caption movies via a corpus of books. **Top:** A shot from *American Pyscho* is captioned with paragraphs from the *Fight Club*, and a shot from *Harry Potter* with paragraphs from *Fight Club*. **Middle** and **Bottom:** We match shots from *Avatar* and *Batman Begins* against 300 books from our BookCorpus, and show the best matched paragraph.

		UNI	SVM	1 layer CNN w/o one feature							CNN-3	CRF
				∅	BLEU	TF-IDF	BOOK	VIS	SCENE	PRIOR		
Fight Club	AP	1.22	0.73	0.45	0.41	0.40	0.50	0.64	0.50	0.48	1.95	**5.17**
	Recall	2.36	10.38	12.26	12.74	11.79	11.79	12.74	11.79	11.79	17.92	**19.81**
The Green Mile	AP	0.00	14.05	14.12	14.09	6.92	10.12	9.83	13.00	14.42	**28.80**	27.60
	Recall	0.00	51.42	62.46	60.57	53.94	57.10	55.52	60.57	62.78	74.13	**78.23**
Harry Potter and the Sorcerers Stone	AP	0.00	10.30	8.09	8.18	5.66	7.84	7.95	8.04	8.20	**27.17**	23.65
	Recall	0.00	44.35	51.05	52.30	46.03	48.54	48.54	49.37	52.72	76.57	**78.66**
American Psycho	AP	0.00	14.78	16.76	17.22	12.29	14.88	14.95	15.68	16.54	**34.32**	32.87
	Recall	0.27	34.25	67.12	66.58	60.82	64.66	63.56	66.58	67.67	**81.92**	80.27
One Flew Over the Cuckoo Nest	AP	0.00	5.68	8.14	6.27	1.93	8.49	8.51	9.32	9.04	14.83	**21.13**
	Recall	1.01	25.25	41.41	34.34	32.32	36.36	37.37	36.36	40.40	49.49	**54.55**
Shawshank Redemption	AP	0.00	8.94	8.60	8.89	4.35	7.99	8.91	9.22	7.86	19.33	**19.96**
	Recall	1.79	46.43	78.57	76.79	73.21	73.21	78.57	75.00	78.57	94.64	**96.79**
The Firm	AP	0.05	4.46	7.91	8.66	2.02	6.22	7.15	7.25	7.26	18.34	**20.74**
	Recall	1.38	18.62	33.79	36.55	26.90	23.45	26.90	30.34	31.03	37.93	**44.83**
Brokeback Mountain	AP	2.36	24.91	16.55	17.82	14.60	15.16	15.58	15.41	16.21	**31.80**	30.58
	Recall	27.0	74.00	88.00	92.00	86.00	86.00	88.00	86.00	87.00	98.00	**100.00**
The Road	AP	0.00	13.77	6.58	7.83	3.04	5.11	5.47	6.09	7.00	**19.80**	19.58
	Recall	1.12	41.90	43.02	48.04	32.96	38.55	37.99	42.46	44.13	**65.36**	65.10
No Country for Old Men	AP	0.00	12.11	9.00	9.39	8.22	9.40	9.35	8.63	9.40	28.75	**30.45**
	Recall	1.12	33.46	48.90	49.63	46.69	47.79	51.10	49.26	48.53	71.69	**72.79**
Mean Recall		3.88	38.01	52.66	52.95	47.07	48.75	50.03	50.77	52.46	66.77	**69.10**
AP		0.40	10.97	9.62	9.88	5.94	8.57	8.83	9.31	9.64	22.51	**23.17**

Table 4: Performance of our model for the movies in our dataset under different settings and metrics.

	BLEU	TF	BOOK	VIS	SCENE	CNN (training)	CNN (inference)	CRF (training)	CRF (inference)
Per movie-book pair	6h	10 min	3 min	2h	1h	3 min	0.2 min	5h	5 min

Table 5: Running time for our model per one movie/book pair.

5.3. Book "Retrieval"

In this experiment, we compute alignment between a movie and all (test) 10 books, and check whether our model retrieves the correct book. Results are shown in Table 6. Under each book we show the computed similarity. In particular, we use the energy from the CRF, and scale all similarities relative to the highest one (100). Notice that our model retrieves the correct book for each movie.

Describing a movie via other books. We can also caption movies by matching shots to paragraphs in a corpus of books. Here we do not encourage a linear timeline (CRF) since the stories are unrelated, and we only match at the local, shot-paragraph level. We show a description for *American Psycho* borrowed from the book *Fight Club* in Fig. 5.

5.4. The CoCoBook: Writing Stories for CoCo

Our next experiment shows that our model is able to "generate" descriptive stories for (static) images. In particular we used the image-text embedding from [13] and generated a simple caption for an image. We used this caption as a query, and used our sentence embedding trained on books to find top 10 nearest sentences (sampled from a few hundred thousand from BookCorpus). We re-ranked these based on the 1-gram precision of non-stop words. Given the best result, we return the sentence as well as the 2 sentences before and after it in the book. The results are in Fig. 6. Our sentence embedding is able to retrieve semantically meaningful *stories* to explain the images.

6. Conclusion

In this paper, we explored a new problem of aligning a book to its movie release. We proposed an approach that computes several similarities between shots and dialogs and the sentences in the book. We exploited our new sentence embedding in order to compute similarities between sentences. We further extended the image-text neural embeddings to video, and proposed a context-aware alignment model that takes into account all the available similarity information. We showed results on a new dataset of movie/book alignments as well as several quantitative results that showcase the power and potential of our approach.

Acknowledgments

We acknowledge the support from NSERC, CIFAR, Samsung, Google, and ONR-N00014-14-1-0232. We also thank Lea Jensterle for helping us with elaborate annotation, and Relu Patrascu for his help with numerous infrastructure related problems.

Appendix

In the Appendix we provide more qualitative results.

A. Qualitative Movie-Book Alignment Results

We show a few qualitative examples of alignment in Fig. 8. In this experiment, we show results obtained with our full model (CRF). For a chosen shot (a node in the CRF) we show the corresponding paragraph in the book.

the club was a little emptier than i would have expected for the late afternoon , and the bartender , in red waistcoat and bowtie , was busy wiping down his counter , replacing peanuts and putting out new coasters . a television with the latest la liga news was hung in an upper corner , and behind him , rows of bottles were reflected in a giant bar mirror . above the stools , a pergola-type overhead structure held rows of wine glasses . it was a classy place , with ferns in the corner , and not the kind of bar to which i was accustomed . my places usually had a more ... relaxed feel .

he felt like an idiot for yelling at the child , but his frustration and trepidation was getting the better of him . he glanced toward the shadowed hall and quickly nodded toward melissa before making his way forward . he came across more children sitting upon a couch in the living room . they watched him , but did n't move and did n't speak . his skin started to feel like hundreds of tiny spiders were running up and down it and he hurried on .

a few miles before tioga road reached highway 395 and the town of lee vining , smith turned onto a narrow blacktop road . on either side were parched , grassy open slopes with barbed-wire fences marking property lines . cattle and horses grazed under trees whose black silhouettes stood stark against the gold-velvet mountains . marty burst into song : " home , home on the range , where the deer and the antelope play ! where seldom is heard a discouraging word and the skies are not cloudy all day ! "

"number seventy-three , second to last from the corner . ' adam slowed the porsche as he approached the quaint-he could think of no other word to use , even though "quaint" was one he normally , manfully , avoided-townhouse , coming to a halt beside a sleek jaguar sedan . it was a quiet street , devoid of traffic at this hour on a monday night . in the bluish-tinted light of a corner street lamp , he developed a quick visual impression of wrought-iron railings on tidy front stoops , window boxes full of bright chrysanthemums , beveled glass in bay windows , and lace curtains . townhouses around here didn't rent cheaply , he could n't help but observe .

Figure 6: CoCoBook: We generate a caption for a CoCo image via [13] and retrieve its best matched sentence (+ 2 before and after) from a large book corpus. One can see a semantic relevance of the retrieved passage to the image.

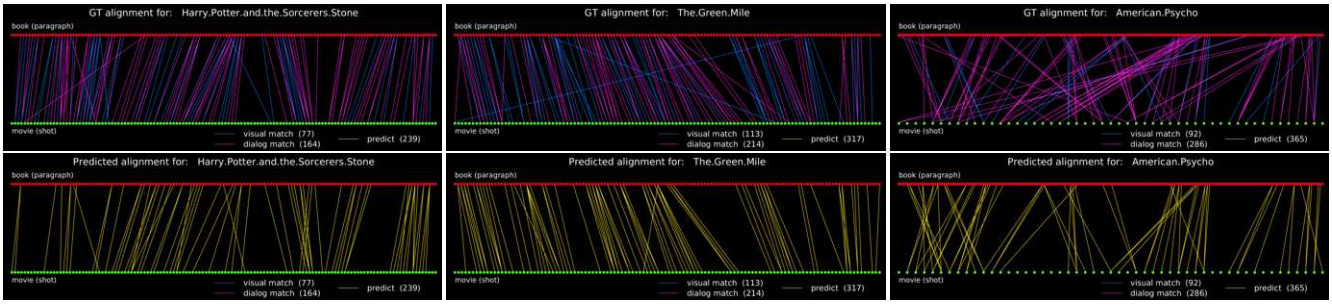


Figure 7: Alignment results of our model (**bottom**) compared to ground-truth alignment (**top**). In ground-truth, blue lines indicate visual matches, and magenta are the dialog matches. Yellow lines indicate predicted alignments.

We can see that some dialogs in the movies closely follow the book and thus help with the alignment. This is particularly important since the visual information is not as strong. Since the text around the dialogs typically describe the scene, the dialogs thus help us ground the visual information contained in the description and the video.

B. Borrowing "Lines" from Other Books

We show a few qualitative examples of top-scoring matches for shot in a movie with a paragraph in another book (a book that does not correspond to this movie).

10 book experiment. In this experiment, we allow a clip in our 10 movie dataset (excluding the training movie) to match to paragraphs in the remaining 9 books (excluding the corresponding book). The results are in Fig. 12. Note that the top-scoring matches chosen from only a small set of books may not be too meaningful.

200 book experiment. We scale the experiment by randomly selecting 200 books from our BookCorpus. The results are in Fig. 15. One can see that by using many more books results in increasingly better "stories".

[00:12:05:00:12:09] we have to promote general social concern...

American Psycho

"But we can't ignore our social needs either. We have to stop people from abusing the welfare system. We have to provide food and shelter for the homeless and oppose racial discrimination and promote civil rights while also promoting equal rights for women but change the abortion laws to protect the right to life yet still somehow maintain women's freedom of choice. We also have to control the influx of illegal immigrants. We have to encourage a return to traditional moral values and curb graphic sex and violence on TV, in movies, in popular music, everywhere. Most importantly we have to promote general social concern and less materialism in young people."

I finish my drink. The table sits facing me in total silence. Courtney's smiling and seems pleased. Timothy just shakes his head in bemused disbelief. Evelyn is completely mystified by the turn the conversation has taken and she stands, unsteadily, and asks if anyone would like dessert.

"I have... sorbet," she says as if in a daze. "Kiwi, carambola, cherimoya, cactus fruit and oh... what is that..."She stops her zombie monotone and tries to remember the last flavor. "Oh yes, Japanese pear."

[00:57:18:00:57:20] You need any help ?

American Psycho

"Yes, Patrick?" She reenters the office trying to downplay her eagerness.

"Would you like to accompany me to dinner?" I ask, still staring at the crossword, gingerly erasing the m in one of the many meats I've filled the puzzle with. "That is, if you're not... doing anything."

"Oh no," she answers too quickly and then, I think, realizing this quickness, says, "I have no plans."

[02:14:29:02:14:32] Good afternoon, Harry.

Harry Potter

... He realized he must be in the hospital wing. He was lying in a bed with white linen sheets, and next to him was a table piled high with what looked like half the candy shop.

"Tokens from your friends and admirers," said Dumbledore, beaming. "What happened down in the dungeons between you and Professor Quirrell is a complete secret, so, naturally, the whole school knows. I believe your friends Misters Fred and George Weasley were responsible for trying to send you a toilet seat. No doubt they thought it would amuse you. Madam Pomfrey, however, felt it might not be very hygienic, and confiscated it."

...

Figure 8: **Examples of movie-book alignment**. We use our model to align a movie to a book. Then for a chosen shot (which is a node in our CRF) we show the corresponding paragraph, plus one before and one after, in the book inferred by our model. On the left we show one (central) frame from the shot along with the subtitle sentence(s) that overlap with the shot. Some dialogs in the movie closely follow the book and thus help with the alignment.

[00:14:47:00:14:49] At the close of Friday's meeting...

One Flew Over the Cuckoo's Nest

The nurse looks at her watch again and pulls a slip of paper out of the folder she's holding, looks at it, and returns it to the folder. She puts the folder down and picks up the log book. Ellis coughs from his place on the wall; she waits until he stops.

"Now. At the close of Friday's meeting. we were discussing Mr. Harding's problem. concerning his young wife. He had stated that his wife was extremely well endowed in the bosom and that this made him uneasy because she drew stares from men on the street." She starts opening to places in the log book; little slips of paper stick out of the top of the book to mark the pages. "According to the notes listed by various patients in the log, Mr. Harding has been heard to say that she 'damn well gives the bastards reason to stare.' He has also been heard to say that he may give her reason to seek further sexual attention. He has been heard to say, 'My dear sweet but illiterate wife thinks any word or gesture that does not smack of brickyard brawn and brutality is a word or gesture of weak dandyism.'"

She continues reading silently from the book for a while, then closes it.

[00:43:16:00:43:19] Okay, I wanna see the hands. Come on.

One Flew Over the Cuckoo's Nest

"Certainly, Mr. Cheswick. A vote is now before the group. Will a show of hands be adequate, Mr. McMurphy, or are you going to insist on a secret ballot?""I want to see the hands. I want to see the hands that don't go up, too."

"Everyone in favor of changing the television time to the afternoon, raise his hand."

[02:15:24:02:15:26] You remember the name of the town, don't you?

Shawshank Redemption

I took the envelope and left the rock where Andy had left it, and Andy's friend before him.

Dear Red, If you're reading this, then you're out. One way or another, you're out. And f you've followed along this far, you might be willing to come a little further. I think you remember the name of the town, don't you? I could use a good man to help me get my project on wheels. Meantime, have a drink on me-and do think it over. I will be keeping an eye out for you. Remember that hope is a good thing, Red, maybe the best of things, and no good thing ever dies. I will be hoping that this letter finds you, and finds you well.

Your friend, Peter Stevensl didn't read that letter in the field.

Figure 9: **Examples of movie-book alignment**. We use our model to align a movie to a book. Then for a chosen shot (which is a node in our CRF) we show the corresponding paragraph, plus one before and one after, in the book inferred by our model. On the left we show one (central) frame from the shot along with the subtitle sentence(s) that overlap with the shot. Some dialogs in the movie closely follow the book and thus help with the alignment.

[00:03:27:00:03:30] Might we ask about the rest of your family?

The Firm

"It's very important to us, Mitch," Royce McKnight said warmly.

They all say that, thought McDeere. "Okay, my father was killed in the coal mines when I was seven years old. My mother remarried and lives in Florida. I had two brothers. Rusty was killed in Vietnam. I have a brother named Ray McDeere."

"Where is he?"

[00:08:52:00:08:55] Meanwhile, he's gonna try not to be embarrassed

The Firm

Oliver Lambert greeted Mitch and introduced him to the gang. There were about twenty in all, most of the associates in, and most barely older than the guest. The partners were too busy, Lamar had explained, and would meet him later at a private lunch. He stood at the end of the table as Mr. Lambert called for quiet.

"Gentlemen, this is Mitchell McDeere. You've all heard about him, and here he is. He is our number one choice this year, our number one draft pick, so to, speak. He is being romanced by the big boys in New York and Chicago and who knows where else, so we have to sell him on our little firm here in Memphis." They smiled and nodded their approval. The guest was embarrassed.

"He will finish at Harvard in two months and will graduate with honors. He's an associate editor of the Harvard Law Review." This made an impression, Mitch could tell. "He did his undergraduate work at Western Kentucky, where he graduated summa cum laude." This was not quite as impressive. "He also played football for four years, starting as quarterback his junior year." Now they were really impressed. A few appeared to be in awe, as if staring at Joe Namath.

[01:00:02:01:00:04] Are you saying my life is in danger?

The Firm

Mitch braced himself and waited.

"Mitch, no lawyer has ever left your law firm alive. Three have tried, and they were killed. Two were about to leave, and they died last summer. Once a lawyer joins Bendini, Lambert & Locke, he never leaves, unless he retires and keeps his mouth shut. And by the time they retire, they are a part of the conspiracy and cannot talk. The Firm has an extensive surveillance operation on the fifth floor. Your house and car are bugged. Your phones are tapped. Your desk and office are wired. Virtually every word you utter is heard and recorded on the fifth floor. They follow you, and sometimes your wife. They are here in Washington as we speak. You see, Mitch, The Firm is more than a firm. It is a division of a very large business, a very profitable business. A very illegal business. The Firm is not owned by the partners."

Mitch turned and watched him closely. The Director looked at the frozen pond as he spoke.

Figure 10: **Examples of movie-book alignment**. We use our model to align a movie to a book. Then for a chosen shot (which is a node in our CRF) we show the corresponding paragraph, plus one before and one after, in the book inferred by our model. On the left we show one (central) frame from the shot along with the subtitle sentence(s) that overlap with the shot. Some dialogs in the movie closely follow the book and thus help with the alignment.

[00:46:06:00:46:08] He's paid what he owed.

The Green Mile

Percy slapped the dead man's cheek. The flat smacking sound of his hand made us all jump. Percy looked around at us with a cocky smile on his mouth, eyes glittering. Then he looked back at Bitterbuck again. "Adios, Chief," he said. "Hope hell's hot enough for you."

"Don't do that," Brutal said, his voice hollow and declamatory in the dripping tunnel. "He's paid what he owed. He's square with the house again. You keep your hands off him."

"Aw, blow it out," Percy said, but he stepped back uneasily when Brutal moved toward him, shadow rising behind him like the shadow of that ape in the story about the Rue Morgue. But instead of grabbing at Percy, Brutal grabbed hold of the gurney and began pushing Arlen Bitterbuck slowly toward the far end of the tunnel, where his last ride was waiting, parked on the soft shoulder of the highway. The gurney's hard rubber wheels moaned on the boards; its shadow rode the bulging brick wall, waxing and waning; Dean and Harry grasped the sheet at the foot and pulled it up over The Chief's face, which had already begun to take on the waxy, characterless cast of all dead faces, the innocent as well as the guilty.

[01:14:09:01:14:10] ...you'll get bit.

The Green Mile

I nodded.

"Oh, yes," Hammersmith said. "He did it. Don't you doubt it, and don't you turn your back on him. You might get away with it once or a hundred times... even a thousand... but in the end -" He raised a hand before my eyes and snapped the fingers together rapidly against the thumb, turning the hand into a biting mouth. "You understand?"

I nodded again.

[01:35:51:01:35:53] You have my whole heart.

The Road

You said you wouldn't ever leave me.

I know. I'm sorry. You have my whole heart. You always did. You're the best guy. You always were. If I'm not here you can still talk to me. You can talk to me and I'll talk to you. You'll see.

Will I hear you?

Figure 11: **Examples of movie-book alignment**. We use our model to align a movie to a book. Then for a chosen shot (which is a node in our CRF) we show the corresponding paragraph, plus one before and one after, in the book inferred by our model. On the left we show one (central) frame from the shot along with the subtitle sentence(s) that overlap with the shot. Some dialogs in the movie closely follow the book and thus help with the alignment.

American.Psycho

[00:13:24:00:13:27] Two: I can only get these sheets in Santa Fe.

Fight.Club

I have your license.

I know who you are. I know where you live. I'm keeping your license, and I'm going to check on you, mister Raymond K. Hessel. In three months, and then in six months, and then in a year, and if you aren't back in school on your way to being a veterinarian, you will be dead.

You didn't say anything.

American.Psycho

[00:21:25:00:21:27] It's okay. I can tell.

Fight.Club

Your head rolled up and away from the gun, and you said, yeah. You said, yes, you lived in a basement.

You had some pictures in the wallet, too. There was your mother. This was a tough one for you, you'd have to open your eyes and see the picture of Mom and Dad smiling and see the gun at the same time, but you did, and then your eyes closed and you started to cry.

You were going to cool, the amazing miracle of death. One minute, you're a person, the next minute, you're an ...

American.Psycho

[00:23:44:00:23:47] You're late, honey. Oh, yes, you are. I am not late.

Fight.Club

I've never been in here before tonight.

"If you say so, sir," the bartender says, "but Thursday night, you came in to ask how soon the police were planning to shut us down." Last Thursday night, I was awake all night with the insomnia, wondering was I awake, was I sleeping. I woke up late Friday morning, bone tired and feeling I hadn't ever had my eyes closed.

"Yes, sir," the bartender says, "Thursday night, you were standing right where you are now and you were asking me about the police crackdown, and you were asking me how many guys we had to turn away from the Wednesday night fight club."

Figure 12: **Examples of of borrowing paragraphs from other books – 10 book experiment**. We show a few examples of top-scoring correspondences between a shot in a movie and a paragraph in a book that does not correspond to the movie. Note that by forcing the model to choose from another book, the top-scoring correspondences may still have a relatively low similarity. In this experiment, we did not enforce a global alignment over the full book – we use the similarity output by our contextual CNN.

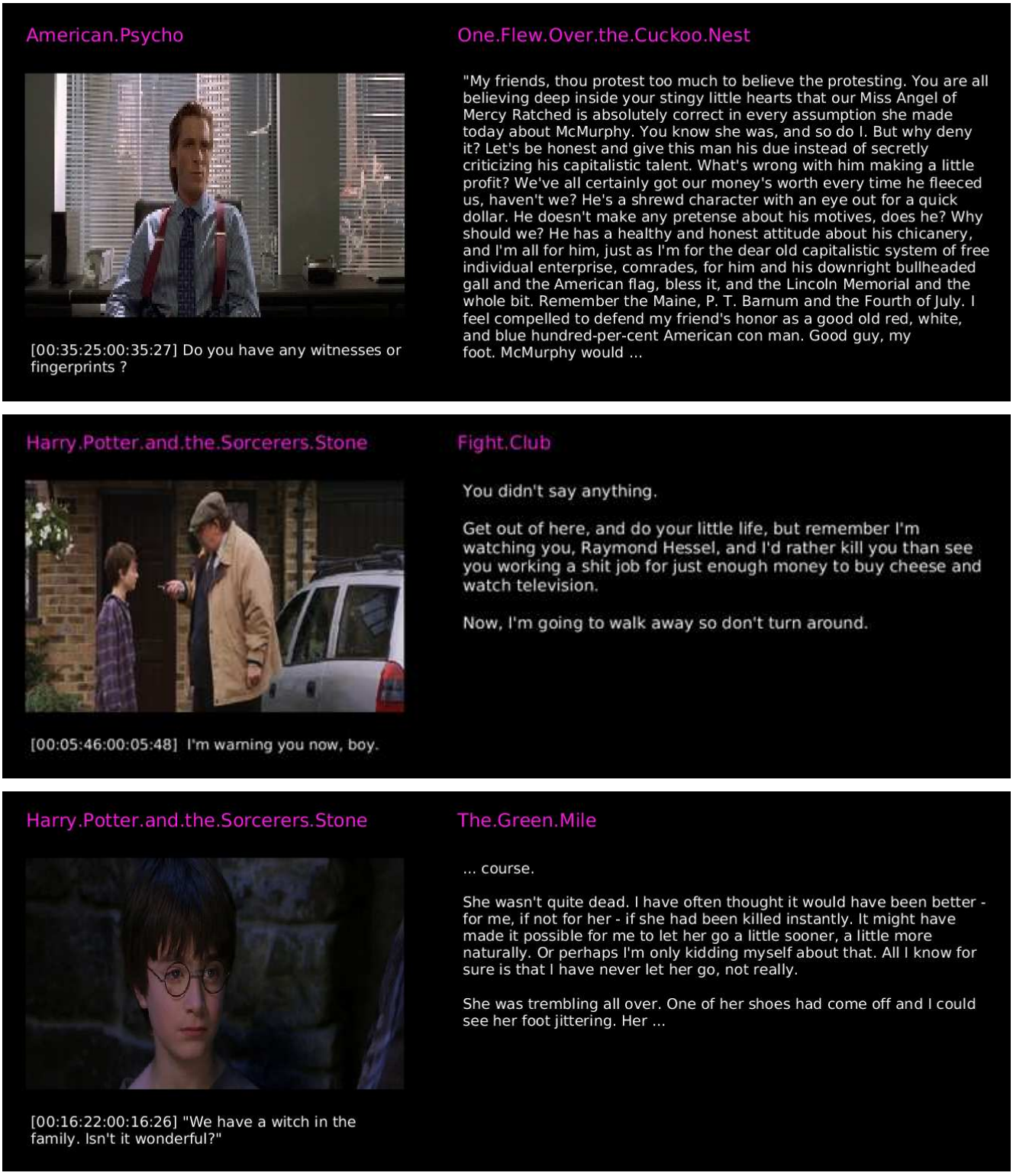


Figure 13: **Examples of of borrowing paragraphs from other books – 10 book experiment**. We show a few examples of top-scoring correspondences between a shot in a movie and a paragraph in a book that does not correspond to the movie. Note that by forcing the model to choose from another book, the top-scoring correspondences may still have a relatively low similarity. In this experiment, we did not enforce a global alignment over the full book – we use the similarity output by our contextual CNN.

Fight.Club

[00:55:19:00:55:23] No, no. I may need to talk to you a little futher, so how about you just let me know if you're gonna leave town.

13th Reality-2

... ya see, the thing is..." He scratched his beard. "See, I done heard yer little twitter feet up on my ceilin' there, so I come up to do some investigatin'. Yep, that's what I reckon, far as I recall."
Tick exchanged a baffled look with Sofia and Paul. It didn't take a genius to realize they'd already caught Sally in his first lie.
"Well," Tick said, "we need a minute to talk about what we're gonna do."

Fight.Club

[01:25:28:01:25:30] - Two pair of black pants? - Yes, sir.

AKissofShadows

... last night, or were the Tears still affecting me more than I realized? I didn't think about it again. I just turned and walked to the bathroom. A quick shower and we'd be on our way to the airport.
Twenty minutes later I was ready, my hair still soaking wet. I was dressed in a pair of navy blue dress slacks, an emerald green silk blouse, and a navy suit jacket that matched the pants. Jeremy had also chosen a pair of black low-heeled pumps and included a pair of black thigh-highs. Since I didn't own any other kind of hose, that I didn't mind. But the rest of it...
"Next time you pick out clothes for me to run for my life in, include some jogging shoes. Pumps, no matter how low-heeled, just aren't made for it."

The.Green.Mile

[01:55:38:01:55:41] I'm thinking I don't know what I would do if you were gone.

Aeons-Gate-1

You, he wanted to say, I'm thinking of you. I'm thinking of your stink and how bad you smell and how I can't stop smelling you. I'm thinking of how you keep staring at me and how I never say anything about it and I don't know why. I'm thinking of you staring at me and why someone's screaming at me inside my head and how someone's screaming inside my head and why it seems odd that I'm not worried about that.

Figure 14: **Examples of of borrowing paragraphs from other books – 200 book experiment**. We show a few examples of top-scoring correspondences between a shot in a movie and a paragraph in a book that does not correspond to the movie. By scaling up the experiment (more books to choose from), our model gets increasingly more relevant "stories".

Harry.Potter

[01:52:05:01:52:09] - How do you know? - Someone's going to try and steal it.

ACaressofTwilight

"A good bodyguard doesn't relax on the job," Ethan said.
"You know we aren't a threat to Ms. Reed, Ethan. I don't know who you're supposed to be protecting her from, but it isn't us."
"They may clean up for the press, but I know what they are, Meredith," Ethan said.

The.Green.Mile

[00:37:32:00:37:35] ...and I'll never do it again, that's for sure.

AScannerDarkly

I could use, he reflected, anything that'd help, anything at all. Any hint, like from that girl, any suggestion. He felt dismal and afraid. Shit, he thought, what am I going to do? If I'm off everything, he thought, then I'll never see any of them again, any of my friends, the people I watched and knew. I'll be out of it; I'll be maybe retired the rest of my life-anyhow, I've seen the last of Arctor and Luckman and Jerry Fabin and Charles Freck and most of all Donna Hawthorne. I'll never see any of my friends again, for the rest of eternity. It's over.

Harry.Potter

[00:55:54:00:55:58] Now, once you've got hold of your broom, I want you to mount it.

ALickofFrost2

He came to his knees and put his hands on my arms, and stared down into my face. "I will love you always. When this red hair is white, I will still love you. When the smooth softness of youth is replaced by the delicate softness of age, I will still want to touch your skin. When your face is full of the line of every smile you have ever smiled, of every surprise I have seen flash through your eyes, when every tear you have ever cried has left its mark upon your face, I will treasure you all the more, because I was there to see it all. I will share your life with you, Meredith, and I ...

Figure 15: **Examples of of borrowing paragraphs from other books – 200 book experiment**. We show a few examples of top-scoring correspondences between a shot in a movie and a paragraph in a book that does not correspond to the movie. By scaling up the experiment (more books to choose from), our model gets increasingly more relevant "stories". Bottom row: failed example.

C. The CoCoBook

We show more results for captioning CoCo images [18] with passages from the books.

“ somewhere you ’ll never find it , ” owens sneered . if never meant five seconds , his claim was true . the little shit ’s gaze cut left , where a laptop sat on a coffee table . trey strode to it . owens ’ email program was open .

seriously . its like a train crashing into another train . a train wreck . just something like that . i try to convince her .

everyone was allowed to rest for the next twenty-four hours . that following evening : the elect , not their entourages , were called to a dining hall for supper with lady dolorous . a table that curved inward was laden with food and drink . the wall behind the table was windows with a view of the planet . girls in pink stood about and at attention .

he had simply ... healed . brian watched his fellow passengers come aboard . a young woman with blonde hair was walking with a little girl in dark glasses . the little girl ’s hand was on the blonde ’s elbow . the woman murmured to her charge , the girl looked immediately toward the sound of her voice , and brian understood she was blind - it was something in the gesture of the head .

this was a beautiful miniature reproduction of a real london town house , and when jessamine touched it , tessa saw that the front of it swung open on tiny hinges . tessa caught her breath . there were beautiful tiny rooms perfectly decorated with miniature furniture , everything built to scale , from the little wooden chairs with needlepoint cushions to the cast-iron stove in the kitchen . there were small dolls , too , with china heads , and real little oil paintings on the walls . “ this was my house . ”

if he had been nearby he would have dragged her out of the room by her hair and strangled her . during lunch break she went with a group back to the encampment . out of view of the house , under a stand of towering trees , several tents were sitting in a field of mud . the rain the night before had washed the world , but here it had made a mess of things . a few women fired up a camp stove and put on rice and lentils .

then a frightened yell . “ hang on ! ” suddenly , jake was flying through the air . nefertiti became airborne , too . he screamed , not knowing what was happening-then he splashed into a pool of water .

grabbing his wristwatch off the bedside table he checked the time , grimacing when he saw that it was just after two in the afternoon . jeanne louise should n’t be up yet . stifling a yawn , he slid out of bed and made his way to the en suite bathroom for a shower . twenty minutes later paul was showered , dressed , and had brushed his teeth and hair . feeling somewhat alive now , he made his way out of his and jeanne louise ’s room , pausing to look in on livy as he passed .

she cried . quentin put a heavy , warm , calming hand on her thigh , saying , “ he should be sober by then . ” a cell phone rang . he pulled his from his back pocket , glanced at it , then used the remote to turn the tv to the channel that showed the feed from the camera at the security gate . “ oh , it ’s rachel . ”

now however she was out of his shot . he had missed it completely until he had ended up on the ground with his shotgun . an old clock hung on the wall near the door . the was obviously broken , the small red hand ticking the same second away over and over again . morgan squeezed the trigger and pellets ripped out of their package , bounced down the barrel , flew through the air and ripped into the old clock tearing it in two before it smashed to the ground .

a man sat in a chair , facing the wall opposite of me . it nearly startled me when i first saw him , and made a bit of a squeak , but he did nothing . he had dark gray hair , a black suit and pants , and a gray and blue striped tie . s-sir ? i said .

its been years since we last played together , but as i recall , he was rather weak at the net . or was it his serving ? all i know is he plays tennis much better than he plays cricket . perhaps , mr brearly , frances eventually replied , we should wait until we actually start playing . then we can ascertain our oppositions faults , and make a plan based on the new information .

since it was the middle of summer , there were candles in the fireplace instead of a fire . but it still cast a romantic glow over the room . there were candles on the mantle and on a table set up in the corner with flowers . as she looked around , her eyes instinctively turned to find max who was behind a bar opening a bottle of champagne . the doors were closed quietly behind her and her mouth felt dry as she looked across the room at the man who had haunted her dreams for so long .

the open doorway of another house provided a view of an ancient game of tiles . it wasnt the game that held reddings attention . it was the four elderly people who sat around a table playing the game . they were well beyond their productive years and the canal township had probably been their whole lives . redding and lin ming stepped away from the doorway right into the path of a wooden pushcart .

along with the fish , howard had given them some other picnic treats that had spoiled ... mushrooms in cream sauce , rotted greens . the bats and temp were only eating from the river now , but the remaining picnic food was running low . there were a few loaves of stale bread , some cheese , some dried vegetables , and a couple of cakes . gregor looked over the supplies and thought about boots wailing for food and water in the jungle . it had been unbearable .

he felt the first stirrings of fear mixing with his anger . a light flicked on in the room and eric jerked , blinking for a minute at the brightness before the images focused . there was a tall , thin man standing over a mannequin . he looked like he was assembling it , since its leg was on the ground next to the man and its arm was in two pieces farther away . then the mannequin 's head turned .

References

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *ICLR*, 2015. 4

[2] K. Cho, B. van Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *EMNLP*, 2014. 4

[3] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014. 4

[4] T. Cour, C. Jordan, E. Miltsakaki, and B. Taskar. Movie/script: Alignment and parsing of video and text transcription. In *ECCV*, 2008. 2

[5] M. Everingham, J. Sivic, and A. Zisserman. "Hello! My name is... Buffy" – Automatic Naming of Characters in TV Video. *BMVC*, pages 899–908, 2006. 2

[6] A. Farhadi, M. Hejrati, M. Sadeghi, P. Young, C. Rashtchian, J. Hockenmaier, and D. Forsyth. Every picture tells a story: Generating sentences for images. In *ECCV*, 2010. 2

[7] S. Fidler, A. Sharma, and R. Urtasun. A sentence is worth a thousand pixels. In *CVPR*, 2013. 2

[8] A. Gupta and L. Davis. Beyond nouns: Exploiting prepositions and comparative adjectives for learning visual classifiers. In *ECCV*, 2008. 1

[9] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 4

[10] N. Kalchbrenner and P. Blunsom. Recurrent continuous translation models. In *EMNLP*, pages 1700–1709, 2013. 4

[11] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *CVPR*, 2015. 1, 2

[12] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5

[13] R. Kiros, R. Salakhutdinov, and R. S. Zemel. Unifying visual-semantic embeddings with multimodal neural language models. *CoRR*, abs/1411.2539, 2014. 1, 2, 3, 5, 9, 10

[14] R. Kiros, Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler. Skip-Thought Vectors. In *Arxiv*, 2015. 3, 4

[15] C. Kong, D. Lin, M. Bansal, R. Urtasun, and S. Fidler. What are you talking about? text-to-image coreference. In *CVPR*, 2014. 1, 2

[16] G. Kulkarni, V. Premraj, S. Dhar, S. Li, Y. Choi, A. Berg, and T. Berg. Baby talk: Understanding and generating simple image descriptions. In *CVPR*, 2011. 2

[17] D. Lin, S. Fidler, C. Kong, and R. Urtasun. Visual Semantic Search: Retrieving Videos via Complex Textual Queries. *CVPR*, pages 2657–2664, 2014. 1, 2

[18] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *ECCV*, pages 740–755. 2014. 1, 19

[19] X. Lin and D. Parikh. Don't just listen, use your imagination: Leveraging visual common sense for non-visual tasks. In *CVPR*, 2015. 1

[20] M. Malinowski and M. Fritz. A multi-world approach to question answering about real-world scenes based on uncertain input. In *NIPS*, 2014. 1

[21] J. Mao, W. Xu, Y. Yang, J. Wang, and A. L. Yuille. Explain images with multimodal recurrent neural networks. In *arXiv:1410.1090*, 2014. 1, 2

[22] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. 4

[23] K. Papineni, S. Roukos, T. Ward, and W. J. Zhu. BLEU: a method for automatic evaluation of machine translation. In *ACL*, pages 311–318, 2002. 6

[24] H. Pirsiavash, C. Vondrick, and A. Torralba. Inferring the why in images. *arXiv.org*, jun 2014. 2

[25] V. Ramanathan, A. Joulin, P. Liang, and L. Fei-Fei. Linking People in Videos with "Their" Names Using Coreference Resolution. In *ECCV*, pages 95–110. 2014. 2

[26] V. Ramanathan, P. Liang, and L. Fei-Fei. Video event understanding using natural language descriptions. In *ICCV*, 2013. 1

[27] A. Rohrbach, M. Rohrbach, N. Tandon, and B. Schiele. A dataset for movie description. In *CVPR*, 2015. 2, 5

[28] P. Sankar, C. V. Jawahar, and A. Zisserman. Subtitle-free Movie to Script Alignment. In *BMVC*, 2009. 2

[29] A. Schwing, T. Hazan, M. Pollefeys, and R. Urtasun. Efficient Structured Prediction with Latent Variables for General Graphical Models. In *ICML*, 2012. 6

[30] J. Sivic, M. Everingham, and A. Zisserman. "Who are you?" - Learning person specific classifiers from video. *CVPR*, pages 1145–1152, 2009. 2

[31] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014. 4

[32] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014. 5

[33] M. Tapaswi, M. Bauml, and R. Stiefelhagen. Book2Movie: Aligning Video scenes with Book chapters. In *CVPR*, 2015. 2

[34] M. Tapaswi, M. Buml, and R. Stiefelhagen. Aligning Plot Synopses to Videos for Story-based Retrieval. *IJMIR*, 4:3–16, 2015. 1, 2, 6

[35] S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. J. Mooney, and K. Saenko. Translating Videos to Natural Language Using Deep Recurrent Neural Networks. *CoRR abs/1312.6229*, cs.CV, 2014. 1, 2

[36] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In *arXiv:1411.4555*, 2014. 1, 2

[37] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *arXiv:1502.03044*, 2015. 2

[38] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning Deep Features for Scene Recognition using Places Database. In *NIPS*, 2014. 5, 7

Deep Networks for Image Super-Resolution with Sparse Prior

Zhaowen Wang[†‡] Ding Liu[†] Jianchao Yang[§] Wei Han[†] Thomas Huang[†]
[†]Beckman Institute, University of Illinois at Urbana-Champaign, Urbana, IL
[‡]Adobe Research, San Jose, CA [§]Snapchat, Venice, CA
[‡]zhawang@adobe.com [†]{dingliu2, weihan3, huang}@ifp.uiuc.edu [§]jcyangenator@gmail.com

Abstract

Deep learning techniques have been successfully applied in many areas of computer vision, including low-level image restoration problems. For image super-resolution, several models based on deep neural networks have been recently proposed and attained superior performance that overshadows all previous handcrafted models. The question then arises whether large-capacity and data-driven models have become the dominant solution to the ill-posed super-resolution problem. In this paper, we argue that domain expertise represented by the conventional sparse coding model is still valuable, and it can be combined with the key ingredients of deep learning to achieve further improved results. We show that a sparse coding model particularly designed for super-resolution can be incarnated as a neural network, and trained in a cascaded structure from end to end. The interpretation of the network based on sparse coding leads to much more efficient and effective training, as well as a reduced model size. Our model is evaluated on a wide range of images, and shows clear advantage over existing state-of-the-art methods in terms of both restoration accuracy and human subjective quality.

1. Introduction

Single image super-resolution (SR) aims at obtaining a high-resolution (HR) image from a low-resolution (LR) input image by inferring all the missing high frequency contents. With the known variables in LR images greatly outnumbered by the unknowns in HR images, SR is a highly ill-posed problem and the current techniques are far from being satisfactory for many real applications [2, 21].

To regularize the solution of SR, people have exploited various priors of natural images. Analytical priors, such as bicubic interpolation, work well for smooth regions; while image models based on statistics of edges [11] and gradients [17, 1] can recover sharper structures. In the patch-based SR methods, HR patch candidates are represented as the sparse linear combination of dictionary atoms trained from external databases [36, 35], or recovered from similar examples in the LR image itself at different locations and across different scales [13, 12, 32]. A comprehensive review of more SR methods can be found in [33].

More recently, inspired by the great success achieved by deep learning [18, 27, 30] in other computer vision tasks, people begin to use neural networks with deep architecture for image SR. Multiple layers of collaborative auto-encoders are stacked together in [6] for robust matching of self-similar patches. Deep convolutional neural networks (CNN) [8] and deconvolutional networks [25] are designed that directly learn the non-linear mapping from LR space to HR space in a way similar to coupled sparse coding [35]. As these deep networks allow end-to-end training of all the model components between LR input and HR output, significant improvements have been observed over their shadow counterparts.

The networks in [6, 8] are built with generic architectures, which means all their knowledge about SR is learned from training data. On the other hand, people's domain expertise for the SR problem, such as natural image prior and image degradation model, is largely ignored in deep learning based approaches. It is then worthy to investigate whether domain expertise can be used to design better deep model architectures, or whether deep learning can be leveraged to improve the quality of handcrafted models.

In this paper, we extend the conventional sparse coding model [36] using several key ideas from deep learning, and show that domain expertise is complementary to large learning capacity in further improving SR performance. First, based on the learned iterative shrinkage and thresholding algorithm (LISTA) [14], we implement a feed-forward neural network whose layers strictly correspond to each step in the processing flow of sparse coding based image SR. In this way, the sparse representation prior is effectively encoded in our network structure; at the same time, all the components of sparse coding can be trained jointly through back-propagation. This simple model, which is named sparse coding based network (SCN), achieves notable improvement over the generic CNN model [8] in terms

of both recovery accuracy and human perception, and yet has a compact model size. Moreover, with the correct understanding of each layer's physical meaning, we have a more principled way to initialize the parameters of SCN, which helps to improve optimization speed and quality.

A single network is only able to perform image SR by a particular scaling factor. In [8], different networks are trained for different scaling factors. In this paper, we also propose a cascade of multiple SCNs to achieve SR for arbitrary factors. This simple approach, motivated by the self-similarity based SR approach [13], not only increases the scaling flexibility of our model, but also reduces artifacts for large scaling factors. The cascade of SCNs (CSCN) can also benefit from the end-to-end training of deep network with a specially designed multi-scale cost function.

In short, the contributions of this paper include:

- combine the domain expertise of sparse coding and the merits of deep learning to achieve better SR performance with faster training and smaller model size;
- use network cascading for large and arbitrary scaling factors;
- conduct a subjective evaluation on several recent state-of-the-art methods.

In the following, we will first review related work in Sec. 2. The SCN and CSCN models are introduced in Sec. 3 and Sec. 4, with implementation details in Sec. 5. Extensive experimental results are reported in Sec. 6, and conclusions are drawn in Sec. 7.

2. Related Work

2.1. Image SR Using Sparse Coding

The sparse representation based SR method [36] models the transform from each local patch $\boldsymbol{y} \in \mathbb{R}^{m_y}$ in the bicubic-upscaled LR image to the corresponding patch $\boldsymbol{x} \in \mathbb{R}^{m_x}$ in the HR image. The dimension m_y is not necessarily the same as m_x when image features other than raw pixel is used to represent patch $\boldsymbol{y}$. It is assumed that the LR(HR) patch $\boldsymbol{y}(\boldsymbol{x})$ can be represented with respect to an overcomplete dictionary $\boldsymbol{D}_y(\boldsymbol{D}_x)$ using some sparse linear coefficients $\boldsymbol{\alpha}_y(\boldsymbol{\alpha}_x) \in \mathbb{R}^n$, which are known as sparse code. Since the degradation process from $\boldsymbol{x}$ to $\boldsymbol{y}$ is nearly linear, the patch pair can share the same sparse code $\boldsymbol{\alpha}_y = \boldsymbol{\alpha}_x = \boldsymbol{\alpha}$ if the dictionaries $\boldsymbol{D}_y$ and $\boldsymbol{D}_x$ are defined properly. Therefore, for an input LR patch $\boldsymbol{y}$, the HR patch can be recovered as

$$\boldsymbol{x} = \boldsymbol{D}_x\boldsymbol{\alpha}, \ \text{ s.t. } \boldsymbol{\alpha} = \arg\min_{\boldsymbol{z}} \|\boldsymbol{y} - \boldsymbol{D}_y\boldsymbol{z}\|_2^2 + \lambda\|\boldsymbol{z}\|_1, \quad (1)$$

where $\|\cdot\|_1$ denotes the ℓ_1 norm which is convex and sparsity-inducing, and λ is a regularization coefficient. The dictionary pair $(\boldsymbol{D}_y, \boldsymbol{D}_x)$ can be learned alternatively with the inference of training patches' sparse codes in their joint space [36] or through bi-level optimization [35].

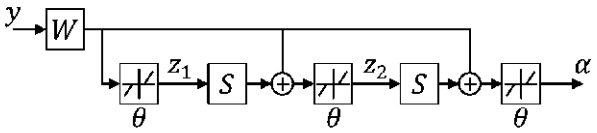


Figure 1. A LISTA network [14] with 2 time-unfolded recurrent stages, whose output $\boldsymbol{\alpha}$ is an approximation of the sparse code of input signal $\boldsymbol{y}$. The linear weights $\boldsymbol{W}$, $\boldsymbol{S}$ and the shrinkage thresholds $\boldsymbol{\theta}$ are learned from data.

2.2. Network Implementation of Sparse Coding

There is an intimate connection between sparse coding and neural network, which has been well studied in [16, 14]. A feed-forward neural network as illustrated in Fig. 1 is proposed in [14] to efficiently approximate the sparse code $\boldsymbol{\alpha}$ of input signal $\boldsymbol{y}$ as it would be obtained by solving (1) for a given dictionary $\boldsymbol{D}_y$. The network has a finite number of recurrent stages, each of which updates the intermediate sparse code according to

$$\boldsymbol{z}_{k+1} = h_{\boldsymbol{\theta}}(\boldsymbol{W}\boldsymbol{y} + \boldsymbol{S}\boldsymbol{z}_k), \quad (2)$$

where $h_{\boldsymbol{\theta}}$ is an element-wise shrinkage function defined as $[h_{\boldsymbol{\theta}}(\boldsymbol{a})]_i = \text{sign}(a_i)(|a_i| - \theta_i)_+$ with positive thresholds $\boldsymbol{\theta}$.

Different from the iterative shrinkage and thresholding algorithm (ISTA) [7, 26] which finds an analytical relationship between network parameters (weights $\boldsymbol{W}$, $\boldsymbol{S}$ and thresholds $\boldsymbol{\theta}$) and sparse coding parameters ($\boldsymbol{D}_y$ and λ), the authors of [14] learn all the network parameters from training data using a back-propagation algorithm called learned ISTA (LISTA). In this way, a good approximation of the underlying sparse code can be obtained within a fixed number of recurrent stages.

3. Sparse Coding based Network for Image SR

Given the fact that sparse coding can be effectively implemented with a LISTA network, it is straightforward to build a multi-layer neural network that mimics the processing flow of the sparse coding based SR method [36]. Same as most patch-based SR methods, our sparse coding based network (SCN) takes the bicubic-upscaled LR image $\boldsymbol{I}_y$ as input, and outputs the full HR image $\boldsymbol{I}_x$. Fig. 2 shows the main network structure, and each of the layers is described in the following.

The input image $\boldsymbol{I}_y$ first goes through a convolutional layer $\boldsymbol{H}$ which extracts feature for each LR patch. There are m_y filters of spatial size $s_y \times s_y$ in this layer, so that our input patch size is $s_y \times s_y$ and its feature representation $\boldsymbol{y}$ has m_y dimensions.

Each LR patch $\boldsymbol{y}$ is then fed into a LISTA network with a finite number of k recurrent stages to obtain its sparse code $\boldsymbol{\alpha} \in \mathbb{R}^n$. Each stage of LISTA consists of two linear

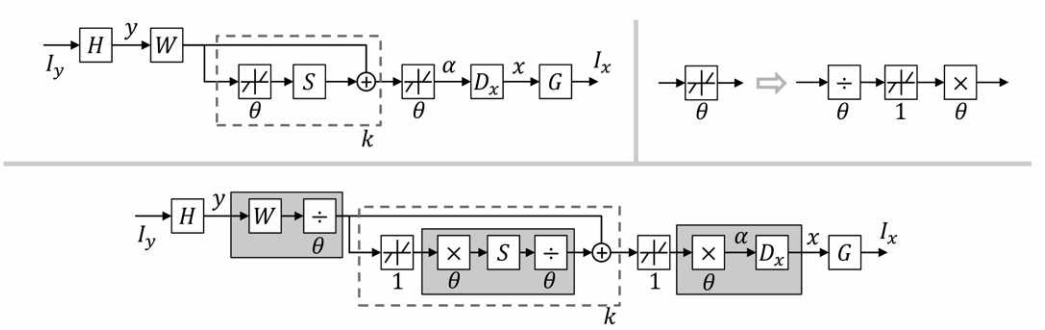


Figure 2. Top left: the proposed SCN model with a patch extraction layer $\boldsymbol{H}$, a LISTA sub-network for sparse coding (with k recurrent stages denoted by the dashed box), a HR patch recovery layer $\boldsymbol{D}_x$, and a patch combination layer $\boldsymbol{G}$. Top right: a neuron with an adjustable threshold decomposed into two linear scaling layers and a unit-threshold neuron. Bottom: the SCN re-organized with unit-threshold neurons and adjacent linear layers merged together in the gray boxes.

layers parameterized by $\boldsymbol{W} \in \mathbb{R}^{n\times m_y}$ and $\boldsymbol{S} \in \mathbb{R}^{n\times n}$, and a nonlinear neuron layer with activation function $h_{\boldsymbol{\theta}}$. The activation thresholds $\boldsymbol{\theta} \in \mathbb{R}^n$ are also to be updated during training, which complicates the learning algorithm. To restrict all the tunable parameters in our linear layers, we do a simple trick to rewrite the activation function as

$$[h_{\boldsymbol{\theta}}(\boldsymbol{a})]_i = \text{sign}(a_i)\theta_i(|a_i|/\theta_i - 1)_+ = \theta_i h_1(a_i/\theta_i). \quad (3)$$

Eq. (3) indicates the original neuron with an adjustable threshold can be decomposed into two linear scaling layers and a unit-threshold neuron, as shown in the top-right of Fig. 2. The weights of the two scaling layers are diagonal matrices defined by $\boldsymbol{\theta}$ and its element-wise reciprocal, respectively.

The sparse code $\boldsymbol{\alpha}$ is then multiplied with HR dictionary $\boldsymbol{D}_x \in \mathbb{R}^{m_x\times n}$ in the next linear layer, reconstructing HR patch $\boldsymbol{x}$ of size $s_x\times s_x = m_x$.

In the final layer $\boldsymbol{G}$, all the recovered patches are put back to the corresponding positions in the HR image $\boldsymbol{I}_x$. This is realized via a convolutional filter of m_x channels with spatial size $s_g\times s_g$. The size s_g is determined as the number of neighboring patches that overlap with the same pixel in each spatial direction. The filter will assign appropriate weights to the overlapped recoveries from different patches and take their weighted average as the final prediction in $\boldsymbol{I}_x$.

As illustrated in the bottom of Fig. 2, after some simple reorganizations of the layer connections, the network described above has some adjacent linear layers which can be merged into a single layer. This helps to reduce the computation load as well as redundant parameters in the network. The layers $\boldsymbol{H}$ and $\boldsymbol{G}$ are not merged because we apply additional nonlinear normalization operations on patches $\boldsymbol{y}$ and $\boldsymbol{x}$, which will be detailed in Sec. 5.

Thus, there are totally 5 trainable layers in our network: 2 convolutional layers $\boldsymbol{H}$ and $\boldsymbol{G}$, and 3 linear layers shown as gray boxes in Fig. 2. The k recurrent layers share the same weights and are therefore conceptually regarded as one. Note that all the linear layers are actually implemented as convolutional layers applied on each patch with filter spatial size of 1×1, a structure similar to the network in network [20]. Also note that all these layers have only weights but no biases (zero biases).

Mean square error (MSE) is employed as the cost function to train the network, and our optimization objective can be expressed as

$$\min_{\boldsymbol{\Theta}} \sum_i \|SCN(\boldsymbol{I}_y^{(i)};\boldsymbol{\Theta}) - \boldsymbol{I}_x^{(i)}\|_2^2, \quad (4)$$

where $\boldsymbol{I}_y^{(i)}$ and $\boldsymbol{I}_x^{(i)}$ are the i-th pair of LR/HR training data, and $SCN(\boldsymbol{I}_y;\boldsymbol{\Theta})$ denotes the HR image for $\boldsymbol{I}_y$ predicted using the SCN model with parameter set $\boldsymbol{\Theta}$. All the parameters are optimized through the standard back-propagation algorithm. Although it is possible to use other cost terms that are more correlated with human visual perception than MSE, our experimental results show that simply minimizing MSE leads to improvement in subjective quality.

Advantages over Previous Models

The construction of our SCN follows exactly each step in the sparse coding based SR method [36]. If the network parameters are set according to the dictionaries learned in [36], it can reproduce almost the same results. However, after training, SCN learns a more complex regression function and can no longer be converted to an equivalent sparse coding model. The advantage of SCN comes from its ability to jointly optimize all the layer parameters from end to end; while in [36] some variables are manually designed and some are optimized individually by fixing all the others.

Technically, our network is also a CNN and it has similar layers as the CNN model proposed in [8] for patch extraction and reconstruction. The key difference is that we have a LISTA sub-network specifically designed to enforce sparse representation prior; while in [8] a generic rectified linear

unit (ReLU) [24] is used for nonlinear mapping. Since SCN is designed based on our domain knowledge in sparse coding, we are able to obtain a better interpretation of the filter responses and have a better way to initialize the filter parameters in training. We will see in the experiments that all these contribute to better SR results, faster training speed and smaller model size than a vanilla CNN.

4. Network Cascade for Scalable SR

Like most SR models learned from external training examples, the SCN discussed previously can only upscale images by a fixed factor. A separate model needs to be trained for each scaling factor to achieve the best performance, which limits the flexibility and scalability in practical use. One way to overcome this difficulty is to repeatedly enlarge the image by a fixed scale until the resulting HR image reaches a desired size. This practice is commonly adopted in the self-similarity based methods [13, 12, 6], but is not so popular in other cases for the fear of error accumulation during repetitive upscaling.

In our case, however, it is observed that a cascade of SCNs (CSCN) trained for small scaling factors can generate even better SR results than a single SCN trained for a large scaling factor, especially when the target scaling factor is large (greater than 2). This is illustrated by the example in Fig. 3. Here an input image is magnified by $\times 4$ times in two ways: with a single SCN$\times$4 model through the processing flow (a) $\rightarrow$ (b) $\rightarrow$ (d); and with a cascade of two SCN$\times$2 models through (a) $\rightarrow$ (c) $\rightarrow$ (e). It can be seen that the input to the second cascaded SCN$\times$2 in (c) is already sharper and contains less artifacts than the bicubic$\times$4 input to the single SCN$\times$4 in (b), which naturally leads to the better final result in (e) than the one in (d). Therefore, each SCN in the cascade serves as a "relaying station" which progressively recovers some useful information lost in bicubic interpolation and compensates for the distortion aggregated from previous stages.

The CSCN is also a deep network, in which the output of each SCN is connected to the input of the next SCN with bicubic interpolation in the between. To construct the cascade, besides stacking several SCNs trained individually with respect to (4), we can also optimize all of them jointly as shown in Fig. 4. Without loss of generality, we assume each SCN in the cascade has the same scaling factor s. Let $\boldsymbol{I}_0$ denote the input image of original size, and $\hat{\boldsymbol{I}}_j$ (j>0) denote the output image of the j-th SCN upscaled by a total of $\times s^j$ times. Each $\hat{\boldsymbol{I}}_j$ can be compared with its associated ground truth image $\boldsymbol{I}_j$ according to the MSE cost, leading to a multi-scale objective function:

$$\min_{\{\boldsymbol{\Theta}_j\}} \sum_i \sum_j \left\| SCN(\hat{\boldsymbol{I}}_{j-1}^{(i)} \uparrow s; \boldsymbol{\Theta}_j) - \boldsymbol{I}_j^{(i)} \right\|_2^2, \qquad (5)$$

where i denotes the data index, and j denotes the SCN index. $\boldsymbol{I}\uparrow s$ is the bicubic interpolated image of $\boldsymbol{I}$ by a factor of s. This multi-scale objective function makes full use of the supervision information in all scales, sharing a similar idea as heterogeneous networks [19, 5]. All the layer parameters $\{\boldsymbol{\Theta}_j\}$ in (5) could be optimized from end to end by back-propagation. We use a greedy algorithm here to train each SCN sequentially from the beginning of the cascade so that we do not need to care about the gradient of bicubic layers. Applying back-propagation through a bicubic layer or its trainable surrogate will be considered in future work.

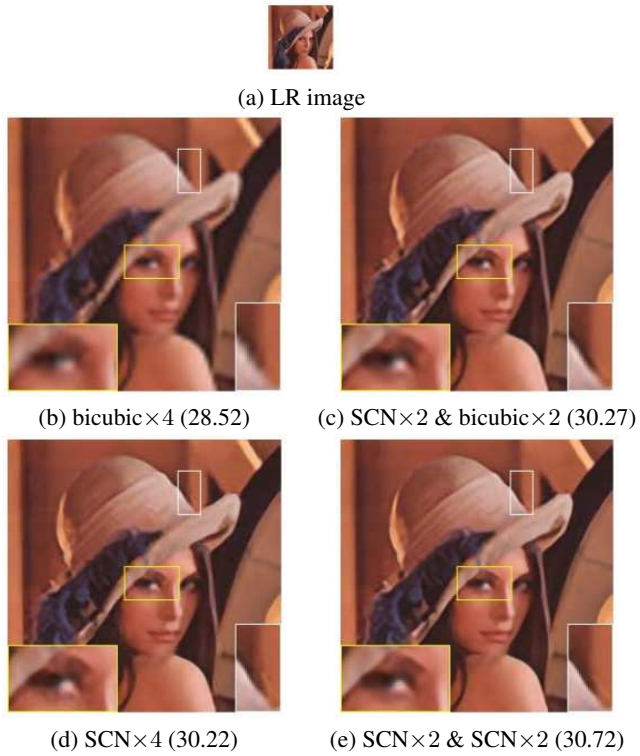


Figure 3. SR results for the "Lena" image upscaled by 4 times. (a) $\rightarrow$ (b) $\rightarrow$ (d) represents the processing flow with a single SCN$\times$4 model. (a) $\rightarrow$ (c) $\rightarrow$ (e) represents the processing flow with two cascaded SCN$\times$2 models. PSNR is given in parentheses.

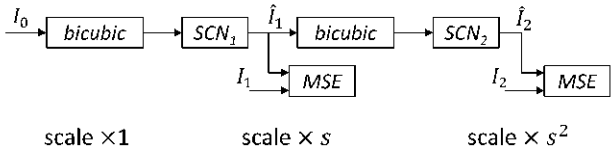


Figure 4. Training cascade of SCNs with multi-scale objectives.

5. Implementation Details

We determine the number of nodes in each layer of our SCN mainly according to the corresponding settings used in sparse coding [35]. Unless otherwise stated, we use

input LR patch size s_y=9, LR feature dimension m_y=100, dictionary size n=128, output HR patch size s_x=5, and patch aggregation filter size s_g=5. All the convolution layers have a stride of 1. Each LR patch $\boldsymbol{y}$ is normalized by its mean and variance, and the same mean and variance are used to restore the final HR patch $\boldsymbol{x}$. We crop 56×56 regions from each image to obtain fixed-sized input samples to the network, which produces outputs of size 44×44.

To reduce the number of parameters, we implement the LR patch extraction layer $\boldsymbol{H}$ as the combination of two layers: the first layer has 4 trainable filters each of which is shifted to 25 fixed positions by the second layer. Similarly, the patch combination layer $\boldsymbol{G}$ is also split into a fixed layer which aligns pixels in overlapping patches and a trainable layer whose weights are used to combine overlapping pixels. In this way, the number of parameters in these two layers are reduced by more than an order, and there is no observable loss in performance.

We employ a standard stochastic gradient descent algorithm to train our networks with mini-batch size of 64. Based on the understanding of each layer's role in sparse coding, we use Harr-like gradient filters to initialize layer $\boldsymbol{H}$, and use uniform weights to initialize layer $\boldsymbol{G}$. All the remaining three linear layers are related to the dictionary pair $(\boldsymbol{D}_x, \boldsymbol{D}_y)$ in sparse coding. To initialize them, we first randomly set $\boldsymbol{D}_x$ and $\boldsymbol{D}_y$ with Gaussian noise, and then find the corresponding layer weights as in ISTA [7]:

$$\boldsymbol{w}_1 = C{\cdot}\boldsymbol{D}_y^T,\ \boldsymbol{w}_2 = \boldsymbol{I}{-}\boldsymbol{D}_y^T\boldsymbol{D}_y,\ \boldsymbol{w}_3 = (CL)^{-1}{\cdot}\boldsymbol{D}_x \quad (6)$$

where $\boldsymbol{w}_1$, $\boldsymbol{w}_2$ and $\boldsymbol{w}_3$ denote the weights of the three subsequent layers after layer $\boldsymbol{H}$. L is the upper bound on the largest eigenvalue of $\boldsymbol{D}_y^T\boldsymbol{D}_y$, and C is the threshold value before normalization. We empirically set L=C=5.

The proposed models are all trained using the CUDA ConvNet package [18] on a workstation with 12 Intel Xeon 2.67GHz CPUs and 1 GTX680 GPU. Training a SCN usually takes less than one day. Note that this package is customized for classification networks, and its efficiency can be further optimized for our SCN model.

In testing, to make the entire image covered by output samples, we crop input samples with overlap and extend the boundary of original image by reflection. Note we shave the image border in the same way as [8] for objective evaluations to ensure fair comparison. Only the luminance channel is processed with our method, and bicubic interpolation is applied to the chrominance channels. To achieve arbitrary upscaling factors using CSCN, we upscale an image by ×2 times repeatedly until it is at least as large as the desired size. Then a bicubic interpolation is used to downscale it to the target resolution if necessary.

When reporting our best results in Sec. 6.2, we also use the multi-view testing strategy commonly employed in image classification. For patch-based image SR, multi-view testing is implicitly used when predictions from multiple overlapping patches are averaged. Here, besides sampling overlapping patches, we also add more views by flipping and transposing the patch. Such strategy is found to improve SR performance for general algorithms at the sheer cost of computation.

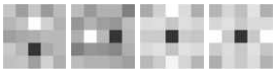


Figure 5. The four learned filters in the first layer $\boldsymbol{H}$.

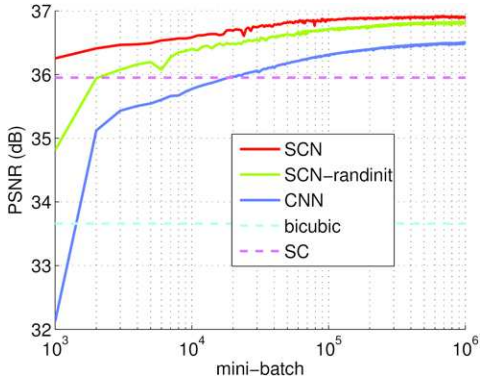


Figure 6. The PSNR change for ×2 SR on Set5 during training using different methods: SCN; SCN with random initialization; CNN. The horizontal dash lines show the benchmarks of bicubic interpolation and sparse coding (SC).

6. Experiments

We evaluate and compare the performance of our models using the same data and protocols as in [28], which are commonly adopted in SR literature. All our models are learned from a training set with 91 images, and tested on Set5 [3], Set14 [37] and BSD100 [23] which contain 5, 14 and 100 images respectively. We have also trained on a different larger data set, and observe little performance change (less than 0.1dB). The original images are downsized by bicubic interpolation to generate LR-HR image pairs for both training and evaluation. The training data are augmented with translation, rotation and scaling.

6.1. Algorithm Analysis

We first visualize the four filters learned in the first layer $\boldsymbol{H}$ in Fig. 5. The filter patterns do not change much from the initial first and second order gradient operators. Some additional small coefficients are introduced in a highly structured form that capture richer high frequency details.

The performance of several networks during training is measured on Set5 in Fig. 6. Our SCN improves significantly over sparse coding (SC) [35], as it leverages data more effectively with end-to-end training. The SCN initialized according to (6) can converge faster and better than the

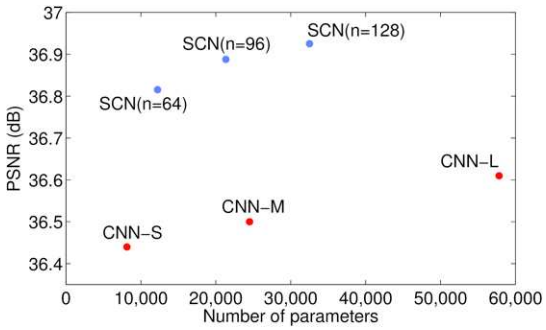


Figure 7. PSNR for $\times 2$ SR on Set5 using SCN and CNN with various network sizes.

Table 1. PSNR of different network cascading schemes on Set5, evaluated for different scaling factors in each column.

upscale factor	$\times 1.5$	$\times 2$	$\times 3$	$\times 4$
SCN$\times 1.5$	40.14	36.41	30.33	29.02
SCN$\times 2$	**40.15**	**36.93**	32.99	30.70
SCN$\times 3$	39.88	36.76	32.87	30.63
SCN$\times 4$	39.69	36.54	32.76	30.55
CSCN	**40.15**	**36.93**	**33.10**	**30.86**

same model with random initialization, which indicates that the understanding of SCN based on sparse coding can help its optimization. We also train a CNN model [8] of the same size as SCN, but find its convergence speed much slower. It is reported in [8] that training a CNN takes 8×10^8 back-propagations (equivalent to 12.5×10^6 mini-batches here). To achieve the same performance as CNN, our SCN requires less than 1% back-propagations.

The network size of SCN is mainly determined by the dictionary size n. Besides the default value n=128, we have tried other sizes and plot their performance versus the number of network parameters in Fig. 7. The PSNR of SCN does not drop too much as n decreases from 128 to 64, but the model size and computation time can be reduced significantly. Fig. 7 also shows the performance of CNN with various sizes. Our smallest SCN can achieve higher PSNR than the largest model (CNN-L) in [9] while only using about 20% parameters.

Different numbers of recurrent stages k have been tested for SCN, and we find increasing k from 1 to 3 only improves performance by less than 0.1dB. As a tradeoff between speed and accuracy, we use k=1 throughout the paper.

In Table 1, different network cascade structures (in each row) are compared at different scaling factors (in each column). SCN$\times a$ denotes the simple cascade of SCN with fixed scaling factor a, where an individually trained SCN is applied one or more times for scaling factors other than a. It is observed that SCN$\times 2$ can perform as well as the scale-specific model for small scaling factor (1.5), and much better for large scaling factors (3 and 4). Note that the cascade of SCN$\times 1.5$ does not lead to good results since artifacts quickly get amplified through many repetitive up-scalings. Therefore, we use SCN$\times 2$ as the default building block for CSCN, and drop the notation $\times 2$ when there is no ambiguity. The last row in Table 1 shows that a CSCN trained using the multi-scale objective in (5) can further improve the SR results for scaling factors 3 and 4, as the second SCN in the cascade is trained to be robust to the artifacts generated by the first one.

6.2. Comparison with State of the Arts

We compare the proposed CSCN with other recent SR methods on all the images in Set5, Set14 and BSD100 for different upscaling factors. Table 2 shows the PSNR and structural similarity (SSIM) [31] for adjusted anchored neighborhood regression (A+) [29], CNN [8], CNN trained with larger model size and more data (CNN-L) [9], the proposed CSCN, and CSCN with our multi-view testing (CSCN-MV). We do not list other methods [35, 28, 37, 17, 15] whose performance is worse than A+ or CNN-L.

It can be seen from Table 2 that CSCN performs consistently better than all previous methods in both PSNR and SSIM, and with multi-view testing the results can be further improved. CNN-L improves over CNN by increasing model parameters and training data. However, it is still not as good as CSCN which is trained with a much smaller size and on a much smaller data set. Clearly, the better model structure of CSCN makes it less dependent on model capacity and training data in improving performance. Our models are generally more advantageous for large scaling factors due to the cascade structure.

The visual qualities of the SR results generated by sparse coding (SC) [35], CNN and CSCN are compared in Fig. 8. Our approach produces image patterns with shaper boundaries and richer textures, and is free of the ringing artifacts observable in the other two methods.

Fig. 9 shows the SR results on the "chip" image compared among more methods including the self-example based method (SE) [12] and the deep network cascade (DNC) [6]. SE and DNC can generate very sharp edges on this image, but also introduce artifacts and blurs on corners and fine structures due to the lack of self-similar patches. On the contrary, the CSCN method recovers all the structures of the characters without any distortion.

We also compare CSCN with other sparse coding extensions [22, 10, 38], and consider the blurring effect introduced in downscaling. A PSNR gain of 0.3$\sim$1.6dB is achieved by CSCN in general. Experiment details and source codes are available online[1].

6.3. Subjective Evaluation

We conducted a subjective evaluation of SR results for several methods including bicubic, SC [35], SE [12],

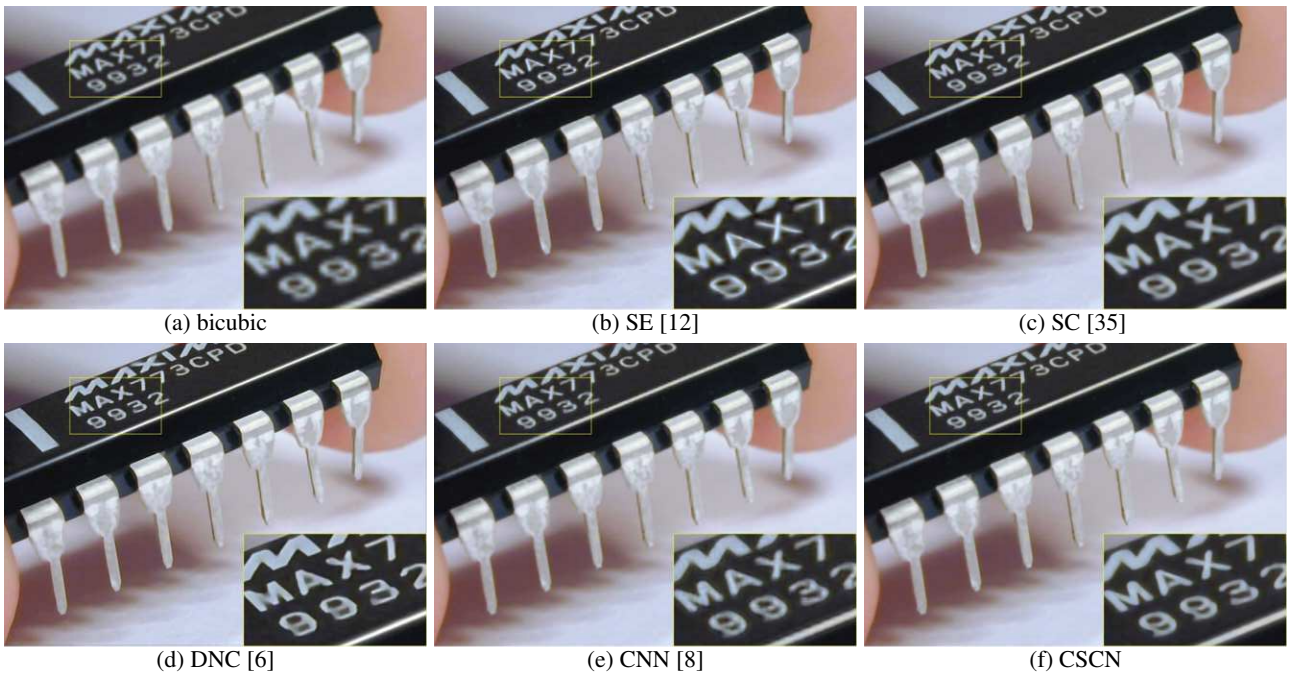
[1] www.ifp.illinois.edu/~dingliu2/iccv15

Table 2. PSNR (SSIM) comparison on three test data sets among different methods. Red indicates the best and blue indicates the second best performance. The performance gain of our best model over all the others' best is shown in the last row.

Data Set	Set5			Set14			BSD100		
Upscaling	×2	×3	×4	×2	×3	×4	×2	×3	×4
A+ [29]	36.55 (0.9544)	32.59 (0.9088)	30.29 (0.8603)	32.28 (0.9056)	29.13 (0.8188)	27.33 (0.7491)	30.78 (0.8773)	28.18 (0.7808)	26.77 (0.7085)
CNN [8]	36.34 (0.9521)	32.39 (0.9033)	30.09 (0.8530)	32.18 (0.9039)	29.00 (0.8145)	27.20 (0.7413)	31.11 (0.8835)	28.20 (0.7794)	26.70 (0.7018)
CNN-L [9]	36.66 (0.9542)	32.75 (0.9090)	30.49 (0.8628)	32.45 (0.9067)	29.30 (0.8215)	27.50 (0.7513)	31.36 (0.8879)	28.41 (0.7863)	26.90 (0.7103)
CSCN	36.93 (0.9552)	33.10 (0.9144)	30.86 (0.8732)	32.56 (0.9074)	29.41 (0.8238)	27.64 (0.7578)	31.40 (0.8884)	28.50 (0.7885)	27.03 (0.7161)
CSCN-MV	37.14 (0.9567)	33.26 (0.9167)	31.04 (0.8775)	32.71 (0.9095)	29.55 (0.8271)	27.76 (0.7620)	31.54 (0.8908)	28.58 (0.7910)	27.11 (0.7191)
Our Improvement	0.48 (0.0023)	0.51 (0.0077)	0.55 (0.0147)	0.26 (0.0028)	0.25 (0.0056)	0.26 (0.0107)	0.18 (0.0029)	0.17 (0.0047)	0.21 (0.0088)

Figure 8. SR results given by SC [35] (first row), CNN [8] (second row) and our CSCN (third row). Images from left to right: the "monarch" image upscaled by ×3; the "zebra" image upscaled by ×3; the "comic" image upscaled by ×3.

self-example regression (SER) [34], CNN [8] and CSCN. Ground truth HR images are also included when they are available as references. Each of the participants in the evaluation is shown a set of HR image pairs, which are



(a) bicubic (b) SE [12] (c) SC [35]

(d) DNC [6] (e) CNN [8] (f) CSCN

Figure 9. The "chip" image upscaled by $\times 4$ times using different methods.

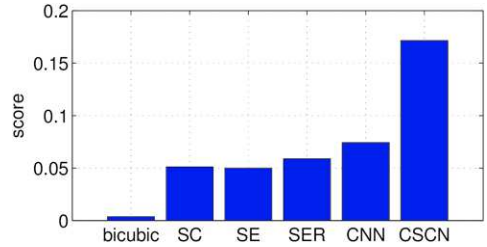


Figure 10. Subjective SR quality scores for different methods including bicubic, SC [35], SE [12], SER [34], CNN [8] and the proposed CSCN. The score for ground truth result is 1.

upscaled from the same LR images using two randomly selected methods. For each pair, the subject needs to decide which one is better in terms of perceptual quality.

We have a total of 270 participants giving 720 pairwise comparisons over 6 images with different scaling factors. Not every participant completed all the comparisons but their partial responses are still useful. All the evaluation results can be summarized into a 7×7 winning matrix for 7 methods (including ground truth), based on which we fit a Bradley-Terry [4] model to estimate the subjective score for each method so that they can be ranked.

Fig. 10 shows the estimated scores for the 6 SR methods in our evaluation, with the score for ground truth method normalized to 1. As expected, all the SR methods have much lower scores than ground truth, showing the great challenge in SR problem. The bicubic interpolation is significantly worse than other SR methods. The proposed CSCN method outperforms other previous state-of-the-art methods by a large margin, demonstrating its superior visual quality. It should be noted that the visual difference between some image pairs is very subtle. Nevertheless, the human subjects are able to perceive such difference when seeing the two images side by side, and therefore make consistent ratings. The CNN model becomes less competitive in the subjective evaluation than it is in PSNR comparison. This indicates that the visually appealing image appearance produced by CSCN should be attributed to the regularization from sparse representation, which can not be easily learned by merely minimizing reconstruction error as in CNN.

7. Conclusions

We propose a new model for image SR by combining the strengths of sparse coding and deep network, and make considerable improvement over existing deep and shallow SR models both quantitatively and qualitatively. Besides producing good SR results, the domain knowledge in the form of sparse coding can also benefit training speed and model compactness. Furthermore, we propose a cascaded network for better flexibility in scaling factors as well as more robustness to artifacts.

In future work, we will apply the SCN model to other problems where sparse coding can be useful. The interaction between deep networks for low-level and high-level vision tasks will also be explored.

References

[1] H. A. Aly and E. Dubois. Image up-sampling using total-variation regularization with a new observation model. *IEEE TIP*, 14(10):1647–1659, 2005. 1

[2] S. Baker and T. Kanade. Limits on super-resolution and how to break them. *IEEE TPAMI*, 24(9):1167–1183, 2002. 1

[3] M. Bevilacqua, A. Roumy, C. Guillemot, and M.-L. A. Morel. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. In *BMVC*, 2012. 5

[4] R. A. Bradley and M. E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, pages 324–345, 1952. 8

[5] S. Chang, W. Han, J. Tang, G.-J. Qi, C. C. Aggarwal, and T. S. Huang. Heterogeneous network embedding via deep architectures. In *ACM SIGKDD*. ACM, 2015. 4

[6] Z. Cui, H. Chang, S. Shan, B. Zhong, and X. Chen. Deep network cascade for image super-resolution. In *ECCV*, pages 49–64, 2014. 1, 4, 6, 8

[7] I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics*, 57(11):1413–1457, 2004. 2, 5

[8] C. Dong, C. C. Loy, K. He, and X. Tang. Learning a deep convolutional network for image super-resolution. In *ECCV*, pages 184–199, 2014. 1, 2, 3, 5, 6, 7, 8

[9] C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. *IEEE TPAMI*, 2015. 6, 7

[10] W. Dong, L. Zhang, and G. Shi. Centralized sparse representation for image restoration. In *ICCV*, pages 1259–1266, 2011. 6

[11] R. Fattal. Image upsampling via imposed edge statistics. In *ACM Transactions on Graphics*, volume 26:3, page 95, 2007. 1

[12] G. Freedman and R. Fattal. Image and video upscaling from local self-examples. *ACM Transactions on Graphics*, 30(2):12, 2011. 1, 4, 6, 8

[13] D. Glasner, S. Bagon, and M. Irani. Super-resolution from a single image. In *ICCV*, 2009. 1, 2, 4

[14] K. Gregor and Y. LeCun. Learning fast approximations of sparse coding. In *ICML*, 2010. 1, 2

[15] J.-B. Huang, A. Singh, and N. Ahuja. Single image super-resolution from transformed self-exemplars. In *CVPR*, 2015. 6

[16] K. Kavukcuoglu, M. Ranzato, and Y. LeCun. Fast inference in sparse coding algorithms with applications to object recognition. *arXiv preprint arXiv:1010.3467*, 2010. 2

[17] K. I. Kim and Y. Kwon. Single-image super-resolution using sparse regression and natural image prior. *IEEE TPAMI*, 32(6):1127–1133, 2010. 1, 6

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012. 1, 5

[19] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. *arXiv preprint arXiv:1409.5185*, 2014. 4

[20] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013. 3

[21] Z. Lin and H.-Y. Shum. Fundamental limits of reconstruction-based superresolution algorithms under local translation. *IEEE TPAMI*, 26(1):83–97, 2004. 1

[22] X. Lu, H. Yuan, P. Yan, Y. Yuan, and X. Li. Geometry constrained sparse coding for single image super-resolution. In *CVPR*, pages 1648–1655, 2012. 6

[23] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *ICCV*, volume 2, pages 416–423, July 2001. 5

[24] V. Nair and G. E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *ICML*, pages 807–814, 2010. 4

[25] C. Osendorfer, H. Soyer, and P. van der Smagt. Image super-resolution with fast approximate convolutional sparse coding. In *Neural Information Processing*, pages 250–257. Springer, 2014. 1

[26] C. J. Rozell, D. H. Johnson, R. G. Baraniuk, and B. A. Olshausen. Sparse coding via thresholding and local competition in neural circuits. *Neural Computation*, 20(10):2526–2563, 2008. 2

[27] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013. 1

[28] R. Timofte, V. De, and L. V. Gool. Anchored neighborhood regression for fast example-based super-resolution. In *ICCV*, pages 1920–1927, 2013. 5, 6

[29] R. Timofte, V. De Smet, and L. Van Gool. A+: Adjusted anchored neighborhood regression for fast super-resolution. In *ACCV*, 2014. 6, 7

[30] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, pages 1096–1103, 2008. 1

[31] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE TIP*, 13(4):600–612, 2004. 6

[32] Z. Wang, Y. Yang, Z. Wang, S. Chang, J. Yang, and T. S. Huang. Learning super-resolution jointly from external and internal examples. *IEEE TIP*, 24(11):4359–4371, 2015. 1

[33] C.-Y. Yang, C. Ma, and M.-H. Yang. Single-image super-resolution: a benchmark. In *ECCV*, pages 372–386, 2014. 1

[34] J. Yang, Z. Lin, and S. Cohen. Fast image super-resolution based on in-place example regression. In *CVPR*, pages 1059–1066. IEEE, 2013. 7, 8

[35] J. Yang, Z. Wang, Z. Lin, S. Cohen, and T. Huang. Coupled dictionary training for image super-resolution. *IEEE TIP*, 21(8):3467–3478, 2012. 1, 2, 4, 5, 6, 7, 8

[36] J. Yang, J. Wright, T. Huang, and Y. Ma. Image super-resolution via sparse representation. *IEEE TIP*, 19(11):1–8, 2010. 1, 2, 3

[37] R. Zeyde, M. Elad, and M. Protter. On single image scale-up using sparse-representations. In *Curves and Surfaces*, pages 711–730. 2012. 5, 6

[38] K. Zhang, X. Gao, D. Tao, and X. Li. Single image super-resolution with non-local means and steering kernel regression. *IEEE TIP*, 21(11):4544–4556, 2012. 6

A Deep Visual Correspondence Embedding Model for Stereo Matching Costs

Zhuoyuan Chen, Xun Sun, Liang Wang,
Baidu Research – Institute of Deep Learning
{chenzhuoyuan,sunxun,wangliang18}@baidu.com

Yinan Yu, Chang Huang
Horizon Robotics
{yinan.yu,chang.huang}@horizon-robotics.com *

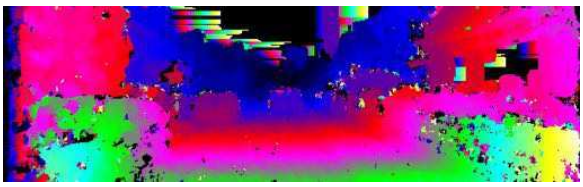
Abstract

This paper presents a data-driven matching cost for stereo matching. A novel deep visual correspondence embedding model is trained via Convolutional Neural Network on a large set of stereo images with ground truth disparities. This deep embedding model leverages appearance data to learn visual similarity relationships between corresponding image patches, and explicitly maps intensity values into an embedding feature space to measure pixel dissimilarities. Experimental results on KITTI and Middlebury data sets demonstrate the effectiveness of our model. First, we prove that the new measure of pixel dissimilarity outperforms traditional matching costs. Furthermore, when integrated with a global stereo framework, our method ranks top 3 among all two-frame algorithms on the KITTI benchmark. Finally, cross-validation results show that our model is able to make correct predictions for unseen data which are outside of its labeled training set.

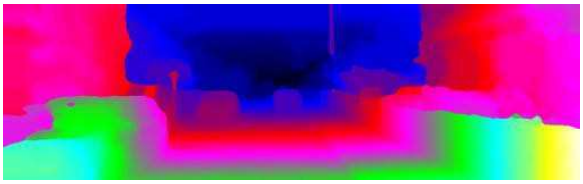
1. Introduction

Stereo matching infers scene geometry by establishing pixel correspondences across multiple images taken from different viewpoints. Scharstein and Szeliski in their seminal taxonomy work [26] argue that stereo algorithms essentially consist of four building blocks: matching cost computation, cost aggregation, disparity computation/optimization and refinement. According to the actual sequence of steps taken, stereo algorithms can be broadly categorized into local and global methods. Local algorithms, where the disparity computation at a pixel depends only on intensity values within a finite support region, usually make implicit smoothness assumptions by aggregating pixel-based matching costs. In contrast, global algorithms typically skip the cost aggregation step by making explicit smoothness

(a) Reference image of a stereo pair



(b) Winner-takes-all result



(c) Result from an MRF-based stereo framework

(d) color-coded depth

Figure 1. A demonstration of our framework: (a) a left image in the KITTI stereo sequences [13]; (b) we use Convolutional Neural Network to extract discriminative features for stereo matching, which is then refined by a global Markov Random Field to obtain the final result (c). Throughout our paper, we color-code all disparity maps as in (d): blue indicates far regions while white indicates near ones.

*This work was done when the fourth and fifth authors were with Baidu Research– Institute of Deep Learning.

assumptions and seek an optimal disparity assignment by solving an MRF (Markov Random Field) based energy optimization problem.

Matching cost computation, as a fundamental step shared by both local and global stereo algorithms, plays an important role in establishing visual correspondences. Typically, the reconstruction accuracy of a stereo method largely depends on the dissimilarity measurement of image patches. However, the visual correspondence search problem is difficult due to matching ambiguities, which generally results from sensor noise, image sampling, lighting variations, textureless or repetitive regions, occlusions, etc.

In this paper, we focus on the matching cost computation step and present a data-driven approach to address ambiguities. Inspired by the recent advances in deep learning, a new *deep visual correspondence embedding model* is trained via Convolutional Neural Network on a large set of stereo images with ground truth disparities. This deep embedding model leverages appearance data to learn visual dissimilarity between image patches, by explicitly mapping raw intensity into a rich embedding space. Quantitative evaluations with ground truth data demonstrate the effectiveness of our learned deep embedding model for dissimilarity computation. It is first proved that the proposed measure of pixel dissimilarity outperforms some widely used matching costs such as sampling-insensitive absolute differences [1], absolute differences of gradient, normalized cross-correlation and census transform [41] for local window-based matching. Furthermore, we incorporate our learning-based matching costs with a semi-global method (SGM) [16] and show that our model matches state-of-the-art performance on the KITTI stereo data set [13] for accuracy while being superior to other top performers for efficiency and simplicity. Lastly, our deep embedding model is evaluated on the Middlebury benchmark [25] to quantitatively assess its capability of generalization. Cross-validation results show that our model is able to make correct predictions across stereo images outside of its labeled training set.

1.1. Previous Work

Stereo has been a highly active research topic of computer vision for decades and this paper owes a lot to a sizable body of literature on stereo matching, more than we hope to account for here. For the scope of this paper, our focus is on the matching costs computation step and we refer interested readers to [18, 34, 26] for more detailed descriptions and evaluations of different components of stereo algorithms.

Common pixel-based matching costs include absolute differences (AD), squared differences (SD) and sampling-insensitive differences (BT) [1]. In practice, truncated versions of these methods are usually adopted because of the robustness brought about by the cost aggregation and global optimization steps. Common window-based matching costs include normalized cross-correlation (NCC), sum of absolute or squared differences (SAD/SSD), gradient-based measures, and non-parametric measures such as rank and census transforms [41]. Methods such as BT, SAD and SSD are built strictly on the brightness constancy assumption while gradient-based and non-parametric measures are more robust to radiometric changes (*e.g.*, due to gain and exposure differences) or non-Lambertian surfaces at the cost of lower discriminative power. Recently, self-adapting dissimilarity measures that combine SAD, Census and gradient-based measures are employed by some state-of-the-art stereo algorithms [19, 4, 39].

CNN (Convolutional Neural Networks) dates back decades in the machine learning community [21] and developed rapidly in recent years, thanks to the advances in learning models [23], larger data sets, and parallel computing devices, as well as the access to many online resources [6, 27]. CNN makes breakthroughs in various computer vision tasks such as image classification [20, 32], object detection [14, 10], face recognition [33, 30], and image parsing [11]. Besides its success in high-level vision applications, deep learning also shows its capability in solving certain low-level vision tasks such as super-resolution [7], denoising [2], and single-view depth estimation [8].

Our work is closely related to a recent paper [35], in which CNN is leveraged to compute stereo matching costs. In comparison to [35], our deep embedding model differs from [35] in two main aspects: 1) Given the feature vectors (corresponding to the left right patches in a stereo pair) output by CNN, we directly compute their similarity in the Euclidean space by a dot product. In contrast, the architecture in [35] is more complicated, in that feature vectors requires further fully-connected DNN (Deep Neural Network) to obtain the final similarity score in a non-Euclidean space. The architecture of our CNN leads to two orders of magnitude ($100\times$) speed-up in computing a dense disparity map compared to [35] with little sacrifice in reconstruction accuracy. 2) Our embedding model is learned from a multi-scale ensemble framework, which automatically fuses features vectors learned at different scale-space. Deep learning is also applied in [12, 22] for feature matching. However, [12] extracts features at sparse locations while [22] targets at matching semantically similar regions. Our model is also related to the Deep Flow model [37], where convolutional matching is applied to estimate optical flow. The difference is that [37] applies bottom-up deep dynamic programming on hand-crafted HOG features, while our model is data-driven and learned.

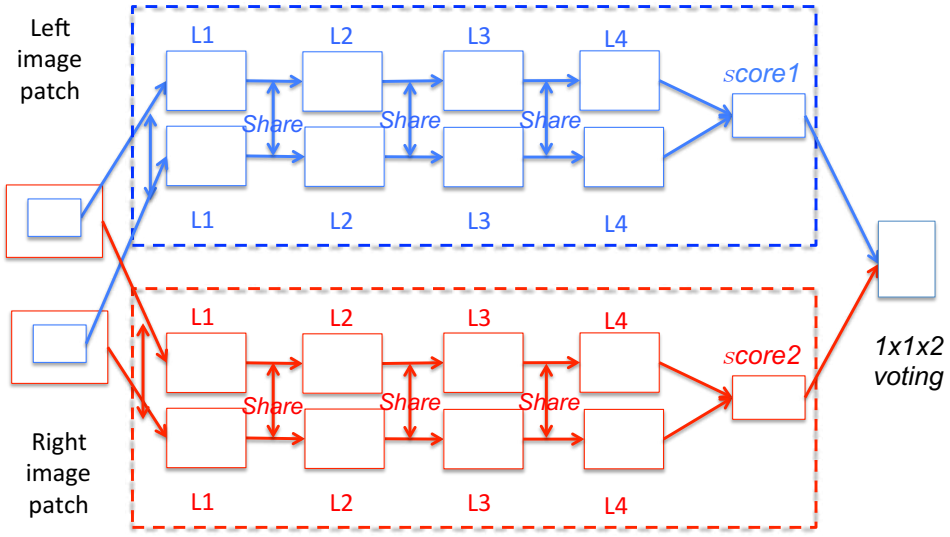


Figure 2. The network architecture of our training model for deep embedding. Features are extracted in a pair of patches at different scales, followed by an inner product to obtain the matching scores. The scores from different scales are then merged for an ensemble.

2. Deep Embedding for Stereo Estimation

Given a pair of rectified left/right images $\{I^L, I^R\}$, a typical stereo algorithm begins by computing the matching quality: denoting patches $I^L(\mathbf{p})$ and $I^R(\mathbf{p}-\mathbf{d})$ centered at $\mathbf{p}^L = (x, y)$ and $\mathbf{p}_d^R = (x-d, y)$, we generate matching score $S(\mathbf{p}, \mathbf{d}) = f(I^L(\mathbf{p}), I^R(\mathbf{p}-\mathbf{d}))$ for each $\mathbf{p}$ with different disparity $\mathbf{d} = (d, 0)$. The score serves as an important initialization, generally followed by a non-local refinement such as cost aggregation or filtering [24, 42]. In this section, we focus on our data-driven model to learn a good matching quality measure.

2.1. Multi-scale Deep Embedding Model

Consider a pair of patches $I^L(\mathbf{p})$, $I^R(\mathbf{p}-\mathbf{d})$ of size 13×13, we propose to learn an embedding model to extract features $f(I)$, such that the inner-product $S =< f(I^L(\mathbf{p})), f(I^R(\mathbf{p}-\mathbf{d})) >$ tends to be large in case of positive matching and small for negative ones. This differs from the binary classification model in [35].

Multi-Scale Embedding: we apply an ensemble model [11, 5, 8] in our deep embedding, which largely improves the matching quality. Denoting $I^L_\downarrow(\mathbf{p})$, $I^R_\downarrow(\mathbf{p}-\mathbf{d})$ as patches at the coarse scale, we have:

$$S(\mathbf{p}, \mathbf{d}) = w_1 < f(I^L(\mathbf{p}), f(I^R(\mathbf{p}-\mathbf{d})) > + \\ w_2 < f(I^L_\downarrow(\mathbf{p}), f(I^R_\downarrow(\mathbf{p}-\mathbf{d}) >$$

As we know, the choice of patch scale and size is very tricky: large patches with richer information are less ambiguous, but more risky of containing multiple objects and producing blurred boundaries; small patches have merits in motion details, but are very noisy. Accordingly, we propose a weighted ensemble of two scales to combine the best of two worlds.

Our embedding framework is shown in Figure 2. The inputs are two pairs of $13 \times 13 \times 1$ image patches, centered at $\mathbf{p} = (x, y)$, $\mathbf{p}-\mathbf{d} = (x-d, y)$ with two different scales. The blue-colored dash box indicates the original resolution while the red is a $\times 2$ down-sampling. We apply a 4-layer CNN model to extract features $f(I)$, followed by an inner-product layer to calculate the matching score $< f(I^L), f(I^R) >$ and an ensemble voting. Layer $L1$ and $L2$ contain $c_1 = c_2 = 32$ kernels of size 3×3; Layer $L3$ and $L4$ contain kernels $c_3 = c_4 = 200$ of size 5×5. At both scales, the weights in all layers $L1...L4$ for left and right patches are tied. Then, features $f_4(I) \in R^{200}$ undergo inner-product operations, outputting two scalars $S_1 =< f_4(I^L), f_4(I^R) >$ and $S_2 =< f_4(I^L_\downarrow), f_4(I^R_\downarrow) >$ as matching scores. Finally, S_1 and S_2 are merged by a $1 \times 1 \times 2$ convolutional layer for a weighted ensemble.

In our training process, we apply a deep regression model and minimize the Euclidean cost:

$$E(w) = ||S(\mathbf{p}, \mathbf{d}) - label(\mathbf{p}, \mathbf{d})||^2 \qquad (1)$$

where $label(\mathbf{p}, \mathbf{d}) = \{0, 1\}$ indicates whether $\mathbf{p}^L = (x, y)$ corresponds to $\mathbf{p}-\mathbf{d} = (x-d, y)$ in the right image. Rectified linear units [23] follow each layer, but we do not use pooling to preserve spatial-variance.

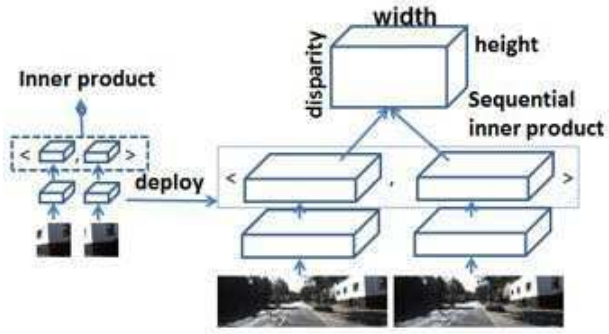


Figure 3. The deployed network architecture of our testing model for deep embedding. Features are extracted in two images **only once** respectively. Then, the sliding-window style inner product can be grouped for **matrix operation**.

2.2. Efficient Embedding for Testing

With deep embedding, we achieve $100\times$ speedup at test time compared to MC-CNN [35]. We attribute this to the largely shared computation in the forward pass as well as grouped matrix operation.

The insight is shown in Figure 3, we extract features $f_4(I)$ in each image **only once** with a fully convolutional neural network, and then the matching score $S(\mathbf{p}, \mathbf{d})$ is computed with all potential offsets $\mathbf{d}$ by the sliding-window style inner product. In comparison, in [35] the fully-connected layers require recomputation every time when d is varied.

Moreover, multiple inner product operations can be **grouped** together as a matrix operation for further acceleration. That is, we multiply $f(I^L(\mathbf{p}))$ with the matrix $F(I^R)$, whose columns contain features $f(I^R(\mathbf{p}-\mathbf{d}))$ with different d. This matrix multiplication is highly parallel in nature.

2.3. Training Details

A training example comprises two pairs of patches $\{I^L(\mathbf{p}), I^R(\mathbf{p}-\mathbf{d})\}, \{I^L_\downarrow(\mathbf{p}), I^R_\downarrow(\mathbf{p}-\mathbf{d})\}$. We sample positive and negative examples at ratio $1:1$ at each location where the disparity $\mathbf{d}$ is known. A negative example is obtained by shifting the right patch to $\mathbf{p_d} = (x-d+o_{neg}, y)$, where $o_{neg} \in \{-N_{hi}, ..., -N_{lo}, N_{lo}, ..., N_{hi}\}$ is a random corrupting offset. Similarly, positive examples are $\mathbf{p_d} = (x-d+o_{pos}, y)$, with $o_{pos} \in \{-P_{hi}, P_{hi}\}$.

In practice, we find that a warm start with large N_{lo}, N_{hi} makes the training converges faster. We gradually decrease N_{lo}, N_{hi} to pursue a hard negative mining.

3. Stereo Framework

To calculate the final disparity map using our proposed matching costs, we adopt an MRF-based stereo framework. First of all, a cost volume is initialized by negating our deep embedding scores as $C(\mathbf{p}, d) = -S(\mathbf{p}, p_d)$.

Secondly, the initial costs are then fed to the semi-global matcher (SGM) [16, 39] to compute a raw disparity map. Thirdly, after removing unreliable matches via a left-right check, the final disparity map is obtained by propagating reliable disparities to non-reliable areas [29]. In the following, we will briefly summarize the key building blocks in our framework.

Formulating the stereo problem as an MRF model, the SGM aims to find a disparity assignment D that minimizes:

$$E(D) = \sum_p (C(\mathbf{p}, d) + \sum_{q \in N(p)} P_1[|d - D_q| = 1] + \sum_{q \in N(p)} P_2[|d - D_q| > 1]) \tag{2}$$

The constant parameters P_1 and P_2 penalize disparity discontinuity and $P_1 < P_2$. While an exact solution to equation (2) is NP-hard, semi-global matcher obtains an approximation by performing multiple dynamic programming (DP)-style $1D$ cost updates along multiple directions as

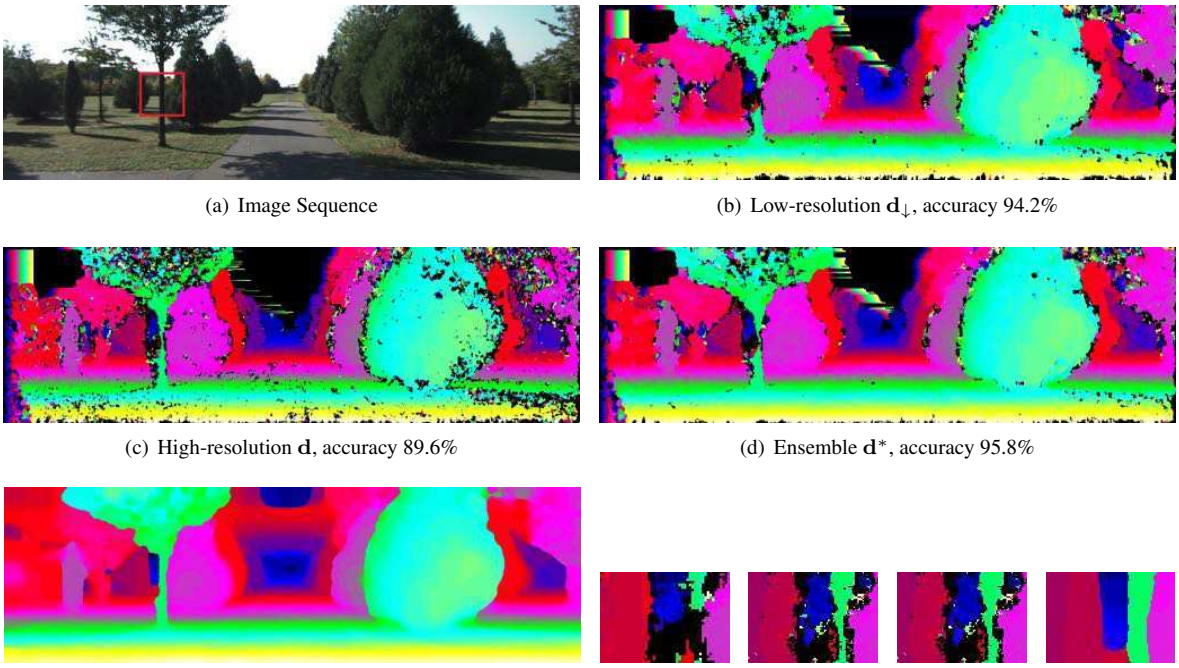
$$\begin{aligned} L_{\mathbf{r}}(\mathbf{p}, d) = & C(\mathbf{p}, d) + \min(L_{\mathbf{r}}(\mathbf{p}-\mathbf{r}, d), \\ & L_r(\mathbf{p}-\mathbf{r}, d \pm 1) + P_1, \\ & \min_i(L_r(\mathbf{p}-\mathbf{r}, i) + P_2)) \end{aligned} \tag{3}$$

where $L_{\mathbf{r}}(p, d)$ is the cost along a direction $\mathbf{r}$. After averaging the costs along 16 directions, the initial disparity map is computed for left and right views by picking the disparity hypothesis with the minimal cost.

To efficiently remove the remaining outliers in the initial disparity map, we adopt a propagation scheme [29, 40]. We start by performing a left-right consistency check to roughly divide all pixels into stable and unstable sets. Assuming that a stable pixel $\mathbf{p}$ should satisfy $D^{Left}(\mathbf{p}) = D^{Right}(\mathbf{p}-d)$, we carry out a propagation on the cost volume as:

$$C(\mathbf{p}, d)_{pro} = \begin{cases} |d - D^{Left}_{Raw}(\mathbf{p})|^2 & \mathbf{p} \text{ is stable,} \\ 0 & \text{else.} \end{cases} \tag{4}$$

Here for stable pixels, we penalize disparity deviation from its initial disparity values. In contrast, for unstable pixels, the costs are set to zero for all disparity hypotheses. Thus, an edge-aware filter applied on this cost volume leads to reliable disparity propagation in the cost domain. We choose the $O(1)$ geodesic filter derived from a tree structure [29] due to its effectiveness and efficiency. After filtering the propagated cost volume $C(\mathbf{p}, d)_{pro}$, we again choose the disparity assignment using winner-takes-all. Finally, we employ a 3×3 median filter to remove remaining isolated noises.



(a) Image Sequence (b) Low-resolution $\mathbf{d}_{\downarrow}$, accuracy 94.2%

(c) High-resolution $\mathbf{d}$, accuracy 89.6% (d) Ensemble $\mathbf{d}^*$, accuracy 95.8%

(e) Refined result of $\mathbf{d}^*$ (Section 3), accuracy 98.5% (f) (g) (h) (i)

Figure 4. An example of how the proposed multi-scale ensemble works. (a) Frame 10 in KITTI [13]; (b,c) winner-takes-all results of embedding at the coarse and fine scales, respectively; (d) ensemble of (b,c) gives smooth results while preserving details at the same time; (f,g,h,i) are close-up views of (b,c,d,e), respectively.

4. Experimental Results

In this section, we evaluate our algorithm extensively by comparing its performance with other state-of-the-art methods. Our matching costs computation is implemented in CUDA. And the subsequent stereo framework is implemented on a PC equipped with a 3.0 GHz Intel i5 CPU. In our stereo framework, we fix the parameter settings throughout our experiments: $\{P_1, P_2, \sigma_s, \sigma_r\} = \{5, 80, 20, 10.5\}$. Here σ_s and σ_r are the spatial and range kernel parameters used in geodesic filters, respectively.

We use off-the-shelf resource Caffe [6] to train our CNN with a Nvidia GeForce GTX Titan. We optimize the Euclidean loss by a stochastic gradient descent with momentum and weight decay [31]. Specially, we preprocess each image by subtracting the mean and dividing by the standard deviation of its pixel intensity values. It takes about 3 hours to train the CNN with KITTI training set.

4.1. Matching Cost Evaluation

We start by comparing our learning-based matching costs against traditional pixel and window-based matching costs, including BT (Birchfield and Tomasi) [1], census transform [41], AD+Gradient (combination of absolute differences and gradient-based measures), Census+Gradient (combination of census transform and gradient-based measures) [39], and NCC. Among these selected counterparts, BT and AD+Gradient are popular choices by many competitive stereo algorithms on the Middlebury evaluation system [19, 24]; census transform is reported as the overall best matching cost function in Hirschmuller and Scharstein's survey [18]; Census+Gradient is used by one of the top performers on the outdoor KITTI data sets [39]; NCC is a classic patch-based measure that is statistically optimal for compensating Gaussian noise [18].

For AD+Gradient and Census+Gradient, we follow the parameter settings as reported in the original literature [19, 39], census transform is computed with a 5×5 local support region. Similar to [18], we use a square window to aggregate the matching costs and calculate the per-pixel disparity value via winner-takes-all. The window size is chosen to be 13×13 because our deep embedding model is learned on the same size. NCC does not go through cost aggregation with the patch width set to 13. Note that no disparity refinement step is applied in this experiment because we want the raw results to provide a more direct assessment of different methods.

We report performance on the 194 KITTI training images in Table 1. As can be seen, our learned matching costs from the deep embedding model substantially outperform traditional matching costs.

Validation of Multi-scale Ensemble: Close scrutiny reveals the complementary power of the embedding results

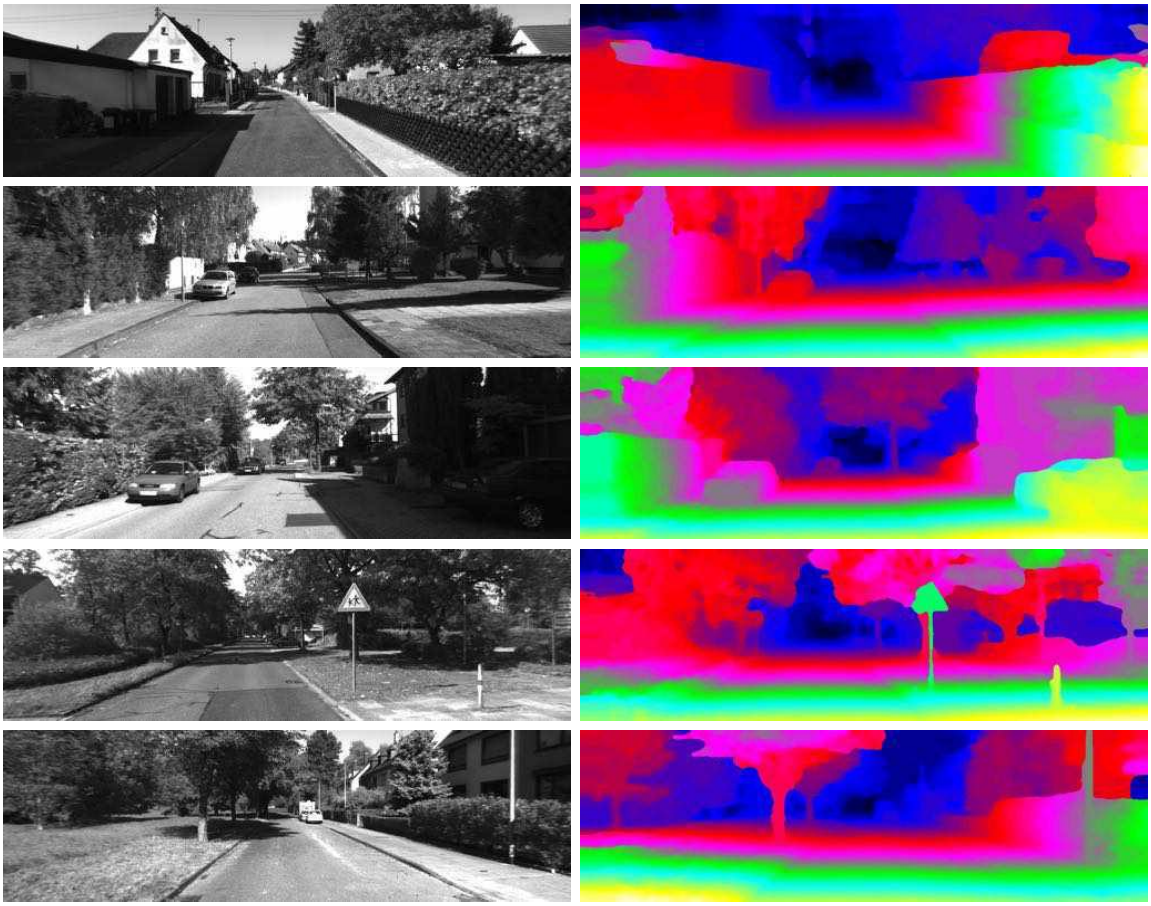


Figure 5. Some disparity map results using our embedding cost on KITTI benchmark. **The left column:** grayscale images of the left frames; **The right column:** our final results. **From top to bottom:** different frames in the KITTI data set.

Methods	All err(%)	Non-occ err(%)
Multi-scale	**11.1**	**9.5**
Single-scale (Coarse)	14.8	12.7
Single-scale (Fine)	17.4	14.6
BT	37.8	36.6
Census	25.1	23.4
AD+Gradient	37.4	36.1
Census+Gradient	23.5	21.7
NCC	21.8	19.9

Table 1. Quantitative evaluation of different costs with local-window matching. We compare errors of winner-takes-all results obtained by classical costs with our deep embedding model (single-scale and the multi-scale ensembles) in *all* and *non-occluded* regions.

at different scales (e.g., the 10-th frame in KITTI training set). As shown in Figure 4, the embedding at low-resolution tends to generate more smooth results, but is at risk of losing motion details. Thin structures like the tree marked in box in (a) are missing in (b,f), where the lost motion proposal can hardly be recovered by post-processing. Meanwhile, noises in high-resolution results (c) can be suppressed by accounting for larger contexts. The accuracy of ensemble (95.8%) is higher than any single scale and a post-processing can produce satisfactory stereo results.

Quantitatively, the average accuracy of winner-takes-all results produced by the fine-scale, coarse-scale and multi-scale ensemble are 85.4%, 87.3% and 90.5% on the KITTI train set, respectively.

4.2. Stereo Pipeline Evaluation

In this section, we incorporate our matching costs with the stereo framework introduced in section 3. The quantitative evaluation results on KITTI test set are shown in table 2, where we compare our model with state-of-the-art methods with only two stereo images for fairness (algorithms that leverage multi-view or temporal/scene flow cues are beyond the scope of this paper). Our method ranks the $3rd$ among these algorithms in terms of accuracy as of September 2015, and is over an order of magnitude faster than [15] and [35]. Note that replacing our deep embedding costs

Method	**Out-Noc**	**Out-All**	**Avg-Noc**	**Avg-All**	**Runtime**	Environment
Displets [15]	2.47 %	3.27 %	0.7 px	0.9 px	265s	8+ cores @ 3.0 Ghz (Matlab + C/C++)
MC-CNN [35]	2.61 %	3.84 %	0.8 px	1.0 px	100s	Nvidia GTX Titan (CUDA, Lua/Torch7)
Ours	3.10 %	4.24 %	0.9 px	1.1 px	3.0s	Nvidia GTX Titan (CUDA, Caffe)
Census+Gradient	3.91 %	5.10 %	0.9 px	1.0 px	2.5s	single core @ 3.0Ghz GPU
CoR[3]	3.30 %	4.10 %	0.8 px	0.9 px	6s	6 cores @ 3.3 Ghz (Matlab + C/C++)
SPS-St[39]	3.39%	4.41%	0.9 px	1.0 px	2s	1 core @3.5 Ghz (C/C++)
DDS-SS[36]	3.83 %	4.59 %	0.9 px	1.0 px	1 min	1 core @ 2.5 Ghz (Matlab + C/C++)
PCBP[38]	4.04 %	5.37 %	0.9 px	1.1 px	5 min	4 cores @ 2.5 Ghz (Matlab + C/C++)
CoR-Conf[3]	4.49 %	5.26 %	1.0 px	1.2 px	6 s	6 cores @ 3.3 Ghz (Matlab + C/C++)
AARBM[9]	4.86 %	5.94 %	1.0 px	1.2 px	0.25 s	1 core @ 3.0 Ghz (C/C++)
wSGM [28]	4.97 %	6.18 %	1.3 px	1.6 px	6s	1 core @ 3.5 Ghz (C/C++)

Table 2. Qualitative evaluation of our pipeline on KITTI benchmark with the error threshold set as 3 pixels. The method "Census+Gradient" uses the same stereo framework, but with Census+Gradient cost function rather than our proposed learning-based matching costs. In terms of runtime, we would like to emphasize that the numbers given in this table are total running time of the whole stereo framework. As reported in the paper [35], the MC-CNN algorithm takes about 95 seconds to compute their learning-based matching costs and 5 seconds for semi-global matching and disparity post-processing steps. In contrast, the running time for our matching cost computation is about 1.0 second, which is around 100 times faster than [35].

with Census+Gradient [39] produces inferior results. A few disparity maps generated by our method are presented in Figure 5, from which we can see that our algorithm produces piecewise smooth and visually plausible results. Our method not only preserves geometry details near depth discontinuities, but also performs well on challenging regions such as textureless and shadow areas.

Regarding the running time, the CNN step takes about 1.0s on average, 734ms at the original resolution and 267ms at the coarse resolution. This is about $100\times$ speedup compared to MC-CNN [35]. The acceleration factor mainly results from fewer model parameters (116,000 versus 600,000 in [35]), fully-convolutional architecture and grouped matrix operation. The not fully optimized MRF stereo implementation takes about 2s on CPU. In total, our approach takes about 3s to process an image at the resolution of 1242×376 with a large disparity searching range from 0 to 255.

4.3. Cross-Validation

It is of interest to evaluate how our model performs on different datasets. Besides KITTI (outdoor street view), we further test our deep embedding model on indoor scenarios without any fine-tuning. We choose stereo sequences from Middlebury benchmarks [25, 17], containing a total number of 27 high resolution image pairs.

Quantitatively, we compare our algorithm with traditional costs such as AD, census transform and NCC. We evaluate the performance with varying error thresholds τ in Figure 6: similar to the setup in section 4.1, we carry out a box aggregation on raw costs and obtain final disparities with a winner-takes-all treatment. It is clear that our embedding model consistently outperforms other methods. Especially, with $\tau = 3$, our method achieves an average error rate of 13.6%, which is much lower than SAD (33.8%), census transform (22.2%), and NCC (21.9%). This demonstrates that our deep embedding model generalizes well on scenes that are sufficiently different from the training data. We provide more qualitative results in Figure 7.

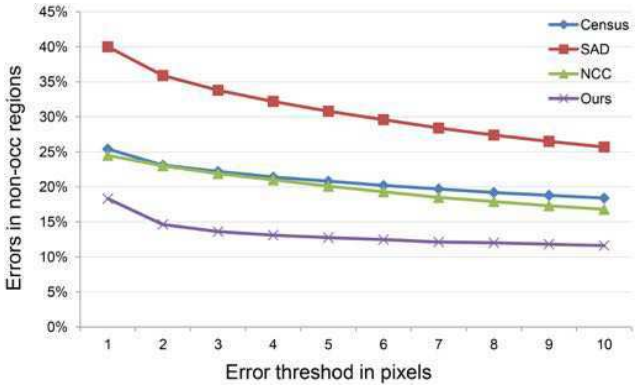


Figure 6. The error curve on Middlebury benchmark with varying error thresholds.

5. Conclusion and Future Work

In this paper, we introduce a novel data-driven patch dissimilarity measure for visual correspondence. We learn a deep embedding model to extract discriminative features from patches in stereo pairs. Our model has fewer parameters and shallower network structures therefore is much more efficient than a previous learning-based model [35]. Deep embedding produces high-quality initialization, which can be refined with an MRF-based stereo algorithm to obtain the state-of-the-art dense disparity estimates. Our results on KITTI [13] and Middlebury [26] benchmarks

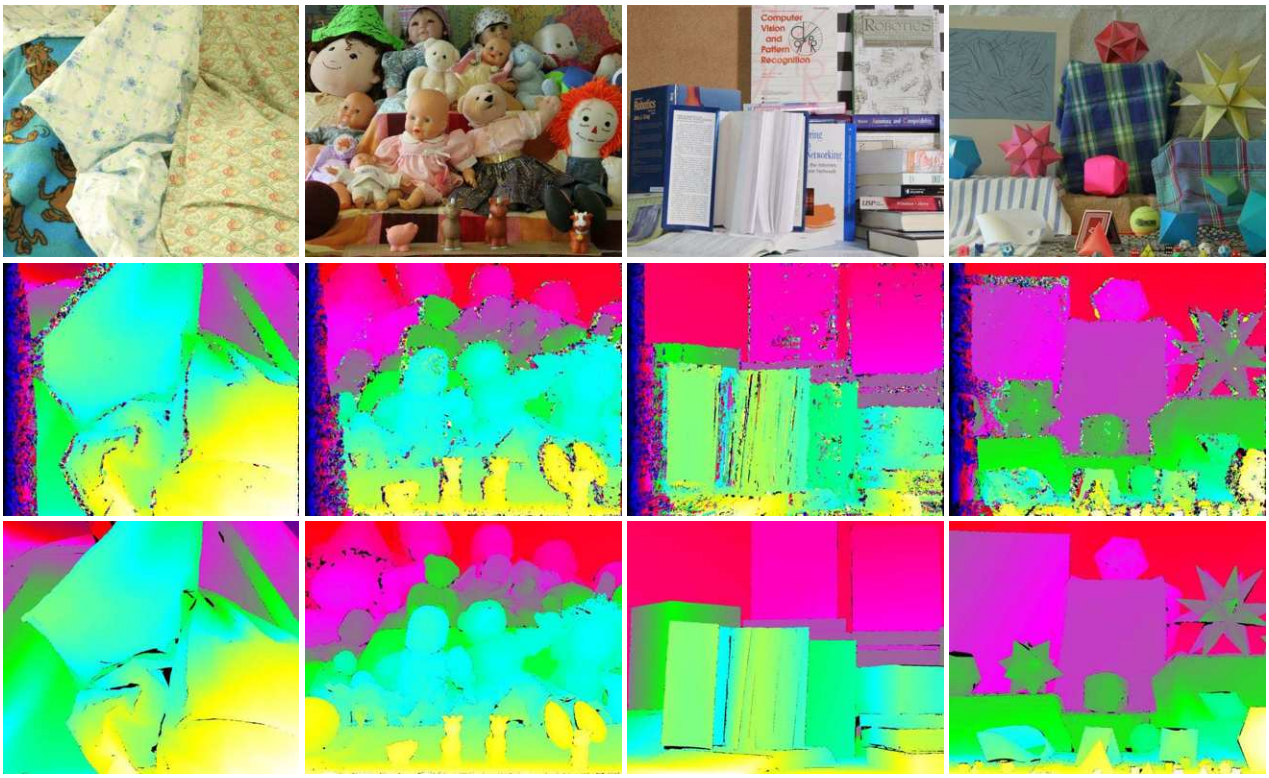


Figure 7. Examples of winner-takes-all results using our deep embedding cost on Middlebury data. **From left to right:** the Cloth, Dolls, Books and Moebius Sequences in Middlebury benchmark; **From top to bottom:** the left images, winner-takes-all results of our deep embedding and ground truth.

suggest that deep embedding is robust across stereo images taken from different environment. In the future, training with larger data sets might have potential to further improve the matching accuracy. We will also accelerate the algorithm for real-time navigation and extend our model to solve other low-level vision problems such as optical flow.

6. Acknowledgements

We would like to thank Haoyuan Gao and Fei Qing for their efforts in data collection. We would also like to thank Kai Yu and Carl Case for discussion and helpful advice.

References

[1] S. Birchfield and C. Tomasi. A pixel dissimilarity measure that is insensitive to image sampling. *PAMI*, 20:401–406, 1998.

[2] H. C. Burger, C. J. Schuler, and S. Harmeling. Image denoising: Can plain neural networks compete with bm3d? *CVPR*, pages 2392–2399, 2012.

[3] A. Chakrabarti, Y. Xiong, S. J. Gortler, and T. Zickler. Low-level vision by consensus in a spatial hierarchy of regions. *CVPR*, 2015.

[4] Z. Chen, H. Jin, Z. Lin, S. Cohen, and Y. Wu. Large displacement optical flow from nearest neighbor fields. *CVPR*, 2013.

[5] D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. *CVPR*, pages 3642–3649, 2012.

[6] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *ICML*, 2014.

[7] C. Dong, C. C. Loy, K. He, and X. Tang. Learning a deep convolutional network for image super-resolution. *ECCV*, 2014.

[8] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. *NIPS*, pages 2366–2374, 2014.

[9] N. Einecke and J. Eggert. Block-matching stereo with relaxed fronto-parallel assumption. In *In Intelligent Vehicles Symposium Proceedings, 2014 IEEE*, pages 700–705, 2014.

[10] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. Scalable object detection using deep neural networks. *CVPR*, 2014.

[11] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *PAMI*, 35(8):1915–1929, 2013.

[12] P. Fischer, A. Dosovitskiy, and T. Brox. Descriptor matching with convolutional neural networks: a comparison to sift. *Technical Report 1405.5769, arXiv(1405.5769)*, 2014.

[13] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, 2012.

[14] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CVPR*, 2014.

[15] F. Güney and A. Geiger. Displets: Resolving stereo ambiguities using object knowledge. In *CVPR*, 2015.

[16] H. Hirschmüller. Stereo processing by semiglobal matching and mutual information. *PAMI*, 30(2):328–341, 2008.

[17] H. Hirschmuller and D. Scharstein. Evaluation of cost functions for stereo matching. *CVPR*, 2007.

[18] H. Hirschmuller and D. Scharstein. Evaluation of stereo matching costs on images with radiometric differences. *PAMI*, 31:1582–1599, 2009.

[19] A. Klaus, M. Sormann, and K. Karner. Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure. *ICPR*, 2006.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS*, 2012.

[21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[22] J. Long, N. Zhang, and T. Darrell. Do convnets learn correspondence? *NIPS*, 2014.

[23] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. *ICML*, 2010.

[24] C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz. Fast cost-volume filtering for visual correspondence and beyond. In *CVPR*, 2011.

[25] D. Scharstein and C. Pal. Learning conditional random fields for stereo. *CVPR*, 2007.

[26] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *IJCV*, 47(1-3):7–42, 2002.

[27] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[28] R. Spangenberg, T. Langner, and R. Rojas. Weighted semi-global matching and center-symmetric census transform for robust driver assistance. In *In Computer Analysis of Images and Patterns*, pages 34–41, 2013.

[29] X. Sun, X. Mei, S. Jiao, M. Zhou, Z. Liu, and H. Wang. Real-time local stereo via edge-aware disparity propagation. *PRL*, 49:201–206, 2014.

[30] Y. Sun, X. Wang, and X. Tang. Deep learning face representation from predicting 10,000 classes. *CVPR*, 2014.

[31] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. *ICML*, 2013.

[32] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv:1409.4842*, 2014.

[33] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. *CVPR*, 2014.

[34] F. Tombari, S. Mattoccia, L. D. Stefano, and E. Addimanda. Classification and evaluation of cost aggregation methods for stereo correspondence. *CVPR*, 2008.

[35] J. Žbontar and Y. LeCun. Computing the stereo matching cost with a convolutional neural network, 2014.

[36] D. Wei, C. Liu, and W. T. Freeman. A data-driven regularization model for stereo and flow. *In 3D Vision (3DV), 2014 2nd International Conference on*, 1:277–284, 2014.

[37] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid. Deepflow: Large displacement optical flow with deep matching. *ICCV*, pages 1385–1392, 2013.

[38] K. Yamaguchi, T. Hazan, D. McAllester, and R. Urtasun. Continuous markov random fields for robust stereo estimation. In *ECCV*, pages 45–58, 2012.

[39] K. Yamaguchi, D. McAllester, and R. Urtasun. Efficient joint segmentation, occlusion labeling, stereo and flow estimation. *ECCV*, 2013.

[40] Q. Yang. A non-local cost aggregation method for stereo matching. In *CVPR*, 2012.

[41] R. Zabih and J. Woodfill. Non-parametric local transforms for computing visual correspondence. *ECCV*, 1994.

[42] K. Zhang, J. Lu, and G. Lafruit. Cross-based local stereo matching using orthogonal integral images. *IEEE Transactions on Circuits and Systems for Video Technology*, 19(7):1073–1079, 2009.

Conditional Random Fields as Recurrent Neural Networks

Shuai Zheng*[1], Sadeep Jayasumana*[1], Bernardino Romera-Paredes[1], Vibhav Vineet†[1,2], Zhizhong Su[3], Dalong Du[3], Chang Huang[3], and Philip H. S. Torr[1]

[1]University of Oxford [2]Stanford University [3]Baidu Institute of Deep Learning

Abstract

Pixel-level labelling tasks, such as semantic segmentation, play a central role in image understanding. Recent approaches have attempted to harness the capabilities of deep learning techniques for image recognition to tackle pixel-level labelling tasks. One central issue in this methodology is the limited capacity of deep learning techniques to delineate visual objects. To solve this problem, we introduce a new form of convolutional neural network that combines the strengths of Convolutional Neural Networks (CNNs) and Conditional Random Fields (CRFs)-based probabilistic graphical modelling. To this end, we formulate mean-field approximate inference for the Conditional Random Fields with Gaussian pairwise potentials as Recurrent Neural Networks. This network, called CRF-RNN, is then plugged in as a part of a CNN to obtain a deep network that has desirable properties of both CNNs and CRFs. Importantly, our system fully integrates CRF modelling with CNNs, making it possible to train the whole deep network end-to-end with the usual back-propagation algorithm, avoiding offline post-processing methods for object delineation.

We apply the proposed method to the problem of semantic image segmentation, obtaining top results on the challenging Pascal VOC 2012 segmentation benchmark.

1. Introduction

Low-level computer vision problems such as semantic image segmentation or depth estimation often involve assigning a label to each pixel in an image. While the feature representation used to classify individual pixels plays an important role in this task, it is similarly important to consider factors such as image edges, appearance consistency and spatial consistency while assigning labels in order to obtain accurate and precise results.

Designing a strong feature representation is a key challenge in pixel-level labelling problems. Work on this topic includes: TextonBoost [52], TextonForest [51], and Random Forest-based classifiers [50]. Recently, supervised deep learning approaches such as large-scale deep Convolutional Neural Networks (CNNs) have been immensely successful in many high-level computer vision tasks such as image recognition [31] and object detection [20]. This motivates exploring the use of CNNs for pixel-level labelling problems. The key insight is to learn a strong feature representation end-to-end for the pixel-level labelling task instead of hand-crafting features with heuristic parameter tuning. In fact, a number of recent approaches including the particularly interesting works FCN [37] and DeepLab [10] have shown a significant accuracy boost by adapting state-of-the-art CNN based image classifiers to the semantic segmentation problem.

However, there are significant challenges in adapting CNNs designed for high level computer vision tasks such as object recognition to pixel-level labelling tasks. Firstly, traditional CNNs have convolutional filters with large receptive fields and hence produce coarse outputs when restructured to produce pixel-level labels [37]. Presence of max-pooling layers in CNNs further reduces the chance of getting a fine segmentation output [10]. This, for instance, can result in non-sharp boundaries and blob-like shapes in semantic segmentation tasks. Secondly, CNNs lack smoothness constraints that encourage label agreement between similar pixels, and spatial and appearance consistency of the labelling output. Lack of such smoothness constraints can result in poor object delineation and small spurious regions in the segmentation output [59, 58, 32, 39].

On a separate track to the progress of deep learning techniques, probabilistic graphical models have been developed as effective methods to enhance the accuracy of pixel-level labelling tasks. In particular, Markov Random Fields (MRFs) and its variant Conditional Random Fields (CRFs) have observed widespread success in this area [32, 29] and have become one of the most successful graphical models used in computer vision. The key idea of CRF inference for semantic labelling is to formulate the label assignment

*Authors contributed equally.

†Work conducted while authors at the University of Oxford.

problem as a probabilistic inference problem that incorporates assumptions such as the label agreement between similar pixels. CRF inference is able to refine weak and coarse pixel-level label predictions to produce sharp boundaries and fine-grained segmentations. Therefore, intuitively, CRFs can be used to overcome the drawbacks in utilizing CNNs for pixel-level labelling tasks.

One way to utilize CRFs to improve the semantic labelling results produced by a CNN is to apply CRF inference as a post-processing step disconnected from the training of the CNN [10]. Arguably, this does not fully harness the strength of CRFs since it is not integrated with the deep network. In this setup, the deep network is unaware of the CRF during the training phase.

In this paper, we propose an end-to-end deep learning solution for the pixel-level semantic image segmentation problem. Our formulation combines the strengths of both CNNs and CRF based graphical models in one unified framework. More specifically, we formulate mean-field approximate inference for the dense CRF with Gaussian pairwise potentials as a Recurrent Neural Network (RNN) which can refine coarse outputs from a traditional CNN in the forward pass, while passing error differentials back to the CNN during training. Importantly, with our formulation, the whole deep network, which comprises a traditional CNN and an RNN for CRF inference, can be trained end-to-end utilizing the usual back-propagation algorithm.

Arguably, when properly trained, the proposed network should outperform a system where CRF inference is applied as a post-processing method on independent pixel-level predictions produced by a pre-trained CNN. Our experimental evaluation confirms that this indeed is the case. We evaluate the performance of our network on the popular Pascal VOC 2012 benchmark, achieving a new state-of-the-art accuracy of 74.7%.

2. Related Work

In this section we review approaches that make use of deep learning and CNNs for low-level computer vision tasks, with a focus on semantic image segmentation. A wide variety of approaches have been proposed to tackle the semantic image segmentation task using deep learning. These approaches can be categorized into two main strategies.

The first strategy is based on utilizing separate mechanisms for feature extraction, and image segmentation exploiting the edges of the image [2, 38]. One representative instance of this scheme is the application of a CNN for the extraction of meaningful features, and using superpixels to account for the structural pattern of the image. Two representative examples are [19, 38], where the authors first obtained superpixels from the image and then used a feature extraction process on each of them. The main disadvantage of this strategy is that errors in the initial proposals (e.g: super-pixels) may lead to poor predictions, no matter how good the feature extraction process is. Pinheiro and Collobert [46] employed an RNN to model the spatial dependencies during scene parsing. In contrast to their approach, we show that a typical graphical model such as a CRF can be formulated as an RNN to form a part of a deep network, to perform end-to-end training combined with a CNN.

The second strategy is to directly learn a nonlinear model from the images to the label map. This, for example, was shown in [17], where the authors replaced the last fully connected layers of a CNN by convolutional layers to keep spatial information. An important contribution in this direction is [37], where Long *et al*. used the concept of fully convolutional networks, and the notion that top layers obtain meaningful features for object recognition whereas low layers keep information about the structure of the image, such as edges. In their work, connections from early layers to later layers were used to combine these cues. Bell *et al*. [5] and Chen *et al*. [10, 41] used a CRF to refine segmentation results obtained from a CNN. Bell *et al*. focused on material recognition and segmentation, whereas Chen *et al*. reported very significant improvements on semantic image segmentation. In contrast to these works, which employed CRF inference as a standalone post-processing step disconnected from the CNN training, our approach is an end-to-end trainable network that jointly learns the parameters of the CNN and the CRF in one unified deep network.

Works that use neural networks to predict structured output are found in different domains. For example, Do *et al*. [14] proposed an approach to combine deep neural networks and Markov networks for sequence labeling tasks. Jain *et al*. [26] has shown Convolutional Neural Networks can perform well like MRFs/CRFs approaches in image restoration application. Another domain which benefits from the combination of CNNs and structured loss is handwriting recognition. In natural language processing, Yao *et al*. [60] shows that the performance of an RNN-based words tagger can be significantly improved by incorporating elements of the CRF model. In [6], the authors combined a CNN with Hidden Markov Models for that purpose, whereas more recently, Peng *et al*. [45] used a modified version of CRFs. Related to this line of works, in [25] a joint CNN and CRF model was used for text recognition on natural images. Tompson *et al*. [57] showed the use of joint training of a CNN and an MRF for human pose estimation, while Chen *et al*. [11] focused on the image classification problem with a similar approach. Another prominent work is [21], in which the authors express deformable part models, a kind of MRF, as a layer in a neural network. In our approach, we cast a different graphical model as a neural network layer.

A number of approaches have been proposed for automatic learning of graphical model parameters and joint

training of classifiers and graphical models. Barbu *et al*. [4] proposed a joint training of a MRF/CRF model together with an inference algorithm in their Active Random Field approach. Domke [15] advocated back-propagation based parameter optimization in graphical models when approximate inference methods such as mean-field and belief propagation are used. This idea was utilized in [28], where a binary dense CRF was used for human pose estimation. Similarly, Ross *et al*. [47] and Stoyanov *et al*. [54] showed how back-propagation through belief propagation can be used to optimize model parameters. Ross *et al*. [21], in particular proposes an approach based on learning messages. Many of these ideas can be traced back to [55], which proposes unrolling message passing algorithms as simpler operations that could be performed within a CNN. In a different setup, Krähenbühl and Koltun [30] demonstrated automatic parameter tuning of dense CRF when a modified mean-field algorithm is used for inference. An alternative inference approach for dense CRF, not based on mean-field, is proposed in [61].

In contrast to the works described above, our approach shows that it is possible to formulate dense CRF as an RNN so that one can form an end-to-end trainable system for semantic image segmentation which combines the strengths of deep learning and graphical modelling.

After our initial publication of the technical report of this work on arXiv.org, a number of independent works [49, 35] appeared on arXiv.org presenting similar joint training approaches for semantic image segmentation.

3. Conditional Random Fields

In this section we provide a brief overview of Conditional Random Fields (CRF) for pixel-wise labelling and introduce the notation used in the paper. A CRF, used in the context of pixel-wise label prediction, models pixel labels as random variables that form a Markov Random Field (MRF) when conditioned upon a global observation. The global observation is usually taken to be the image.

Let X_i be the random variable associated to pixel i, which represents the label assigned to the pixel i and can take any value from a pre-defined set of labels $\mathcal{L} = \{l_1, l_2, \ldots, l_L\}$. Let $\mathbf{X}$ be the vector formed by the random variables $X_1, X_2, \ldots, X_N$, where N is the number of pixels in the image. Given a graph $G = (V, E)$, where $V = \{X_1, X_2, \ldots, X_N\}$, and a global observation (image) $\mathbf{I}$, the pair $(\mathbf{I}, \mathbf{X})$ can be modelled as a CRF characterized by a Gibbs distribution of the form $P(\mathbf{X} = \mathbf{x}|\mathbf{I}) = \frac{1}{Z(\mathbf{I})} \exp(-E(\mathbf{x}|\mathbf{I}))$. Here $E(\mathbf{x})$ is called the energy of the configuration $\mathbf{x} \in \mathcal{L}^N$ and $Z(\mathbf{I})$ is the partition function [33]. From now on, we drop the conditioning on $\mathbf{I}$ in the notation for convenience.

In the fully connected pairwise CRF model of [29], the energy of a label assignment $\mathbf{x}$ is given by:

$$E(\mathbf{x}) = \sum_i \psi_u(x_i) + \sum_{i<j} \psi_p(x_i, x_j), \qquad (1)$$

where the unary energy components $\psi_u(x_i)$ measure the inverse likelihood (and therefore, the cost) of the pixel i taking the label x_i, and pairwise energy components $\psi_p(x_i, x_j)$ measure the cost of assigning labels x_i, x_j to pixels i, j simultaneously. In our model, unary energies are obtained from a CNN, which, roughly speaking, predicts labels for pixels without considering the smoothness and the consistency of the label assignments. The pairwise energies provide an image data-dependent smoothing term that encourages assigning similar labels to pixels with similar properties. As was done in [29], we model pairwise potentials as weighted Gaussians:

$$\psi_p(x_i, x_j) = \mu(x_i, x_j) \sum_{m=1}^{M} w^{(m)} k_G^{(m)}(\mathbf{f}_i, \mathbf{f}_j), \qquad (2)$$

where each $k_G^{(m)}$ for $m = 1, \ldots, M$, is a Gaussian kernel applied on feature vectors. The feature vector of pixel i, denoted by $\mathbf{f}_i$, is derived from image features such as spatial location and RGB values [29]. We use the same features as in[29]. The function $\mu(.,.)$, called the label compatibility function, captures the compatibility between different pairs of labels as the name implies.

Minimizing the above CRF energy $E(\mathbf{x})$ yields the most probable label assignment $\mathbf{x}$ for the given image. Since this exact minimization is intractable, a mean-field approximation to the CRF distribution is used for approximate maximum posterior marginal inference. It consists in approximating the CRF distribution $P(\mathbf{X})$ by a simpler distribution $Q(\mathbf{X})$, which can be written as the product of independent marginal distributions, i.e., $Q(\mathbf{X}) = \prod_i Q_i(X_i)$. The steps of the iterative algorithm for approximate mean-field inference and its reformulation as an RNN are discussed next.

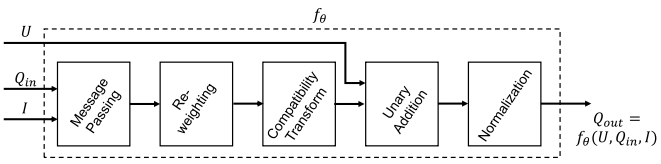


Figure 1. **A mean-field iteration as a CNN.** A single iteration of the mean-field algorithm can be modelled as a stack of common CNN layers.

4. A Mean-field Iteration as a Stack of CNN Layers

A key contribution of this paper is to show that the mean-field CRF inference can be reformulated as a Recurrent

Algorithm 1 Mean-field in dense CRFs [29], broken down to common CNN operations.

$Q_i(l) \leftarrow \frac{1}{Z_i} \exp\left(U_i(l)\right)$ for all i ▷ Initialization

while not converged **do**

$\tilde{Q}_i^{(m)}(l) \leftarrow \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l)$ for all m ▷ Message Passing

$\check{Q}_i(l) \leftarrow \sum_m w^{(m)} \tilde{Q}_i^{(m)}(l)$ ▷ Weighting Filter Outputs

$\hat{Q}_i(l) \leftarrow \sum_{l' \in \mathcal{L}} \mu(l, l') \check{Q}_i(l')$ ▷ Compatibility Transform

$\breve{Q}_i(l) \leftarrow U_i(l) - \hat{Q}_i(l)$ ▷ Adding Unary Potentials

$Q_i \leftarrow \frac{1}{Z_i} \exp\left(\breve{Q}_i(l)\right)$ ▷ Normalizing

end while

Neural Network (RNN). To this end, we first consider individual steps of the mean-field algorithm summarized in Algorithm 1 [29], and describe them as CNN layers. Our contribution is based on the observation that filter-based approximate mean-field inference approach for dense CRFs relies on applying Gaussian spatial and bilateral filters on the mean-field approximates in each iteration. Unlike the standard convolutional layer in a CNN, in which filters are fixed after the training stage, we use edge-preserving Gaussian filters [56, 42], coefficients of which depend on the original spatial and appearance information of the image. These filters have the additional advantages of requiring a smaller set of parameters, despite the filter size being potentially as big as the image.

While reformulating the steps of the inference algorithm as CNN layers, it is essential to be able to calculate error differentials in each layer w.r.t. its inputs in order to be able to back-propagate the error differentials to previous layers during training. We also discuss how to calculate error differentials with respect to the parameters in each layer, enabling their optimization through the back-propagation algorithm. Therefore, in our formulation, CRF parameters such as the weights of the Gaussian kernels and the label compatibility function can also be optimized automatically during the training of the full network.

Once the individual steps of the algorithm are broken down as CNN layers, the full algorithm can then be formulated as an RNN. We explain this in Section 5 after discussing the steps of Algorithm 1 in detail below. In Algorithm 1 and the remainder of this paper, we use $U_i(l)$ to denote the negative of the unary energy introduced in the previous section, i.e., $U_i(l) = -\psi_u(X_i = l)$. In the conventional CRF setting, this input $U_i(l)$ to the mean-field algorithm is obtained from an independent classifier.

4.1. Initialization

In the initialization step of the algorithm, the operation $Q_i(l) \leftarrow \frac{1}{Z_i} \exp\left(U_i(l)\right)$, where $Z_i = \sum_l \exp(U_i(l))$, is performed. Note that this is equivalent to applying a softmax function over the unary potentials U across all the labels at each pixel. The softmax function has been extensively used in CNN architectures before and is therefore well known in the deep learning community. This operation does not include any parameters and the error differentials received at the output of the step during back-propagation could be passed down to the unary potential inputs after performing usual backward pass calculations of the softmax transformation.

4.2. Message Passing

In the dense CRF formulation, message passing is implemented by applying M Gaussian filters on Q values. Gaussian filter coefficients are derived based on image features such as the pixel locations and RGB values, which reflect how strongly a pixel is related to other pixels. Since the CRF is potentially fully connected, each filter's receptive field spans the whole image, making it infeasible to use a brute-force implementation of the filters. Fortunately, several approximation techniques exist to make computation of high dimensional Gaussian filtering significantly faster. Following [29], we use the Permutohedral lattice implementation [1], which can compute the filter response in $O(N)$ time, where N is the number of pixels of the image [1].

During back-propagation, error derivatives w.r.t. the filter inputs are calculated by sending the error derivatives w.r.t. the filter outputs through the same M Gaussian filters in reverse direction. In terms of permutohedral lattice operations, this can be accomplished by only reversing the order of the separable filters in the blur stage, while building the permutohedral lattice, splatting, and slicing in the same way as in the forward pass. Therefore, back-propagation through this filtering stage can also be performed in $O(N)$ time. Following [29], we use two Gaussian kernels, a spatial kernel and a bilateral kernel. In this work, for simplicity, we keep the bandwidth values of the filters fixed. It is also possible to use multiple spatial and bilateral kernels with different bandwidth values and learn their optimal linear combination.

4.3. Weighting Filter Outputs

The next step of the mean-field iteration is taking a weighted sum of the M filter outputs from the previous step, for each class label l. When each class label is considered individually, this can be viewed as usual convolution with a 1×1 filter with M input channels, and one output channel. Since both inputs and the outputs to this step are known during back-propagation, the error derivative w.r.t. the filter

weights can be computed, making it possible to automatically learn the filter weights (relative contributions from each Gaussian filter output from the previous stage). Error derivative w.r.t. the inputs can also be computed in the usual manner to pass the error derivatives down to the previous stage. To obtain a higher number of tunable parameters, in contrast to [29], we use independent kernel weights for each class label. The intuition is that the relative importance of the spatial kernel vs the bilateral kernel depends on the visual class. For example, bilateral kernels may have on the one hand a high importance in bicycle detection, because similarity of colours is determinant; on the other hand they may have low importance for TV detection, given that whatever is inside the TV screen may have many different colours.

4.4. Compatibility Transform

In the compatibility transform step, outputs from the previous step (denoted by $\check{Q}$ in Algorithm 1) are shared between the labels to a varied extent, depending on the compatibility between these labels. Compatibility between the two labels l and l' is parameterized by the label compatibility function $\mu(l, l')$. The Potts model, given by $\mu(l, l') = [l \neq l']$, where $[.]$ is the Iverson bracket, assigns a fixed penalty if different labels are assigned to pixels with similar properties. A limitation of this model is that it assigns the same penalty for all different pairs of labels. Intuitively, better results can be obtained by taking the compatibility between different label pairs into account and penalizing the assignments accordingly. For example, assigning labels "person" and "bicycle" to nearby pixels should have a lesser penalty than assigning labels "sky" and "bicycle". Therefore, learning the function μ from data is preferred to fixing it in advance with Potts model. We also relax our compatibility transform model by assuming that $\mu(l, l') \neq \mu(l', l)$ in general.

Compatibility transform step can be viewed as another convolution layer where the spatial receptive field of the filter is 1×1, and the number of input and output channels are both L. Learning the weights of this filter is equivalent to learning the label compatibility function μ. Transferring error differentials from the output of this step to the input can be done since this step is a usual convolution operation.

4.5. Adding Unary Potentials

In this step, the output from the compatibility transform stage is subtracted element-wise from the unary inputs U. While no parameters are involved in this step, transferring error differentials can be done trivially by copying the differentials at the output of this step to both inputs with the appropriate sign.

4.6. Normalization

Finally, the normalization step of the iteration can be considered as another softmax operation with no parameters. Differentials at the output of this step can be passed on to the input using the softmax operation's backward pass.

5. The End-to-end Trainable Network

We now describe our end-to-end deep learning system for semantic image segmentation. To pave the way for this, we first explain how repeated mean-field iterations can be organized as an RNN.

5.1. CRF as RNN

In the previous section, it was shown that one iteration of the mean-field algorithm can be formulated as a stack of common CNN layers (see Fig. 1). We use the function $f_{\boldsymbol{\theta}}$ to denote the transformation done by one mean-field iteration: given an image I, pixel-wise unary potential values U and an estimation of marginal probabilities Q_{in} from the previous iteration, the next estimation of marginal distributions after one mean-field iteration is given by $f_{\boldsymbol{\theta}}(U, Q_{\text{in}}, I)$. The vector $\boldsymbol{\theta} = \{w^{(m)}, \mu(l, l^{'})\}, m \in \{1, ..., M\}, l, l^{'} \in \{l_1, ..., l_L\}$ represents the CRF parameters described in Section 4.

Multiple mean-field iterations can be implemented by repeating the above stack of layers in such a way that each iteration takes Q value estimates from the previous iteration and the unary values in their original form. This is equivalent to treating the iterative mean-field inference as a Recurrent Neural Network (RNN) as shown in Fig. 2. Using the notation in the figure, the behaviour of the network is given by the following equations where T is the number of mean-field iterations:

$$H_1(t) = \begin{cases} \text{softmax}(U), & t = 0 \\ H_2(t-1), & 0 < t \leq T, \end{cases} \tag{3}$$

$$H_2(t) = f_{\boldsymbol{\theta}}(U, H_1(t), I), \quad 0 \leq t \leq T, \tag{4}$$

$$Y(t) = \begin{cases} 0, & 0 \leq t < T \\ H_2(t), & t = T. \end{cases} \tag{5}$$

We name this RNN structure CRF-RNN. Parameters of the CRF-RNN are the same as the mean-field parameters described in Section 4 and denoted by $\boldsymbol{\theta}$ here. Since the calculation of error differentials w.r.t. these parameters in a single iteration was described in Section 4, they can be learnt in the RNN setting using the standard back-propagation through time algorithm [48, 40]. It was shown in [29] that the mean-field iterative algorithm for dense CRF converges in less than 10 iterations. Furthermore, in practice, after about 5 iterations, increasing the number of iterations usually does not significantly improve results [29]. Therefore,

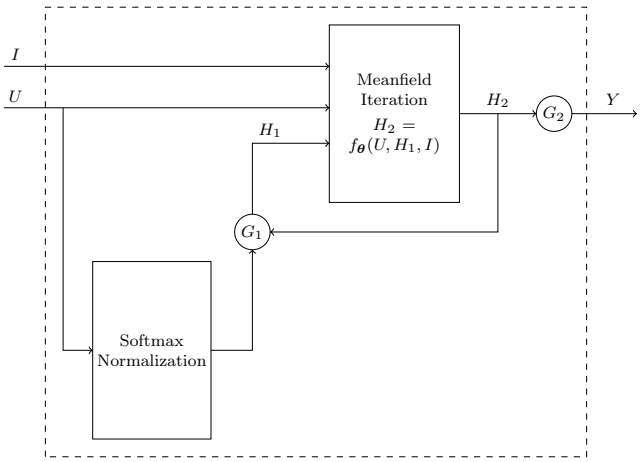


Figure 2. **The CRF-RNN Network.** We formulate the iterative mean-field algorithm as a Recurrent Neural Network (RNN). Gating functions G_1 and G_2 are fixed as described in the text.

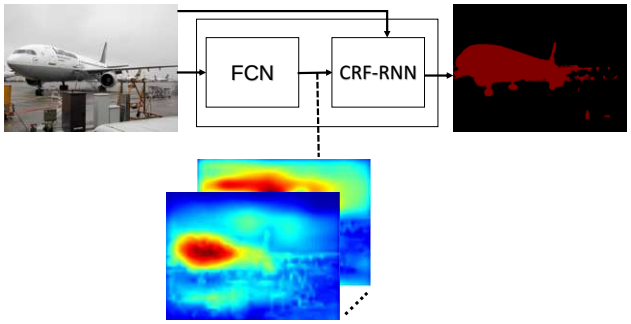


Figure 3. **The End-to-end Trainable Network.** Schematic visualization of our full network which consists of a CNN and the CNN-CRF network. Best viewed in colour.

it does not suffer from the vanishing and exploding gradient problem inherent to deep RNNs [7, 43]. This allows us to use a plain RNN architecture instead of more sophisticated architectures such as LSTMs in our network.

5.2. Completing the Picture

Our approach comprises a fully convolutional network stage, which predicts pixel-level labels without considering structure, followed by a CRF-RNN stage, which performs CRF-based probabilistic graphical modelling for structured prediction. The complete system, therefore, unifies strengths of both CNNs and CRFs and is trainable end-to-end using the back-propagation algorithm [34] and the Stochastic Gradient Descent (SGD) procedure. During training, a whole image (or many of them) can be used as the mini-batch and the error at each pixel output of the network can be computed using an appropriate loss function such as the softmax loss with respect to the ground truth segmentation of the image. We used the FCN-8s architecture of [37] as the first part of our network, which provides unary potentials to the CRF. This network is based on the VGG-16 network [53] but has been restructured to perform pixel-wise prediction instead of image classification.

In the forward pass through the network, once the computation enters the CRF-RNN after passing through the CNN stage, it takes T iterations for the data to leave the loop created by the RNN. Neither the CNN that provides unary values nor the layers after the CRF-RNN (i.e., the loss layers) need to perform any computations during this time since the refinement happens only inside the RNN's loop. Once the output Y leaves the loop, next stages of the deep network after the CRF-RNN can continue the forward pass. In our setup, a softmax loss layer directly follows the CRF-RNN and terminates the network.

During the backward pass, once the error differentials reach the CRF-RNN's output Y, they similarly spend T iterations within the loop before reaching the RNN input U in order to propagate to the CNN which provides the unary input. In each iteration inside the loop, error differentials are computed inside each component of the mean-field iteration as described in Section 4. We note that unnecessarily increasing the number of mean-field iterations T could potentially result in the vanishing and exploding gradient problems in the CRF-RNN. We, however, did not experience this problem during our experiments.

6. Implementation Details

In the present section we describe the implementation details of the proposed network, as well as its training process. The high-level architecture of our system, which was implemented using the popular Caffe [27] deep learning library, is shown in Fig. 3. The full source code and the trained models of our approach will be made publicly available [1].

We initialized the first part of the network using the publicly available weights of the FCN-8s network [37]. The compatibility transform parameters of the CRF-RNN were initialized using the Potts model, and kernel width and weight parameters were obtained from a cross-validation process. We found that such initialization results in faster convergence of training. During the training phase, parameters of the whole network were optimized end-to-end using the back-propagation algorithm. In particular we used full image training described in [37], with learning rate fixed at 10^{-13} and momentum set to 0.99. These extreme values of the parameters were used since we employed only one image per batch to avoid reaching memory limits of the GPU.

In all our experiments, during training, we set the number of mean-field iterations T in the CRF-RNN to 5 to avoid

[1] https://github.com/torrvision/crfasrnn/.

vanishing/exploding gradient problems and to reduce the training time. During the test time, iteration count was increased to 10. The effect of this parameter value on the accuracy is discussed in section 7.1.

Loss function During the training of the models that achieved the best results reported in this paper, we used the standard softmax loss function, that is, the log-likelihood error function described in [30]. The standard metric used in the Pascal VOC challenge is the average intersection over union (IU), which we also use here to report the results. In our experiments we found that high values of IU on the validation set were associated to low values of the averaged softmax loss, to a large extent. We also tried the robust log-likelihood in [30] as a loss function for CRF-RNN training. However, this did not result in increased accuracy nor faster convergence.

Normalization techniques As described in Section 4, we use the exponential function followed by pixel-wise normalization across channels in several stages of the CRF-RNN. Since this operation has a tendency to result in small gradients with respect to the input when the input value is large, we conducted several experiments where we replaced this by a rectifier linear unit (ReLU) operation followed by a normalization across the channels. Our hypothesis was that this approach may approximate the original operation adequately while speeding up the training due to improved gradients. Furthermore, ReLU would induce sparsity on the probability of labels assigned to pixels, implicitly pruning low likelihood configurations, which could have a positive effect. However, this approach did not lead to better results, obtaining 1% IU lower than the original setting performance.

Figure 4. **Qualitative results on the validation set of Pascal VOC 2012.** FCN [37] is a CNN-based model that does not employ CRF. Deeplab [10] is a two-stage approach, where the CNN is trained first, and then CRF is applied on top of the CNN output. Our approach is an end-to-end trained system that integrates both CNN and CRF-RNN in one deep network. Best viewed in colour.

7. Experiments

We present experimental results with the proposed CRF-RNN framework. We use these datasets: the Pascal VOC 2012 dataset, and the Pascal Context dataset. We use the Pascal VOC 2012 dataset as it has become the golden standard to comprehensively evaluate any new semantic segmentation approach in comparison to existing methods. We also use the Pascal Context dataset to assess how well our approach performs on a dataset with different characteristics.

Pascal VOC Datasets

In order to evaluate our approach with existing methods under the same circumstances, we conducted two main experiments with the Pascal VOC 2012 dataset, followed by a qualitative experiment.

In the first experiment, following [37, 38, 41], we used a training set consisted of VOC 2012 training data (1464 images), and training and validation data of [23], which amounts to a total of 11,685 images. After removing the overlapping images between VOC 2012 validation data and this training dataset, we were left with 346 images from the original VOC 2012 validation set to validate our models on. We call this set the reduced validation set in the sequel. Annotations of the VOC 2012 test set, which consists of 1456 images, are not publicly available and hence the final results on the test set were obtained by submitting the results to the Pascal VOC challenge evaluation server [18]. Regardless of the smaller number of images, we found that the relative improvements of the accuracy on our validation set were in good agreement with the test set.

As a first step we directly compared the potential advantage of learning the model end-to-end with respect to alternative learning strategies. These are plain FCN-8s without applying CRF, and with CRF as a postprocessing method disconnected from the training of FCN, which is comparable to the approach described in [10] and [41]. The results are reported in Table 1 and show a clear advantage of the end-to-end strategy over the offline application of CRF as a

post-processing method. This can be attributed to the fact that during the SGD training of the CRF-RNN, the CNN component and the CRF component learn how to co-operate with each other to produce the optimum output of the whole network.

We then proceeded to compare our approach with all state-of-the-art methods that used training data from the standard VOC 2012 training and validation sets, and from the dataset published with [22]. The results are shown in Table 2, above the bar, and we can see that our approach outperforms all competitors.

In the second experiment, in addition to the above training set, we used data from the Microsoft COCO dataset [36] as was done in [41] and [12]. We selected images from MS COCO 2014 training set where the ground truth segmentation has at least 200 pixels marked with classes labels present in the VOC 2012 dataset. With this selection, we ended up using 66,099 images from the COCO dataset and therefore a total of 66,099 + 11,685 = 77,784 training images were used in the second experiment. The same reduced validation set was used in this second experiment as well. In this case, we first fine-tuned the plain FCN-32s network (without the CRF-RNN part) on COCO data, then we built an FCN-8s network with the learnt weights and finally train the CRF-RNN network end-to-end using VOC 2012 training data only. Since the MS COCO ground truth segmentation data contains somewhat coarse segmentation masks where objects are not delineated properly, we found that fine-tuning our model with COCO did not yield significant improvements. This can be understood because the primary advantage of our model comes from delineating the objects and improving fine segmentation boundaries. The VOC 2012 training dataset therefore helps our model learn this task effectively. The results of this experiment are shown in Table 2, below the bar, and we see that our approach sets a new state-of-the-art on the VOC 2012 dataset.

Note that in both setups, our approach outperforms competing methods due to the end-to-end training of the CNN and CRF in the unified CRF-RNN framework. We also evaluated our models on the VOC 2010, and VOC 2011 test set (see Table 2). In all cases our method achieves the state-of-the-art performance.

In order to have a qualitative evidence about how CRF-RNN learns, we visualize the compatibility function learned after the training stage of the CRF-RNN as a matrix representation in Fig. 5. Element (i, j) of this matrix corresponds to $\mu(i, j)$ defined earlier: a high value at (i, j) implies high penalty for assigning label i to a pixel when a similar pixel (spatially or appearance wise) is assigned label j. For example we can appreciate that the learned compatibility matrix assigns a low penalty to pairs of labels that tend to appear together, such as [*Motorbike*, *Person*], and [*Dining table*, *Chair*].

Method	Without COCO	With COCO
Plain FCN-8s	61.3	68.3
FCN-8s and CRF disconnected	63.7	69.5
End-to-end training of CRF-RNN	69.6	72.9

Table 1. Mean IU accuracy of our approach, CRF-RNN, compared with similar methods, evaluated on the reduced VOC 2012 validation set.

Method	VOC 2010 test	VOC 2011 test	VOC 2012 test
BerkeleyRC [3]	n/a	39.1	n/a
O2PCPMC [8]	49.6	48.8	47.8
Divmbest [44]	n/a	n/a	48.1
NUS-UDS [16]	n/a	n/a	50.0
SDS [23]	n/a	n/a	51.6
MSRA-CFM [13]	n/a	n/a	61.8
FCN-8s [37]	n/a	62.7	62.2
Hypercolumn [24]	n/a	n/a	62.6
Zoomout [38]	64.4	64.1	64.4
Context-Deep-CNN-CRF [35]	n/a	n/a	70.7
DeepLab-MSc [10]	n/a	n/a	71.6
Our method w/o COCO	73.6	72.4	72.0
BoxSup [12]	n/a	n/a	71.0
DeepLab [10, 41]	n/a	n/a	72.7
Our method with COCO	**75.7**	**75.0**	**74.7**

Table 2. Mean IU accuracy of our approach, CRF-RNN, compared to the other approaches on the Pascal VOC 2010-2012 test datasets. Methods from the first group do not use MS COCO data for training. The methods from the second group use both COCO and VOC datasets for training.

Pascal Context Dataset

We conducted an experiment on the Pascal Context dataset [39], which differs from the previous one in the larger number of classes considered, 59. We used the provided partitions of training and validation sets, and the obtained results are reported in Table 3.

Method	O_2P [8]	CFM [13]	FCN-8s [37]	CRF-RNN
Mean IU	18.1	34.4	37.78	**39.28**

Table 3. Mean IU accuracy of our approach, CRF-RNN, evaluated on the Pascal Context validation set.

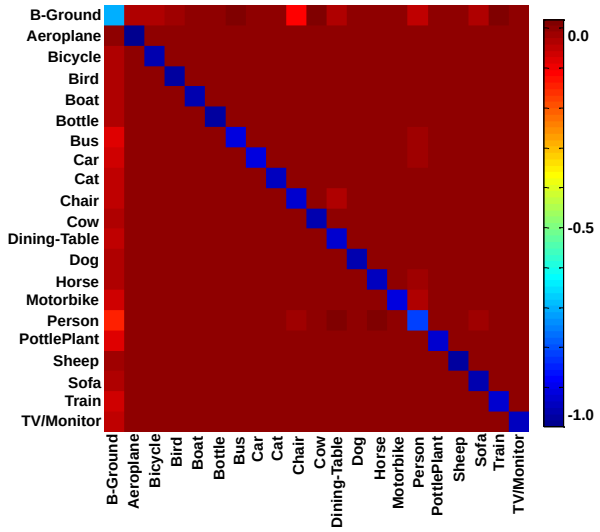


Figure 5. **Visualization of the learnt label compatibility matrix.** In the standard Potts model, diagonal entries are equal to -1, while off-diagonal entries are zero. These values have changed after the end-to-end training of our network. Best viewed in colour.

7.1. Effect of Design Choices

We performed a number of additional experiments on the Pascal VOC 2012 validation set described above to study the effect of some design choices we made.

We first studied the performance gains attained by our modifications to the CRF over the CRF approach proposed by [29]. We found that using different filter weights for different classes improved the performance by 1.8 percentage points, and that introducing the asymmetric compatibility transform further boosted the performance by 0.9 percentage points.

Regarding the RNN parameter iteration count T, incrementing it to $T = 10$ during the test time, from $T = 5$ during the train time, produced an accuracy improvement of 0.2 percentage points. Setting $T = 10$ also during training reduced the accuracy by 0.7 percentage points. We believe that this might be due to a vanishing gradient effect caused by using too many iterations. In practice that leads to the first part of the network (the one producing unary potentials) receiving a very weak error gradient signal during training, thus hampering its learning capacity.

End-to-end training after the initialization of CRF parameters improved performance by 3.4 percentage points. We also conducted an experiment where we froze the FCN-8s part and fine-tuned only the RNN part (i.e., CRF parameters). It improved the performance over initialization by only 1 percentage point. We therefore conclude that end-to-end training significantly contributed to boost the accuracy of the system.

Treating each iteration of mean-field inference as an independent step with its own parameters, and training end-to-end with 5 such iterations yielded a final mean IU score of only 70.9, supporting the hypothesis that the recurrent structure of our approach is important for its success.

8. Conclusion

We presented CRF-RNN, an interpretation of dense CRFs as Recurrent Neural Networks. Our formulation fully integrates CRF-based probabilistic graphical modelling with emerging deep learning techniques. In particular, the proposed CRF-RNN can be plugged in as a part of a traditional deep neural network: It is capable of passing on error differentials from its outputs to inputs during back-propagation based training of the deep network while learning CRF parameters. We demonstrate the use of this approach by utilizing it for the semantic segmentation task: we form an end-to-end trainable deep network by combining a fully convolutional neural network with the CRF-RNN. Our system achieves a new state-of-the-art on the popular Pascal VOC segmentation benchmark. This improvement can be attributed to the uniting of the strengths of CNNs and CRFs in a single deep network.

In the future, we plan to investigate the advantages/disadvantages of restricting the capabilities of the RNN part of our network to mean-field inference of dense CRF. A sensible baseline to the work presented here would be to use more standard RNNs (*e.g.* LSTMs) that learn to iteratively improve the input unary potentials to make them closer to the ground-truth.

Acknowledgement This work was supported by grants EP/M013774/1 and ERC 321162-HELIOS. We thank the Caffe team, Baidu IDL, and the Oxford ARC team for their support. We gratefully acknowledge GPU donations from NVIDIA.

References

[1] A. Adams, J. Baek, and M. A. Davis. Fast high-dimensional filtering using the permutohedral lattice. *Computer Graphics Forum*, 29(2):753–762, 2010.

[2] P. Arbeláez, B. Hariharan, C. Gu, S. Gupta, L. Bourdev, and J. Malik. Semantic segmentation using regions and parts. In *IEEE CVPR*, 2012.

[3] P. Arbeláez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE TPAMI*, 33(5):898–916, 2011.

[4] A. Barbu. Training an active random field for real-time image denoising. *IEEE TIP*, 18(11):2451–2462, 2009.

[5] S. Bell, P. Upchurch, N. Snavely, and K. Bala. Material recognition in the wild with the materials in context database. In *IEEE CVPR*, 2015.

[6] Y. Bengio, Y. LeCun, and D. Henderson. Globally trained handwritten word recognizer using spatial representation, convolutional neural networks, and hidden markov models. In *NIPS*, pages 937–937, 1994.

[7] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 1994.

[8] J. Carreira, R. Caseiro, J. Batista, and C. Sminchisescu. Free-form region description with second-order pooling. *IEEE TPAMI*, 2014.

[9] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *BMVC*, 2014.

[10] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *ICLR*, 2015.

[11] L.-C. Chen, A. G. Schwing, A. L. Yuille, and R. Urtasun. Learning deep structured models. In *ICLRW*, 2015.

[12] J. Dai, K. He, and J. Sun. Boxsup: Exploiting bounding boxes to supervise convolutional networks for semantic segmentation. In *arXiv:1503.01640*, 2015.

[13] J. Dai, K. He, and J. Sun. Convolutional feature masking for joint object and stuff segmentation. In *IEEE CVPR*, 2015.

[14] T.-M.-T. Do and T. Artieres. Neural conditional random fields. In *NIPS*, 2010.

[15] J. Domke. Learning graphical model parameters with approximate marginal inference. *IEEE TPAMI*, 35(10):2454–2467, 2013.

[16] J. Dong, Q. Chen, S. Yan, and A. Yuille. Towards unified object detection and semantic segmentation. In *ECCV*, 2014.

[17] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. In *NIPS*, 2014.

[18] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *IJCV*, 111(1):98–136, 2015.

[19] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE TPAMI*, 2013.

[20] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE CVPR*, 2014.

[21] R. Girshick, F. Iandola, T. Darrell, and J. Malik. Deformable part models are convolutional neural networks. In *CVPR*, 2015.

[22] B. Hariharan, P. Arbelaez, L. D. Bourdev, S. Maji, and J. Malik. Semantic contours from inverse detectors. In *IEEE ICCV*, 2011.

[23] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Simultaneous detection and segmentation. In *ECCV*, 2014.

[24] B. Hariharan, P. Arbelaez, R. Girshick, and J. Malik. Hypercolumns for object segmentation and fine-grained localization. In *IEEE CVPR*, 2015.

[25] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman. Deep structured output learning for unconstrained text recognition. In *ICLR*, 2015.

[26] V. Jain, J. F. Murray, F. Roth, S. C. Turaga, V. P. Zhigulin, K. L. Briggman, M. Helmstaedter, W. Denk, and H. S. Seung. Supervised learning of image restoration with convolutional networks. In *IEEE ICCV*, 2007.

[27] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM Multimedia*, pages 675–678, 2014.

[28] M. Kiefel and P. V. Gehler. Human pose estmation with fields of parts. In *ECCV*, 2014.

[29] P. Krähenbühl and V. Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *NIPS*, 2011.

[30] P. Krähenbühl and V. Koltun. Parameter learning and convergent inference for dense random fields. In *ICML*, 2013.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[32] L. Ladicky, C. Russell, P. Kohli, and P. H. Torr. Associative hierarchical crfs for object class image segmentation. In *IEEE ICCV*, 2009.

[33] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

[34] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[35] G. Lin, C. Shen, I. Reid, and A. van dan Hengel. Efficient piecewise training of deep structured models for semantic segmentation. In *arXiv:1504.01013*, 2015.

[36] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollar. Microsoft coco: Common objects in context. In *arXiv:1405.0312*, 2014.

[37] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *IEEE CVPR*, 2015.

[38] M. Mostajabi, P. Yadollahpour, and G. Shakhnarovich. Feedforward semantic segmentation with zoom-out features. In *IEEE CVPR*, 2015.

[39] R. Mottaghi, X. Chen, X. Liu, N.-G. Cho, S.-W. Lee, S. Fidler, R. Urtasun, and A. Yuille. The role of context for object detection and semantic segmentation in the wild. In *IEEE CVPR*, 2014.

[40] M. C. Mozer. Backpropagation. In Y. Chauvin and D. E. Rumelhart, editors, *Backpropagation*, chapter A Focused Backpropagation Algorithm for Temporal Pattern Recognition, pages 137–169. L. Erlbaum Associates Inc., 1995.

[41] G. Papandreou, L.-C. Chen, K. Murphy, and A. L. Yuille. Weakly- and semi-supervised learning of a dcnn for semantic image segmentation. In *arXiv:1502.02734*, 2015.

[42] S. Paris and F. Durand. A fast approximation of the bilateral filter using a signal processing approach. *IJCV*, 81(1):24–52, 2013.

[43] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio. On the difficulty of training recurrent neural networks. In *ICML*, 2013.

[44] G. S. Payman Yadollahpour, Dhruv Batra. Discriminative re-ranking of diverse segmentations. In *IEEE CVPR*, 2013.

[45] J. Peng, L. Bo, and J. Xu. Conditional neural fields. In *NIPS*, 2009.

[46] P. H. O. Pinheiro and R. Collobert. Recurrent convolutional neural networks for scene labeling. In *ICML*, 2014.

[47] S. Ross, D. Munoz, M. Hebert, and J. A. Bagnell. Learning message-passing inference machines for structured prediction. In *IEEE CVPR*, 2011.

[48] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: explorations in the microstructure of cognition. In J. A. Anderson and E. Rosenfeld, editors, *Parallel distributed processing: explorations in the microstructure of cognition*, chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, 1986.

[49] A. G. Schwing and R. Urtasun. Fully connected deep structured networks. In *arXiv:1503.02351*, 2015.

[50] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *IEEE CVPR*, 2011.

[51] J. Shotton, M. Johnson, and R. Cipolla. Semantic texton forests for image categorization and segmentation. In *IEEE CVPR*, 2008.

[52] J. Shotton, J. Winn, C. Rother, and A. Criminisi. Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context. *IJCV*, 81(1):2–23, 2009.

[53] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *arXiv:1409.1556*, 2014.

[54] V. Stoyanov, A. Ropson, and J. Eisner. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *AISTATS*, 2011.

[55] S. C. Tatikonda and M. I. Jordan. Loopy belief propagation and gibbs measures. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, 2002.

[56] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *IEEE CVPR*, 1998.

[57] J. J. Tompson, A. Jain, Y. LeCun, and C. Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. In *NIPS*, 2014.

[58] Z. Tu. Auto-context and its application to high-level vision tasks. In *IEEE CVPR*, 2008.

[59] Z. Tu, X. Chen, A. L. Yuille, and S.-C. Zhu. Image parsing: Unifying segmentation, detection, and recognition. *IJCV*, 63(2):113–140, 2005.

[60] K. Yao, B. Peng, G. Zweig, D. Yu, X. Li, and F. Gao. Recurrent conditional random field for language understanding. In *ICASSP*, 2014.

[61] Y. Zhang and T. Chen. Efficient inference for fully-connected crfs with stationarity. In *CVPR*, 2012.

Methods trained with COCO	Mean IU									
Our method	**74.7**	**90.4**	**55.3**	**88.7**	**68.4**	69.8	88.3	82.4	85.1	32.6
DeepLab[10, 41]	72.7	89.1	38.3	88.1	63.3	69.7	87.1	83.1	85.0	29.3
BoxSup[12]	71.0	86.4	35.5	79.7	65.2	65.2	84.3	78.5	83.7	30.5
Methods trained w/o COCO										
Our method trained w/o COCO	72.0	87.5	39.0	79.7	64.2	68.3	87.6	80.8	84.4	30.4
DeepLab-MSc-CRF-LargeFOV[10]	71.6	84.4	54.5	81.5	63.6	65.9	85.1	79.1	83.4	30.7
Context_Deep_CNN_CRF[35]	70.7	87.5	37.7	75.8	57.4	72.3	88.4	82.6	80.0	33.4
Zoomout[38]	64.4	81.9	35.1	78.2	57.4	56.5	80.5	74.0	79.8	22.4
Hypercolumn[24]	62.6	68.7	33.5	69.8	51.3	70.2	81.1	71.9	74.9	23.9
FCN-8s[37]	62.2	76.8	34.2	68.9	49.4	60.3	75.3	74.7	77.6	21.4
MSRA_CFM[13]	61.8	75.7	26.7	69.5	48.8	65.6	81.0	69.2	73.3	30.0
SDS[23]	51.6	63.3	25.7	63.0	39.8	59.2	70.9	61.4	54.9	16.8
NUS_UDS [16]	50.0	67.0	24.5	47.2	45.0	47.9	65.3	60.6	58.5	15.5
TTIC-divmbest-rerank[44]	48.1	62.7	25.6	46.9	43.0	54.8	58.4	58.6	55.6	14.6
BONN_O2PCPMC_FGT_SEGM [8]	47.8	64.0	27.3	54.1	39.2	48.7	56.6	57.7	52.5	14.2

Methods trained with COCO											
Our method	**78.5**	**64.4**	79.6	**81.9**	**86.4**	81.8	58.6	82.4	53.5	77.4	**70.1**
DeepLab[10, 41]	76.5	56.5	79.8	77.9	85.8	82.4	57.4	84.3	54.9	80.5	64.1
BoxSup[12]	76.2	62.6	79.3	76.1	82.1	81.3	57.0	78.2	55.0	72.5	68.1
Methods trained w/o COCO											
Our method trained w/o COCO	78.2	60.4	80.5	77.8	83.1	80.6	59.5	82.8	47.8	78.3	67.1
DeepLab-MSc-CRF-LargeFOV [10]	74.1	59.8	79.0	76.1	83.2	80.8	59.7	82.2	50.4	73.1	63.7
Context_Deep_CNN_CRF[35]	71.5	55.0	79.3	78.4	81.3	82.7	56.1	79.8	48.6	77.1	66.3
TTI_zoomout_16[38]	69.6	53.7	74.0	76.0	76.6	68.8	44.3	70.2	40.2	68.9	55.3
Hypercolumn[24]	60.6	46.9	72.1	68.3	74.5	72.9	52.6	64.4	45.4	64.9	57.4
FCN-8s[37]	62.5	46.8	71.8	63.9	76.5	73.9	45.2	72.4	37.4	70.9	55.1
MSRA_CFM[13]	68.7	51.5	69.1	68.1	71.7	67.5	50.4	66.5	44.4	58.9	53.5
SDS[23]	45.0	48.2	50.5	51.0	57.7	63.3	31.8	58.7	31.2	55.7	48.5
NUS_UDS[16]	50.8	37.4	45.8	59.9	62.0	52.7	40.8	48.2	36.8	53.1	45.6
TTIC-divmbest-rerank[44]	47.5	31.2	44.7	51.0	60.9	53.5	36.6	50.9	30.1	50.2	46.8
BONN_O2PCPMC_FGT_SEGM[8]	54.8	29.6	42.2	58.0	54.8	50.2	36.6	58.6	31.6	48.4	38.6

Table 4. Intersection over Union (IU) accuracy of our approach, CRF-RNN, compared to the other state-of-the-art approaches on the Pascal VOC 2012 test set. Scores for other methods were taken the results published by the original authors. The symbols are from Chatfield *et al*. [9].

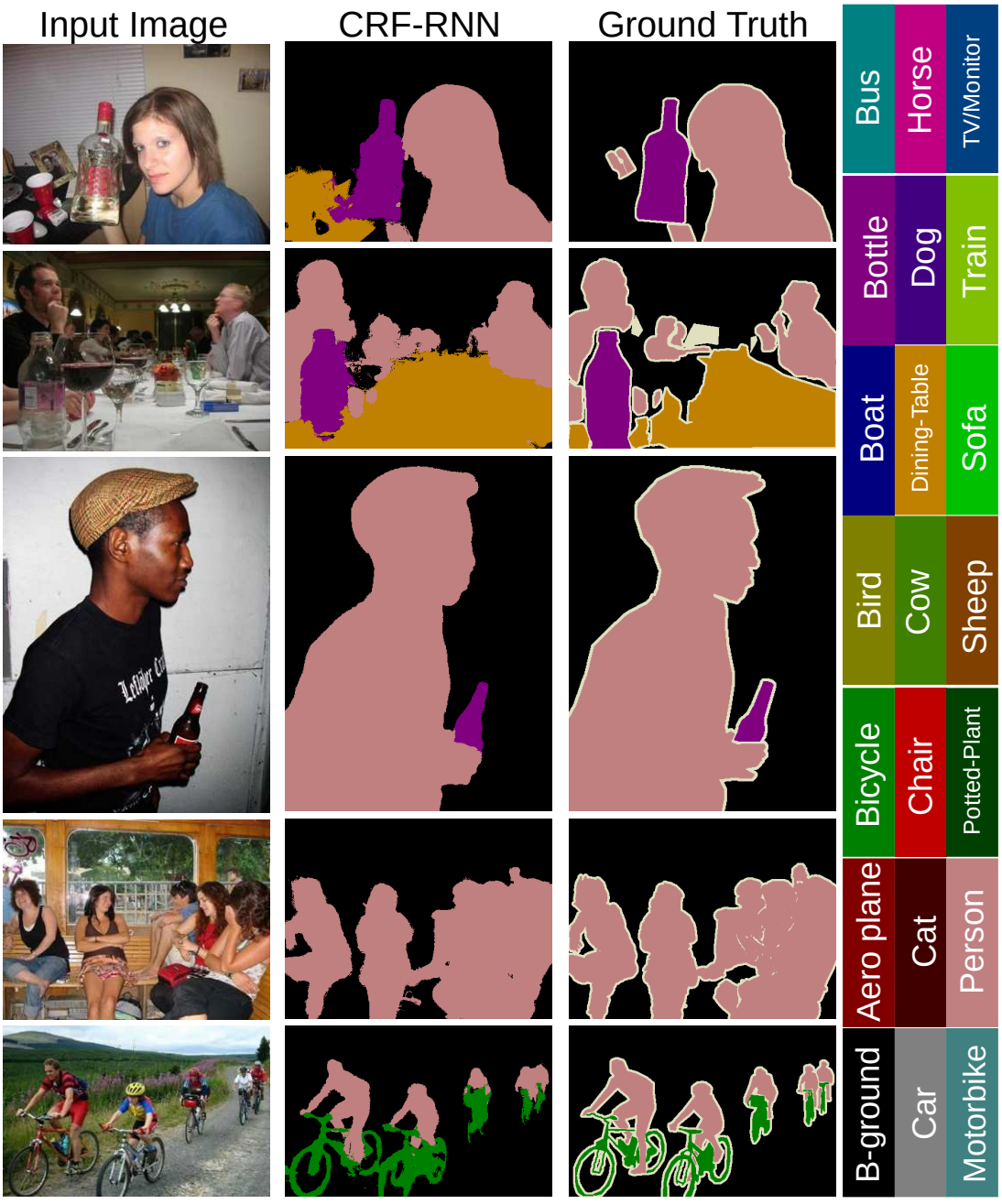


Figure 6. **Typical good quality segmentation results I.** Illustration of sample results on the validation set of the Pascal VOC 2012 dataset. Note that in some cases our method is able to pick correct segmentations that are not marked correctly in the ground truth. Best viewed in colour.

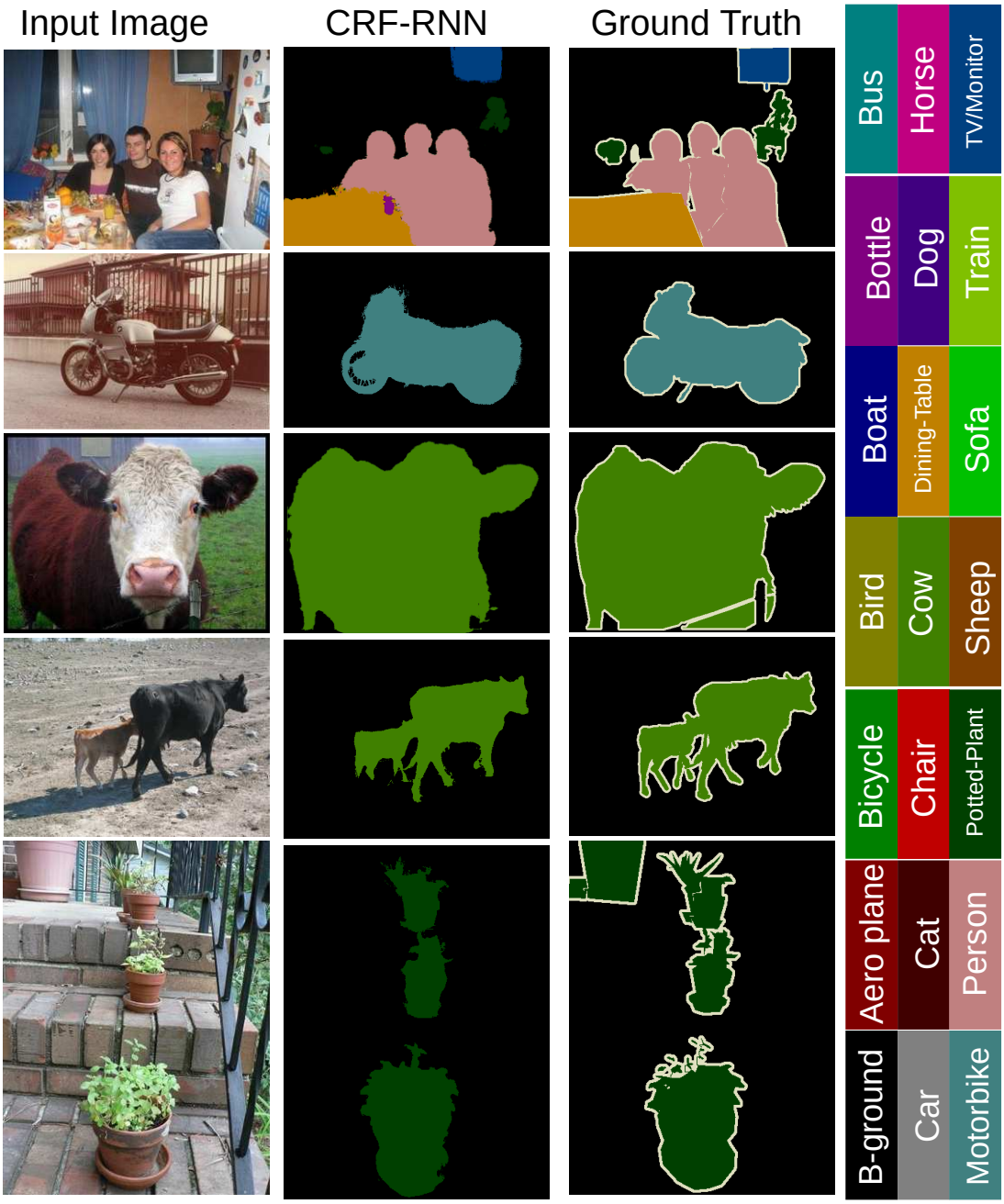


Figure 7. **Typical good quality segmentation results II.** Illustration of sample results on the validation set of the Pascal VOC 2012 dataset. Note that in some cases our method is able to pick correct segmentations that are not marked correctly in the ground truth. Best viewed in colour.

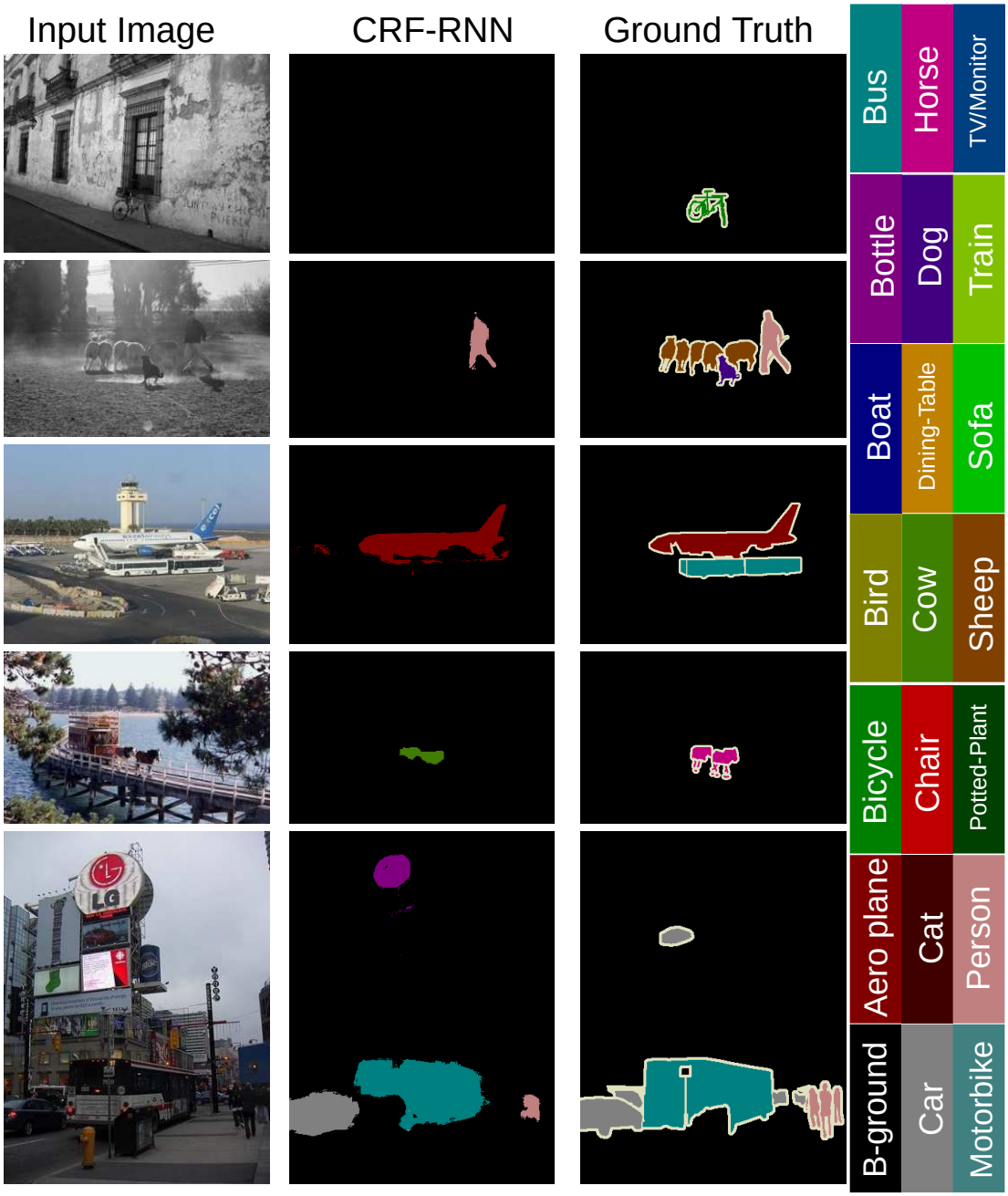


Figure 8. **Failure cases I.** Illustration of sample failure cases on the validation set of the Pascal VOC 2012 dataset. Best viewed in colour.

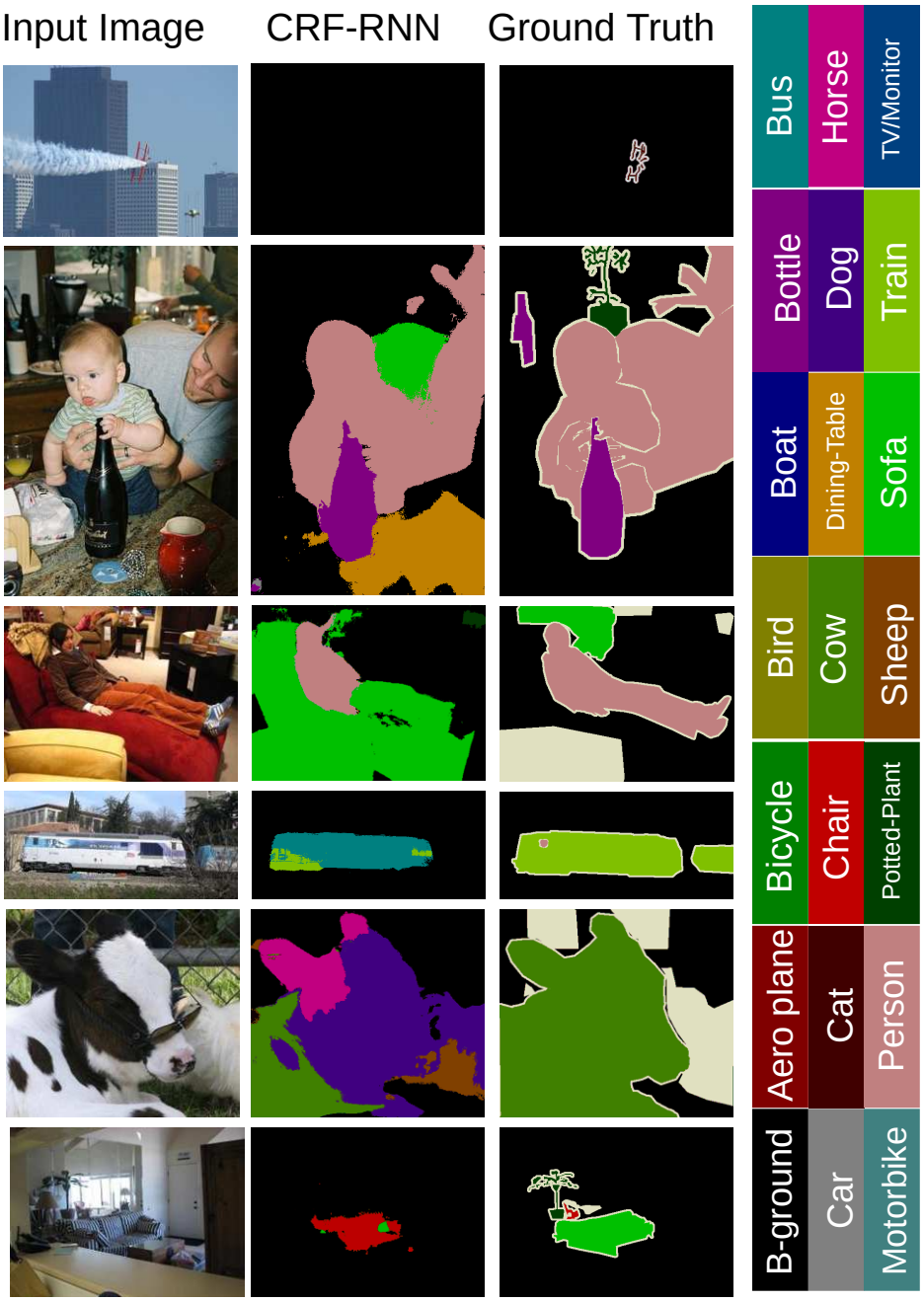


Figure 9. **Failure cases II.** Illustration of sample failure cases on the validation set of the Pascal VOC 2012 dataset. Best viewed in colour.

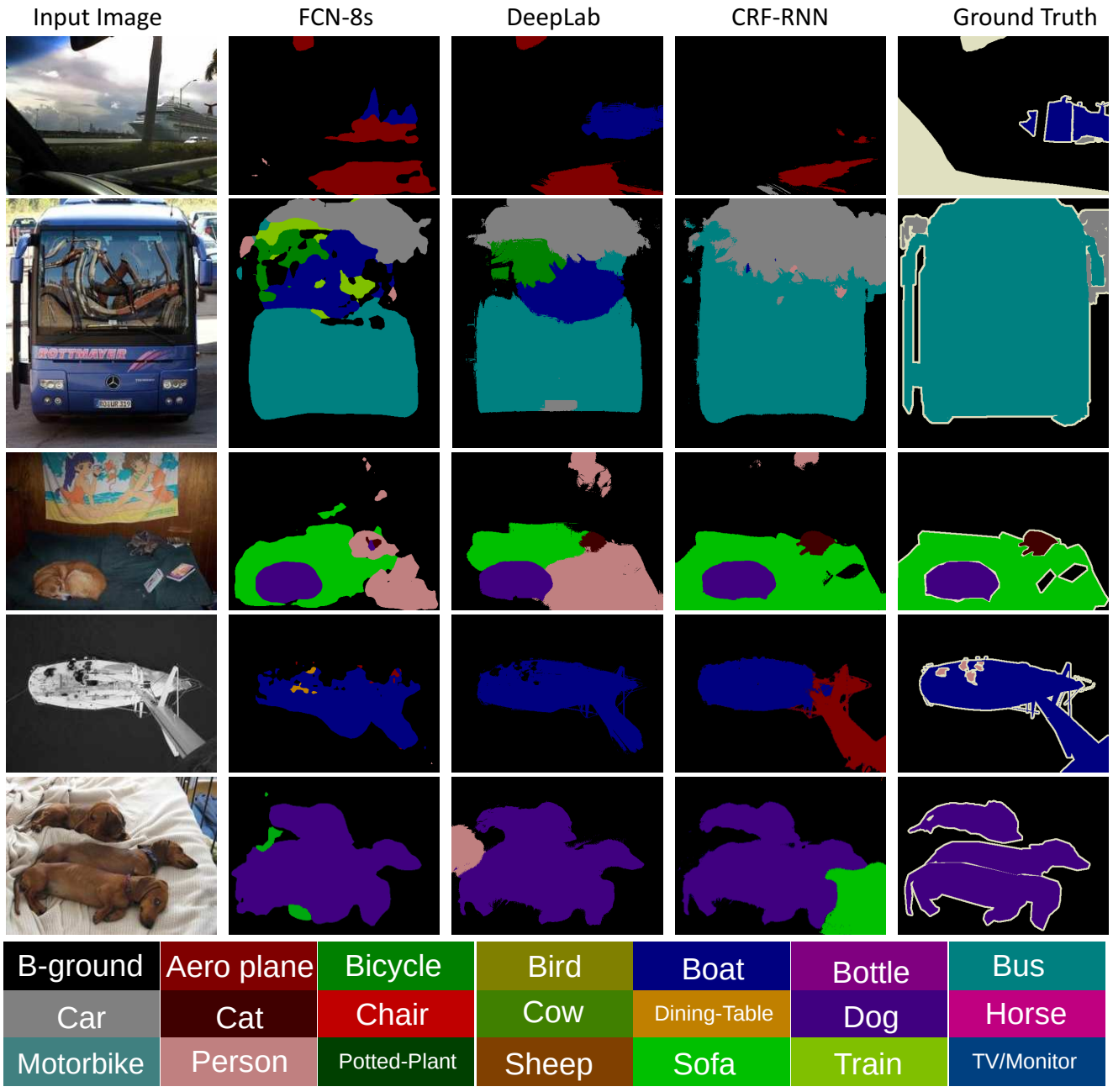


Figure 10. **Qualitative comparison with the other approaches.** Sample results with our method on the validation set of the Pascal VOC 2012 dataset, compared with previous state-of-the-art methods. Segmentation results with DeepLap approach were reproduced from the original publication. Best viewed in colour.

Local Convolutional Features with Unsupervised Training for Image Retrieval

Mattis Paulin, Matthijs Douze, Zaid Harchaoui, Julien Mairal, Florent Perronnin, Cordelia Schmid

Local Convolutional Features with Unsupervised Training for Image Retrieval

Mattis Paulin [1] Matthijs Douze[1] Zaid Harchaoui[1,2]
Julien Mairal[1] Florent Perronnin[3] Cordelia Schmid[1]

[1] Inria * [2] NYU [3] Facebook AI Research†

Abstract

Patch-level descriptors underlie several important computer vision tasks, such as stereo-matching or content-based image retrieval. We introduce a deep convolutional architecture that yields patch-level descriptors, as an alternative to the popular SIFT descriptor for image retrieval. The proposed family of descriptors, called Patch-CKN, adapt the recently introduced Convolutional Kernel Network (CKN), an unsupervised framework to learn convolutional architectures. We present a comparison framework to benchmark current deep convolutional approaches along with Patch-CKN for both patch and image retrieval, including our novel "RomePatches" dataset. Patch-CKN descriptors yield competitive results compared to supervised CNN alternatives on patch and image retrieval.

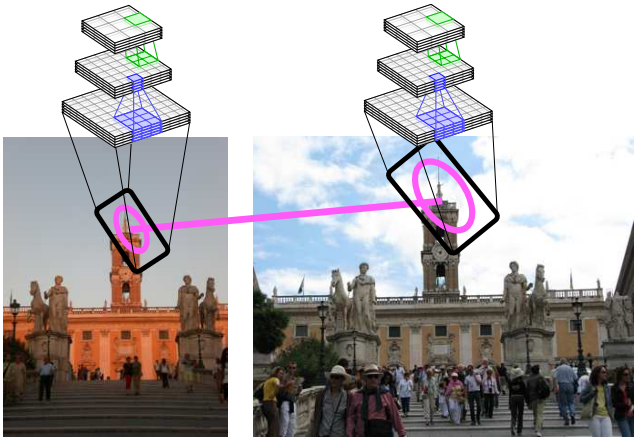


Figure 1. Proposed approach: interest regions are first extracted from images; then the neighborhood of each point is affine- and orientation-normalized to sample a patch; finally, deep convolutional nets are used to get a descriptor for each patch.

1. Introduction

This paper introduces a deep kernel-based convolutional approach for the description of image patches that does not require supervision. The kernel-based feature representation can be effectively approximated using a simple stochastic gradient optimization procedure, yielding a patch-level descriptor that can be used for image retrieval tasks. Image retrieval is a challenging problem as different images of the same object/scene may exhibit large variations in viewpoint, illumination, scaling, occlusion, etc – see Figure 1.

State-of-the-art instance-level retrieval systems involve three steps: 1) *interest point detection*, 2) *description* and 3) *matching*. The goal of step 1) is to select key points that are reproducible under scale and viewpoint changes – see [30, 43] for a detailed comparison of detectors. The choice of a good local representation at step 2) is crucial to ensure robustness to viewing conditions. As an example, the popular SIFT descriptor [26] is robust to illumination variations or to small rotations.[1] As for step 3), the goal is to define a suitable metric between two patch sets. To avoid the cost of matching individual patches, scalable alternatives have been proposed that encode and aggregate local patch statistics, such as bag-of-words [41] and VLAD [19]. In this work, we focus on step 2), *i.e*. the description step, while we rely on state-of-the-art components for the detection and matching steps.

Our inspiration comes from the expressive feature representations output by deep convolutional neural nets (CNNs) [23] used in image classification [22]. Features output by a CNN's intermediate layers can be used as *image-level descriptors* [10] which can be transferred to a variety of tasks – see *e.g*. [3, 13, 31]. More recently, the question of whether suitable *patch-level descriptors* could be derived from such architectures has been raised. [25, 12, 39] provide a preliminary positive answer to this question by comparing favorably with the SIFT descriptor. While these works exhibit significant differences, it is worth noting they

*LEAR team, Inria Grenoble Rhône-Alpes, Laboratoire Jean Kuntzmann, CNRS, Univ. Grenoble Alpes, France.

†Research conducted while at the Xerox Research Centre Europe.

[1] SIFT refers only to the description part.

all rely on *supervised learning* processes.

While the penultimate layer outputs a suitable image-level descriptor [3, 13, 31], the output of previous layers, typically the 4th one, is actually preferable to obtain expressive patch-level descriptors, as noticed in [25]. As shown in [47], earlier layers tend to encode more task-independent information with respect to the later ones. In particular, the filters learned by the first layer tend to be similar regardless of the task, the objective function or the *level of supervision*. This motivates the following question: *is supervised learning required to make good local convolutional features for patch matching and image retrieval?*

Our contribution is the family of patch descriptors Patch-CKN based on Convolutional Kernel Networks (CKNs) [27]. CKNs were initially introduced for image classification in [27]. We introduce a feature representation for patches that is based on the kernel feature map of a convolutional match kernel, and therefore does not depend on examples nor labels. A finite-dimensional explicit feature embedding can be computed to approximate this kernel feature map [34, 44, 27]. We present a fast and simple procedure to compute an explicit feature embedding, using random sub-sampling of the patches, suitable for large-scale experiments. The experiments suggest that Patch-CKN gives competitive patch-level descriptors compared to supervised CNNs.

Several works have focused on learning patch representations [7, 46], yet few have analyzed the impact of improving patch retrieval on image retrieval performance. For this purpose, we introduce a new dataset, "RomePatches", using images and the 3D reconstruction of [1]. The 16K Flickr images of "RomePatches" represent views of 66 different locations in Rome. The 3D reconstruction provides sparse patch matches, yielding the ground truth for our patch retrieval dataset. This allows relating performance improvements for both patch and image retrieval tasks. In a nutshell, our main contributions are three-fold:

1. We propose a patch descriptor based on a CKN architecture [27], using a fast and simple stochastic procedure to compute an explicit feature embedding.

2. We introduce and make available a dataset, named "Rome-Patches", for the evaluation of patch *and* image retrieval, enabling to systematically study the correlation between patch matching and image retrieval performance.

3. We show that, for the purpose of patch and image retrieval, it is possible to learn competitive patch-level descriptors without supervision, and therefore at a fraction of the computational and annotation cost compared to previous supervised alternatives [25, 11].

Overview. We review related work in Sec. 2. In Sec. 3, we describe our image retrieval pipeline, and devote Sec. 4 to convolutional descriptors. Our new dataset is introduced in Sec. 5, and experimental results are presented in Sec. 6. Our new dataset as well as the code to extract Patch-CKN are available online[2].

2. Related Work

Our literature review focuses on the works which are closest to ours: shallow patch descriptors, deep learning for image retrieval and deep learning for patch description.

Traditional patch descriptors. Among the variety of standard patch descriptors, SIFT [26] is the most widely used. Interpreted as a convolutional net, SIFT is a two-layer architecture, the first layer computing patch gradient orientations, average-pooled in the second one. SIFT has been successfully used for many tasks such as stereo matching [1], content-based retrieval [16], or classification [33]. Mikolajczyk *et al*. [29] provide a detailed survey of local descriptors and demonstrate the excellent performance of SIFT. Improved local descriptors include BRIEF [8] and LIOP [45]. All these descriptors are hand-crafted and have been optimized by grid-search on a relatively small amount of parameters. When the number of parameters to be set is large, such as approach is infeasible and the optimal parametrization needs to be learned from data.

Most works on hand-crafted descriptor learning use supervision. Brown *et al*. [7, 46] designed a matching dataset from reprojected 3D models of landmarks, obtained by structure from motion, with a descriptor consisting of several existing parts, including but not limited to SIFT, GLOH [29] and Daisy [42]. We do not include their dataset in our experiments, because of significant differences in early stages of the pipeline as the Multi-view Stereo Correspondence Dataset contains few images of just three locations, with grey-scale patches extracted (while in this work we leverage the additional color information), that were extracted with a detector that is incompatible with ours (DoG instead of Hessian-affine). Philbin *et al*. [37] learn a Mahalanobis metric for SIFT descriptors to compensate the binarization error, with excellent results in instance-based retrieval. Simonyan *et al*. [40] propose the "Pooling Regions" descriptor and learn its parameters, as well as a linear projection using stochastic optimization. Their learning objective can be cast as a convex optimization problem, which is not the case for classical convolutional networks.

An exception is [5] which presents a match-kernel interpretation of SIFT, and a family of kernel descriptors whose parameters are learned in an unsupervised fashion. The Patch-CKN we introduce generalizes kernel descriptors; the proposed procedure for computing an explicit feature embedding is faster and simpler.

[2] lear.inrialpes.fr/people/paulin/projects/RomePatches

descriptor	application	supervision	#parameters		optimization method
SIFT [26]	sparse features	N/A	2	0	N/A
Daisy [46]	patch matching	class = 3D location	0	10	Powel's conjugate direction method
AlexNet [22]	image classification	object classes	50	70M	backpropagation, SGD
Neural codes [3]	same-image recognition	landmark images	50	70M	fine-tuning on top of AlexNet
PhilippNet [12]	patch matching	artificial classes	10	10k	backpropagation, SGD
Fracking [39]	patch matching	match/non-match	10	46k	backpropagation, SGD
CKN [27]	image classification	no supervision	10	256k	layer-wise SGD

Table 1. Levels of supervision and optimization methods used by the approaches related to this work. There are two columns for parameters: hyper-parameters (tuned by hand) and parameters determined by the optimization method.

Deep learning for image retrieval. With a CNN learned on a sufficiently large labeled set such as ImageNet [9], the output of its intermediate layers can be used as image descriptors for a wide variety of tasks including image retrieval [3, 38] – the focus of this work. The output of one of the fully-connected layers is often chosen because it is compact, usually 4,096 D. However, global CNN descriptors lack geometric invariance [14], so they produce results below the state-of-the-art in instance-level image retrieval. Hence, improvements have been proposed.

In [38, 14], CNN responses at different scales and positions are extracted. We proceed similarly, yet we replace the (coarse) dense grid with a patch detector. There are important differences between [38, 14] and our work. While [14, 38] use the output of the penultimate layer as patch descriptor, we show in our experiments that we can get improved results with the output of preceding layers, that are cheaper to compute. In [3], the authors use a single global CNN descriptor for instance-based image retrieval and fine-tune the descriptor on a surrogate landmark dataset. While fine-tuning improves results, it would be difficult to replicate this success beyond landmarks. Finally, [21] proposes a Siamese architecture to train image retrieval descriptors but does not report results on standard retrieval benchmarks.

Deep patch descriptors. Recently [25, 12, 39] reported superior results to SIFT for tasks such as patch matching or patch classification. The three works use different levels of supervision to train a CNN: category labels in [25], surrogate patch labels in [12] (each class is a given patch under different transformations) and matching/non-matching pairs in [39]. There are two key differences between those works and ours. First, they focus on patch-level metrics, instead of actual image retrieval. Second, and more importantly, while all these approaches require some kind of supervision, we show that our Patch-CKN yields competitive performance in both patch matching and image retrieval without requiring supervision. Especially, with respect to [25, 39] we do not need costly labels. And compared to [12] we do not need to make arbitrary choices in the definition of classes (*i.e*. the set of transformations). Table 1 summarizes the competing approaches.

3. Image Retrieval Pipeline

We briefly present the three-step pipeline: interest point detection, patch description, and patch matching.

Interest point detection. Interest point detectors provide locations invariant to certain image transformations. This ensures that two views of the same scene even with changes in viewpoint or illumination share similar "interest points", see [30] for a review of detectors. We use the popular Hessian-Affine detector [28]. The idea is to extract points at their characteristic scale and estimate for each point an affine-invariant local region, see Fig. 1. Rotation invariance is obtained by rotating patches to align the dominant gradient orientation. This results in a set of interest points associated with locally affine-invariant regions.

Interest point description. Given a normalized patch M obtained by mapping the affine region to a fixed-size square, we compute its feature representation $\phi(M)$ in a Euclidean space. The representation is expected to be robust to the perturbations that are not covered by the detector (lighting changes, small rotations, blur,...).

Patch matching. Because matching all possible pairs of patches is too expensive, we follow the standard practice of encoding the patch descriptors and aggregating them into a fixed-length image descriptor, using the VLAD representation [18]. Given a clustering of the feature space consisting of k centroids $\{c_1, \ldots, c_k\}$, VLAD encodes a set of descriptors as the total shift with respect to their assigned centroid. A power normalization with exponent 0.5 is then applied to the VLAD descriptor, as well as an L_2 normalization.

4. Convolutional Descriptors

We use convolutional features to encode fixed-size image patches (size 51×51 pixels). CNNs are normally trained with class supervision for a classification task. This can be extended to image retrieval by either: (i) encoding local descriptors with a model that has been trained for an unrelated image classification task, see Section 4.1; (ii) devising a surrogate classification problem that is as related as possible to image retrieval; (iii) using unsupervised learning, such as a convolutional kernel network, see Sec. 4.2.

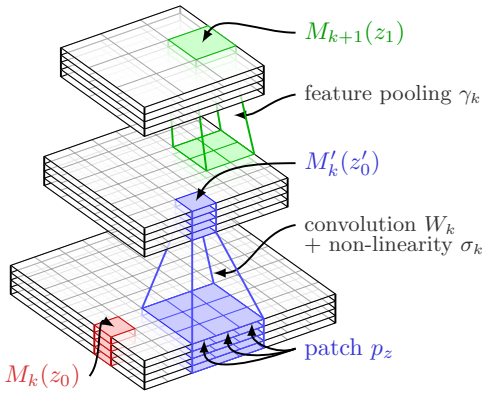


Figure 2. A typical organization for two successive layers of a CNN. The spatial map $M_{k'}$ is obtained from M_k by convolution and pointwise non-linearity, and the top layer M_{k+1} is obtained from M'_k by a downsampling operation called feature pooling. By convention the map M_0 corresponds to the input image x.

4.1. Convolutional Neural Networks

Convolutional neural nets transform an input image by a sequence of simple operations or layers. Each layer performs a linear operation followed by a pointwise non-linearity. Formally, the output $f(x)$ of a CNN for some image x represented as a vector is

$$f(x) = \gamma_K(\sigma_K(W_K \ldots \gamma_2(\sigma_2(W_2\gamma_1(\sigma_1(W_1 x))\ldots)), \quad (1)$$

where the terms W_k are matrices corresponding to linear operations, the functions σ_k are pointwise non-linear functions, e.g., sigmoids or rectified linear units, and the functions γ_k perform a downsampling operation (feature pooling). For a convolutional layer, the matrices W_k have a particular structure and correspond to convolutions of spatial maps, as illustrated in Fig. 2. When they are dense and unstructured, the layer is called "fully connected".

Learning from category labels. The most popular off-the-shelf CNN is AlexNet [22], which won the ImageNet 2012 challenge. AlexNet has 7 layers: the first five are convolutional and the last ones are fully connected. The network is designed to process images of size 224×224, but convolutional layers may be fed with smaller inputs to produce 1x1 maps that we can use as low-dimensional patch descriptors – see the "coverage" column in Table 3. To ensure a fair comparison between all approaches, we rescale patches to always produce a 1×1 map.

Learning from surrogate labels. Most CNNs such as AlexNet augment the dataset with perturbed versions of training patches to learn the filters W_k in (1). The authors of [11, 12] use "virtual patches", obtained as transformations of randomly extracted ones to fall back to a classification problem. For a set of patches $\mathcal{P}$, and a set a transformations $\mathcal{T}$, the dataset consists of all $\tau(p)$, $(\tau, p) \in \mathcal{T} \times \mathcal{P}$. Transformed versions of the same patch share the same label, thus defining surrogate classes. In this paper, we evaluate this strategy by using the same architecture and filter values, called PhilippNet, as in [12]. The network has three convolutional and one fully connected layers, takes as input 64x64 patches, and produces a 512-dimensional output.

4.2. Convolutional Kernels Networks

CKNs have the same architecture as classical CNNs presented in Eq. (1) and in Fig. 2. The feature representation of CNNs relies on filters that are learned and hence defined in a data-dependent manner. We define here a feature representation that is based on a kernel (feature) map. The exact version of this feature representation is therefore data-independent. An explicit kernel (feature) map can be computed [34, 44, 27] to approximate it for computational efficiency. We present here a fast and simple procedure for this purpose, using sub-sampling of patches and stochastic gradient optimization, yielding a CKN that outputs patch descriptors.

Let M and M' be two patches of size $m \times m$ ($m = 51$ in this paper), and $\Omega = \{1, \ldots, m\}^2$ be the set of pixel locations. Let us also consider a fixed sub-patch size and denote by p_z the sub-patch from M (resp. p'_z the sub-patch from M') centered at location $z \in \Omega$.[3]

Single-layer kernel definition. We consider the following kernel [27]:

$$K_1(M, M') = \sum_{z, z' \in \Omega} e^{-\|z - z'\|^2 / 2\beta_1^2} k_1(p_z, p'_{z'}), \quad (2)$$

where

$$k_1(p_z, p'_{z'}) = \|p_z\| \|p'_{z'}\| e^{-\|\tilde{p}_z - \tilde{p}'_{z'}\|^2 / 2\alpha_1^2}, \quad (3)$$

α_1 and β_1 are two kernel hyperparameters, $\|\cdot\|$ denotes the usual L_2 norm, and $\tilde{p}_z$ and $\tilde{p}'_{z'}$ are L_2-normalized versions of the sub-patches p_z and $p'_{z'}$.

The corresponding kernel (feature) map defines a feature representation for patches and images. Furthermore, the kernel is a match kernel. Therefore, the kernel offers a tunable level of invariance through the choice of hyperparameters, and produces hierarchical convolutional representations that are well-suited for natural images.

Kernel embedding approximation. Since the exact computation of (2-3) is overwhelming, Mairal *et al.* propose an explicit finite-dimensional embedding [34, 44] to approximate it. The embedding of [27] keeps the 2-D spatial structure, similar to CNN feature maps. For the one-layer CKN, the approximation of [27] is:

$$K_1(M, M') \approx \sum_{u \in \Omega_1} g_1(u; M)^T g_1(u; M')$$

[3] In practice, sub-patches near the border of M which have values outside of the domain Ω are discarded from the sum (2).

with for all $u \in \Omega_1$,

$$g_1(u; M) := \sum_{z \in \Omega} e^{-\|u-z\|^2/2\beta_1^2} h_1(z; M),$$

and, for all $z \in \Omega$,

$$h_1(z; M) := \|p_z\| \left[\sqrt{\eta_j} e^{-\|w_j - \tilde{p}_z\|^2/\alpha_1^2}\right]_{j=1}^{n_1},$$

where Ω_1 is a subset of Ω as in [27] and w and η are learned parameters. There are two distinct approximations, one in the subsampling defined by $|\Omega_1| \leq |\Omega|$ that corresponds to the stride of a CNN pooling operation, and one in the embedding of the Gaussian kernel of the subpatches: $k_1(p_z, p'_{z'}) \approx h_1(z; M) h_1(z'; M')$.

Since $K_1(M, M')$ is a sum of the match-kernel terms, we can approximate it at sub-patch level by solving an optimization problem. In contrast to the original formulation in Eq. 4 of [27], we introduce the change of variables

$$b_j = \log(\eta_j)/2 - (1 + \|w_j\|^2)/\alpha_1^2$$
$$\omega_j = 2w_j/\alpha_1^2$$

and, considering a sample of n pairs of sub-patches $\{(p_i, p'_i)\}_{i=1,\dots,n}$, we solve:

$$\min_{w_j, \eta_j} \sum_{i=1}^{n} \left(e^{-\frac{\|\tilde{p}_i - \tilde{p}'_i\|^2}{2\alpha_1^2}} - \sum_{j=1}^{n_1} \eta_j e^{-\frac{\|w_j - \tilde{p}_i\|^2}{\alpha_1^2}} e^{-\frac{\|w_j - \tilde{p}'_i\|^2}{\alpha_1^2}} \right)^2$$

We use stochastic gradient optimization to find a stationary point of this (non-convex) objective. This is much faster than the original L-BFGS optimizer [27]; see Sec. 6.

Multi-layer CKN kernel. A kernel can be overlaid on top of the single kernel for a "deeper" and potentially better feature representation [4]. Given an input patch M, the single-layer CKN defines an approximation $f_1(M)$ that can be interpreted as a spatial map. It is possible to define a kernel K_2 on this map in the same way as we have done for input patches. For that, we simply define a patch size, new hyper-parameters β_2 and α_2, and replace M, M' by $f_1(M), f_1(M')$ in all equations of the previous section. Figure 3 gives an illustration of the corresponding two-layer convolutional kernel. Training a multi-layer CKN is naturally sequential, one layer after another.

Input types. We investigate three possible inputs for our CKNs. The first, CKN-raw, directly feeds the raw RGB patch to the network. This scheme captures the hue information, which can prove a drawback in certain situations.

CKN-white consists in pre-processing each sub-patch of the CKN's first layer, by subtracting their mean color, and using PCA-whitening, with a PCA learned on all sub-patches of the initial patch. This responds only to local variations inside the sub-patch, and makes the network more invariant to color.

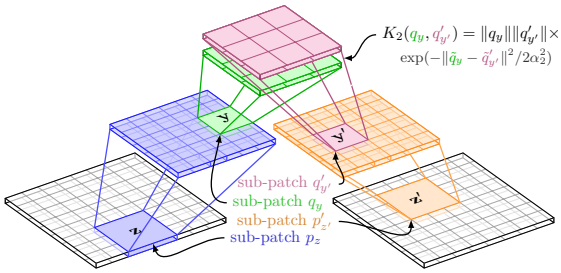


Figure 3. The two-layer convolutional kernel architecture. Each layer is a weighted match kernel between all patches of the previous one; q_z (resp $q'_{z'}$) is a sub-patch of p_z (resp $p'_{z'}$), which is a itself a sub-patch of M (resp M'). The two-layer CKN provides an approximate explicit feature map of this kernel. See [27] for details.

CKN-grad is fully invariant to color. It is the gradient along each spatial dimension with 1×1 sub-patches – that is, the sub-patch $\tilde{p}_z$ for this first layer is simply two-dimensional and can be written $p_z = (G_x, G_y)$. Because the features are normalized, the inner part of the match kernel $\|\tilde{p}_z - \tilde{p}'_{z'}\|$ is directly linked to the cosine of the angle between the two gradients, see [5, 27]. Indeed, an explicit approximation of the kernel K_1 with n_1 evenly distributed orientations $\theta_j = 2j\pi/n_1,\ j \in \{1, \dots, n_1\}$ writes:

$$e^{-\|\tilde{p}_z - \tilde{p}'_{z'}\|^2/2\alpha_1^2} \approx \sum_{j=1}^{n_1} \varphi_1(j; p_z) \varphi_1(j; p'_{z'}),$$

where for all j,

$$\varphi_1(j, p_z) = e^{-\left((\cos\theta_j - G_x/\rho)^2 + (\sin\theta_j - G_y/\rho)^2\right)/\alpha_1^2}$$

and $\rho = \sqrt{G_x + G_y}$. This formulation can be interpreted as a soft-binning of gradient orientations in a "histogram" of size n_1. To ensure an adequate distribution in each bin, we set $\alpha_1 = \left((1 - \cos(2\pi/n_1))^2 + \sin(2\pi/n_1)^2\right)^{1/2}$.

5. Datasets

We conduct experiments for two tasks, patch and image retrieval. We introduce a new dataset for both, which we describe in this section, together with the standard benchmarks.

5.1. Patch retrieval

The Mikolajczyk Dataset. Designed to benchmark interest points detectors and descriptors, the Mikolajczyk dataset [29] contains a set of 8 scenes with 6 images for each. Images of a scene are linked by a homography. We extract regions with the Hessian-Affine detector, and match the corresponding descriptors using Euclidean nearest neighbor. The match between a pair of ellipses is counted correct if the projection of the region with the

ground-truth homography to the second image overlaps by at least 50%. Mean average precision (mAP) is used as performance measure.

RomePatches. Since the existing patch retrieval datasets we are aware of do not contain color information and are not extracted with our detector (Hessian-Affine), we introduce a new dataset[4]. Similar to [46], we use the 3D-reconstruction of landmarks to get different views of the same location. We use the Rome16K dataset [24], which consists of 16,179 images of locations in Rome, downloaded from photo sharing sites. Images are partitioned in 66 "bundles", each one containing a set of viewpoints of a given location in Rome (e.g. "Trevi Fountain"). Within a bundle, consistent camera parameters are available for all images[5]. We match the SIFT descriptors of all images using product quantization [17]. Then we keep only matches that verify the epipolar constraint within a tolerance of 3 pixels. Pairwise point matches are then aggregated greedily to form larger groups of 2D points viewed from several cameras. Groups are merged while the reproduction error from the estimated 3D position is below the 3 pixel threshold. Fig. 4 shows matching patches extracted with this algorithm. We split the dataset into two sets of bundles, the train set with 44 bundles on which we are allowed to learn parameters and tune hyper-parameters. The remaining 22 bundles form the test set. From the train as well as the test set, we select 1,000 3D points that are viewed in at least 10 different images and use one as a query and nine randomly sampled as the targets. Our dataset therefore contains 9,000 target points, and 1,000 queries for the train as well as the test set, i.e., a total of 20,000 patches. We report mean average precision (mAP).

5.2. Image Retrieval

RomePatches-Image. Using the aforementioned bundle split, we select 1,000 query images and 1,000 target images evenly distributed over all bundles for both train and test splits. Two images are considered to match if they come from the same bundle, as illustrated in Fig. 4.

Oxford. The Oxford dataset [35] involves 5,000 images of Oxford landmarks. 11 locations in the city are selected as queries. Each location is represented by 5 bounding boxes each extracted from a different image. Given one of the 55 bounding boxes, the task is to find all images of the same location.

UKbench and Holidays. The University of Kentucky benchmark is a set of 10,200 photos. Each group of 4 images represents the same object. Each image is used as a query in turn. The Holidays dataset contains 1,491 photos of scenes and objects. 500 images are used as queries and the queries are excluded from the datasets.

The standard metrics are mAP for Oxford, Paris and Holidays and 4×recall@4 for UKB.

[4]Available online at http://lear.inrialpes.fr/people/paulin/projects/RomePatches/

[5]http://www.cs.cornell.edu/projects/p2f/

Figure 4. Patch and image retrieval on the Rome dataset. Top: examples of matching patches. Bottom: Images of the same bundle, that therefore share the same class for image retrieval.

6. Experimental Results

After describing implementation details, we report results for patch and image retrieval.

6.1. Implementation details

As our goal is to optimize local descriptors, all methods are given the same patch information as input (computed at Hessian-affine interest points), and are evaluated with the same global descriptor (VLAD with 256 centroids). We believe that improvements in feature detection and aggregation would benefit all architectures equally, without changing the relative performance of patch descriptors.

Patch extraction. As input for all methods, we use 51×51 pixel patches, which was found to be optimal on SIFT descriptors for the Oxford dataset.

CNN implementation. For CNNs, we use the popular Caffe framework [20], and the provided AlexNet (learned on ImageNet 2012). For the PhilippNet [12], we used the model provided by the authors. As explained in section 4, we rescale the 51x51 input patches to the size that, when fed to the CNN, produces 1x1 output maps. Rescaling artifacts do not have a noticeable impact compared to re-extracting patches.

Details of CKN learning. AlexNet and PhilippNet are provided with their parameters, we only learn CKNs. To do so, we randomly select a set of 100K patches in the train split of RomePatches. For each layer, 1 million sub-patches corresponding to convolution areas are extracted and all pairs of patches are fed to the objective function (4.2). The SGD optimization is run for 300K iterations with a batchsize of

1000. Because the objective is nonconvex, several tricks were used, such as random initialization, preconditioning (optimization is conducted in a space where the patch entries are decorrelated), selecting an initial learning rate in the range $\{1, 2^{-1/2}, 2^{-1}, \ldots, 2^{-20}\}$ by performing 1K iterations and choosing the one giving the lowest objective evaluated on a validation set [6]; after choosing the learning rate, we keep monitoring the objective on a validation set every 1K iteration, and perform backtracking in case of divergence. The learning rate is also divided by $\sqrt{2}$ every 50K iterations. These heuristics are fixed over all experiments. Training a CKN takes roughly 10 min on a GPU compared to 2-3 days for the L-BFGS implementation of [27]. As CKN and CNN share the same architecture, the descriptor extraction time is similar for all convolutional methods.

6.2. Patch retrieval

Because the evaluation is computationally cheaper for the patch retrieval task than for image retrieval (10K patches to encode for RomePatches, against more than 4M for Holidays), we optimize the hyperparameters of our CKNs on the RomePatches dataset. We select the best parameters on the train split, without accessing the test data.

Parametric exploration of CKNs. We explore the three input types separately. For each layer, four hyperparameters have to be determined: the size of the convolutional mask (sub-patch size), the coefficient α_k, the pooling factor and the number of outputs (n_k). The spatial comparison coefficient β_k is related to the pooling factor and is set as in [27]—that is, to the pooling factor divided by $\sqrt{2}$. We determine α_k as a quantile σ_k of the distribution of pairwise distances between sub-patches. This value was found optimal at 10^{-3} for all architectures, a much smaller value than reported in [27], which suggests that image classification requires more invariance than patch matching.

As mentioned before, we optimize these parameters over the train split of RomePatches. We try the values 2, 3, 4 and 5 for the sub-patch sizes and pooling factors with 128, 256, 512 or 1024 outputs. The α parameter was selected in the $\{0.1, 0.01, 0.001\}$ quantiles. The retained parameters are given in Table 2. To the notable exception of color, architectures perform better with two layers. In general, the higher the number of features, the better performance.

Input	Layer 1	Layer 2	dim.
CKN-raw	5x5, 5, 512	—	41472
CKN-white	3x3, 3, 512	2x2, 2, 512	32768
CKN-grad	1x1, 3, 16	4x4,2,1024	50176

Table 2. For each layer we indicate the sub-patch size, the subsampling factor and the number of filters. For the gradient network, the value 16 corresponds to the number of orientations.

In the following, we use the best architectures given in Table 2 for each input type.

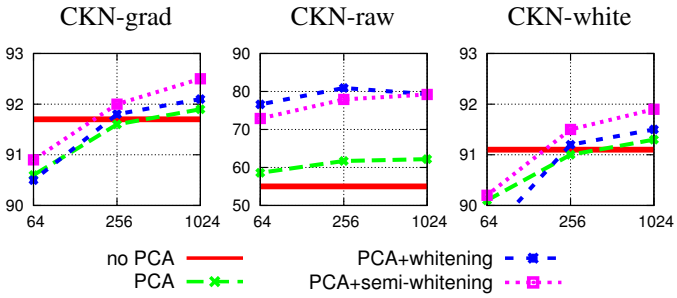


Figure 5. Influence of dimensionality reduction on patch retrieval performance. Results reported in mAP (%) on the train split of RomePatches as a function of the PCA dimension. As a comparison, SIFT reports 91.6%.

Architecture	coverage	Dim	RomePatches train	RomePatches test	Miko.
SIFT	51x51	128	91.6	87.9	57.8
AlexNet-conv1	11x11	96	66.4	65.0	40.9
AlexNet-conv2	51x51	256	73.8	69.9	46.4
AlexNet-conv3	99x99	384	81.6	79.2	53.7
AlexNet-conv4	131x131	384	78.4	75.7	43.4
AlexNet-conv5	163x163	256	53.9	49.6	24.4
PhilippNet	64x64	512	86.1	81.4	59.7
CKN-grad	51x51	1024	**92.5**	**88.1**	59.5
CKN-raw	51x51	1024	79.3	76.3	50.9
CKN-white	51x51	1024	91.9	87.7	**62.5**

Table 3. Results of convolutional architectures for patch retrieval.

Comparative results. We compare the convolutional architectures on our three patch datasets: RomePatches-train, RomePatches-test and Mikolajczyk. Results are given in Table 3. For AlexNet CNNs, we report results for all outputs of the 5 convolutional layers (after ReLU). We note that SIFT is an excellent baseline for these methods, and that CNN architectures that were designed for local invariances perform better than the ones used in AlexNet, as observed in [12]. The results of the PhilippNet on the Mikolajczyk dataset are different from the ones reported in [12], for several reasons. First, we evaluate on Hessian-Affine descriptors while they use MSER. To have a comparable setting, we use their network with an input of 64x64, while they slide it on 91x91 patches. Such an additional layer results in a small increase of performance (2% for patch retrieval and 1% for image retrieval). We observe that PhilippNet outperforms both SIFT and AlexNet, which was the conclusion of [12]; CKN trained on whitened patches do however yield better results.

6.3. Image Retrieval

Settings. We learn a vocabulary of 256 centroids on a related database: for Holidays and UKB we use 5000 Flickr images and for Oxford, we train on Paris [36]. For

	Holidays	UKB	Oxford	Rome	
				train	test
SIFT	64.0	3.44	43.7	52.9	62.7
AlexNet-conv1	59.0	3.33	18.8	28.9	36.8
AlexNet-conv2	62.7	3.19	12.5	36.1	21.0
AlexNet-conv3	**79.3**	3.74	33.3	47.1	54.7
AlexNet-conv4	77.1	3.73	34.3	47.9	55.4
AlexNet-conv5	75.3	3.69	33.4	45.7	53.1
PhilippNet	74.1	3.66	38.3	50.2	60.4
CKN-grad	66.5	3.42	**49.8**	**57.0**	**66.2**
CKN-raw	69.9	3.54	23.0	33.0	43.8
CKN-white	78.7	3.74	41.8	51.9	62.4
CKN-mix	**79.3**	**3.76**	43.4	54.5	65.3

Table 4. Image retrieval results. CKN-mix is the result of the concatenation of the VLAD descriptors for the three channels.

RomePatches-Train and RomePatches-Test the vocabulary is learned the other one. The final VLAD descriptor size is 256 times the local descriptor dimension.

Comparative results. We compare all convolutional approaches as well as the SIFT baseline in the image retrieval settings. Results are summarized in Table 4.

On datasets for which color is dominant (e.g. Holidays or UKB), the best individual CKN results are attained by CKN-white, improved by combining the three channels. On images of buildings, gradients still perform best and the addition of color channels is harmful, which also explains the poor performance of AlexNet. On the other hand, PhilippNet was trained to be invariant to colorimetric transformations, and therefore yields better results than its CNN counterpart.

Comparison with the state of the art. Table 5 compares our approach to recently published results. Approaches based on VLAD with SIFT [2, 19] can be improved significantly by CKN local descriptors (+15% on Holidays). To compare to the state of the art with SIFT on Oxford [2], we use the same Hessian-Affine patches extracted with gravity assumption [32]. Note that this alone results in a 7% gain.

We also compare with global CNNs [3]. Our approach outperforms it on Oxford and UKB and is on par on Holidays. On Holidays, our approach is slightly below the one of [14], that uses AlexNet descriptors and VLAD pooling on large, densely extracted patches. Note that they perform dimensionality reduction and whitening, which results in a 2% improvement. We plan to investigate dimensionality reduction methods [3, 15] as well as quantization [17] in future work.

7. Conclusion

We propose a new descriptor Patch-CKN for patch and image retrieval, that performs on par or better than supervised CNNs on standard patch and image retrieval benchmark datasets and on the proposed RomePatches benchmark dataset.

Method \ Dataset	Holidays	UKB	Oxford
VLAD [19]	63.4	3.47	-
VLAD++ [2]	64.6	-	55.5*
Global-CNN [3]	79.3	3.56	54.5
MOP-CNN [14]	**80.2**	-	-
Ours	79.3	**3.76**	49.8 (**56.5***)

Table 5. Comparison with state-of-the-art image retrieval results. Results with * use a Hessian-Affine detector with gravity assumption [32].

Acknowledgements. This work was partially supported by projects "Allegro" (ERC), "Titan" (CNRS-Mastodons), "Macaron" (ANR-14-CE23-0003-01), the Moore-Sloan Data Science Environment at NYU and a Xerox Research Center Europe collaboration contract.

References

[1] S. Agarwal, Y. Furukawa, N. Snavely, I. Simon, B. Curless, S. M. Seitz, and R. Szeliski. Building Rome in a day. *Communications of the ACM*, 2011. 2

[2] R. Arandjelovic and A. Zisserman. All about VLAD. In *CVPR*, 2013. 8

[3] A. Babenko, A. Slesarev, A. Chigorin, and V. Lempitsky. Neural codes for image retrieval. In *ECCV*, 2014. 1, 2, 3, 8

[4] L. Bo, K. Lai, X. Ren, and D. Fox. Object recognition with hierarchical kernel descriptors. In *CVPR*, 2011. 5

[5] L. Bo, X. Ren, and D. Fox. Kernel descriptors for visual recognition. In *NIPS*, 2010. 2, 5

[6] L. Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*. Springer, 2012. 7

[7] M. Brown, G. Hua, and S. Winder. Discriminative learning of local image descriptors. *PAMI*, 2011. 2

[8] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. BRIEF: Binary robust independent elementary features. In *ECCV*, 2010. 2

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009. 3

[10] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, 2014. 1

[11] A. Dosovitskiy, J. T. Springenberg, M. Riedmiller, and T. Brox. Discriminative unsupervised feature learning with convolutional neural networks. *NIPS*, 2014. 2, 4

[12] P. Fischer, A. Dosovitskiy, and T. Brox. Descriptor matching with convolutional neural networks: a comparison to SIFT. arXiv Preprint, 2014. 1, 3, 4, 6, 7

[13] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 1, 2

[14] Y. Gong, L. Wang, R. Guo, and S. Lazebnik. Multi-scale orderless pooling of deep convolutional activation features. In *ECCV*, 2014. 3, 8

[15] H. Jégou and O. Chum. Negative evidences and cooccurrences in image retrieval: the benefit of PCA and whitening. In *ECCV*, 2012. 8

[16] H. Jegou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *ECCV*. 2008. 2

[17] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *PAMI*, 2011. 6, 8

[18] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *CVPR*, 2010. 3

[19] H. Jégou, F. Perronnin, M. Douze, J. Sánchez, P. Pérez, and C. Schmid. Aggregating local image descriptors into compact codes. *PAMI*, 2012. 1, 8

[20] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. 2014. 6

[21] J. Jiang, Y. Song, T. Leung, C. Rosenberg, J. Wang, J. Philbin, B. Chen, and Y. Wu. Learning fine-grained image similarity with deep ranking. In *CVPR*, 2014. 3

[22] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012. 1, 3, 4

[23] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Handwritten digit recognition with a back-propagation network. *NIPS*, 1989. 1

[24] Y. Li, N. Snavely, and D. P. Huttenlocher. Location recognition using prioritized feature matching. In *ECCV*. 2010. 6

[25] J. Long, N. Zhang, and T. Darrell. Do Convnets learn correspondances? In *NIPS*, 2014. 1, 2, 3

[26] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004. 1, 2, 3

[27] J. Mairal, P. Koniusz, Z. Harchaoui, and C. Schmid. Convolutional kernel networks. In *NIPS*, 2014. 2, 3, 4, 5, 7

[28] K. Mikolajczyk and C. Schmid. Scale & affine invariant interest point detectors. *IJCV*, 2004. 3

[29] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *PAMI*, 2005. 2, 5

[30] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool. A comparison of affine region detectors. *IJCV*, 2005. 1, 3

[31] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *CVPR*, 2014. 1, 2

[32] M. Perdoch, O. Chum, and J. Matas. Efficient representation of local geometry for large scale object retrieval. In *CVPR*, 2009. 8

[33] F. Perronnin and C. Dance. Fisher kernels on visual vocabularies for image categorization. In *CVPR*, 2007. 2

[34] F. Perronnin, J. Sánchez, and Y. Liu. Large-scale image categorization with explicit data embedding. In *CVPR*, 2010. 2, 4

[35] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *CVPR*, 2007. 6

[36] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *CVPR*, 2008. 8

[37] J. Philbin, M. Isard, J. Sivic, and A. Zisserman. Descriptor learning for efficient retrieval. In *ECCV*. 2010. 2

[38] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. *arXiv Preprint*, 2014. 3

[39] E. Simo-Serra, E. Trulls, L. Ferraz, I. Kokkinos, and F. Moreno-Noguer. Fracking deep convolutional image descriptors. *Arxiv preprint*, 2015. 1, 3

[40] K. Simonyan, A. Vedaldi, and A. Zisserman. Learning local feature descriptors using convex optimisation. *PAMI*, 2014. 2

[41] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *ICCV*, 2003. 1

[42] E. Tola, V. Lepetit, and P. Fua. Daisy: An efficient dense descriptor applied to wide-baseline stereo. *PAMI*, 2010. 2

[43] T. Tuytelaars and K. Mikolajczyk. Local invariant feature detectors: A survey. *Foundations and Trends in Computer Graphics and Vision*, 2008. 1

[44] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. *TPAMI*, 2012. 2, 4

[45] Z. Wang, B. Fan, and F. Wu. Local intensity order pattern for feature description. In *ICCV*, 2011. 2

[46] S. Winder, G. Hua, and M. Brown. Picking the best daisy. In *CVPR*, 2009. 2, 3, 6

[47] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *NIPS*, 2014. 2

FlowNet: Learning Optical Flow with Convolutional Networks

Alexey Dosovitskiy*, Philipp Fischer†,*, Eddy Ilg*, Philip Häusser, Caner Hazırbaş, Vladimir Golkov†
University of Freiburg — Technical University of Munich
{fischer,dosovits,ilg}@cs.uni-freiburg.de, {haeusser,hazirbas,golkov}@cs.tum.edu

Patrick van der Smagt
Technical University of Munich
smagt@brml.org

Daniel Cremers
Technical University of Munich
cremers@tum.de

Thomas Brox
University of Freiburg
brox@cs.uni-freiburg.de

Abstract

Convolutional neural networks (CNNs) have recently been very successful in a variety of computer vision tasks, especially on those linked to recognition. Optical flow estimation has not been among the tasks CNNs succeeded at. In this paper we construct CNNs which are capable of solving the optical flow estimation problem as a supervised learning task. We propose and compare two architectures: a generic architecture and another one including a layer that correlates feature vectors at different image locations. Since existing ground truth data sets are not sufficiently large to train a CNN, we generate a large synthetic Flying Chairs dataset. We show that networks trained on this unrealistic data still generalize very well to existing datasets such as Sintel and KITTI, achieving competitive accuracy at frame rates of 5 to 10 fps.

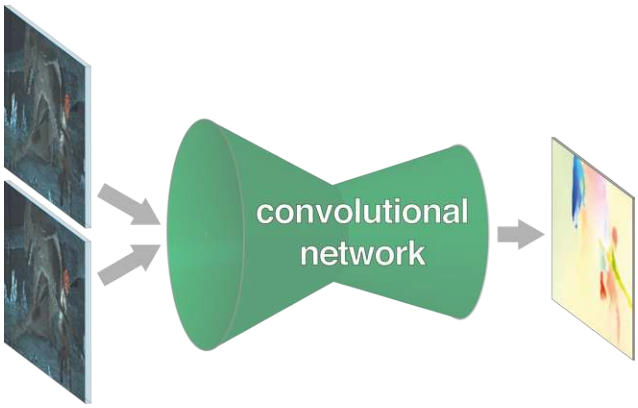


Figure 1. We present neural networks which learn to estimate optical flow, being trained end-to-end. The information is first spatially compressed in a contractive part of the network and then refined in an expanding part.

1. Introduction

Convolutional neural networks have become the method of choice in many fields of computer vision. They are classically applied to classification [25, 24], but recently presented architectures also allow for per-pixel predictions like semantic segmentation [28] or depth estimation from single images [10]. In this paper, we propose training CNNs end-to-end to learn predicting the optical flow field from a pair of images.

While optical flow estimation needs precise per-pixel localization, it also requires finding correspondences between two input images. This involves not only learning image feature representations, but also learning to match them at different locations in the two images. In this respect, optical flow estimation fundamentally differs from previous applications of CNNs.

Since it was not clear whether this task could be solved with a standard CNN architecture, we additionally developed an architecture with a correlation layer that explicitly provides matching capabilities. This architecture is trained end-to-end. The idea is to exploit the ability of convolutional networks to learn strong features at multiple levels of scale and abstraction and to help it with finding the actual correspondences based on these features. The layers on top of the correlation layer learn how to predict flow from these matches. Surprisingly, helping the network this way is not necessary and even the raw network can learn to predict optical flow with competitive accuracy.

Training a network to predict generic optical flow requires a sufficiently large training set. Although data augmentation does help, the existing optical flow datasets are still too small to train a network on par with state of the art. Getting optical flow ground truth for realistic video material is known to be extremely difficult [7]. Trading in realism

*These authors contributed equally

†Supported by the Deutsche Telekom Stiftung

for quantity, we generate a synthetic Flying Chairs dataset which consists of random background images from Flickr on which we overlay segmented images of chairs from [1]. These data have little in common with the real world, but we can generate arbitrary amounts of samples with custom properties. CNNs trained on just these data generalize surprisingly well to realistic datasets, even without fine-tuning.

Leveraging an efficient GPU implementation of CNNs, our method is faster than most competitors. Our networks predict optical flow at up to 10 image pairs per second on the full resolution of the Sintel dataset, achieving state-of-the-art accuracy among real-time methods.

2. Related Work

Optical Flow. Variational approaches have dominated optical flow estimation since the work of Horn and Schunck [19]. Many improvements have been introduced [29, 5, 34]. The recent focus was on large displacements, and combinatorial matching has been integrated into the variational approach [6, 35]. The work of [35] termed DeepMatching and DeepFlow is related to our work in that feature information is aggregated from fine to coarse using sparse convolutions and max-pooling. However, it does not perform any learning and all parameters are set manually. The successive work of [30] termed EpicFlow has put even more emphasis on the quality of sparse matching as the matches from [35] are merely interpolated to dense flow fields while respecting image boundaries. We only use a variational approach for optional refinement of the flow field predicted by the convolutional net and do not require any handcrafted methods for aggregation, matching and interpolation.

Several authors have applied machine learning techniques to optical flow before. Sun *et al*. [32] study statistics of optical flow and learn regularizers using Gaussian scale mixtures; Rosenbaum *et al*. [31] model local statistics of optical flow with Gaussian mixture models. Black *et al*. [4] compute principal components of a training set of flow fields. To predict optical flow they then estimate coefficients of a linear combination of these 'basis flows'. Other methods train classifiers to select among different inertial estimates [21] or to obtain occlusion probabilities [27].

There has been work on unsupervised learning of disparity or motion between frames of videos using neural network models. These methods typically use multiplicative interactions to model relations between a pair of images. Disparities and optical flow can then be inferred from the latent variables. Taylor *et al*. [33] approach the task with factored gated restricted Boltzmann machines. Konda and Memisevic [23] use a special autoencoder called 'synchrony autoencoder'. While these approaches work well in a controlled setup and learn features useful for activity recognition in videos, they are not competitive with classical methods on realistic videos.

Convolutional Networks. Convolutional neural networks trained with backpropagation [25] have recently been shown to perform well on large-scale image classification by Krizhevsky *et al*. [24]. This gave the beginning to a surge of works on applying CNNs to various computer vision tasks.

While there has been no work on estimating optical flow with CNNs, there has been research on matching with neural networks. Fischer *et al*. [12] extract feature representations from CNNs trained in supervised or unsupervised manner and match these features based on Euclidean distance. Zbontar and LeCun [36] train a CNN with a Siamese architecture to predict similarity of image patches. A drastic difference of these methods to our approach is that they are patch based and leave the spatial aggregation to postprocessing, whereas the networks in this paper directly predict complete flow fields.

Recent applications of CNNs include semantic segmentation [11, 15, 17, 28], depth prediction [10], keypoint prediction [17] and edge detection [13]. These tasks are similar to optical flow estimation in that they involve per-pixel predictions. Since our architectures are largely inspired by the recent progress in these per-pixel prediction tasks, we briefly review different approaches.

The simplest solution is to apply a conventional CNN in a 'sliding window' fashion, hence computing a single prediction (e.g. class label) for each input image patch [8, 11]. This works well in many situations, but has drawbacks: high computational costs (even with optimized implementations involving re-usage of intermediate feature maps) and per-patch nature, disallowing to account for global output properties, for example sharp edges. Another simple approach [17] is to upsample all feature maps to the desired full resolution and stack them together, resulting in a concatenated per-pixel feature vector that can be used to predict the value of interest.

Eigen *et al*. [10] refine a coarse depth map by training an additional network which gets as inputs the coarse prediction and the input image. Long *et al*. [28] and Dosovitskiy *et al*. [9] iteratively refine the coarse feature maps with the use of 'upconvolutional' layers [1] . Our approach integrates ideas from both works. Unlike Long *et al*., we 'upconvolve' not just the coarse prediction, but the whole coarse feature maps, allowing to transfer more high-level information to the fine prediction. Unlike Dosovitskiy *et al*., we concatenate the 'upconvolution' results with the features from the 'contractive' part of the network.

[1] These layers are often named 'deconvolutional', although the operation they perform is technically convolution, not deconvolution

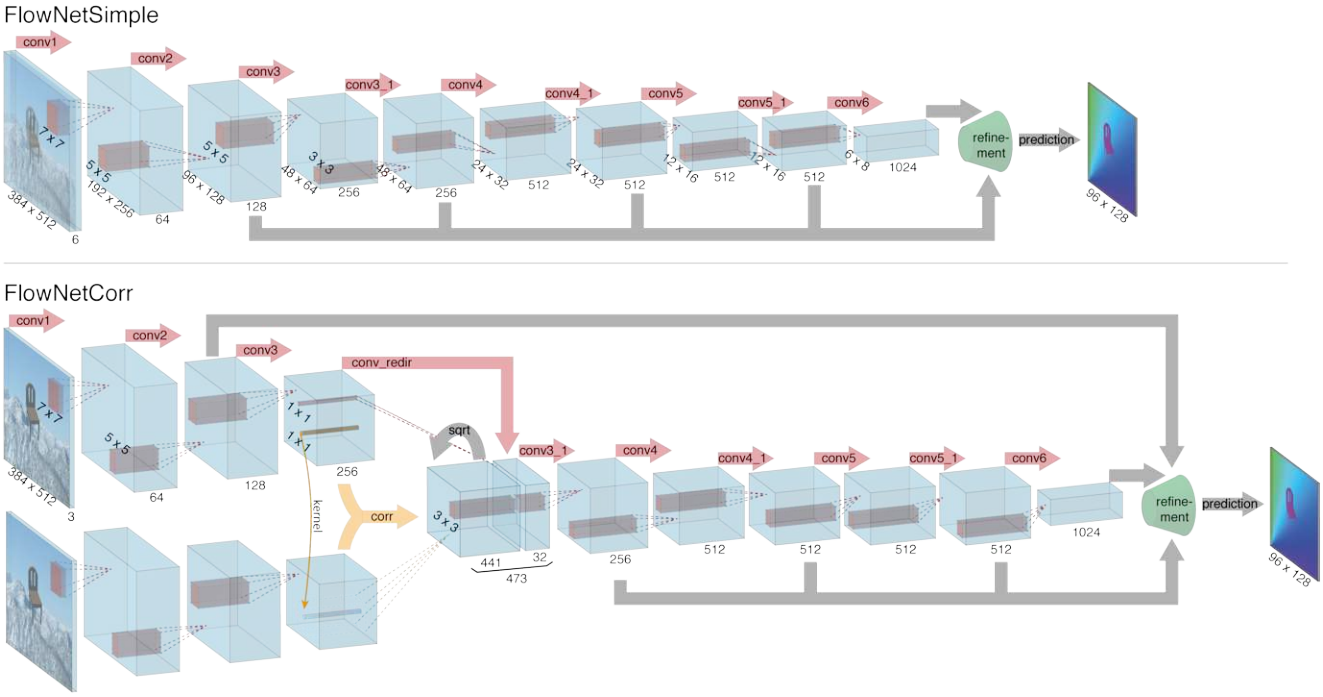


Figure 2. The two network architectures: FlowNetSimple (top) and FlowNetCorr (bottom). The green funnel is a placeholder for the expanding refinement part shown in Fig 3. The networks including the refinement part are trained end-to-end.

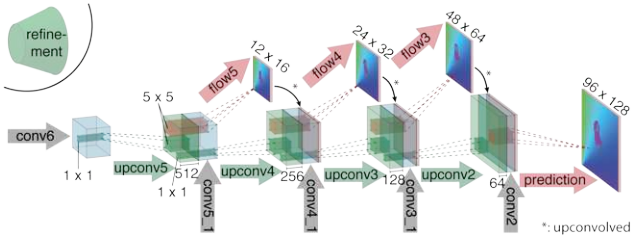


Figure 3. Refinement of the coarse feature maps to the high resolution prediction.

3. Network Architectures

Convolutional neural networks are known to be very good at learning input–output relations given enough labeled data. We therefore take an end-to-end learning approach to predicting optical flow: given a dataset consisting of image pairs and ground truth flows, we train a network to predict the x–y flow fields directly from the images. But what is a good architecture for this purpose?

Pooling in CNNs is necessary to make network training computationally feasible and, more fundamentally, to allow aggregation of information over large areas of the input images. But pooling results in reduced resolution, so in order to provide dense per-pixel predictions we need to refine the coarse pooled representation. To this end our networks contain an expanding part which intelligently refines the flow to high resolution. Networks consisting of contracting and expanding parts are trained as a whole using backpropagation. Architectures we use are depicted in Figures 2 and 3. We now describe the two parts of networks in more detail.

Contracting part. A simple choice is to stack both input images together and feed them through a rather generic network, allowing the network to decide itself how to process the image pair to extract the motion information. This is illustrated in Fig. 2 (top). We call this architecture consisting only of convolutional layers 'FlowNetSimple'.

Another approach is to create two separate, yet identical processing streams for the two images and to combine them at a later stage as shown in Fig. 2 (bottom). With this architecture the network is constrained to first produce meaningful representations of the two images separately and then combine them on a higher level. This roughly resembles the standard matching approach when one first extracts features from patches of both images and then compares those feature vectors. However, given feature representations of two images, how would the network find correspondences?

To aid the network in this matching process, we introduce a 'correlation layer' that performs multiplicative patch comparisons between two feature maps. An illustration of the network architecture 'FlowNetCorr' containing this layer is shown in Fig. 2 (bottom). Given two multi-channel feature maps $\mathbf{f}_1, \mathbf{f}_2 : \mathbb{R}^2 \rightarrow \mathbb{R}^c$, with w, h, and c being their width, height and number of channels, our correlation layer

lets the network compare each patch from $\mathbf{f}_1$ with each path from $\mathbf{f}_2$.

For now we consider only a single comparison of two patches. The 'correlation' of two patches centered at $\mathbf{x}_1$ in the first map and $\mathbf{x}_2$ in the second map is then defined as

$$c(\mathbf{x}_1, \mathbf{x}_2) = \sum_{\mathbf{o} \in [-k,k] \times [-k,k]} \langle \mathbf{f}_1(\mathbf{x}_1 + \mathbf{o}), \mathbf{f}_2(\mathbf{x}_2 + \mathbf{o}) \rangle \quad (1)$$

for a square patch of size $K := 2k + 1$. Note that Eq. 1 is identical to one step of a convolution in neural networks, but instead of convolving data with a filter, it convolves data with other data. For this reason, it has no trainable weights.

Computing $c(\mathbf{x}_1, \mathbf{x}_2)$ involves $c \cdot K^2$ multiplications. Comparing all patch combinations involves $w^2 \cdot h^2$ such computations, yields a large result and makes efficient forward and backward passes intractable. Thus, for computational reasons we limit the maximum displacement for comparisons and also introduce striding in both feature maps.

Given a maximum displacement d, for each location $\mathbf{x}_1$ we compute correlations $c(\mathbf{x}_1, \mathbf{x}_2)$ only in a neighborhood of size $D := 2d + 1$, by limiting the range of $\mathbf{x}_2$. We use strides s_1 and s_2, to quantize $\mathbf{x}_1$ globally and to quantize $\mathbf{x}_2$ within the neighborhood centered around $\mathbf{x}_1$.

In theory, the result produced by the correlation is four-dimensional: for every combination of two 2D positions we obtain a correlation value, i.e. the scalar product of the two vectors which contain the values of the cropped patches respectively. In practice we organize the relative displacements in channels. This means we obtain an output of size $(w \times h \times D^2)$. For the backward pass we implemented the derivatives with respect to each bottom blob accordingly.

Expanding part. The main ingredient of the expanding part are 'upconvolutional' layers, consisting of unpooling (extending the feature maps, as opposed to pooling) and a convolution. Such layers have been used previously [38, 37, 16, 28, 9]. To perform the refinement, we apply the 'upconvolution' to feature maps, and concatenate it with corresponding feature maps from the 'contractive' part of the network and an upsampled coarser flow prediction (if available). This way we preserve both the high-level information passed from coarser feature maps and fine local information provided in lower layer feature maps. Each step increases the resolution twice. We repeat this 4 times, resulting in a predicted flow for which the resolution is still 4 times smaller than the input. Overall architecture is shown in Figure 3. We found that further refinement from this resolution does not significantly improve the results, compared to a computationally less expensive bilinear upsampling to full image resolution.

Variational refinement. In an alternative scheme, at this very last stage instead of bilinear upsampling we use the variational approach from [6] without the matching term: we start at the 4 times downsampled resolution and then use the coarse to fine scheme with 20 iterations to bring the flow field to the full resolution. Finally, we run 5 more iterations at the full image resolution. We additionally compute image boundaries with the approach from [26] and respect the detected boundaries by replacing the smoothness coefficient by $\alpha = \exp(-\lambda b(x, y)^\kappa)$, where $b(x, y)$ denotes the thin boundary strength resampled at the respective scale and between pixels. This upscaling method is more computationally expensive than simple bilinear upsampling, but adds the benefits of variational methods to obtain smooth and subpixel-accurate flow fields. In the following, we denote the results obtained by this variational refinement with a '+v' suffix. An example of variational refinement can be seen in Fig. 4.

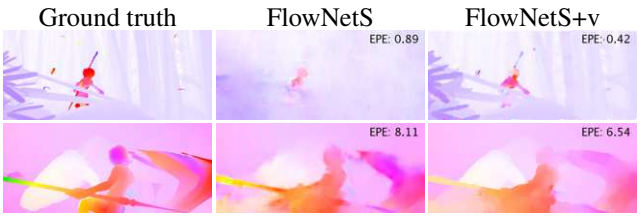


Figure 4. The effect of variational refinement. In case of small motions (first row) the predicted flow is changed dramatically. For larger motions (second row), big errors are not corrected, but the flow field is smoothed, resulting in lower EPE.

4. Training Data

Unlike traditional approaches, neural networks require data with ground truth not only for optimizing several parameters, but to learn to perform the task from scratch. In general, obtaining such ground truth is hard, because true pixel correspondences for real world scenes cannot easily be determined. An overview of the available datasets is given in Table 1.

4.1. Existing Datasets

The Middlebury dataset [2] contains only 8 image pairs for training, with ground truth flows generated using four different techniques. Displacements are very small, typi-

	Frame pairs	Frames with ground truth	Ground truth density per frame
Middlebury	72	8	100%
KITTI	194	194	∽50%
Sintel	1,041	1,041	100%
Flying Chairs	22,872	22,872	100%

Table 1. Size of already available datasets and the proposed Flying Chair dataset.

cally below 10 pixels.

The KITTI dataset [14] is larger (194 training image pairs) and includes large displacements, but contains only a very special motion type. The ground truth is obtained from real world scenes by simultaneously recording the scenes with a camera and a 3D laser scanner. This assumes that the scene is rigid and that the motion stems from a moving observer. Moreover, motion of distant objects, such as the sky, cannot be captured, resulting in sparse optical flow ground truth.

The MPI Sintel [7] dataset obtains ground truth from rendered artificial scenes with special attention to realistic image properties. Two versions are provided: the Final version contains motion blur and atmospheric effects, such as fog, while the Clean version does not include these effects. Sintel is the largest dataset available (1,041 training image pairs for each version) and provides dense ground truth for small and large displacement magnitudes.

4.2. Flying Chairs

The Sintel dataset is still too small to train large CNNs. To provide enough training data, we create a simple synthetic dataset, which we name Flying Chairs, by applying affine transformations to images collected from Flickr and a publicly available set of renderings of 3D chair models [1]. We retrieve 964 images from Flickr[2] with a resolution of $1,024 \times 768$ from the categories 'city' (321), 'landscape' (129) and 'mountain' (514). We cut the images into 4 quadrants and use the resulting 512×384 image crops as background. As foreground objects we add images of multiple chairs from [1] to the background. From the original dataset we remove very similar chairs, resulting in 809 chair types and 62 views per chair available. Examples are shown in Figure 5.

To generate motion, we randomly sample 2D affine transformation parameters for the background and the chairs. The chairs' transformations are relative to the background transformation, which can be interpreted as both the camera and the objects moving. Using the transformation parameters we generate the second image, the ground truth optical flow and occlusion regions.

All parameters for each image pair (number, types, sizes and initial positions of the chairs; transformation parameters) are randomly sampled. We adjust the random distributions of these parameters in such a way that the resulting displacement histogram is similar to the one from Sintel (details can be found in the supplementary material). Using this procedure, we generate a dataset with 22,872 image pairs and flow fields (we re-use each background image multiple times). Note that this size is chosen arbitrarily and could be larger in principle.

[2]Non-commercial public license. We use the code framework by Hays and Efros [18]

4.3. Data Augmentation

A widely used strategy to improve generalization of neural networks is data augmentation [24, 10]. Even though the Flying Chairs dataset is fairly large, we find that using augmentations is crucial to avoid overfitting. We perform augmentation online during network training. The augmentations we use include geometric transformations: *translation*, *rotation* and *scaling*, as well as additive *Gaussian noise* and changes in *brightness*, *contrast*, *gamma*, and *color*. To be reasonably quick, all these operations are processed on the GPU. Some examples of augmentation are given in Fig. 5.

As we want to increase not only the variety of images but also the variety of flow fields, we apply the same strong geometric transformation to both images of a pair, but additionally a smaller relative transformation between the two images.

Specifically we sample *translation* from a the range $[-20\%, 20\%]$ of the image width for x and y; *rotation* from $[-17°, 17°]$; *scaling* from $[0.9, 2.0]$. The Gaussian *noise* has a sigma uniformly sampled from $[0, 0.04]$; *contrast* is sampled within $[-0.8, 0.4]$; multiplicative color changes to the RGB channels per image from $[0.5, 2]$; gamma values from $[0.7, 1.5]$ and additive brightness changes using Gaussian with a sigma of 0.2.

5. Experiments

We report the results of our networks on the Sintel, KITTI and Middlebury datasets, as well as on our synthetic Flying Chairs dataset. We also experiment with fine-tuning of the networks on Sintel data and variational refinement of the predicted flow fields. Additionally, we report runtimes of our networks, in comparison to other methods.

5.1. Network and Training Details

The exact architectures of the networks we train are shown in Fig. 2. Overall, we try to keep the architectures of different networks consistent: they have nine convolutional layers with stride of 2 (the simplest form of pooling) in six of them and a ReLU nonlinearity after each layer. We do not have any fully connected layers, which allows the networks to take images of arbitrary size as input. Convolutional filter sizes decrease towards deeper layers of networks: 7×7 for the first layer, 5×5 for the following two layers and 3×3 starting from the fourth layer. The number of feature maps increases in the deeper layers, roughly doubling after each layer with a stride of 2. For the correlation layer in FlowNetC we chose the parameters $k = 0$, $d = 20$, $s_1 = 1$, $s_2 = 2$. As training loss we use the endpoint error (EPE), which is the standard error measure for optical flow estimation. It is the Euclidean distance between the predicted flow vector and the ground truth, averaged over all pixels.

Figure 5. **Flying Chairs.** Generated image pair and color coded flow field (first three columns), augmented image pair and corresponding color coded flow field respectively (last three columns).

For training CNNs we use a modified version of the caffe [20] framework. We choose Adam [22] as optimization method because for our task it shows faster convergence than standard stochastic gradient descent with momentum. We fix the parameters of Adam as recommended in [22]: $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Since, in a sense, every pixel is a training sample, we use fairly small mini-batches of 8 image pairs. We start with learning rate $\lambda = 1e{-}4$ and then divide it by 2 every 100k iterations after the first 300k. With FlowNetCorr we observe exploding gradients with $\lambda = 1e{-}4$. To tackle this problem, we start by training with a very low learning rate $\lambda = 1e{-}6$, slowly increase it to reach $\lambda = 1e{-}4$ after 10k iterations and then follow the schedule just described.

To monitor overfitting during training and fine-tuning, we split the Flying Chairs dataset into $22,232$ training and 640 test samples and split the Sintel training set into 908 training and 133 validation pairs.

We found that upscaling the input images during testing may improve the performance. Although the optimal scale depends on the specific dataset, we fixed the scale once for each network for all tasks. For FlowNetS we do not upscale, for FlowNetC we chose a factor of 1.25.

Fine-tuning. The used datasets are very different in terms of object types and motions they include. A standard solution is to fine-tune the networks on the target datasets. The KITTI dataset is small and only has sparse flow ground truth. Therefore, we choose to fine-tune on the Sintel training set. We use images from the Clean and Final versions of Sintel together and fine-tune using a low learning rate $\lambda = 1e{-}6$ for several thousand iterations. For best performance, after defining the optimal number of iterations using a validation set, we then fine-tune on the whole training set for the same number of iterations. In tables we denote fine-tuned networks with a '+ft' suffix.

5.2. Results

Table 2 shows the endpoint error (EPE) of our networks and several well-performing methods on public datasets (Sintel, KITTI, Middlebury), as well as on our Flying Chairs dataset. Additionally we show runtimes of different methods on Sintel.

The networks trained just on the non-realistic Flying Chairs perform very well on real optical flow datasets, beating for example the well-known LDOF [6] method. After fine-tuning on Sintel our networks can outperform the competing real-time method EPPM [3] on Sintel Final and KITTI while being twice as fast.

Sintel. From Table 2 one can see that FlowNetC is better than FlowNetS on Sintel Clean, while on Sintel Final the situation changes. On this difficult dataset, FlowNetS+ft+v is even on par with DeepFlow. Since the average endpoint error often favors over-smoothed solutions, it is interesting to see qualitative results of our method. Figure 6 shows examples of the raw optical flow predicted by the two FlowNets (without fine-tuning), compared to ground truth and EpicFlow. The figure shows how the nets often produce visually appealing results, but are still worse in terms of endpoint error. Taking a closer look reveals that one reason for this may be the noisy non-smooth output of the nets especially in large smooth background regions. This we can partially compensate with variational refinement.

KITTI. The KITTI dataset contains strong projective transformations which are very different from what the networks encountered during training on Flying Chairs. Still, the raw network output is already fairly good, and additional fine-tuning and variational refinement give a further boost. Interestingly, fine-tuning on Sintel improves the results on KITTI, probably because the images and motions in Sintel are more natural than in Flying Chairs. The FlowNetS outperforms FlowNetC on this dataset.

Flying Chairs. Our networks are trained on the Flying Chairs, and hence are expected to perform best on those. When training, we leave aside a test set consisting of 640 images. Table 2 shows the results of various methods on this

Method	Sintel Clean		Sintel Final		KITTI		Middlebury train		Middlebury test		Chairs	Time (sec)	
	train	test	train	test	train	test	AEE	AAE	AEE	AAE	test	CPU	GPU
EpicFlow [30]	2.27	4.12	3.57	6.29	3.47	3.8	0.31	3.24	0.39	3.55	2.94	16	-
DeepFlow [35]	3.19	5.38	4.40	7.21	4.58	5.8	0.21	3.04	0.42	4.22	3.53	17	-
EPPM [3]	-	6.49	-	8.38	-	9.2	-	-	0.33	3.36	-	-	0.2
LDOF [6]	4.19	7.56	6.28	9.12	13.73	12.4	0.45	4.97	0.56	4.55	3.47	65	2.5
FlowNetS	4.50	7.42	5.45	8.43	8.26	-	1.09	13.28	-	-	2.71	-	0.08
FlowNetS+v	3.66	6.45	4.76	7.67	6.50	-	0.33	3.87	-	-	2.86	-	1.05
FlowNetS+ft	(3.66)	6.96	(4.44)	7.76	7.52	9.1	0.98	15.20	-	-	3.04	-	0.08
FlowNetS+ft+v	(2.97)	6.16	(4.07)	7.22	6.07	7.6	0.32	3.84	0.47	4.58	3.03	-	1.05
FlowNetC	4.31	7.28	5.87	8.81	9.35	-	1.15	15.64	-	-	2.19	-	0.15
FlowNetC+v	3.57	6.27	5.25	8.01	7.45	-	0.34	3.92	-	-	2.61	-	1.12
FlowNetC+ft	(3.78)	6.85	(5.28)	8.51	8.79	-	0.93	12.33	-	-	2.27	-	0.15
FlowNetC+ft+v	(3.20)	6.08	(4.83)	7.88	7.31	-	0.33	3.81	0.50	4.52	2.67	-	1.12

Table 2. Average endpoint errors (in pixels) of our networks compared to several well-performing methods on different datasets. The numbers in parentheses are the results of the networks on data they were trained on, and hence are not directly comparable to other results.

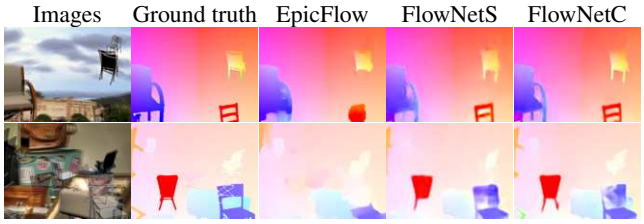


Figure 7. Examples of optical flow prediction on the Flying Chairs dataset. The images include fine details and small objects with large displacements which EpicFlow often fails to find. The networks are much more successful.

test set, some example predictions are shown in Fig. 7. One can see that FlowNetC outperforms FlowNetS and that the nets outperform all state-of-the-art methods. Another interesting finding is that this is the only dataset where the variational refinement does not improve performance but makes things worse. Apparently the networks can do better than variational refinement already. This indicates that with a more realistic training set, the networks might also perform even better on other data.

Timings. In Table 2 we show the per-frame runtimes of different methods in seconds. Unfortunately, many methods only provide the runtime on a single CPU, whereas our FlowNet uses layers only implemented on GPU. While the error rates of the networks are below the state of the art, they are the best among real-time methods. For both training and testing of the networks we use an *NVIDIA GTX Titan* GPU. The CPU timings of DeepFlow and EpicFlow are taken from [30], while the timing of LDOF was computed on a single 2.66GHz core.

5.3. Analysis

Training data. To check if we benefit from using the Flying Chairs dataset instead of Sintel, we trained a network just on Sintel, leaving aside a validation set to control the performance. Thanks to aggressive data augmentation, even Sintel alone is enough to learn optical flow fairly well. When testing on Sintel, the network trained exclusively on Sintel has EPE roughly 1 pixel higher than the net trained on Flying Chairs and fine-tuned on Sintel.

The Flying Chairs dataset is fairly large, so is data augmentation still necessary? The answer is positive: training a network without data augmentation on the Flying Chairs results in an EPE increase of roughly 2 pixels when testing on Sintel.

Comparing the architectures. The results in Table 2 allow to draw conclusions about strengths and weaknesses of the two architectures we tested.

First, FlowNetS generalizes to Sintel Final better than FlowNetC. On the other hand, FlowNetC outperforms FlowNetS on Flying chairs and Sintel Clean. Note that Flying Chairs do not include motion blur or fog, as in Sintel Final. These results together suggest that even though the number of parameters of the two networks is virtually the same, the FlowNetC slightly more overfits to the training data. This does not mean the network remembers the training samples by heart, but it adapts to the kind of data it is presented during training. Though in our current setup this can be seen as a weakness, if better training data were available it could become an advantage.

Second, FlowNetC seems to have more problems with large displacements. This can be seen from the results on KITTI discussed above, and also from detailed performance analysis on Sintel Final (not shown in the tables). FlowNetS+ft achieves s40+ error (EPE on pixels with displacements of at least 40 pixels) of 43.3px, and for FlowNetC+ft this value is 48px. One explanation is that the maximum displacement of the correlation does not allow to predict very large motions. This range can be increased at the cost of computational efficiency.

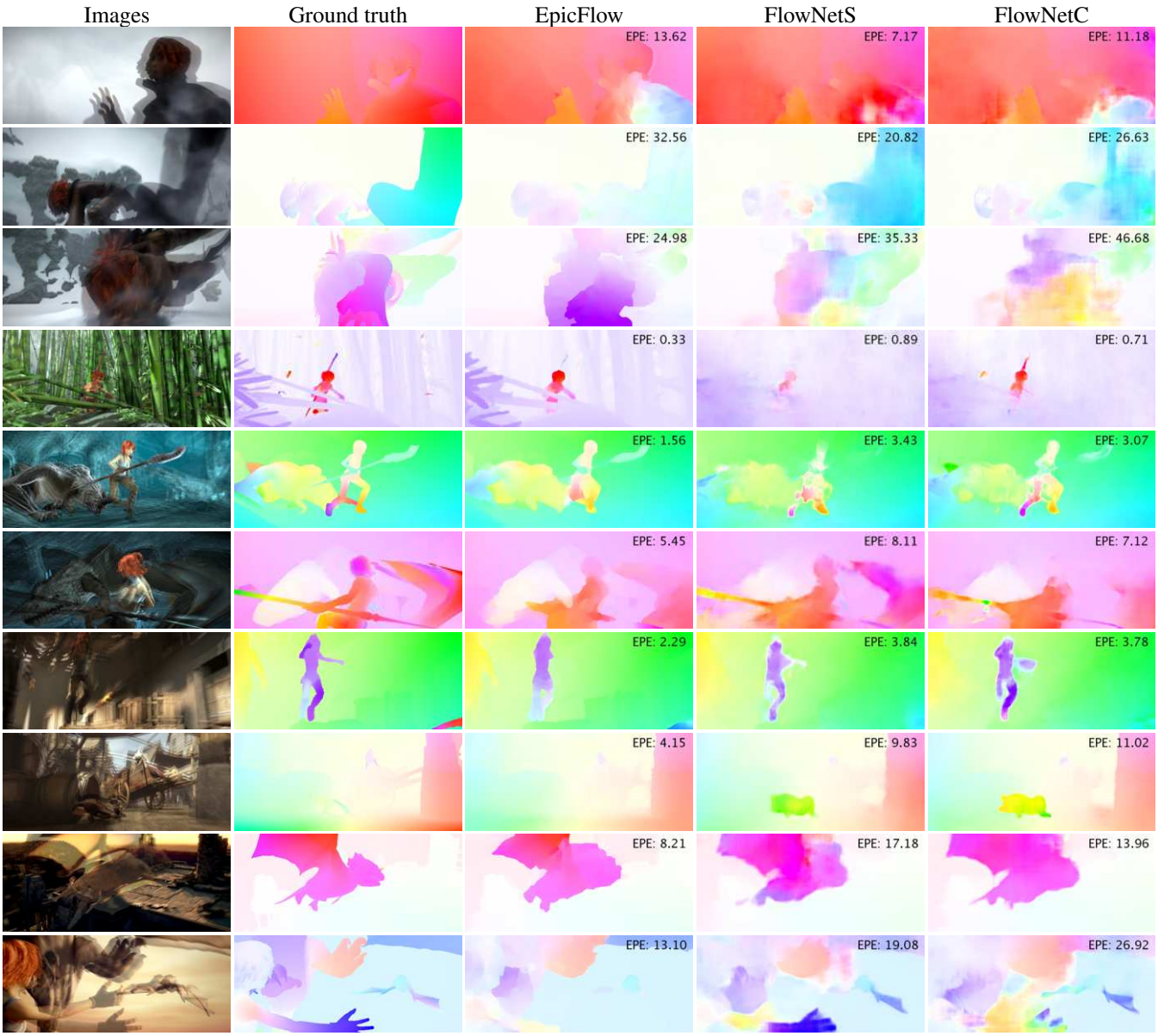


Figure 6. Examples of optical flow prediction on the Sintel dataset. In each row left to right: overlaid image pair, ground truth flow and 3 predictions: EpicFlow, FlowNetS and FlowNetC. Endpoint error is shown for every frame. Note that even though the EPE of FlowNets is usually worse than that of EpicFlow, the networks often better preserve fine details.

6. Conclusion

Building on recent progress in design of convolutional network architectures, we have shown that it is possible to train a network to directly predict optical flow from two input images. Intriguingly, the training data need not be realistic. The artificial Flying Chairs dataset including just affine motions of synthetic rigid objects is sufficient to predict optical flow in natural scenes with competitive accuracy. This proves the generalization capabilities of the presented networks. On the test set of the Flying Chairs the CNNs even outperform state-of-the-art methods like DeepFlow and EpicFlow. It will be interesting to see how future networks perform as more realistic training data becomes available.

Acknowledgments

The work was partially funded by the ERC Starting Grants VideoLearn and ConvexVision, by the DFG Grants BR-3815/7-1 and CR 250/13-1, and by the EC FP7 project 610967 (TACMAN).

References

[1] M. Aubry, D. Maturana, A. Efros, B. Russell, and J. Sivic. Seeing 3d chairs: exemplar part-based 2d-3d alignment using a large dataset of cad models. In *CVPR*, 2014. 2, 5

[2] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. Technical Report MSR-TR-2009-179, December 2009. 4

[3] L. Bao, Q. Yang, and H. Jin. Fast edge-preserving patchmatch for large displacement optical flow. In *CVPR*, 2014. 6, 7

[4] M. J. Black, Y. Yacoob, A. D. Jepson, and D. J. Fleet. Learning parameterized models of image motion. In *CVPR*, 1997. 2

[5] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. In *ECCV*, volume 3024, pages 25–36. Springer, 2004. 2

[6] T. Brox and J. Malik. Large displacement optical flow: descriptor matching in variational motion estimation. *PAMI*, 33(3):500–513, 2011. 2, 4, 6, 7

[7] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In A. Fitzgibbon et al. (Eds.), editor, *ECCV*, Part IV, LNCS 7577, pages 611–625. Springer-Verlag, Oct. 2012. 1, 5

[8] D. C. Ciresan, L. M. Gambardella, A. Giusti, and J. Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In *NIPS*, pages 2852–2860, 2012. 2

[9] A. Dosovitskiy, J. T. Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. In *CVPR*, 2015. 2, 4

[10] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. *NIPS*, 2014. 1, 2, 5

[11] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *PAMI*, 35(8):1915–1929, 2013. 2

[12] P. Fischer, A. Dosovitskiy, and T. Brox. Descriptor matching with convolutional neural networks: a comparison to SIFT. 2014. pre-print, arXiv:1405.5769v1 [cs.CV]. 2

[13] Y. Ganin and V. S. Lempitsky. N^4 -fields: Neural network nearest neighbor fields for image transforms. In *ACCV*, pages 536–551, 2014. 2

[14] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013. 5

[15] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 2

[16] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, 2014. 4

[17] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Hypercolumns for object segmentation and fine-grained localization. *CVPR*, 2015. 2

[18] J. Hays and A. A. Efros. im2gps: estimating geographic information from a single image. In *CVPR*, 2008. 5

[19] B. K. P. Horn and B. G. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981. 2

[20] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. 6

[21] R. Kennedy and C. Taylor. Optical flow with geometric occlusion estimation and fusion of multiple frames. In *EMMCVPR*. 2015. 2

[22] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 6

[23] K. R. Konda and R. Memisevic. Unsupervised learning of depth and motion. *CoRR*, abs/1312.3429, 2013. 2

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1106–1114, 2012. 1, 2, 5

[25] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989. 1, 2

[26] M. Leordeanu, R. Sukthankar, and C. Sminchisescu. Efficient closed-form solution to generalized boundary detection. In *Proceedings of the 12th European Conference on Computer Vision - Volume Part IV*, ECCV'12, pages 516–529, Berlin, Heidelberg, 2012. Springer-Verlag. 4

[27] M. Leordeanu, A. Zanfir, and C. Sminchisescu. Locally affine sparse-to-dense matching for motion and occlusion estimation. *ICCV*, 0:1721–1728, 2013. 2

[28] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 1, 2, 4

[29] E. Mémin and P. Pérez. Dense estimation and object-based segmentation of the optical flow with robust techniques. 7(5):703–719, May 1998. 2

[30] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid. EpicFlow: Edge-Preserving Interpolation of Correspondences for Optical Flow. In *CVPR*, Boston, United States, June 2015. 2, 7

[31] D. Rosenbaum, D. Zoran, and Y. Weiss. Learning the local statistics of optical flow. In *NIPS*, 2013. 2

[32] D. Sun, S. Roth, J. Lewis, and M. J. Black. Learning optical flow. In *ECCV*, 2008. 2

[33] G. W. Taylor, R. Fergus, Y. LeCun, and C. Bregler. Convolutional learning of spatio-temporal features. In *ECCV*, pages 140–153, 2010. 2

[34] A. Wedel, D. Cremers, T. Pock, and H. Bischof. Structure- and motion-adaptive regularization for high accuracy optic flow. In *ICCV*, Kyoto, Japan, 2009. 2

[35] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid. DeepFlow: Large displacement optical flow with deep matching. In *ICCV*, Sydney, Australia, Dec. 2013. 2, 7

[36] J. Zbontar and Y. LeCun. Computing the stereo matching cost with a convolutional neural network. *CoRR*, abs/1409.4326, 2014. 2

[37] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014. 4

[38] M. D. Zeiler, G. W. Taylor, and R. Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *ICCV*, pages 2018–2025, 2011. 4

Active Object Localization with Deep Reinforcement Learning

Juan C. Caicedo
Fundación Universitaria Konrad Lorenz
Bogotá, Colombia
juanc.caicedor@konradlorenz.edu.co

Svetlana Lazebnik
University of Illinois at Urbana Champaign
Urbana, IL, USA
slazebni@illinois.edu

Abstract

We present an active detection model for localizing objects in scenes. The model is class-specific and allows an agent to focus attention on candidate regions for identifying the correct location of a target object. This agent learns to deform a bounding box using simple transformation actions, with the goal of determining the most specific location of target objects following top-down reasoning. The proposed localization agent is trained using deep reinforcement learning, and evaluated on the Pascal VOC 2007 dataset. We show that agents guided by the proposed model are able to localize a single instance of an object after analyzing only between 11 and 25 regions in an image, and obtain the best detection results among systems that do not use object proposals for object localization.

1. Introduction

The process of localizing objects with bounding boxes can be seen as a control problem with a sequence of steps to refine the geometry of the box. Determining the exact location of a target object in a scene requires active engagement to understand the context, change the fixation point, identify distinctive parts that support recognition, and determine the correct proportions of the box.

During the last decade, the problem of object detection or localization has been studied by the vision community with the goal of recognizing the category of an object, and identifying its spatial extent with a tight bounding box that covers all its visible parts [10, 26]. This is a challenging setup that requires computation and analysis in multiple image regions, and a good example of a task driven by active attention.

Important progress for improving the accuracy of object detectors has been recently possible with Convolutional Neural Networks (CNNs), which leverage big visual data and deep learning for image categorization. A successful model is the R-CNN detector proposed by Girshick et al. [12, 25], which combines object proposals and CNN features to achieve state-of-the-art results in the Pascal and ImageNet benchmarks. Several other works have proposed the use of high-capacity CNNs to directly predict bounding boxes under a regression setting also with good results [27, 28, 9].

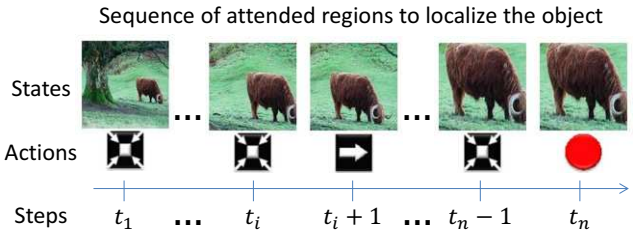


Figure 1. A sequence of actions taken by the proposed algorithm to localize a cow. The algorithm attends regions and decides how to transform the bounding box to progressively localize the object.

In this work, we propose a class-specific active detection model that *learns* to localize target objects known by the system. The proposed model follows a top-down search strategy, which starts by analyzing the whole scene and then proceeds to narrow down the correct location of objects. This is achieved by applying a sequence of transformations to a box that initially covers a large region of the image and is finally reduced to a tight bounding box. The sequence of transformations is decided by an *agent* that analyzes the content of the currently visible region to select the next best action. Each transformation should keep the object inside the visible region while cutting off as much background as possible. Figure 1 illustrates some steps of the dynamic decision process to localize a cow in an image.

The proposed approach is fundamentally different from most localization strategies. In contrast to sliding windows, our approach does not follow a fixed path to search objects; instead, different objects in different scenes will end up in different search paths. Unlike object proposal algorithms, candidate regions in our approach are selected by a high-level reasoning strategy instead of following low-level cues. Also, compared to bounding box regression algorithms, our approach does not localize objects following a single, struc-

tured prediction method. We propose a dynamic *attention-action* strategy that requires to pay attention to the contents of the current region, and to transform the box in such a way that the target object is progressively more focused.

To stimulate the attention of the proposed agent, we use a reward function proportional to how well the current box covers the target object. We incorporate the reward function in a reinforcement learning setting to learn a *localization policy*, based on the DeepQNetwork algorithm [23]. As a result, the trained agent can localize a single instance of an object in about 11 steps, which means that the algorithm can correctly find an object after processing only 11 regions of the image. We conducted a comprehensive experimental evaluation in the challenging Pascal VOC dataset, obtaining competitive results in terms of precision and recall. In what follows, we present and discuss the components of the proposed approach and provide a detailed analysis of experimental results.

2. Previous Works

Object localization has been successfully approached with sliding window classifiers. A popular sliding window method, based on HOG templates and SVM classifiers, has been extensively used to localize objects [11, 21], parts of objects [8, 20], discriminative patches [29, 17] and even salient components of scenes [24]. Sliding windows are related to our work because they are category-specific localization algorithms, which is also part of our design. However, unlike our work, sliding windows make an exhaustive search over the location-scale space.

A recent trend for object localization is the generation of category independent object proposals. Hosang et al. [15] provide an in depth analysis of ten object proposal methods, whose goal is to generate the smallest set of candidate regions with the highest possible recall. Substantial acceleration is achieved by reducing the set of candidates in this way, compared to sliding windows. Nonetheless, object detection based on proposals follows the same design of window-based classification on a set of reduced regions, which is still large (thousands of windows) for a single image that may contain a few interesting objects.

Several works attempt to reduce the number of evaluated regions during the detection process. For instance, Lampert et al. [18] proposed a branch-and-bound algorithm to find high-scoring regions only evaluating a few locations. Recently, Gonzalez-Garcia et al. [13] proposed an active search strategy to accelerate category-specific R-CNN detectors. These methods are related to ours because they try to optimize computational resources for localization. Also related is the work of Divvala et al. [7], which uses context to determine the localization of objects.

Visual attention models have been investigated with the goal of predicting where an observer is likely to orient the gaze (see [3] for a recent survey). These models are generally based on a saliency map that aggregates low-level features to identify interesting regions. These models are designed to predict human fixations and evaluate performance with user studies [33], while our work aims to localize objects and we evaluate performance in this task.

There is recent interest in attention capabilities for visual recognition in the machine learning community. Xu et al. [35] use a Recurrent Neural Network (RNN) to generate captions for images, using an attention mechanism that explains where the system focused attention to predict words. Mnih et al. [22] and Ba et al. [2] also used RNNs to select a sequence of regions that need more attention, which are processed at higher resolution for recognizing multiple characters. Interestingly, these models are trained with Reinforcement Learning as we do; however, our work uses a simpler architecture and intuitive actions to transform boxes.

3. Object Localization as a Dynamic Decision Process

We cast the problem of object localization as a Markov decision process (MDP) since this setting provides a formal framework to model an agent that makes a sequence of decisions. Our formulation considers a single image as the environment, in which the agent transforms a bounding box using a set of actions. The goal of the agent is to land a tight box in a target object that can be observed in the environment. The agent also has a state representation with information of the currently visible region and past actions, and receives positive and negative rewards for each decision made during the training phase. During testing, the agent does not receive rewards and does not update the model either, it just follows the learned policy.

Formally, the MDP has a set of actions A, a set of states S, and a reward function R. This section presents details of these three components, and the next section presents technical details of the training and testing strategies.

3.1. Localization Actions

The set of actions A is composed of eight transformations that can be applied to the box and one action to terminate the search process. These actions are illustrated in Figure 2, and are organized in four sub-sets: actions to move the box in the horizontal and vertical axes, actions to change scale, and actions to modify aspect ratio. In this way, the agent has four degrees of freedom to transform the box during any interaction with the environment.

A box is represented by the coordinates in pixels of its two corners: $b = [x_1, y_1, x_2, y_2]$. Any of the transformation actions make a discrete change to the box by a factor relative to its current size in the following way:

$$\alpha_w = \alpha * (x_2 - x_1) \qquad \alpha_h = \alpha * (y_2 - y_1) \qquad (1)$$

Figure 2. Illustration of the actions in the proposed MDP, giving 4 degrees of freedom to the agent for transforming boxes.

where $\alpha \in [0, 1]$. Then, the transformations are obtained by adding or removing α_w or α_h to the x or y coordinates, depending on the desired effect. For instance, a horizontal move to the right adds α_w to x_1 and x_y, while decreasing aspect ratio subtracts α_w from x_1, and adds it to x_2. Note that the origin in the image plane is located in the top-left corner.

We set $\alpha = 0.2$ in all our experiments, since this value gives a good trade-off between speed and localization accuracy. In early exploration experiments we noticed that smaller values make the agent slower to localize objects, while larger values make it harder to place the box correctly.

Finally, the only action that does not transform the box is a trigger to indicate that an object is correctly localized by the current box. This action terminates the sequence of the current search, and restarts the box in an initial position to begin the search for a new object. The trigger also modifies the environment: it marks the region covered by the box with a black cross as shown in the final frame of the top two examples in Figure 6. This mark serves as a inhibition-of-return (IoR) mechanism by which the currently attended region is prevented from being attended again. IoR mechanisms have been widely used in visual attention models (see [16] for a review) to suppress the attended location and avoid endless attractions towards the most salient stimulus.

3.2. State

The state representation is a tuple (o, h), where o is a feature vector of the observed region, and h is a vector with the history of taken actions. The set of possible states S is very large as it includes arbitrary boxes from a large set of images, and is expanded with all the combinations of actions that lead to those boxes. Therefore, generalization is important to design an effective state representation.

The feature vector o is extracted from the current region using a pre-trained CNN following the architecture of Zeiler and Fergus [36]. Any attended region by the agent is warped to match the input of the network (224×224) regardless of its size and aspect ratio, following the technique proposed by Girshick et al. [12]. We also expand the region to include 16 pixels of context around the original box. We forward the region up to the layer 6 (fc6) and use the 4,096 dimensional feature vector to represent its content.

The history vector h is a binary vector that informs which actions have been used in the past. Each action in the history vector is represented by a 9-dimensional binary vector, where all values are zero except the one corresponding to the taken action. The history vector encodes 10 past actions, which means that $h \in \mathbb{R}^{90}$. Although h is very low-dimensional compared to o, it has enough energy to inform what has happened in the past. This information demonstrated to be useful to stabilize search trajectories that might get stuck in repetitive cycles, improving average precision by approximately 3 percent points. The history vector also works better than appending a few more frames to the state representation with the additional benefit of increasing dimensionality by a negligible factor.

3.3. Reward Function

The reward function R is proportional to the improvement that the agent makes to localize an object after selecting a particular action. Improvement in our setup is measured using the Intersection-over-Union (IoU) between the target object and the predicted box at any given time. More specifically, the reward function is estimated using the differential of IoU from one state to another. The reward function can be estimated during the training phase only because it requires ground truth boxes to be calculated.

Let b be the box of an observable region, and g the ground truth box for a target object. Then, IoU between b and g is defined as $IoU(b, g) = area(b \cap g)/area(b \cup g)$.

The reward function $R_a(s, s')$ is granted to the agent when it chooses the action a to move from state s to s'. Each state s has an associated box b that contains the attended region. Then, the reward is as follows[1]:

$$R_a(s, s') = sign\left(IoU(b', g) - IoU(b, g)\right) \qquad (2)$$

Intuitively, equation 2 says that the reward is positive if IoU improved from state s to state s', and negative otherwise. This reward scheme is binary $r \in \{-1, +1\}$, and applies to any action that transforms the box. Without quantization, the difference in IoU is small enough to confuse the agent about which actions are good or bad choices. Binary rewards communicate more clearly which transformations keep the object inside the box, and which ones take the box away the target. In this way, the agent pays a penalty for taking the box away the target, and is rewarded to keep the target object in the visible region until there is no other transformation that improves localization. In that case, the best action to choose should be the trigger.

The trigger has a different reward scheme because it leads to a terminal state that does not change the box, and thus, the differential of IoU will always be zero for this action. The reward for the trigger is a thresholding function of IoU as follows:

[1] Notice that the ground truth box g is part of the environment and cannot be modified by the agent.

$$R_\omega(s, s') = \begin{cases} +\eta & \text{if } IoU(b, g) \geq \tau \\ -\eta & \text{otherwise} \end{cases} \quad (3)$$

where ω is the trigger action, η is the trigger reward, set to 3.0 in our experiments, and τ is a threshold that indicates the minimum IoU allowed to consider the attended region as a positive detection. The standard threshold for object detection evaluation is 0.5, but we used $\tau = 0.6$ during training to encourage better localization. A larger value for τ has a negative effect in performance because the agent learns that only clearly visible objects are worth the trigger, and tends to avoid truncated or naturally occluded objects.

Finally, the proposed reward scheme implicitly considers the number of steps as a cost because of the way in which Q-learning models the discount of future rewards (positive and negative). The agent follows a greedy strategy, which prefers short sequences because any unnecessary step pays a penalty that reduces the accumulated utility.

4. Finding a Localization Policy with Reinforcement Learning

The goal of the agent is to transform a bounding box by selecting actions in a way that maximizes the sum of the rewards received during an interaction with the environment (an episode). The core problem is to find a policy that guides the decision making process of this agent. A policy is a function $\pi(s)$ that specifies the action a to be chosen when the current state is s. Since we do not have the state transition probabilities and the reward function is data-dependent, the problem is formulated as a reinforcement learning problem using Q-learning [31].

In our work, we follow the deep Q-learning algorithm recently proposed by Mnih et al. [23]. This approach estimates the action-value function using a neural network, and has several advantages over previous Q-learning methods. First, the output of the Q-network has as many units as actions in the problem. This makes the model efficient because the input image is forwarded through the network only once to estimate the value of all possible actions. Second, the algorithm incorporates a replay-memory to collect various experiences and learns from them in the long run. In this way, transitions in the replay-memory are used in many model updates resulting in greater data efficiency. Third, to update the model the algorithm selects transitions from the replay-memory uniformly at random to break short-term correlations between states. This makes the algorithm more stable and prevents divergence of the parameters. After learning the action-value function $Q(s, a)$, the policy that the agent follows is to select the action a with the maximum estimated value.

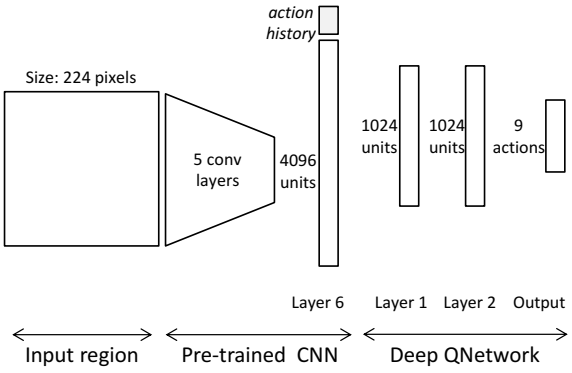


Figure 3. Architecture of the proposed QNetwork. The input region is first warped to 224×224 pixels and processed by a pre-trained CNN with 5 convolutional layers and 1 fully connected layer. The output of the CNN is concatenated with the action history vector to complete the state representation. It is processed by the Q-network which predicts the value of the 9 actions.

4.1. Q-learning for Object Localization

We use a Q-network that takes as input the state representation discussed in section 3.2 and gives as output the value of the nine actions presented in section 3.1. We train category specific Q-networks following the architecture illustrated in Figure 3. Notice that in our design we do not learn the full feature hierarchy of the convolutional network; instead, we rely on a pre-trained CNN.

Using a pre-trained CNN has two advantages: First, learning the Q function is faster because we need to update the parameters of the Q-Network only, while using the deep CNN just as a feed-forward feature extractor. Second, the hierarchy of features is trained with a larger dataset, leveraging generic discriminative features in our method. Learning the full hierarchy of features is possible under the deep Q-learning framework, and we hypothesize that performance could be improved because the learned features would be adapted for the localization task instead of classification. However, this mainly requires larger object detection datasets, so we leave this possibility for future work.

4.2. Training Localization Agents

The parameters of the Q-network are initialized at random. Then, the agent is set to interact with the environment in multiple episodes, each presenting a different training image. The policy followed during training is ϵ-greedy [31], which gradually shifts from exploration to exploitation according to the value of ϵ. During exploration, the agent selects actions randomly to observe different transitions and collects a varied set of experiences. During exploitation, the agent selects actions greedily according to the learned policy, and learns from its own successes and mistakes.

In our work, exploration does not proceed with random

actions. Instead, we use a guided exploration strategy following the principles of apprenticeship learning [1, 6, 19], which is based on demonstrations made by an expert to the agent. Since the environment knows the ground truth boxes, and the reward function is calculated with respect to the IoU with the current box, we can identify which actions will give positive and negative rewards. Then, during exploration, the agent is allowed to choose one random action from the set of positive actions [2]. Notice that for one particular state s, there might be multiple positive actions because there is no single path to localize an object. Using this strategy, the algorithm terminates in a small number of epochs.

The ϵ-greedy training strategy is run for 15 epochs, each completed after the agent has had interaction with all training images. During the first 5 epochs, ϵ is annealed linearly from 1.0 to 0.1 to progressively let the agent use its own learned model. After the fifth epoch, ϵ is fixed to 0.1 so the agent adjusts the model parameters from experiences produced by its own decisions. The parameters are updated using stochastic gradient descent and the back-propagation algorithm, and we also use dropout regularization [30].

4.3. Testing a Localization Agent

Once an agent is trained with the procedure described above, it learns to attend regions that contain objects of the target category. Since we do not know the number of objects present in a single image beforehand, we let the agent run for a maximum of 200 steps, so only 200 regions are evaluated per test image. An alternative to stop the search after a fixed number of steps is to include an extra termination action to let the agent indicate when the search is done. However, additional actions introduce new errors and make the problem more difficult to learn. We simplified the model with a minimum set of actions to act locally in time.

At each step, the agent makes a decision to transform the current box or selects the trigger to indicate that an object has been found. When the trigger is used, the search for other objects continues from a new box that covers a large portion of the image. The search for objects is restarted from the beginning due to two possible events: the agent used the trigger, or 40 steps passed without using the trigger. We found that 40 steps are enough to localize most objects, even the smaller ones, and when it takes longer is usually because the agent is stuck searching in an ambiguous region. Restarting the box helps to take a new perspective of the scene. The very first box covers the entire scene, and after any restarting event, the new box has a reduced size set to 75% of the image size, and is placed in one of the four corners of the image always in the same order (from top-left to right-bottom).

[2] Or any action if this set of positive actions is empty

5. Experiments and Results

We evaluated the performance of the proposed model using the Pascal VOC dataset. The localization method is sensitive to the amount of data used for training, and for that reason, we used the combined training sets of 2007 and 2012. Using this joint set, results improved nearly 10% relative to using either one alone. We evaluate performance on the test set of VOC 2007 and report our findings under this setting.

5.1. Evaluation of Precision

As an object detector, our algorithm can be evaluated in two modes: 1) All attended regions (AAR), a detector that scores all regions processed by the agent during a search episode. This is useful to consider well-localized regions that were not explicitly marked by the agent as detections. 2) Terminal regions (TR), a detector that only considers regions in which the agent explicitly used the trigger to indicate the presence of an object. In both cases, we use an external linear SVM trained with the same procedure as R-CNN (with hard-negative mining on region proposals using the VOC2012 training set) to score the attended regions. This classifier is useful to rerank candidate regions assuming that our model generates object proposals. The scores computed by the Q-network are not useful for object detection evaluation because they estimate the value of actions instead of discriminative scores.

We compare performance with other methods recently proposed in the literature. Table 1 presents detailed, per-category Average Precision (AP) for all systems. The only baseline method that does not use CNN features is the DPM system [11]. MultiBox [9] and DetNet [32] predict bounding boxes from input images using deep CNNs with a regression objective, and the work of Zou et al. [38] adapts the regionlets framework with CNN features too. Finally, we compare with the R-CNN system [12] configured with a network architecture that has the same number of layers as ours, and without bounding box regression.

Overall, the R-CNN system still has the best performance and remains as a strong baseline. The major drawback of R-CNN is that it relies on a large number of object proposals to reach that performance, and demands significant computing power. MultiBox and Regionlets attempt to leverage deep CNNs in more efficient ways, by using only a few boxes to make predictions or avoiding re-computing features multiple times. Our approach also aims to localize objects by attending to a small number of regions, and this has an impact in performance. However, our result is significantly better than the other baseline methods, reaching 46.1 MAP, while the next best reaches 40.2 MAP.

The main difference in performance between the TR and the ARR settings is that the first one only ranks regions explicitly marked by the agent with the trigger action. The

Method	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	MAP
DPM [11]	33.2	60.3	10.2	16.1	**27.3**	54.3	58.2	23.0	20.0	24.1	26.7	12.7	58.1	48.2	43.2	12.0	21.1	36.1	46.0	43.5	33.7
MultiBox [9]	41.3	27.7	30.5	17.6	3.2	45.4	36.2	53.5	6.9	25.6	27.3	46.4	31.2	29.7	37.5	7.4	29.8	21.1	43.6	22.5	29.2
DetNet [32]	29.2	35.2	19.4	16.7	3.7	53.2	50.2	27.2	10.2	34.8	30.2	28.2	46.6	41.7	26.2	10.3	32.8	26.8	39.8	47.0	30.5
Regionlets [38]	44.6	55.6	24.7	23.5	6.3	49.4	51.0	**57.5**	14.3	35.9	45.9	41.3	*61.9*	54.7	44.1	16.0	28.6	**41.7**	*63.2*	44.2	40.2
Ours TR	**57.9**	56.7	38.4	33.0	17.5	51.1	52.7	53.0	17.8	39.1	*47.1*	52.2	58.0	**57.0**	45.2	19.3	42.2	35.5	54.8	49.0	43.9
Ours AAR	55.5	**61.9**	**38.4**	**36.5**	21.4	**56.5**	**58.8**	55.9	**21.4**	**40.4**	46.3	**54.2**	56.9	55.9	**45.7**	**21.1**	**47.1**	41.5	54.7	**51.4**	**46.1**
R-CNN [12]	*64.2*	*69.7*	*50.0*	*41.9*	*32.0*	*62.6*	*71.0*	*60.7*	*32.7*	*58.5*	**46.5**	*56.1*	**60.6**	*66.8*	*54.2*	*31.5*	*52.8*	*48.9*	**57.9**	*64.7*	*54.2*

Table 1. Average Precision (AP) per category in the Pascal VOC 2007 test set. The DPM system is the only baseline that does not use CNN features. R-CNN is the only method that uses object proposals. Our system is significantly better at localizing objects than other recent systems that predict bounding boxes from CNN features without object proposals. Numbers in bold are the second best result per column, and underlined numbers are the overall best result.

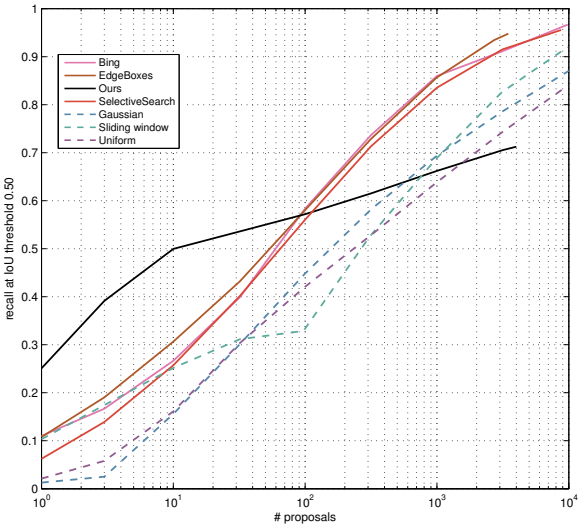


Figure 4. Recall as a function of the number of proposed regions. Solid lines are state-of-the-art methods, and dashed lines are simple baselines. Our approach is significantly better at early recall: only 10 proposals per image reach 50% recall. The tendency is also fundamentally different: overall recall does not depend strongly on a large number of proposed regions. Notice that our approach is not a category-independent region proposal algorithm.

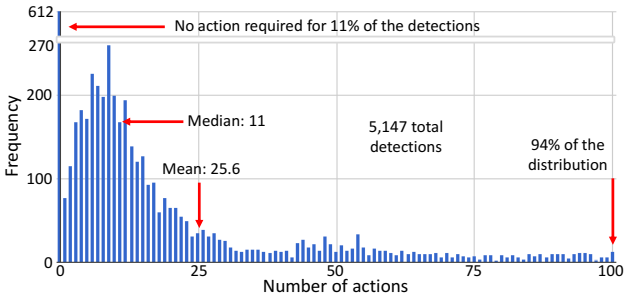


Figure 5. Distribution of detections explicitly marked by the agent as a function of the number of actions required to reach the object. For each action, one region in the image needs to be processed. Most detections can be obtained with about 11 actions only.

difference in performance is 2.2 percent points, which is relatively small, considering that TR proposes an average of only 1.3 regions per category per image (including true and false positives in all test images). Both settings require the same computational effort[3], because the agent has to attend the same number of regions. However, TR is able to identify which of those regions are promising objects with very high accuracy.

5.2. Evaluation of Recall

All the regions attended by the agent can be understood as object proposal candidates, so we evaluate them following the methodology proposed by Hosang et al. [15]. Overall, our method running for 200 steps per category, processes a total of 4,000 candidates per image reaching 71% of recall. This is between 10 and 25 points less recall than most methods achieve at a similar number of proposals. However, the recall of our proposals is significantly superior for the top 100 candidates per image.

For this evaluation we score attended regions using the Q-values predicted by the agent, and we add a large constant only to those regions for which the agent used the trigger. This gives priority to regions that the agent considers correctly localized objects. We use this scoring function instead of the classification scores for fairness in the evaluation [4], since other methods rank proposals using an estimated objectness score, and high Q-values can be interpreted in the same way. Figure 4 shows the number of proposals vs. recall for several baseline proposal methods. In this evaluation, we include selective search [34], BING [5] and EdgeBoxes [37]. The results show that our method reaches 50% of recall with just 10 proposals per image, while all the other methods are around or below 30%.

This evaluation emphasizes two important points: first, our method uses category-specific knowledge to find objects, and this is clearly an advantage over category-independent proposals, but is also a drawback since objects that cannot be recognized given the feature representation are never localized. Second, the main challenge of our approach is to improve overall recall, that is, to localize ev-

[3]The cost of the classifier is negligible compared to the cost of feature extraction and analysis.

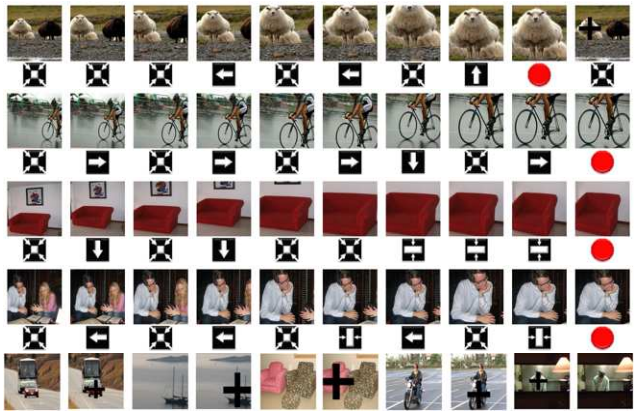


Figure 6. Example sequences observed by the agent and the actions selected to focus objects. Regions are warped in the same way as they are fed to the CNN. Actions keep the object in the center of the box. More examples in the supplementary material. Last row: example Inhibition of Return marks placed during test.

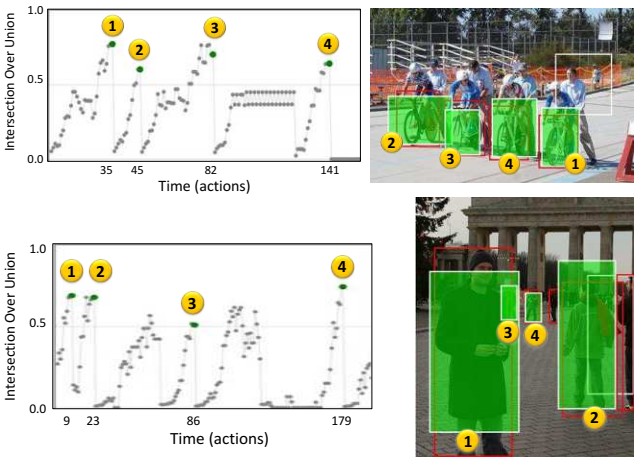


Figure 7. Examples of multiple objects localized by the agent in a single scene. Numbers in yellow indicate the order in which each instance was localized. Notice that IoU between the attended region and ground truth increases quickly before the trigger is used.

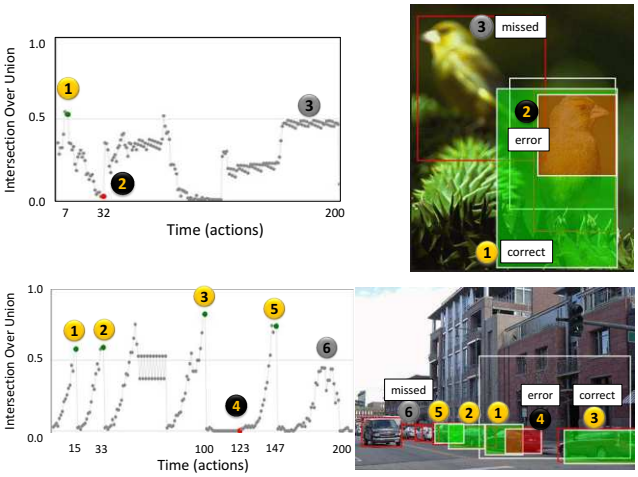


Figure 8. Examples of images with common mistakes that include: duplicated detections due to objects not fully covered by the IoR mark, and missed objects due to size or other difficult patterns.

ery single object without missing any instance. Notice that more candidates do not improve recall as much as other methods do, so we hypothesize that fixing overall recall will improve early recall even more.

To illustrate this result further, we plot the distribution of correctly detected objects according to the number of steps necessary to localize them in Figure 5. The distribution has a long tail, with 83% of detections requiring less than 50 steps to be obtained, and an average of 25.6. A more robust statistic for long-tailed distributions is the median, which in our experiments is just 11 steps, indicating that most of the correct detections happen around that number of steps. Also, the agent is able to localize 11% of the objects immediately without processing more regions, because they are big instances that occupy most of the image.

5.3. Qualitative Evaluation

We present a number of example sequences of regions that the agent attended to localize objects. Figure 7 shows two example scenes with multiple objects, and presents green boxes where a correct detection was explicitly marked by the agent. The plot to the left presents the evolution of IoU as the agent transforms the bounding box. These plots show that correct detections are usually obtained with a small number of steps increasing IoU with the ground truth rapidly. Points of the plots that oscillate below the minimum accepted threshold (0.5) indicate periods of the search process that were difficult or confusing to the agent.

Figure 6 shows sequences of attended regions as seen by the agent, as well as the actions selected in each step. Notice that the actions chosen attempt to keep the object in the center of the box, and also that the final object appears to have normalized scale and aspect ratio. The top two examples also show the IoR mark that is placed in the environment after the agent triggers a detection. The reader can find more examples and videos in the supplementary material.

5.4. Error Modes

We also evaluate performance using the diagnostic tool proposed by Hoiem et al. [14]. In summary, object localization is the most frequent error of our system, and it is sensitive to object size. Here we include the report of sensitivity to characteristics of objects in Figure 9, and compare to the R-CNN system. Our system is more sensitive to the size of objects than any other characteristic, which is mainly explained by the difficulty of the agent to attend cluttered re-

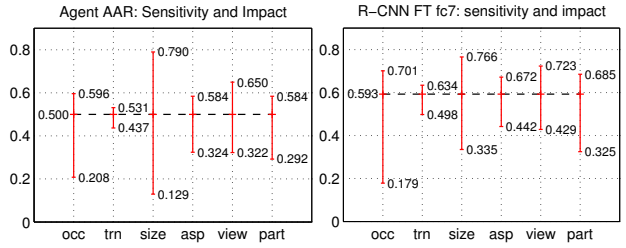


Figure 9. Sensitivity analysis following the method of Hoiem et al. [14]. Error bars are the average normalized AP over categories with the highest and lowest performance for each characteristic. Evaluated characteristics are: occlusion (occ), truncation (trn), size, aspect ratio (asp), viewpoint of objects (view), and visibility of parts (parts).

gions with small objects. Interestingly, our system seems to be less sensitive to occlusion and truncation than R-CNN.

Figure 8 presents object search episodes with some common errors. The figure shows examples of objects that the agent tried to localize but IoU never passed the minimum threshold. Those objects are missed because none of the attended regions surpass the threshold. In both examples, the agent triggered an additional detection in a location where a correct detection was placed before. This happens because the inhibition-of-return (IoR) mark leaves part of the object visible that may be observed again by the agent.

The IoR mark also generates other types of errors such as covering additional objects in the scene that cannot be recovered. We found that 78% of all missed detections happen in images with multiple instances of the same object, and the other 22% occur in images with a single instance. Not all missed objects are explained by obstruction of the IoR mark, however, we experimented with different strategies to change the focus of attention. We used a black box of the same size of the predicted box, but this results in a highly intrusive mark. We also experimented with a memory of detected instances to give negative rewards for going back to a known region, but training becomes unstable. The final cross that we adopted is centered in the predicted box and covers a third of its area, leaving parts of the region visible in favor of other overlapping objects. Overlapping objects are in general a challenge for most object detectors, and the problem requires special research attention.

5.5. Runtime

We conducted runtime experiments using a workstation equipped with a K-40 GPU. Feature extraction and action-decision run in the GPU. Our algorithm proceeds by analyzing a single region at a time, and processing each region takes an average of 7.74ms. This time is divided between feature extraction with the CNN (4.5 ms) and decision making with the Q-network (3.2ms), imposing an overhead of about 70% more computing power per region in our prototype, compared to systems that only extract features. This overhead is likely to be reduced using a more optimized implementation. Since we run the system for 200 steps, the total test processing time is 1.54s in average (wall clock time). Notice that we do not add segmentation or any other preprocessing time, because our system directly decides which boxes need to be evaluated.

6. Conclusions and Future Work

A system that learns to localize objects in scenes using an attention-action strategy has been presented. This approach is fundamentally different from previous works for object detection, because it follows a top-down scene analysis to narrow down the correct location of objects. Reinforcement learning (RL) demonstrated to be an efficient strategy to learn a localization policy, which is a challenging task since an object can be localized following different search paths. Nevertheless, the proposed agent learns from its own mistakes and optimizes the policy to find objects.

Experiments show that the system can localize a single instance of an object processing between 11 and 25 regions only. This results in a very efficient strategy for applications where a few number of categories are required. Our formulation needs to be extended in other ways for scaling to large numbers of categories; for instance, making it category-independent or using hierarchical ontologies to decide the fine-grained category later. An important challenge to improve recall needs to be addressed. Part of our future work includes training the system end-to-end instead of using a pre-trained CNN, and using deeper CNN architectures for improving the accuracy of predictions.

Acknowledgements. Special thanks to David Forsyth and Derek Hoiem for discussions and advice, and to Oscar Sánchez for his feedback. We gratefully acknowledge NVIDIA corporation for the donation of Tesla GPUs for this research. This research was part of the Blue Waters sustained-petascale computing project. This work was partially supported by NSF grants IIS 1228082, CIF 1302438, and the DARPA Computer Science Study Group (D12AP00305).

References

[1] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML*, 2004. 5

[2] J. Ba, V. Mnih, and K. Kavukcuoglu. Multiple object recognition with visual attention. *arXiv:1412.7755*, 2014. 2

[3] A. Borji and L. Itti. State-of-the-art in visual attention modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):185–207, 2013. 2

[4] N. Chavali, H. Agrawal, A. Mahendru, and D. Batra. Object-proposal evaluation protocol is' gameable'. *arXiv preprint arXiv:1505.05836*, 2015. 6

[5] M.-M. Cheng, Z. Zhang, W.-Y. Lin, and P. Torr. BING: Binarized normed gradients for objectness estimation at 300fps. In *CVPR*. IEEE, 2014. 6

[6] A. Coates, P. Abbeel, and A. Y. Ng. Learning for control from multiple demonstrations. In *ICML*, 2008. 5

[7] S. K. Divvala, D. Hoiem, J. H. Hays, A. Efros, M. Hebert, et al. An empirical study of context in object detection. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1271–1278. IEEE, 2009. 2

[8] I. Endres, K. J. Shih, J. Jiaa, and D. Hoiem. Learning collections of part models for object recognition. In *CVPR*. IEEE, 2013. 2

[9] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. Scalable object detection using deep neural networks. In *CVPR*. IEEE, 2014. 1, 5, 6

[10] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *IJCV*, 88(2):303–338, 2010. 1

[11] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010. 2, 5, 6

[12] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*. IEEE, 2014. 1, 3, 5, 6

[13] A. Gonzalez-Garcia, A. Vezhnevets, and V. Ferrari. An active search strategy for efficient object class detection. In *CVPR*. IEEE, 2015. 2

[14] D. Hoiem, Y. Chodpathumwan, and Q. Dai. Diagnosing error in object detectors. In *ECCV*. Springer, 2012. 7, 8

[15] J. Hosang, R. Benenson, and B. Schiele. How good are detection proposals, really? *arXiv:1406.6962*, 2014. 2, 6

[16] L. Itti and C. Koch. Computational modelling of visual attention. *Nature reviews neuroscience*, 2(3):194–203, 2001. 3

[17] M. Juneja, A. Vedaldi, C. Jawahar, and A. Zisserman. Blocks that shout: Distinctive parts for scene classification. In *CVPR*. IEEE, 2013. 2

[18] C. H. Lampert, M. B. Blaschko, and T. Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *CVPR*. IEEE, 2008. 2

[19] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *arXiv:1504.00702*, 2015. 5

[20] J. J. Lim, H. Pirsiavash, and A. Torralba. Parsing ikea objects: Fine pose estimation. In *ICCV*. IEEE, 2013. 2

[21] T. Malisiewicz, A. Gupta, and A. A. Efros. Ensemble of exemplar-SVMs for object detection and beyond. In *ICCV*. IEEE, 2011. 2

[22] V. Mnih, N. Heess, A. Graves, et al. Recurrent models of visual attention. In *NIPS*, 2014. 2

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015. 2, 4

[24] M. Pandey and S. Lazebnik. Scene recognition and weakly supervised object localization with deformable part-based models. In *ICCV*. IEEE, 2011. 2

[25] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015. 1

[26] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *arXiv:1409.0575*, 2014. 1

[27] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv:1312.6229*, 2013. 1

[28] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014. 1

[29] S. Singh, A. Gupta, and A. A. Efros. Unsupervised discovery of mid-level discriminative patches. In *ECCV*. Springer, 2012. 2

[30] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014. 5

[31] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 1998. 4

[32] C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. In *NIPS*, 2013. 5, 6

[33] A. Torralba, A. Oliva, M. S. Castelhano, and J. M. Henderson. Contextual guidance of eye movements and attention in real-world scenes: the role of global features in object search. *Psychological review*, 113(4):766, 2006. 2

[34] J. R. Uijlings, K. E. van de Sande, T. Gevers, and A. W. Smeulders. Selective search for object recognition. *IJCV*, 104(2):154–171, 2013. 6

[35] K. Xu, J. Ba, R. Kiros, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv:1502.03044*, 2015. 2

[36] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV*. Springer, 2014. 3

[37] C. L. Zitnick and P. Dollár. Edge boxes: Locating object proposals from edges. In *ECCV*. Springer, 2014. 6

[38] W. Y. Zou, X. Wang, M. Sun, and Y. Lin. Generic object detection with dense neural patterns and regionlets. *arXiv:1404.4316*, 2014. 5, 6

Deep Neural Decision Forests

Peter Kontschieder[†], **Madalina Fiterau**[◇], **Antonio Criminisi**[†], **Samuel Rota Bulò**[⋆]
[†] Microsoft Research Cambridge, UK
[◇] Stanford University California, USA
[⋆] Fondazione Bruno Kessler Trento, Italy

Abstract

We present a novel approach to enrich classification trees with the representation learning ability of deep (neural) networks within an end-to-end trainable architecture. We combine these two worlds via a stochastic and differentiable decision tree model, which steers the formation of latent representations within the hidden layers of a deep network. The proposed model differs from conventional deep networks in that a decision forest provides the final predictions and it differs from conventional decision forests by introducing a principled, joint and global optimization of split and leaf node parameters. Our approach compares favourably to other state-of-the-art deep models on a large-scale image classification task like ImageNet.

1 Introduction

Random forests [Amit and Geman, 1997; Breiman, 2001; Criminisi and Shotton, 2013] have found successfull application in machine learning in general and the computer vision community in particular. They empirically outperform most state-of-the-art learners on high dimensional data problems [Caruana *et al.*, 2008], they are inherently able to deal with multi-class problems and are easily distributable on parallel hardware architectures. Moreover, they are considered to be close to an ideal learner [Hastie *et al.*, 2009]. These facts and many (computationally) appealing properties make them attractive for various research areas and commercial products [Bosch *et al.*, 2007; Brostow *et al.*, 2008; Shotton *et al.*, 2013]. On the downside, random forests lack a machanism to *efficiently* learn internal representations that help to capture the main factors of variation in the data [Bengio *et al.*, 2010].

One of the consolidated findings of modern, deep learning approaches [Krizhevsky *et al.*, 2012; Lin *et al.*, 2013; Szegedy *et al.*, 2014] is that their joint and unified way of learning internal data representations along with the classifiers greatly outperforms conventional feature descriptor & classifier pipelines on different tasks, given enough training data and computation capabilities (see *e.g.* [He *et al.*, 2015] for image classification, [Yu and Deng, 2014] for speech recognition, [Fei-Fei, 2015] for image description).

An interesting open question, which has received little attention in the literature so far, is how to endow random forests with the ability of learning proper internal representations of the input data in order to improve on the generalization capacity of the final classifier. Notable but limited exceptions are [Kontschieder *et al.*, 2013; Montillo *et al.*, 2013] where random forests were trained in an entangled setting, stacking intermediate classifier outputs with the original input data. The approach in [Rota Bulò and Kontschieder, 2014] introduced a way to integrate multi-layer perceptrons as split functions, however, representations were learned only locally at split node level and independently among split nodes. While these attempts can be considered early forms of representation learning in random forests, their prediction accuracies remained below the state-of-the-art.

In this work we present *Deep Neural Decision Forests* – a novel approach to unify appealing properties from representation learning as known from deep architectures with the divide-and-conquer principle of decision trees. We introduce a stochastic, differentiable, and therefore back-propagation compatible version of decision trees, guiding the representation learning in hidden layers of deep networks. Thus, the task for representation learning is to reduce the uncertainty on the routing decisions of a sample taken at the split nodes, such that a globally-defined loss function is minimized. Additionally, for given split node parameters we obtain optimal predictions for all leaves of our trees by minimizing a convex objective, and we provide an optimization algorithm for it that does not depend on tedious step-size selection. Consequently, we can take the optimal decision for a test sample ending up in the leaves, with respect to all the training data and the current state of the network. We show the efficacy of our approach on the challenging ImageNet dataset for large-scale image classification, where we obtain state-of-the-art results with no data augmentation.

2 Decision Trees with Stochastic Routing

Decision trees can be used to tackle a wide range of learning problems [Criminisi and Shotton, 2013] including classification, which will be the focus of this work. Consider a classification problem with input and (finite) output spaces given by $\mathcal{X}$ and $\mathcal{Y}$, respectively. A *decision tree* is a classifier consisting of decision (or split) nodes and prediction (or leaf) nodes organized into a tree structure. Decision nodes indexed

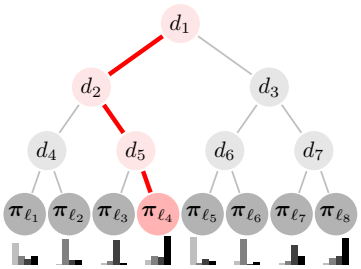


Figure 1: Routing of sample $\boldsymbol{x}$ along a decision tree to leaf ℓ_4. Each node $n \in \mathcal{N}$ along the path steers the sample to the left with probability $d_n(\boldsymbol{x})$ and to the right with probability $\bar{d}_n(\boldsymbol{x}) = 1 - d_n(\boldsymbol{x})$. So, the probability of reaching ℓ_4 is $\mu_{\ell_4}(\boldsymbol{x}) = d_1(\boldsymbol{x})\bar{d}_2(\boldsymbol{x})\bar{d}_5(\boldsymbol{x})$ and the related prediction is $\boldsymbol{\pi}_{\ell_4}$.

by $\mathcal{N}$ are internal nodes of the tree, while prediction nodes indexed by $\mathcal{L}$ are the terminal nodes of the tree. A decision tree classifies a sample by routing it through the tree to a leaf node $\ell \in \mathcal{L}$, where the prediction takes place via a probability distribution $\boldsymbol{\pi}_\ell$ over $\mathcal{Y}$. The routing along the tree is determined through decision functions $d_n(\cdot;\Theta) : \mathcal{X} \to [0,1]$ parametrized by Θ and located in each decision node $n \in \mathcal{N}$, which locally indicate the path to follow. When a sample $\boldsymbol{x} \in \mathcal{X}$ reaches a decision node n it will be sent to the left or right subtree based on the output of $d_n(\boldsymbol{x};\Theta)$. In standard decision forests, d_n is binary and the routing is deterministic. In this paper we will consider instead a probabilistic routing, *i.e.* the routing direction is the output of a Bernoulli random variable with mean $d_n(\boldsymbol{x};\Theta)$. Once a sample ends in a leaf node ℓ, the related tree prediction is given by the class-label distribution $\boldsymbol{\pi}_\ell$ (see Fig. 1 for an illustration). In the case of stochastic routings, the leaf predictions will be weighted by the probability of reaching the leaf. Accordingly, the final prediction for sample $\boldsymbol{x}$ from tree T with decision nodes parametrized by Θ is given by

$$\mathbb{P}_T[y|\boldsymbol{x},\Theta,\boldsymbol{\pi}] = \sum_{\ell\in\mathcal{L}} \pi_{\ell y}\mu_\ell(\boldsymbol{x}|\Theta)\,, \tag{1}$$

where $\boldsymbol{\pi} = (\boldsymbol{\pi}_\ell)_{\ell\in\mathcal{L}}$ and $\pi_{\ell y}$ denotes the probability of a sample reaching leaf ℓ to take on class y, while $\mu_\ell(\boldsymbol{x}|\Theta)$ is regarded as the *routing function* providing the probability that sample $\boldsymbol{x}$ will reach leaf ℓ. Clearly, $\sum_\ell \mu_\ell(\boldsymbol{x}|\Theta) = 1$ for all $\boldsymbol{x} \in \mathcal{X}$. To provide an explicit form for the routing function we introduce the following binary relations that depend on the tree's structure: $\ell \swarrow n$, which is true if ℓ belongs to the left subtree of node n, and $n \searrow \ell$, which is true if ℓ belongs to the right subtree of node n. These relations allow us to express μ_ℓ as follows:

$$\mu_\ell(\boldsymbol{x}|\Theta) = \prod_{n\in\mathcal{N}} d_n(\boldsymbol{x};\Theta)^{\mathbb{1}_{\ell\swarrow n}}\bar{d}_n(\boldsymbol{x};\Theta)^{\mathbb{1}_{n\searrow\ell}}\,, \tag{2}$$

where $\bar{d}_n(\boldsymbol{x};\Theta) = 1 - d_n(\boldsymbol{x};\Theta)$, and $\mathbb{1}_P$ is an indicator function conditioned on the argument P. Although the product in (2) runs over all nodes, only decision nodes along the path from the root to the leaf ℓ contribute to μ_ℓ (assuming $0^0 = 1$), because for all other nodes $\mathbb{1}_{\ell\swarrow n}$ and $\mathbb{1}_{n\searrow\ell}$ will be both 0.

Decision nodes. We consider decision trees having decision functions of the following form to deliver stochastic routings:

$$d_n(\boldsymbol{x};\Theta) = \sigma(f_n(\boldsymbol{x};\Theta))\,, \tag{3}$$

where $\sigma(x) = (1+e^{-x})^{-1}$ is the sigmoid function, and $f_n(\cdot;\Theta) : \mathcal{X} \to \mathbb{R}$ is a real-valued function depending on the sample and the parametrization Θ. The outcome of $d_n(\boldsymbol{x};\Theta) \in [0,1]$ represents the probability of sample $\boldsymbol{x}$ to be steered left in node $n \in \mathcal{N}$. Further details about the functions f_n can be found in Sec. 4, but intuitively depending on how we choose these functions we can model trees having shallow decisions (*e.g.* such as in oblique forests [Heath *et al.*, 1993]) as well as deep ones.

Forests of decision trees. A forest is an ensemble of decision trees $\mathcal{F} = \{T_1,\dots,T_{\mathsf{k}}\}$, which delivers a prediction for sample $\boldsymbol{x}$ by averaging the output of each tree, *i.e.*

$$\mathbb{P}_{\mathcal{F}}[y|\boldsymbol{x}] = \frac{1}{\mathsf{k}}\sum_{h=1}^{\mathsf{k}} \mathbb{P}_{T_h}[y|\boldsymbol{x}]\,, \tag{4}$$

omitting the tree parameters for notational convenience.

3 Learning Trees by Back-Propagation

In order to learn a decision tree defined as per Sec. 2, we need to estimate both, the decision node parametrization Θ and the leaf predictions $\boldsymbol{\pi}$. To carry out the estimation we follow the minimum empirical risk principle with respect to a given data set $\mathcal{T} \subset \mathcal{X}\times\mathcal{Y}$ under log-loss, *i.e.* we pursue minimizers of the following risk term:

$$R(\Theta,\boldsymbol{\pi};\mathcal{T}) = \frac{1}{|\mathcal{T}|}\sum_{(\boldsymbol{x},y)\in\mathcal{T}} L(\Theta,\boldsymbol{\pi};\boldsymbol{x},y)\,, \tag{5}$$

where $L(\Theta,\boldsymbol{\pi};\boldsymbol{x},y)$ is the log-loss term for the training sample $(\boldsymbol{x},y) \in \mathcal{T}$, which is given by

$$L(\Theta,\boldsymbol{\pi};\boldsymbol{x},y) = -\log(\mathbb{P}_T[y|\boldsymbol{x},\Theta,\boldsymbol{\pi}])\,, \tag{6}$$

and $\mathbb{P}_T$ is defined as in (1). We use a two-step optimization strategy, described in the rest of this section, alternating updates of Θ with updates of $\boldsymbol{\pi}$ in a way to minimize (5).

Learning Decision Nodes. All decision functions depend on a common parameter Θ, which in turn parametrizes each function f_n in (3). The minimization of the empirical risk with respect to Θ for a given $\boldsymbol{\pi}$ is, in general, a difficult and large-scale optimization problem, since no assumption has been made about the structure of f_n. As an example, Θ could absorb all the parameters of a deep neural network having f_n as one of its output units. For this reason, we will employ a Stochastic Gradient Descent (SGD) approach to minimize (5) with respect to Θ, as commonly done in the context of deep neural networks:

$$\begin{aligned}\Theta^{(t+1)} &= \Theta^{(t)} - \eta\frac{\partial R}{\partial\Theta}(\Theta^{(t)},\boldsymbol{\pi};\mathcal{B}) \\ &= \Theta^{(t)} - \frac{\eta}{|\mathcal{B}|}\sum_{(\boldsymbol{x},y)\in\mathcal{B}}\frac{\partial L}{\partial\Theta}(\Theta^{(t)},\boldsymbol{\pi};\boldsymbol{x},y)\,.\end{aligned} \tag{7}$$

Here, $\eta > 0$ is the learning rate and $\mathcal{B} \subseteq \mathcal{T}$ is a random subset (*a.k.a.* mini-batch) of samples from the training set. Although not shown explicitly, we additionally consider a momentum term to smooth out the variations of the gradients. The gradient of the loss L with respect to Θ can be decomposed by the chain rule as follows

$$\frac{\partial L}{\partial \Theta}(\Theta, \boldsymbol{\pi}; \boldsymbol{x}, y) = \sum_{n \in \mathcal{N}} \frac{\partial L(\Theta, \boldsymbol{\pi}; \boldsymbol{x}, y)}{\partial f_n(\boldsymbol{x}; \Theta)} \frac{\partial f_n(\boldsymbol{x}; \Theta)}{\partial \Theta} \,. \quad (8)$$

Here, the gradient term that depends on the decision tree is

$$\frac{\partial L(\Theta, \boldsymbol{\pi}; \boldsymbol{x}, y)}{\partial f_n(\boldsymbol{x}; \Theta)} = d_n(\boldsymbol{x}; \Theta) A_{n_r} - \bar{d}_n(\boldsymbol{x}; \Theta) A_{n_l} \,, \quad (9)$$

where n_l and n_r indicate the left and right child of node n, respectively, and we define A_m for a generic node $m \in \mathcal{N}$ as

$$A_m = \frac{\sum_{\ell \in \mathcal{L}_m} \pi_{\ell y} \mu_\ell(\boldsymbol{x}|\Theta)}{\mathbb{P}_T[y|\boldsymbol{x}, \Theta, \boldsymbol{\pi}]} \,,$$

where $\mathcal{L}_m \subseteq \mathcal{L}$ denotes the set of leaves held by the subtree rooted in node m. The actual computation of the gradient term in Eq. (9) can be carried out by traversing the tree twice [Kontschieder *et al.*, 2015].

Learning prediction nodes. We consider now the minimization of (5) with respect to $\boldsymbol{\pi}$ when Θ is fixed, *i.e.*

$$\min_{\boldsymbol{\pi}} R(\Theta, \boldsymbol{\pi}; \mathcal{T}) \,. \quad (10)$$

This is a *convex* optimization problem and a global solution can be easily recovered. A similar problem has been encountered in the context of decision trees in [Rota Bulò and Kontschieder, 2014], but only at the level of a single node. In our case, however, the whole tree is taken into account, and we are jointly estimating *all* the leaf predictions.

In order to compute a global minimizer of (10) we propose the following iterative scheme:

$$\pi_{\ell y}^{(t+1)} = \frac{1}{Z_\ell^{(t)}} \sum_{(\boldsymbol{x}, y') \in \mathcal{T}} \frac{\mathbb{1}_{y=y'} \pi_{\ell y}^{(t)} \mu_\ell(\boldsymbol{x}|\Theta)}{\mathbb{P}_T[y|\boldsymbol{x}, \Theta, \boldsymbol{\pi}^{(t)}]} \,, \quad (11)$$

for all $\ell \in \mathcal{L}$ and $y \in \mathcal{Y}$, where $Z_\ell^{(t)}$ is a normalizing factor ensuring that $\sum_y \pi_{\ell y}^{(t+1)} = 1$. The starting point $\boldsymbol{\pi}^{(0)}$ can be arbitrary as long as every element is positive. A typical choice is to start from the uniform distribution in all leaves, *i.e.* $\pi_{\ell y}^{(0)} = |\mathcal{Y}|^{-1}$. It is interesting to note that the update rule in (11) is step-size free and it guarantees a strict decrease of the risk at each update until a fixed-point is reached [Kontschieder *et al.*, 2015].

Learning a forest. So far we have dealt with a single decision tree setting. Now, we consider an ensemble of trees $\mathcal{F}$, where all trees can possibly share the same parameters in Θ, but each tree can have a different structure with a different set of decision functions (still defined as in (3)), and independent leaf predictions $\boldsymbol{\pi}$. Since each tree in the forest $\mathcal{F}$ has its own set of leaf parameters $\boldsymbol{\pi}$, we can update the prediction nodes of each tree independently via the update rule (11), given the current estimate of Θ. As for Θ, instead, we randomly select a tree in $\mathcal{F}$ for each mini-batch and then we proceed with the SGD update in (7), assuming that its leaf nodes' parameters $\boldsymbol{\pi}$ are fixed. This strategy somewhat resembles the basic idea of Dropout [Srivastava *et al.*, 2014], where each SGD update is potentially applied to a different network topology, which is sampled according to a specific distribution. In addition, updating individual trees instead of the entire forest reduces the computational load during training. At test time we average over tree predictions to produce the final outcome,(4).

4 Deep Decision Nodes

We have defined decision functions d_n in terms of real-valued functions $f_n(\cdot; \Theta)$, which are not necessarily independent, but coupled through the shared parametrization Θ. Our intention is to endow the trees with feature learning capabilities by embedding functions f_n within a deep neural network with parameters Θ. To this end, we regard each function f_n as a linear output unit of a deep network that will be turned into a probabilistic routing decision by the action of d_n, which applies a sigmoid activation to obtain a response in the $[0, 1]$ range. Fig. 2 provides a schematic illustration of this idea. The number of split nodes is determined by the number of output nodes of the preceding fully-connected layer. Under the proposed construction, the output units of the deep network are therefore not directly delivering the final predictions, *e.g.* through a Softmax layer, but each unit is responsible for driving the decision of a node in the forest. Indeed, during the forward pass through the deep network, a data sample $\boldsymbol{x}$ produces soft activations of the routing decisions of the tree that induce via the routing function a mixture of leaf predictions as per (1), which will form the final output.

5 Experiments on ImageNet

ImageNet [Russakovsky *et al.*, 2014] is a benchmark for large-scale image recognition tasks and its images are assigned to one out of 1000 possible ground truth labels. The dataset contains ≈1.2M training images, 50.000 validation images and 100.000 test images with average dimensionality of 482x415 pixels. Training and validation data are publicly available and we followed the commonly agreed protocol by reporting *Top5-Errors* on validation data. The GoogLeNet architecture [Szegedy *et al.*, 2014], which has a reported Top5-Error of 10.07% when used in a single-model, single-crop setting (see first row in Tab. 3 in [Szegedy *et al.*, 2014]), served as basis for our experiments. As opposed to conventional architectures, GoogLeNet uses 3 Softmax layers at different stages of the network to encourage the construction of informative features, due to its very deep architecture.

In order to obtain a *Deep Neural Decision Forest* architecture coined dNDF**.**NET, we have replaced each Softmax layer from GoogLeNet with a forest consisting of 10 trees (each fixed to depth 15), resulting in a total number of 30 trees. We refer to the individual forests as dNDF_0 (closest to raw input), dNDF_1 (replacing middle loss layer in GoogLeNet) and dNDF_2 (as terminal layer). Further details about our dNDF**.**NET architecture are provided in [Kontschieder *et al.*,

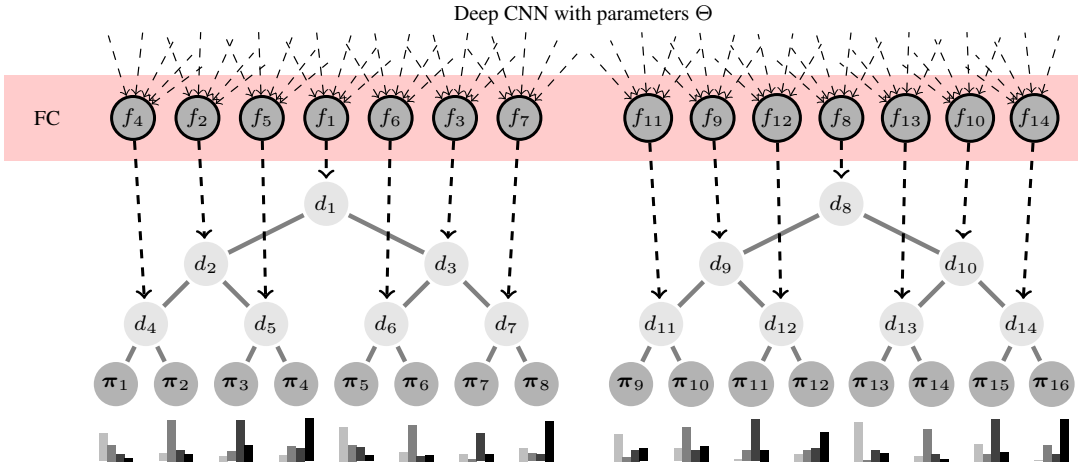


Figure 2: Illustration how to implement a deep neural decision forest (dNDF). Top: Deep neural network with variable number of layers, represented via parameters Θ. FC block: Fully Connected layer used to provide functions $f_n(\cdot;\Theta)$ (here: inner products), described in Eq. (3). Each output of f_n is brought in correspondence with a split node in a tree, eventually producing the routing (split) decisions $d_n(\boldsymbol{x}) = \sigma(f_n(\boldsymbol{x}))$. The order of the assignments of output units to decision nodes can be arbitrary (the one we show allows a simple visualization). The circles at the bottom correspond to leaf nodes, holding probability distributions $\boldsymbol{\pi}_\ell$ as a result of solving the convex optimization problem defined in Eq. (10).

	GoogLeNet [Szegedy *et al.*, 2014]						dNDF**.**NET			dNDF_0	dNDF_1	dNDF_2
#models/#crops	1/1	1/10	1/144	7/1	7/10	7/144	1/1	1/10	7/1	1/1		
Top5-Errors	10.07%	9.15%	7.89%	8.09%	7.62%	6.67%	7.84%	7.08%	**6.38%**	9.26%	8.49%	7.92%

Table 1: Top5-Errors obtained on ImageNet validation data, comparing our dNDF**.**NET to GoogLeNet.

2015]. Following the implementation guideline for decision nodes in Section 4, we randomly selected 500 output dimensions of the respectively preceding layers in GoogLeNet for each decision function f_n. We trained the network for 1000 epochs using (mini-) batches composed of 100.000 images (which was feasible due to distribution of the computational load to a cluster of 52 CPUs and 12 hosts, where each host is equipped with a NVIDIA Tesla K40 GPU).

As for the posterior learning, we only update the leaf node predictions of the tree that also receives split node parameter updates, *i.e.* the randomly selected one as described in Section 3. To improve computational efficiency, we consider only the samples of the current mini-batch for posterior learning, while *all* the training data could be used in principle. However, since we use mini-batches composed of 100.000 samples, we can approximate the training set sufficiently well and, at the same time, introduce a positive, regularizing effect.

Tab. 1 provides a summary of Top5-Errors on validation data for our proposed dNDF**.**NET against GoogLeNet. We ascribe the improvements on the single crop, single model setting (Top5-Error of only 7.84%) to our proposed approach, as the only architectural difference to GoogLeNet is deploying our dNDFs. By using an ensemble of 7 dNDF**.**NETs (still single crop inputs), we can reduce the error further and obtain a Top5-Error of 6.38%, which is better than the best result of 6.67%, obtained with 7 GoogLeNets using 144 crops per image [Szegedy *et al.*, 2014]. In the last three columns of Tab. 1, we report also the performance of each individual forest dNDF_x ($x = 0, 1, 2$) within the architecture. As expected, dNDF_0 (closest to the input layer) performs worse than dNDF_2, which is the final layer of dNDF**.**NET, but only by 1.34%. Averaging over all three dNDF forest outputs yields the lowest Top5-Error of 7.84%.

6 Conclusions

In this paper we have shown how to model and train stochastic, differentiable decision trees, usable as alternative classifiers for end-to-end learning in (deep) convolutional networks. Prevailing approaches for decision tree training typically operate in a greedy and local manner, making representation learning impossible. To overcome this problem, we introduced stochastic routing for decision trees, enabling split node parameter learning via back-propagation. Moreover, we showed how to populate leaf nodes with their optimal predictors, given the current state of the tree/underlying network. We have successfully validated our new decision forest model on ImageNet and surpassed state-of-the-art when integrating it in the GoogLeNet architecture, without any form of dataset augmentation, further detailed in [Kontschieder *et al.*, 2015].

Acknowledgments Peter and Samuel were partially supported by Novartis Pharmaceuticals and the ASSESS MS project. Madalina was partially supported by NSH Award 1320347, DARPA Contract FA8750-12-2-0324 and the Mobilize Center (NIH U54 EB020405).

References

[Amit and Geman, 1997] Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *(NC)*, 9(7):1545–1588, 1997.

[Bengio *et al.*, 2010] Yoshua Bengio, Olivier Delalleau, and Clarence Simard. Decision trees do not generalize to new variations. *Computational Intelligence*, 26(4):449–467, 2010.

[Bosch *et al.*, 2007] A. Bosch, A. Zisserman, and X. Muñoz. Image classification using random forests and ferns. In *(ICCV)*, 2007.

[Breiman, 2001] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[Brostow *et al.*, 2008] Gabriel J. Brostow, Jamie Shotton, Julien Fauqueur, and Roberto Cipolla. Segmentation and recognition using structure from motion point clouds. In *(ECCV)*. Springer, 2008.

[Caruana *et al.*, 2008] R. Caruana, N. Karampatziakis, and A. Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *(ICML)*, pages 96–103, 2008.

[Criminisi and Shotton, 2013] A. Criminisi and J. Shotton. *Decision Forests in Computer Vision and Medical Image Analysis*. Springer, 2013.

[Fei-Fei, 2015] Andrej Karpathy Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *(CVPR)*, 2015.

[Hastie *et al.*, 2009] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, 2009.

[He *et al.*, 2015] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[Heath *et al.*, 1993] David Heath, Simon Kasif, and Steven Salzberg. Induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2(2):1–32, 1993.

[Kontschieder *et al.*, 2013] P. Kontschieder, P. Kohli, J. Shotton, and A. Criminisi. GeoF: Geodesic forests for learning coupled predictors. In *(CVPR)*, pages 65–72, 2013.

[Kontschieder *et al.*, 2015] P. Kontschieder, M. Fiterau, A. Criminisi, and S. Rota Bulò. Deep neural decision forests. In *(ICCV)*, 2015.

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. Imagenet classification with deep convolutional neural networks. In *(NIPS)*, 2012.

[Lin *et al.*, 2013] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *CoRR*, abs/1312.4400, 2013.

[Montillo *et al.*, 2013] A. Montillo, J. Tu, J. Shotton, J. Winn, J. E. Iglesias, D. N. Metaxas, and A. Criminisi. Entangled forests and differentiable information gain maximization. In *Decision Forests in Computer Vision and Medical Image Analysis*. Springer, 2013.

[Rota Bulò and Kontschieder, 2014] Samuel Rota Bulò and Peter Kontschieder. Neural decision forests for semantic image labelling. In *(CVPR)*, 2014.

[Russakovsky *et al.*, 2014] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 2014.

[Shotton *et al.*, 2013] Jamie Shotton, Ross Girshick, Andrew Fitzgibbon, Toby Sharp, Mat Cook, Mark Finocchio, Richard Moore, Pushmeet Kohli, Antonio Criminisi, Alex Kipman, and Andrew Blake. Efficient human pose estimation from single depth images. *(PAMI)*, 2013.

[Srivastava *et al.*, 2014] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[Szegedy *et al.*, 2014] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[Yu and Deng, 2014] D. Yu and L. Deng. *Automatic Speech Recognition: A Deep Learning Approach*. Springer, 2014.

Im2Calories: towards an automated mobile vision food diary

Austin Myers, Nick Johnston, Vivek Rathod, Anoop Korattikara, Alex Gorban
Nathan Silberman, Sergio Guadarrama, George Papandreou, Jonathan Huang, Kevin Murphy
amyers@umd.edu, (nickj, rathodv, kbanoop, gorban)@google.com
(nsilberman, sguada, gpapan, jonathanhuang, kpmurphy)@google.com

We present a system which can recognize the contents of your meal from a single image, and then predict its nutritional contents, such as calories. The simplest version assumes that the user is eating at a restaurant for which we know the menu. In this case, we can collect images offline to train a multi-label classifier. At run time, we apply the classifier (running on your phone) to predict which foods are present in your meal, and we lookup the corresponding nutritional facts. We apply this method to a new dataset of images from 23 different restaurants, using a CNN-based classifier, significantly outperforming previous work. The more challenging setting works outside of restaurants. In this case, we need to estimate the size of the foods, as well as their labels. This requires solving segmentation and depth / volume estimation from a single image. We present CNN-based approaches to these problems, with promising preliminary results.

1. Introduction

Many people are interested in tracking what they eat to help them achieve weight loss goals or manage their diabetes or food allergies. However, most current mobile apps (MyFitnessPal, LoseIt, etc) require manual data entry, which is tedious and time consuming. Consequently, most users do not use such apps for very long [9]. Furthermore, amateur self-reports of calorie intake typically have an error rate that exceeds 400 calories per day [31, 5].

Rather than rely purely on data entry, several approaches make use of a mobile camera to aid in this task. Cordeiro *et al*. [8] records a photo of the meal but does not perform any visual analysis of the image. Several previous approaches [23] [29] rely on an expert nutritionists to analyse the image offline (at the end of each day). Other approaches [26] [25] use crowd sourcing to interpret the image, in lieu of an expert. However, crowd sourcing is both costly and slow, which hinders widespread adoption.

Several existing works [39, 21, 37] do use computer vision algorithms to reason about meals but only work in laboratory conditions where the food items are well separated and the number of categories is small. Furthermore, most of these methods use traditional, hand-crafted visual features, and only use machine learning at the classification stage.

The holy grail is an automatic method for estimating the nutritional contents of a meal from one or more images. Unfortunately, even a perfect visual interpretation of the scene cannot perfectly predict what is inside many foods, e.g., a burrito. Consequently, an ideal system is one which correctly infers what is knowable and prompts the user for feedback on inherantly ambiguous components of a meal. Our goal is to minimize user effort for completing food diaries by offering smart "auto-complete" functionality, rather than complete automation.

In this paper, we take some initial steps towards such a system. Our approach utilizes several deep learning algorithms, tailored to run on a conventional mobile phone, trained to recognize food items and predict the nutrional contents meals from images taken "in the wild". We refer to this task as the "Im2Calories" problem, by analogy to the recent line of work on the "Im2Text" problem. It should be stressed, however, that we are interested in estimating various other properties of a meal (such as fat and carbohydrates) and not just calories.

We start by building on [1], who developed a system that can predict the calorie content of a meal from a single image. Their key insight (also made in [2]) was that the problem becomes much simpler if we restrict the setting to one where the user is eating in a restaurant whose menu is known. In this case, the problem reduces to one of detecting which items, out of the K possible items on the menu, the user has chosen. Each item typically has a standard serving size[1], and we typically know its nutritional contents.[2], whereas getting nutritional information for arbitrary cooked foods is much harder, as discussed in Section 7.

In Section 3, we show that by simply replacing the hand-crafted features used in [1] with a convolutional neural network (CNN), we can significantly improve performance,

[1] There may be variants, but these are typically few in number (e.g., small, medium or large fries). Hence we an treat these as different items.

[2] The US Food and Drug Administration has passed a law requiring all major chain restaurants to post the nutritional contents of their meals, starting in December 2016. See [15] for details.

both at labeling the foods and estimating total calories. We then extend their method from 3 restaurants to 23 restaurants, increasing the coverage from 41 food items to 2517. We show that the same basic multilabel classification system continues to work.

Unfortunately, it is hard to get enough images for all the menu items for all the restaurants in the world. And even if we could, this would not help us when the user is not eating at a restaurant. Therefore, in Section 4, we develop a set of 201 generic, restaurant-independent food tags. We then extend the existing public Food101 dataset [3] with these tags using crowdsourcing. We call the resulting dataset Food201-multilabel and plan to release it publicly.[3] We show that the same kind of CNN-based multi-label classifier also works fairly well on this new (larger and more challenging) dataset, although we found it necessary to perform some clustering of visually indistinguishable labels in order to get acceptable performance.

Of course, detecting the presence of a food item in an image is not sufficient, since most items can be "parameterized" in terms of size, toppings, etc. Note that this is true even in the restaurant setting. We could of course ask the user about these variants, but we would like to do as much as possible automatically.

To be able to perform such fine grained classification, we need to be able to localize the objects within the image and extract detailed features. We argue that a segmentation-based approach is more appropriate than the traditional bounding-box approach, since food items can have highly variable shape. In Section 5, we present the new Food201-segmented dataset, and our approach to semantic image segmentation. We show that leveraging the multilabel classifier from the earlier stage can help with segmentation, since it provides a form of "context".

Once we have segmented the foods, we can try to estimate their volume. To do this, we need to know the surface height of the foods above the plate. In Section 6, we present some preliminary results on estimating the depth of each pixel from a single RGB image, using a CNN. We then show promising results on estimating the volumes of foods.

In summary, this paper makes 3 main contributions. First, we develop a system that can recognize the contents of a restaurant meal much more accurately than the previous state of the art, and at a much larger scale. Second, we introduce a new dataset, Food201, and show how it can be used to train and test image tagging and segmentation systems. Third, we show some promising preliminary results on the challenging problem of mapping image to calories from images taken in the wild, in a non-restaurant setting. Our overall system is illustrated in Figure 1.

[3] See https://storage.googleapis.com/food201/food201.zip.

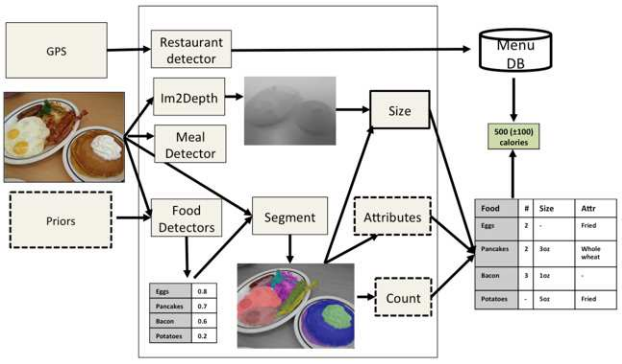


Figure 1. Illustration of the overall system. Dotted boxes denote components that have not yet been implemented. The input is one or more images, and optionally a GPS signal and user priors (e.g., concerning food preferences). The output is a food diary, and an estimated total calorie count (in cases where a suitable nutritional database is available, such as from a restaurant's menu).

Name	#Classes	#Train	#Test	Comments
Food101	101	75,750	25,250	[3]
Food101-Background	2	151,500	50,500	Food vs non-food
Food201-MultiLabel	201	35,242	15,132	Multi label
Food201-Segmented	201	10,100	2,525	Label per pixel
Restaurant	2517	75k	25k	Single label
Gfood-3d	-	150k	2471	Depth per pixel
Nfood-3d	-	-	1050	Depth per pixel

Table 1. Summary of the datasets used in this paper. The Restaurant dataset contains web images for 23 restaurants. Gfood-3d is from 50 Google meals. Nfood-3d is from 11 meals made with Nasco food replicas.

2. Meal detection

The first step in our pipeline is to determine if the image is an image of a meal at a "reasonable" distance from the camera. We can phrase this as a simple binary classification problem. To tackle this problem, we need to have a suitable dataset. We start by taking the Food101 dataset developed in [3], which contains 101 classes of food, 1000 images each (750 train, 250 test). Food101 is the largest publicly available dataset of food images we are aware of (see Table 1 for a comparison with some other public datasets).

The Food101 dataset is designed for multi-class classification. To make a dataset suitable for binary classification, we combined all the food classes into one generic "food" class, and then randomly extracted an equal number of images from the ImageNet challenge dataset [30] to create the "non-food" class.[4]

Each image is rescaled (if necessary) so that its maximum height or width is 512 pixels, but preserving the original aspect ratio. We call this new dataset the **Food101-**

[4] ImageNet actually contains 51 food classes, so we removed these images from the negative set.

Background dataset.

To train a classifier for this problem, we took the GoogLeNet CNN model from [34], which had been pre-trained on ImageNet, removed the final 1000-way softmax, and replaced it with a single logistic node. Then we fine tune the whole model on the Food101-Background dataset; this takes about 12 hours using a single Titan X GPU with 12 GB of memory. The final test set accuracy is 99.02%.

3. Restaurant-specific im2calories

Once we have determined that the image contains a meal, we try to analyze its contents. The first step is to determine which restaurant the user is in. In this paper, we use Google's Places API [27] for this. We then retrieve the menu of the nearest restaurant from the web,[5] parse the menu into a list of K food items, and retrieve images for each of these, either by performing web search (as in [2]) or by asking users to collect images (as in [1]).[6]

Once we have the dataset, we can train a classifier to map from image to label. Since the user may have multiple food items on their plate, it is better to use a multi-label classifier rather than using a multi-class classifier, which assumes the labels are mutually exclusive. Next we can estimate the set of foods that are present by picking a suitable threshold ϕ, and then computing $\mathcal{S} = \{k : p(y_k = 1|x) > \phi\}$, where $p(y_k = 1|x)$ is the probability that food k is present in image x. Finally, we can lookup each of these food items in our (restaurant specific) database, and then estimate the total calories as follows: $\hat{C} = \sum_{k \in \mathcal{S}} C_k$, where C_k is the calorie content of menu item k. Alternatively, to avoid having to specify the threshold ϕ, we can compute $\bar{C} = \sum_{k=1}^{K} p(y_k = 1|x) C_k$. We compare these methods below.

3.1. Experiments on MenuMatch dataset

To evaluate this approach, we used the dataset from [1], known as "MenuMatch". This consists of 646 images, tagged with 41 food items, taken from 3 restaurants. Unlike other datasets, each image in MenuMatch has a corresponding ground truth calorie estimate, computed by an expert nutritionist. In addition, each restaurant's menu has corresponding ground truth calorie content per item.

[1] used various hand-crafted features together with a linear SVM, trained in a one-vs-all fashion. Their best performing system achieved a mean average precision (mAP) of 51.2% on the test set. By contrast, we get much better results using a deep learning approach, as we explain below.

Method	Mean error	Mean absolute error
Baseline	-37.3 ± 3.9	239.9 ± 1.4
Meal Snap	-268.5 ± 13.3	330.9 ± 11.0
Menu Match	-21.0 ± 11.6	232.0 ± 7.2
$\hat{C}$	-31.90 ± 28.10	163.43 ± 16.32
$\bar{C}$	-25.35 ± 26.37	152.95 ± 15.61

Table 2. Errors in calorie estimation on the MenuMatch dataset. $\hat{C}$ and $\bar{C}$ are our methods. Numbers after the $\pm$ sign are standard errors estimated by 5-fold cross-validation. See text for details.

We took the GoogLeNet CNN model from [34], which had been pre-trained on ImageNet, removed the final 1000-way softmax, replaced it with a 101-way softmax, and fine-tuned the model on the Food101 dataset [3]. The resulting model has a classification accuracy on the Food101 test set of 79%, which is significantly better than the 50.76% reported by [3] (they used hand crafted features plus an SVM classifer using a spatial pyramid matching kernel).

Next, we took the model which was trained on Food101, removed the 101-way softmax, replaced it with 41 logistic nodes, and fine-tuned on the MenuMatch training set. (Pre-training on Food101 was necessary since the MenuMatch dataset is so small.) The resulting mAP on the test set is 81.4%, which is significantly higher than the best result of 51.2% reported in [1].

Finally, we wanted to assess the accuracy of calorie prediction. We compared 5 methods: the MenuMatch system of [1], the MealSnap app [25] that uses crowdsourcing, our method using $\hat{C}$, our method using $\bar{C}$, and finally, a baseline method, which simply computes the empirical mean of the calorie content of all meals from a specific restaurant. The results are shown in Table 2. We see that our system has considerably lower error than MenuMatch and the crowdsourced MealSnap app. (The unreliability of MealSnap was also noted in [26].) In fact, we see that MenuMatch barely beats the baseline approach of predicting the prior mean.

3.2. Scaling up to more restaurants

The MenuMatch results are based on 646 images of 41 food items from 3 restaurants. In this Section, we discuss our attempts to scale up these experiments.

First, we downloaded the menus for the top 25 restaurants in the USA, as ranked by sales.[7] We decided to drop "Pizza Hut" and "Chipotle", since gathering images for their menu items was tricky.[8] From the remaining 23

[5] This data is available for many US chain restaurants in semi-structured form from https://www.nutritionix.com.

[6] In practice, it is suprisingly complicated to parse the menus and retrieve relevant images. For example, the restaurant "16 handles" contains the folowing items: NSA Blueberry Tease, NSA Chocolate Eruption, NSA Strawberry Fields, and 65 other similar entries. You need to know that these are from the frozen yogurt section of the menu, and that NSA stands for "no sugar added", in order to make any sense of this data.

[7] Source: http://nrn.com/us-top-100/top-100-chains-us-sales.

[8] For example, "Pizza Hut" has menu items such as "chicken", "pepperoni", etc. But these are toppings for a pizza, not individual food items. Similarly, "Chipotle" lists "chicken", "beans", etc. But these are fillings for a burrito, not individual food items.

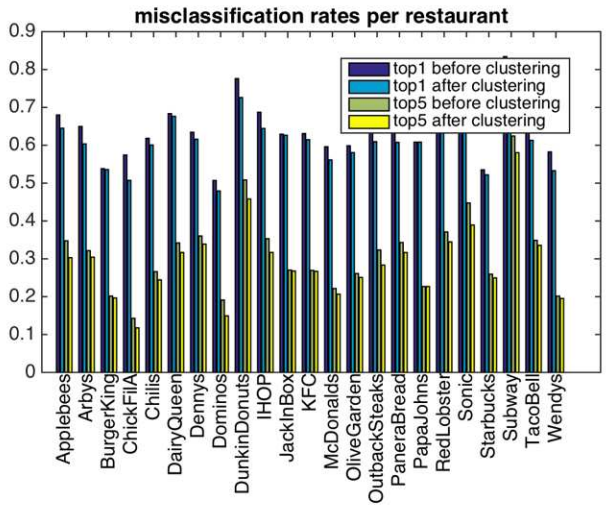


Figure 2. Top 1 and top 5 error rates on the test set for 23 different restaurants, before and after clustering the most confusable labels.

restaurants, we collected 4857 menu items. We manually removed drinks and other miscellaneous items, and then scraped 1.2 million images by issuing image search queries of the form "<restaurant name> <item name> (yelp | flickr | instagram | pinterest | foodspotting)". (These site restricts were chosen to increase the chances that we collected user-generated photos, rather than "official" photos from the restaurant itself, which tend to be too easy.) Of the scraped images, 270k were sent to Amazon Mechanical Turk for verification, producing a final set of 2517 menu items and 99k images. We call this the **Restaurant** dataset.

We then took our GoogleLeNet model, which had been trained on ImagetNet and then Food101, and replaced the 101-softmax with 2517 logistic nodes. We trained the final layer of this model on 75% of the data, and tested on the rest. We then computed the top 1 and top 5 error rates, averaged over the food items, for each of the 23 restaurants. The results are shown in Figure 2.

The top 1 error rate is quite high. This is because many food items are extremely similar, and it is hard, even for a human expert, to tell them apart. For example, McDonalds has the following items on its menu: Quarter Pounder Deluxe, Quarter Pounder Bacon Cheese, Quarter Pounder with Cheese, etc. (See Figure 3 for an illustration of these food items.)

To combat this issue, we computed the class confusion matrix on the training set for each restaurant, and then performed a very simple clustering of similar items. In particular, we computed the K nearest neighbor graph, in which we connected each label to the K other labels it is most often mapped to (which can include itself). We then computed the connected components to form a set of clusters.

Figure 3. The first two images are put into the same visual cluster, the third is kept distinct. Image sources: http://www.mcdonaldsmenu.mobi/beefburgersmenu/deluxequarterpounder/. http://www.mcdonaldsmenu.mobi/beefburgersmenu/baconcheesequarterpounder/. http://www.mcdonalds.com/us/en/food/product_nutrition.burgerssandwiches.7.quarter-pounder-with-cheese.html

We found qualitatively that using $K = 1$ gave a good tradeoff between merging the most confusable classes and overclustering. With $K = 1$, the number of clusters was about 0.9 times the original number of items; most clusters were singletons, but some had 2 or 3 items in them. Figure 3 gives an example of two clusters we created from the McDonald's menu; the first cluster contains two very visually similar items (Quarter Pounder Deluxe and Quarter Pounder Bacon Cheese), and the second cluster contains a visually distinctive item (Quarter Pounder with Cheese).

Finally, we evaluated performance of the classifier on the clustered test labels, by defining the probability of a cluster as the max probability of any label in the cluster. Figure 2 shows that, not surprisingly, the error rates decrease. In the future, we hope to try using a hierarchical classifier, which can tradeoff specificity with error rate c.f., [11].

4. Generic food detection

The results from Section 3 required images for each menu item from each restaurant. It is difficult to acquire such data, as previously remarked. To create a more generic dataset, we took half of the Food101 dataset (50k images), and asked raters on Mechanical Turk to name all the food items they could see in each image. We included the original class label as one of the choices, and manually created a list of commonly co-occuring foods as additional choices in the drop-down menu (e.g., eggs often co-occur with bacon). We also allowed raters to enter new terms in a text box. We used 2 raters per image. After manually merging synonymous terms, and removing terms that occurred less than 100 times, we ended up with a vocabulary of 201 labels. On average, each image had 1.9 labels (with a median of 1 and a max of 8).

We split the resulting dataset into 35,242 training images and 15,132 test images, in a way which is consistent with the Food101 train/ test split. We call this the **Food201-MultiLabel** dataset. See Table 1 for a summary of how this dataset compares to other public food datasets.

Next, we developed a multi-label classifier for this

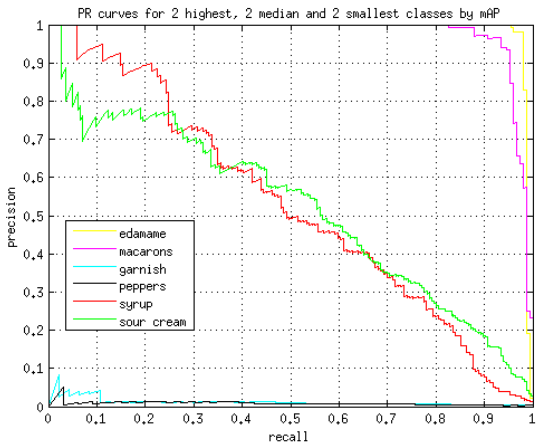


Figure 4. Precision-recall curves for 6 classes, ranging from best to worst, on the Food201-MultLabel dataset.

dataset, using the same method as in Section 3 (namely taking the GoogLeNet model, replacing the 101-way softmax with 201 logistic nodes). This takes about half a day to train on a single GPU. We then compute the average precision (area under the precision-recall curve) for each class, and average this over the classes, to compute the mean average precision (mAP).

The mAP is 0.8 for classes in Food 101, 0.2 for classes outside Food 101, and 0.5 overall. Not surprisingly, we do better on the original Food101 classes, since the new classes often correspond to side dishes or smaller food items, and are much less frequent in the training data. The top 3 classes were: edamame (0.987), macarons (0.976), hot and sour soup (0.956). The bottom 3 classes were: cream (0.015), garnish (0.014), peppers (0.010). Figure 4 shows the precision-recall curves for some of these classes.

5. Semantic image segmentation

In addition to predicting the presence of certain foods, it is useful to localize them in the image. Since most foods are amorphous, it is better to segment out the region corresponding to each food, rather than putting a bounding box around them. Such a segmented image will enable further analysis, such as counting and size estimation (see below), which is essential for nutrition estimation. We can also allow the user to interactively edit the estimated segmentation mask, in order to improve accuracy of the system (although we leave this to future work).

To train and evaluate semantic image segmentation systems, we took a subset of the Food201-MultiLabel dataset (12k images), and asked raters on a crowd computing platform to manually segment each of the food items that have been tagged (in an earlier stage) in that each image. We used 1 rater per image, and an internal tool that leverages grabcut to make this labeling process reasonably fast. Raters had the option of skipping foods that were too hard to segment. Thus some foods are not segmented, resulting in false negatives. We call this the **Food201-segmented** dataset.

Note that we are segmenting each class, as in the PASCAL VOC semantic segmentation challenge [13]. Arguably, instance-level segmentation (see e.g., [17]) would be more useful, since it distinguishes different instances of the same class, which would enable us to count instances.[9]. However, this is quite difficult. For example, consider distinguishing pancake 1 from pancake 2 in the food image in Figure 1: this is quite challenging, since the top pancake almost completely covers the bottom one. Furthemore, segmenting the eggs into 2 instances is even harder, since the boundary is not well defined. We leave instance-level segmentation to future work.

To tackle the segmentation problem, we use the "DeepLab" system from [6].[10] This model uses a CNN to provide the unary potentials of a CRF, and a fully connected graph to perform edge-sensitive label smoothing (as in bilateral filtering).

The model is initialized on ImageNet, and then fine-tuned on Food201-segmented, which takes about 1 day on a single GPU. For the 3 CRF parameters, which control the strength of the edge potentials, we use the parameter values from [6]; these were chosen by grid search, to minimize validation error on held-out training data.

The label set in the Food201-Segmented dataset is much larger than in the VOC challenge (201 labels instead of just 20). Consequently, the baseline model has a tendency to generate a lot of false positives. To improve performance, we take the probability vector $p(y_k = 1|x)$ computed by the multi-label classifier from Section 4, find the top 5 entries, and then create a binary mask vector, which is all 0s except for the top 5 labels, plus the background. We then multiply the per-pixel label distribution from the segmentation CNN by this sparse vector, before running the CRF smoothing. This provides a form of global image context and improves the results considerably (see below).

We show some sample results in Figure 5. Consider the last row, for example. We see that the context from the multilabel model helps by eliminating false positives (e.g., caprese salad and caesar salad). We also see that the ground truth is sometimes incomplete (e.g., the burrito is not actu-

[9] It is more natural for the user if the system records in their food diary that they ate 3 slices of pizza rather than, say, 120 ounces of pizza. It is also much easier for the user to interactively fix errors related to discrete counts than continuous volumes. Of course, this assumes we know what the size of each slice is. We leave further investigation to future work.

[10] At the time this paper was submitted, DeepLab was the best performing method on the PASCAL VOC challenge (see http://host.robots.ox.ac.uk:8080/leaderboard/displaylb.php?challengeid=11&compid=6). At the time of writing the camera ready version, the best performing method is an extension of DeepLab that was additionally trained on MS-COCO data.

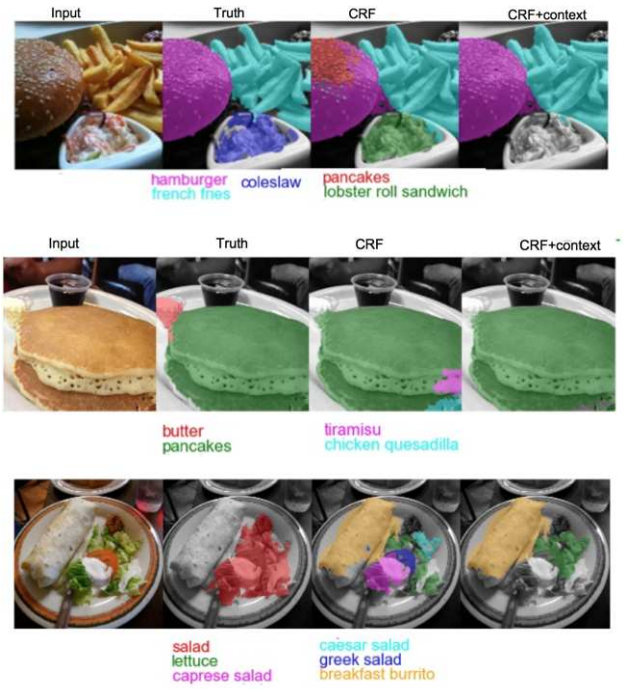


Figure 5. Examples of semantic image segmentation on some images from the Food201-Segmented test set. First column: original image. Second column: ground truth. Third column: predictions using CNN/CRF. Fouth column: predictions using CNN/CRF with multilabel context. Best viewed in color.

CRF?	Context?	Acc	Recall	IoU
0	0	0.71	0.30	0.19
1	0	0.74	0.32	0.22
0	1	0.74	0.32	0.23
1	1	0.76	0.33	0.25

Table 3. Performance on the Food101-Segmented test set.

ally labeled by the human). Finally, we see that the label space is somewhat arbitrary: the ground truth uses the term "salad", whereas the model predicts "lettuce". Currently we ignore any semantic similarities between the labels.

The performance of our system with and without the CRF, and with and without multilabel context, is shown in Table 3. The performance in terms of IoU is much lower than on the PASCAL VOC 2012 segmentation challenge. We believe this is due to several factors: (1) we have 201 foreground labels to predict, whereas VOC just has 20; (2) our labeled dataset suffers from incompleteness, which can result in correct predictions being erroneously being counted as false positives, as well as "polluting" the background class during training; (3) our task is arguably intrinsically harder, since the categories we wish to segment are more deformable and varied in their appearance than most of the VOC categories.

6. Volume estimation

Having segmented the foods, the next step is to estimate their physical size (3d volume).

We first predict the distance of every pixel from the camera, using the same CNN architecture as in [12] applied to a single RGB image. We trained our model on the NYU v2 RGBD dataset of indoor scenes, and then fine tuned it on a new 3d food dataset we collected which we call the **GFood3d** dataset (G for Google). This consists of short RGBD videos of 50 different meals from various Google cafes collected using the Intel RealSense F200 depth sensor.[11] In total, the dataset has about 150k frames. (Note that each RGB image has a corresponding depth image, but otherwise this is unlabeled data.)

On a test set of 2,471 images (recorded in a different set of Google cafes), we get an average relative error of 0.18 meters, which is slightly better to the performance reported in [12] on the NYU dataset. Figure 6 shows a qualitative example. We see that the CNN is able to predict the depth map fairly well, albeit at a low spatial resolution of 75 x 55. (For comparison with the F200 data, we upsample the predicted depth map).

The next step is to convert the depthmap into a voxel representation. To do this, we first detect the table surface using RANSAC. Next, we project each pixel into 3d space, exploiting the known intrinsic parameters of the F200 sensor. Finally, we tile the table surface with a 2d grid (using a cell size of 2mm x 2mm), and compute the average height of all the points in each cell, thus creating a "tower" of that height. For an example of the result of this process, see Figure 7.

Given a voxel representation of the food, and a segmentation mask, we can estimate the volume of each food item. To measure the accuracy of this approach, we purchased the "MyPlate" kit from Nasco.[12] This contains rubber replicas of 42 food items in standard sizes, and is used for training nutritionists. We verified the actual size of these items using the water displacement method (we found that the measured size was somewhat smaller than the quoted sizes). We then created 11 "meals" from these food replicas, and recorded images of them using the F200. Specifically, we put the meals on a turntable, and automatically took 100 images as the plate went through a full rotation. The resulting dataset has 1050 depth images. We call this the **NFood-3d** dataset (N for Nasco).

To measure performance of our system, we compute the absolute error in the volume estimate. We can estimate the volume using the true depth map (from F200) or the pre-

[11] The F200 has a spatial resolution of 1920 x 1080 pixels, and a sensing range of 0.2–1.2m. The Kinect2 camera has the same resolution, but a sensing range of 0.8–3.5m. Thus the F200 is a better choice for close up images.

[12] Source: http://www.enasco.com/product/WA29169HR.

Figure 6. (a) An image from the GFood-3d test set. (b) Depth recorded from RealSense RGBD sensor. (c) Depth predicted by CNN.

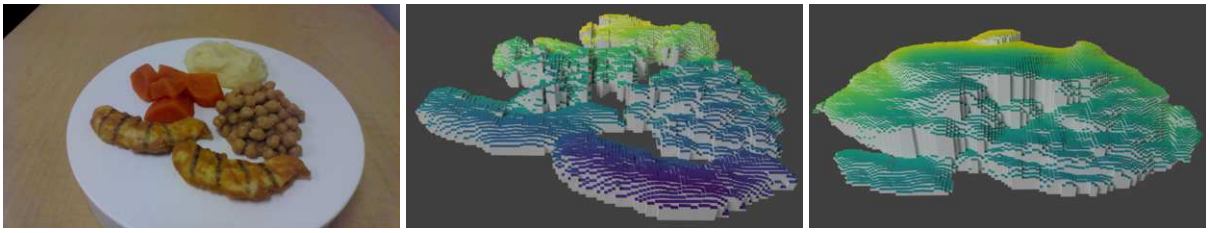


Figure 7. (a) An image from the NFood-3d dataset. (b) Voxel grid derived from RealSense depthmap. (c) Voxel grid estimated from CNN predicted depthmap. Size of grid cell is 2mm x 2mm.

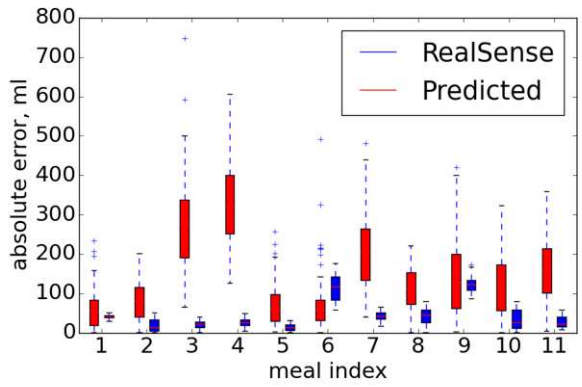


Figure 8. Absolute error in volume estimation (in ml) across the 11 meals in the NFood-3d dataset. Each meal has 100 images, taken from different viewing angles. The boxplots plot the distribution of errors across these 100 images.

dicted depth map (from CNN), and using the true segmentation mask (from human) or the predicted segmentation mask (from CNN). Unfortunately, we have found that our segmentation system does not work well on the NFood images, because their color and texture properties are too dissimilar to real food. However, we were able to compare the quality of using the true depth map and predicted depth map. The results are shown in Figure 8. We see that for most of the meals, our CNN volume predictor is quite accurate.

7. Calorie estimation

The final step is to map from the volume to the calorie content. This requires knowing the calorific density of each kind of food. The standard source for this is the USDA National Nutrient Database (NNDB). The latest version (May 2015) is Standard Release 27 [35], which lists nutritional facts about 8618 basic foods. However, we have noted a discrepancy of up to 35% in the calorie contents between the USDA NNDB and the numbers quoted by Nasco for their food replicas.[13]

A more serious problem is that the NNDB focuses on "raw" foods, rather than cooked meals. For example, NNDB contains information such as the calorie content of a pound of beef, but this does not help us estimate the calorie content of a cooked burger. For prepared foods, it is better to use the USDA Food and Nutrient Database for Dietary Studies (FNDDS) [16], which is derived from NNDB.[14] However, the calorie content can vary a lot depending on exactly how the food was prepared (e.g., grilling vs frying). Consequently, we do not yet have a broad coverage nutritional database for prepared foods, and therefore we have not yet been able to conduct an end-to-end test of our system outside of the restaurant setting.[15]

[13] According to http://www.enasco.com/product/WA29109HR, "nutrition data is provided by The Nutrition Company using their FoodWorks nutrient analysis software". Recall that these food replicas are used to train professional nutritionists, so the information cards that accompany them are designed to be as accurate as possible.

[14] For a detailed comparison of NDB-SR and FNDDS, see http://www.nutrientdataconf.org/PastConf/NDBC36/W-3_Montville_NNDC2012.pdf.

[15] Companies such as MyFitnessPal have developed much larger nutri-

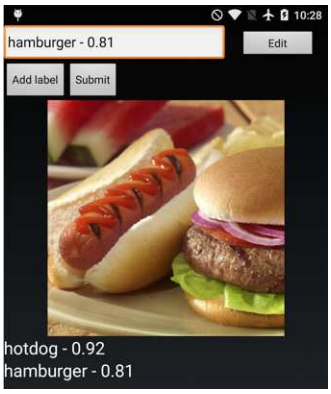


Figure 9. Screenshot of the mobile app. Note that it is running in airplane mode (no internet connection).

8. Mobile app

The complete system is sketched in Figure 1. We have ported the restaurant detector, meal detector, and food detectors (the multi-label classifiers) to the Android mobile phone system. The app can classify any image in under 1 second. The memory footprint for the model is less than 40MB (using unquantized floats to represent the CNN weights). However, we have not yet ported the segmentation or depth estimation CNNs.

To use the app, the user takes a photo and then our system processes it. First the system determines if the image contains a meal, and if you are at a known restaurant. If so, it applies the multilabel classifiers. We sort the possible labels by their predicted probability, taking all those above a threshold of 0.5, and then truncating to the top 5, if necessary. We then show these labels to the user (see Figure 9 for a screenshot). The user can dismiss any labels that are incorrect. He/ she can also enter new labels, either by selecting from a pull-down menu (sorted in order of probability), or by entering free text. The user's image and labels are then optionally stored in the cloud for subsequent offline model retraining (although we have not yet implemented this).

9. Related work

There is a large number of related prior publications. Here we mention a few of them.

In terms of public datasets, we have already mentioned Food101 [3], which has 101 classes and 101k images. In addition, there are various smaller datasets, such as: PFID [7], which has 61 classes (of fast food) and 1098 images; UNICT-FD889 [14], which has 889 classes and 3853 images; and UECFOOD-100 [24], which has 100 classes (of Japanese food), and 9060 images.

tional databases using crowd sourcing [22], but this data is proprietary, and its quality is not guaranteed (since most casual users will not know the true nutritional content of their food).

In terms of classifiers, [3, 18] use SVMs for generic foods. [2] and [1] develop restaurant-specific multi-label classifiers. Some recent papers on food segmentation include [28, 33, 38, 37].

There are many papers that leverage structure from motion to develop a 3d model of the food, including [28, 21, 10, 36, 32]. However, all these methods require multiple images and calibration markers. In terms of single images, [4] use parametric shape models for a small set of food types (e.g., sphere for an apple), and [19] use a nearest neighbor regression method to map the 2d image area to physical mass of each food item.

There are very few papers that predict calories from images. [33] apply semantic image segmentation, and then train a support vector regression to map from the number of pixels of each class to the overall number of calories in the meal. [4, 19] calculate calories by multiplying the estimated volume of each food by the calorie density. [26] use crowdsourcing to estimate calories. [1] relies on the restaurant's menu having the calorie information.

10. Discussion

Besides its obvious practical use, the Im2Calories problem is also very interesting from the point of view of computer vision research. In particular, it requires solving various problems, such as: fine-grained recognition (to distinguish subtly different forms of food); hierarchical label spaces (to handle related labels); open-world recognition (to handle an unbounded number of class names); visual attribute recognition (to distinguish fried vs baked, or with or without mayo); instance segmentation; instance counting; amodal completion of occluded shapes [20]; depth estimation from a single image; information fusion from multiple images in real time, on-device; etc. We have tackled some of these problems, but it is clear that there is much more work to do. Nevertheless, we believe that even a partial solution to these problems could be of great value to people.

References

[1] O. Beijbom, N. Joshi, D. Morris, S. Saponas, and S. Khullar. Menu-match: restaurant-specific food logging from images. In *WACV*, 2015.

[2] V. Bettadapura, E. Thomaz, A. Parnami, G. D. Abowd, and I. Essa. Leveraging context to support automated food recognition in restaurants. In *WACV*, pages 580–587, 2015.

[3] L. Bossard, M. Guillaumin, and L. Van Gool. Food-101: Mining discriminative components with random forests. In *ECCV*, 2014.

[4] J. Chae, I. Woo, S. Kim, R. Maciejewski, F. Zhu, E. J. Delp, C. J. Boushey, and D. S. Ebert. Volume estimation using food specific shape templates in mobile image-based dietary assessment. In *Proc. SPIE*, 2011.

[5] C. Champagne, G. Bray, A. Kurtz, J. Montiero, E. Tucker, J. Voaufova, and J. Delany. Energy intake and energy expenditure: a controlled study comparing dietitians and non-dietitians. *J. Am. Diet. Assoc.*, 2002.

[6] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected CRFs. In *ICLR*, 2015.

[7] M. Chen, K. Dhingra, W. Wu, L. Yang, R. Sukthankar, and J. Yang. PFID: Pittsburgh fast-food image dataset. In *ICIP*, pages 289–292, 2009.

[8] F. Cordeiro, E. Bales, E. Cherry, and J. Fogarty. Rethinking the mobile food journal: Exploring opportunities for lightweight Photo-Based capture. In *CHI*, 2015.

[9] F. Cordeiro, D. Epstein, E. Thomaz, E. Bales, A. K. Jagannathan, G. D. Abowd, and J. Fogarty. Barriers and negative nudges: Exploring challenges in food journaling. In *CHI*, 2015.

[10] J. Dehais, S. Shevchik, P. Diem, and S. G. Mougiakakou. Food volume computation for self dietary assessment applications. In *13th IEEE Conf. on Bioinfo. and Bioeng.*, pages 1–4, Nov. 2013.

[11] J. Deng, J. Krause, A. C. Berg, and L. Fei-Fei. Hedging your bets: Optimizing accuracy-specificity trade-offs in large scale visual recognition. In *CVPR*, pages 3450–3457, June 2012.

[12] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common Multi-Scale convolutional architecture. *Arxiv*, 18 Nov. 2014.

[13] M. Everingham, S. M. Ali Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *IJCV*, 111(1):98–136, 25 June 2014.

[14] G. M. Farinella, D. Allegra, and F. Stanco. A benchmark dataset to study the representation of food images. In *ECCV Workshop Assistive CV*, 2014.

[15] FDA. www.fda.gov/Food/ IngredientsPackagingLabeling/LabelingNutrition/ucm248732.htm.

[16] USDA FNDDS. www.ars.usda.gov/ba/bhnrc/fsrg.

[17] B. Hariharan, P. Arbel, R. Girshick, and J. Malik. Simultaneous detection and segmentation. In *ECCV*, 2014.

[18] H. He, F. Kong, and J. Tan. DietCam: Multiview Food Recognition Using a MultiKernel SVM. *IEEE J. of Biomedical and Health Informatics*, 2015.

[19] Y. He, C. Xu, N. Khanna, C. J. Boushey, and E. J. Delp. Food image analysis: Segmentation, identification and weight estimation. In *ICME*, pages 1–6, July 2013.

[20] A. Kar, S. Tulsiani, J. Carreira, and J. Malik. Amodal completion and size constancy in natural scenes. In *Intl. Conf. on Computer Vision*, 2015.

[21] F. Kong and J. Tan. DietCam: Automatic dietary assessment with mobile camera phones. *Pervasive Mob. Comput.*, 8(1):147–163, Feb. 2012.

[22] R. Macmanus. How myfitnesspal became the king of diet trackers, 2015. readwrite.com/2015/02/23/trackers-myfitnesspal-excerpt.

[23] C. K. Martin, H. Han, S. M. Coulon, H. R. Allen, C. M. Champagne, and S. D. Anton. A novel method to remotely measure food intake of free-living individuals in real time: the remote food photography method. *British J. of Nutrition*, 101(03):446–456, 2009.

[24] Y. Matsuda, H. Hoashi, and K. Yanai. Recognition of Multiple-Food images by detecting candidate regions. In *ICME*, pages 25–30, July 2012.

[25] Mealsnap app. tracker.dailyburn.com/apps.

[26] J. Noronha, E. Hysen, H. Zhang, and K. Z. Gajos. PlateMate: Crowdsourcing nutrition analysis from food photographs. In *Proc. Symp. User interface software and tech.*, 2011.

[27] Google places API. https://developers.google.com/places/.

[28] M. Puri, Z. Zhu, Q. Yu, A. Divakaran, and H. Sawhney. Recognition and volume estimation of food intake using a mobile device. In *WACV*, pages 1–8, Dec. 2009.

[29] Rise app. https://www.rise.us/.

[30] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *arXiv:1409.0575*, 2014.

[31] D. Schoeller, L. Bandini, and W. Dietz. Inaccuracies in self-reported intake identified by comparison with the doubly labelled water method. *Can. J. Physiol. Pharm.*, 1990.

[32] T. Stutz, R. Dinic, M. Domhardt, and S. Ginzinger. Can mobile augmented reality systems assist in portion estimation? a user study. In *Intl. Symp. Mixed and Aug. Reality*, pages 51–57, 2014.

[33] K. Sudo, K. Murasaki, J. Shimamura, and Y. Taniguchi. Estimating nutritional value from food images based on semantic segmentation. In *CEA workshop*, pages 571–576, 13 Sept. 2014.

[34] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[35] USDA National Nutrient Database for Standard Reference, Release 27 (revised). http://www.ars.usda.gov/ba/bhnrc/ndl.

[36] C. Xu, N. Khanna, C. B. Y. He, and A. Parra. Image-Based food volume estimation. In *CAE workshop*, 2013.

[37] W. Zhang, Q. Yu, B. Siddiquie, A. Divakaran, and H. Sawhney. 'Snap-n-eat': Food recognition and nutrition estimation on a smartphone. *J. Diabetes Science and Technology*, 9(3):525–533, 2015.

[38] F. Zhu, M. Bosch, N. Khanna, C. J. Boushey, and E. J. Delp. Multiple hypotheses image segmentation and classification with application to dietary assessment. *IEEE J. of Biomedical and Health Informatics*, 19(1):377–388, Jan. 2015.

[39] F. Zhu, M. Bosch, I. Woo, S. Kim, C. J. Boushey, D. S. Ebert, and E. J. Delp. The use of mobile devices in aiding dietary assessment and evaluation. *IEEE J. Sel. Top. Signal Process.*, 4(4):756–766, Aug. 2010.

DeepBox: Learning Objectness with Convolutional Networks

Weicheng Kuo Bharath Hariharan Jitendra Malik
University of California, Berkeley
{wckuo, bharath2, malik}@eecs.berkeley.edu

Abstract

Existing object proposal approaches use primarily bottom-up cues to rank proposals, while we believe that "objectness" is in fact a high level construct. We argue for a data-driven, semantic approach for ranking object proposals. Our framework, which we call DeepBox, uses convolutional neural networks (CNNs) to rerank proposals from a bottom-up method. We use a novel four-layer CNN architecture that is as good as much larger networks on the task of evaluating objectness while being much faster. We show that DeepBox significantly improves over the bottom-up ranking, achieving the same recall with 500 proposals as achieved by bottom-up methods with 2000. This improvement generalizes to categories the CNN has never seen before and leads to a 4.5-point gain in detection mAP. Our implementation achieves this performance while running at 260 ms per image.

1. Introduction

Object detection methods have moved from scanning window approaches [9] to ones based on bottom-up object proposals [11]. Bottom-up proposals [1] have two major advantages: 1) by reducing the search space, they allow the usage of more sophisticated recognition machinery, and 2) by pruning away false positives, they make detection easier. [14, 10]

Most object proposal methods rely on simple bottom-up grouping and saliency cues. The rationale for this is that this step should be reasonably fast and generally applicable to all object categories. However, we believe that there is more to objectness than bottom-up grouping or saliency. For instance, many disparate object categories might share high-level structures (such as the limbs of animals and robots) and detecting such structures might hint towards the presence of objects. A proposal method that incorporates these and other such cues is likely to perform much better.

In this paper, we argue for a *semantic*, *data-driven* notion of objectness. Our approach is to present a large database of images with annotated objects to a learning algorithm, and let the algorithm figure out what low-, mid- and high-level cues are most discriminative of objects. Following recent work on a range of recognition tasks [11, 12, 18], we use convolutional networks (CNNs) [19] for this task. Concretely, we train a CNN to rerank a large pool of object proposals produced by a bottom-up proposal method (we use Edge boxes [30] for most experiments in this paper). For ease of reference, we call our approach DeepBox. Figure 1 shows our framework.

We propose a lightweight four layer network architecture that significantly improves over bottom-up proposal methods in terms of ranking (26% relative improvement on AUC over Edge boxes on VOC 2007 [8]). Our network architecture is as effective as state-of-the-art classification networks on this task while being much smaller and thus much faster. In addition, using ideas from SPP [13] and Fast R-CNN [10], our implementation runs in 260 ms per image, comparable to some of the fastest bottom-up proposal approaches like Edge Boxes (250 ms). We also provide evidence that what our network learns is category-agnostic: our improvements in performance generalize to categories that the CNN has not seen before (16% improvement over Edge boxes on COCO [20]). Our results suggest that a) there is indeed a generic, semantic notion of objectness beyond bottom-up saliency, and that b) this semantic notion of objectness can be learnt effectively by a lightweight CNN.

Object proposals are just the first step in an object detection system, and the final evaluation of a proposal system is the impact it has on detection performance. We show that the Fast R-CNN detection system [10], using 500 DeepBox proposals per image, is 4.5 points better than the same object detector using 500 Edge box proposals. Thus our high quality proposals directly lead to better object detection.

The rest of the paper is laid out as follows. In Section 2 we discuss related work. We describe our network architecture and training and testing procedures in Section 3. Section 4 describes experiments and we end with a discussion.

2. Related work

Russell et al. [26] were one of the first to suggest a category-independent method to propose putative objects.

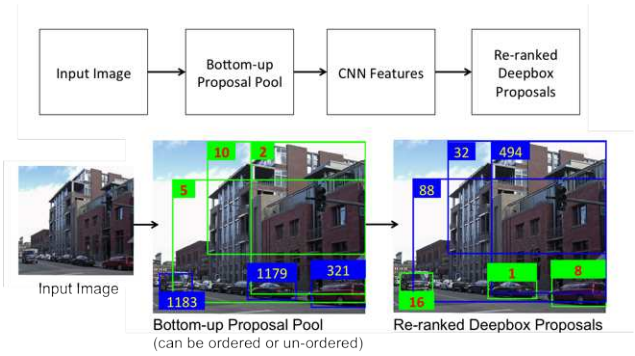


Figure 1. The DeepBox framework. Given any RGB image, we first generate bottom-up proposals and then rerank them using a CNN. High ranked boxes are shown in green and low ranked ones are blue. The number in each box is its ranking in the proposal pool. DeepBox corrects the ranking of Edge box, ranking objects higher than background.

Their method involved sampling regions from multiple segmentations of the image. More recently, Alexe et al. [1] and Endres et al. [6] propose using bottom-up object proposals as a first step in recognition. Expanding on the multiple segmentations idea, Selective search [29] uses regions from hierarchical segmentations in multiple color spaces as object proposals. CPMC [4] uses multiple graph-cut based segmentations with multiple foreground seeds and multiple foreground biases to propose objects. GOP [17] replaces graph cuts with a much faster geodesic based segmentation. MCG [2] also uses multiple hierarchical segmentations from different scales of the image, but produces proposals by combinatorially grouping regions. Edge boxes [30] uses contour information instead of segments: bounding boxes which have fewer contours straggling the boundary of the box are considered more likely to be objects.

Many object proposal methods also include a ranking of the regions. This ranking is typically based on low level region features such as saliency [1], and is sometimes learnt [2, 4]. Relatively simple ranking suffices when the goal is a few thousand proposals as in MCG [2], but to narrow the list down to a few hundred as in CPMC [4] requires more involved reasoning. DeepBox aims at such a ranking.

Multibox [7, 28] directly produces object proposals from images using a sophisticated neural network. In contemporary work, Faster R-CNN [25] uses the same large network to propose objects and classify them. DeepMask [22] also uses a very deep network to directly produce segment proposals. In comparison, our architecture is quite lightweight and can be used out of the box to rerank any bottom-up proposals.

Finally, we direct the reader to [15, 14] for a more thorough evaluation of bottom-up proposal methods.

3. Method

The pipeline consists of two steps: 1) Generate an initial pool of N bottom-up proposals. Our method is agnostic to the precise bottom-up proposal method. The main point of this step is to prune out the obviously unlikely windows so that DeepBox can focus on the hard negatives. 2) Rerank the proposals using scores obtained by the DeepBox network. We rerank each proposal by cropping out the proposal box and feeding it into a CNN, as described by Girshick et al. [11]. Because highly overlapping proposals are handled independently, this strategy is computationally wasteful and thus slow. A more sophisticated and much faster approach, using ideas from [13, 10] is described in Section 3.2.

For datasets without many small objects (e.g. PASCAL), we often only need to re-rank the top 2000 proposals to obtain good enough recall. As shown by [30], increasing the number of Edge box proposals beyond 2000 leads to only marginal increase in recall. For more challenging datasets with small objects (e.g. COCO [20]), reranking more proposals continues to provide gains in recall beyond 2000 proposals.

3.1. Network Architecture

When it comes to the network architecture, we would expect that predicting the precise category of an object is harder than predicting objectness, and so we would want a simpler network for objectness. This also makes sense from a computational standpoint, since we do not want the object proposal scoring stage to be as expensive as the detector itself.

We organized our search for a suitable network architecture by starting from the architecture of [18] and gradually ablating it while trying to preserve performance. The ablation here can be performed by reducing the number of channels in the different layers (thus reducing the number of parameters in the layer), by removing some layers, or by decreasing the input resolution so that the features computed become coarser.

The original architecture gave an AUC on PASCAL VOC of $0.76(0.62)$ for IoU=0.5 (0.7). First, we changed the number of outputs of $fc6$ to 1024 with other things fixed and found that performance remained unchanged. Then we adjusted the input image crop from 227×227 of the network to 120×120 and observed that the AUC dropped 1.5 points for IoU=0.5 and 2.9 points for IoU=0.7 on PASCAL. With this input size, we tried removing $fc6$ (drop: 2.3 points), $conv5$ (drop: 2.9 points), $conv5+conv4$ (drop: 10.6 points) and $conv5 + conv4 + conv3$ layers (drop: 6.7 points). This last experiment meant that dropping all of $conv5, conv4$ and $conv3$ was *better* than just dropping $conv5$ and $conv4$. This might be because $conv3, conv4$ and $conv5$ while adding to the capacity of the network are

also likely overfit to the task of image classification (as described below, the convolutional layers are initialized from a model trained on ImageNet). We stuck to this architecture (i.e.,without $conv5$, $conv4$ and $conv3$) and explored different input sizes for the net. For an input size of 140×140, we obtained a competitive AUC of 0.74 (for IoU=0.5) and 0.60 (for IoU=0.7) on PASCAL, or equivalently a 4.4 point drop against the baseline.

Our final architecture can be written down as follows. Denote by $conv(k, c, s)$ a convolutional layer with kernel size k, stride s and number of output channels c. Similarly, $pool(k, s)$ denotes a pooling layer with kernel size k and stride s, and $fc(c)$ a fully connected layer with c outputs. Then, our network architecture is:
$conv(11, 96, 4) - pool(3, 2) - conv(5, 256, 1) - fc(1024) - fc(2)$.
Each layer except the last is followed by a ReLU nonlinearity. Our problem is a binary classification problem (object or not), so we only have two outputs which are passed through a softmax. Our input size is 140×140.

While we finalized this architecture on PASCAL for Edge boxes, we show in Section 4 that the same architecture works just as well for other datasets such as COCO [20] and for other proposal methods such as Selective Search [29] or MCG [2].

3.2. Sharing computation for faster reranking

Running the CNN separately on highly overlapping boxes wastes a lot of computation. He et al. [13] pointed out that the convolutional part of the network could be shared among all the proposals. Concretely, instead of cropping out individual boxes, we pass in the entire image into the network (at a high resolution). After passing through all the convolutional and pooling layers, the result is a feature map which is some fraction of the image size. Given this feature map and a set of bounding boxes, we want to compute a fixed length vector for each box that we can then feed into the fully connected layers. To do this, note that each bounding box B in the image space corresponds to a box b in the feature space of the final convolutional layer, with the scale and aspect ratio of b being dependent on B and thus different for each box. He et al. propose to use a *fixed* spatial pyramid grid to max-pool features for each box. While the size of the grid cell varies with the box, the number of bins don't, thus resulting in a fixed feature vector for each box. From here on, each box is handled separately. However, the shared convolutional feature maps means that we have saved a lot of computation.

One issue with this approach is that all the convolutional feature maps are computed at just one image scale, which may not be appropriate for all objects. He et al. [13] suggest a multiscale version where the feature maps are computed at a few fixed scales, and for each box we pick the best scale, which they define as the one where the area of the scaled box is closest to a predefined value. We experiment with both the single scale version and a multiscale version using three scales.

We use the implementation proposed by Girshick in Fast R-CNN [10]. Fast R-CNN implements this pooling as a layer in the CNN (the RoI Pooling layer) allowing us to train the network end-to-end.

To differentiate this version of DeepBox from the slower alternative based on cropping and warping, we call this version Fast DeepBox in the rest of the paper.

3.3. Training Procedure

3.3.1 Initialization

The first two convolutional layers were initialized using the publicly available Imagenet model [18]. This model was pretrained on 1000 Imagenet categories for the classification task. The fc layers are initialized randomly from Gaussian distribution with $\sigma = 0.01$. Our DeepBox training procedure consists of two stages. Similar to the classical notion of bootstrapping in object detection, we first train an initial model to distinguish between object boxes and randomly sampled sliding windows from the background. This teaches it the difference between objects and background. To enable it to do better at correcting the errors made by bottom-up proposal methods, we run a second training round where we train the model on bottom-up proposals from a method such as Edge boxes.

3.3.2 Training on Sliding Windows

First we generate negative windows by simple raster scanning. The sliding window step size is selected based on the box-searching strategy of Edge boxes[30]. Following Zitnick et al [30], we use α to denote the IoU threshold for neighboring sliding windows, and set it to 0.65. We generated windows in 5 aspect ratios: $(w : h) = (1 : 1), (2 : 3), (1 : 3), (3 : 2)$, and $(3 : 1)$. Negative windows which overlap with a ground truth object by more than $\beta_{-} = 0.5$ are discarded.

To obtain positives, we randomly perturb the corners of ground truth bounding boxes. Suppose a ground truth bounding box has coordinates $(x_{min}, y_{min}, x_{max}, y_{max})$, with the width denoted by w and the height denoted by h. Then the perturbed coordinates are distributed as:

$$x'_{min} \sim \text{unif}(x_{min} - \gamma w, x_{min} + \gamma w) \quad (1)$$
$$y'_{min} \sim \text{unif}(y_{min} - \gamma h, y_{min} + \gamma h) \quad (2)$$
$$x'_{max} \sim \text{unif}(x_{max} - \gamma w, x_{max} + \gamma w) \quad (3)$$
$$y'_{max} \sim \text{unif}(y_{max} - \gamma h, y_{max} + \gamma h) \quad (4)$$

where $\gamma = 0.2$ defines the level of noise. Larger γ introduces more robustness into positive training samples, but

might hurt localization. In practice we found that $\gamma = 0.2$ works well. In case some perturbed points go out of the image, we set them to stay on the image border. Perturbed windows that overlap with ground truth boxes by less than $\beta_+ = 0.5$ are discarded.

3.3.3 Training on Hard-negatives

Next, we trained the net on bottom-up proposals from a method such as Edge boxes [30]. Compared to sliding windows, these proposals contain more edges, texture and parts of objects, and thus form hard negatives. While sliding windows trained the net to distinguish primarily between background and objects, training on these proposals allows the net to learn the notion of complete objects and better adapt itself to the errors made by the bottom-up proposer. This is critical: without this stage, performance drops by 18% on PASCAL from 0.60 AUC at IoU=0.7 to 0.48.

The window preparation and training procedure is as described in Section 3.3.2, except that sliding windows are replaced with Edge boxes proposals. We set the overlap threshold $\beta_+ = 0.7$ slightly higher, so that the net can learn to distinguish between tight and loose bounding boxes. We set $\beta_- = 0.3$ below which the windows are labeled negative. Windows with IoU $\in [0.3, 0.7]$ are discarded lest they confuse the net.

We balanced the ratio of positive and negative windows at $1:3$ at training time for both stages. Momentum is set to 0.9 and weight decay to 0.0005. The initial learning rate is 0.001, which we decrease by 0.1 every 20,000 iterations. We train DeepBox for 60,000 iterations with a mini-batch size of 128.

The Fast DeepBox network was trained for $120k$ iterations in both sliding window and hard negative training. Three scales $[400, 600, 900]$ were used for both training and testing.

4. Experiments

All experiments except those in Section 4.7 and 4.8 were done using DeepBox. We evaluate Fast DeepBox in Section 4.7, and as a final experiment plug it into an object detection system in Section 4.8.

4.1. Learning objectness on PASCAL and COCO

In our first set of experiments, we evaluated our approach on PASCAL VOC 2007 [8] and on the newly released COCO [20] dataset. For these experiments, we used Edge boxes [30] for the initial pool of proposals. On PASCAL we reranked the top 2048 Edge box proposals, whereas on COCO we re-ranked all. We used the network architecture described in Section 3.1. For results on PASCAL VOC 2007, we trained our network on the trainval set and tested on the test set. For results on COCO, we trained our network on the train set and evaluated on the val set.

4.2. Comparison to Edge boxes

We first compare our ranking to the ranking output by Edge boxes. Figure 2 plots Recall vs Number of Proposals in PASCAL VOC 2007 for IoU=0.7[1]. We observe that DeepBox outperforms Edge boxes in all regimes, especially with a low number of proposals. The same is true for IoU=0.5 (not shown). The AUCs (Areas Under Curve) for DeepBox are $0.74(0.60)$ vs Edge box's $0.60(0.47)$ for IoU=0.5(0.7), suggesting that DeepBox proposals are 24% better at IoU=0.5 and 26% better at IoU=0.7 compared to Edge boxes. Figure 3 plots the same in COCO. The AUCs (Areas Under Curve) for DeepBox are $0.40(0.28)$ vs Edge boxes's $0.28(0.20)$ for IoU=0.5(0.7), suggesting DeepBox proposals are $40\%(43\%)$ better than Edge boxes. If we re-ranked top 2048 proposals instead, the AUCs are $0.38(0.27)$.

On PASCAL, we also plot the Recall vs IoU threshold curves at 1000 proposals in Figure 4. At 1000 proposals, the gain of DeepBox is not as big as at 100-500 proposals, but we still see that it is superior in all regimes of IoU.

Part of this performance gain is due to small objects, defined as objects with area less than 2000. On COCO the AUCs for DeepBox (trained on all categories) are $(0.161, 0.080)$, while the numbers for Edge boxes are $(0.061, 0.030)$ for IoU(0.5,0.7). DeepBox outperforms Edge boxes by more than 160% on small objects.

Comparison to other proposal methods We can also compare our ranked set of proposals to all the other proposals in the literature. We show this comparison for IoU=0.7 in Table 1. The numbers are obtained from [30] except for MCG and DeepBox which we computed ourselves. In Table 1, the metrics are the number of proposals needed to achieve 25%, 50% and 75% recall and the maximum recall using 5000 boxes.

4.3. Visualization of DeepBox proposals

Figures 5 and 6 visualizes DeepBox and Edge box performance on PASCAL and COCO images. Figure 5 shows the ground truth boxes that are detected ("hits", shown in green, with the corresponding best overlapping proposal shown in blue) and those that are missed (shown in red) for both proposal rankings. We observe that in complicated scenes with multiple small objects or cluttered background, DeepBox significantly outperforms Edge box. The tiny boats, the cars parked by the road, the donuts and people in the shade are all correctly captured.

[1] We computed recall using the code provided by [30]

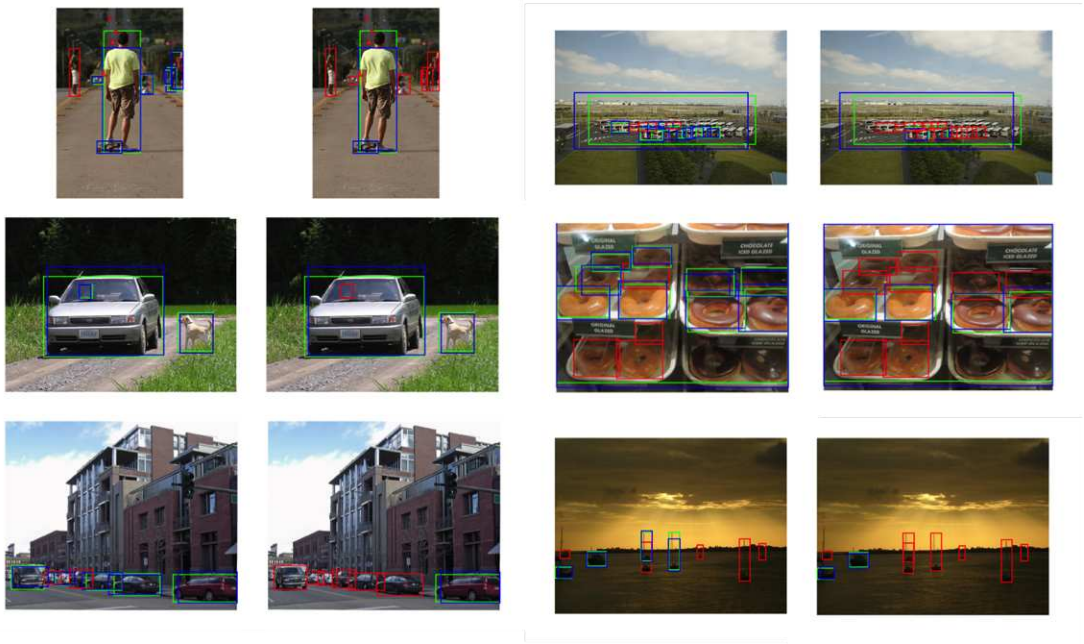


Figure 5. Visualization of hits and misses. In each image, the green boxes are ground truth boxes for which a highly overlapping proposal exists (with the corresponding proposals shown as blue boxes) and red boxes are ground truth boxes that are missed. The IoU threshold is 0.7. We evaluated 1000 proposals per image for COCO and 500 proposals per image for PASCAL. The left image in every pair shows the result of DeepBox ranking, and the right image shows the ranking from Edge boxes. In cluttered scenes, DeepBox has a higher recall. See Section 4.3 for a detailed discussion.

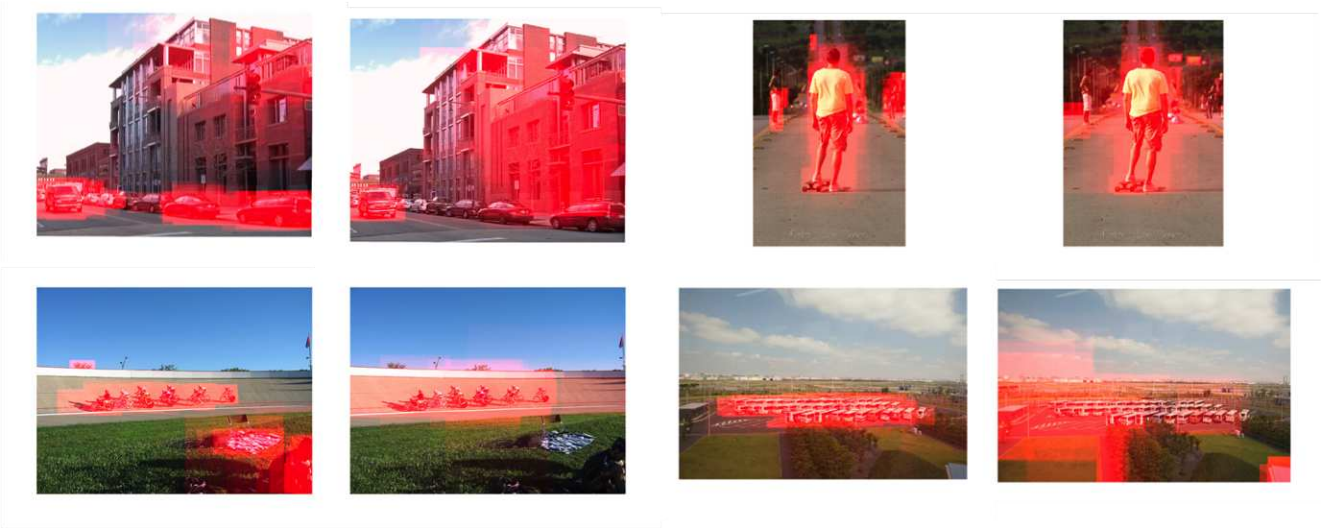


Figure 6. In each image we show the distribution of the proposals produced by pasting a red box for each proposal. Only the top 100 proposals are shown. For each pair of images, DeepBox is on the left and Edge boxes is the right. DeepBox proposals are more tightly concentrated on the objects. See Section 4.3 for a detailed discussion.

This is also validated by looking at the distribution of top 100 proposals (Figure 6), which is shown in red. In general, DeepBox's bounding boxes are very densely focused on the objects of interest while Edge boxes primarily recognize contours and often spread evenly across the image in a complicated scene.

4.4. Generalization to unseen categories

The high recall our method achieves on the PASCAL 2007 test set does not guarantee that our net is truly learning

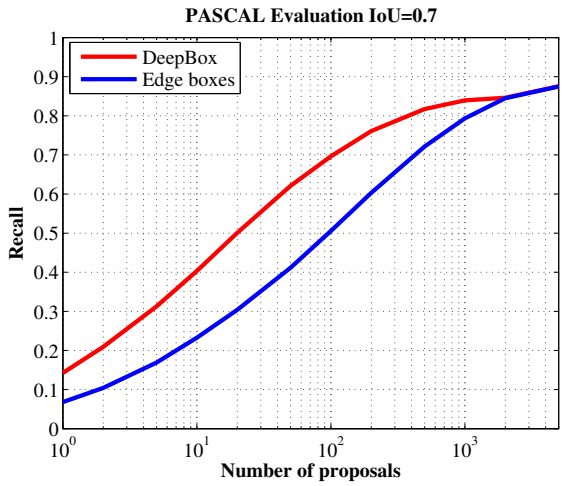


Figure 2. PASCAL Evaluation IoU=0.7. DeepBox starts off much higher than Edge boxes. The wide margin continues all the way until 500 proposals and gradually decays. The two curves join at 2048 proposals because we chose to re-rank this number of proposals.

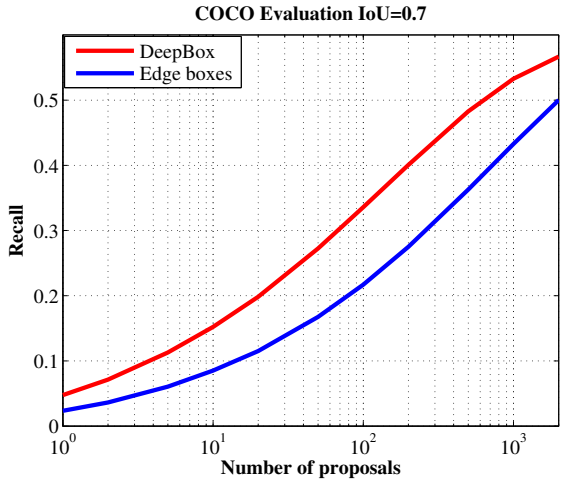


Figure 3. MS COCO Evaluation IoU=0.7. The strong gain demonstrated by DeepBox on COCO suggests that our learnt objectness is particularly helpful in a complicated dataset.

general objectness. It is possible that the net is learning just the union of 20 object categories and using that knowledge to rank proposals. To evaluate whether the net is indeed learning a more general notion of objectness that extends beyond the categories it has seen during training, we did the following experiment:

- We identified the 36 overlapping categories between Imagenet and COCO.
- We trained the net just on these overlapping categories on COCO with initialization from Imagenet. This means that during training, only boxes that overlapped highly with ground truth from the 36 overlapping categories were labeled positives, and others were labeled negatives. Also, when sampling positives, only the ground truth boxes corresponding to these overlapping categories were used to produce perturbed positives. This is equivalent to training on a dataset where all the other categories have not been labeled at all.
- We then evaluated our performance on the rest of the categories in COCO (44 in number). This means that at test time, only proposed boxes that overlapped with ground truth from the other 44 categories were considered true positives. Again, this corresponds to evaluat-

	AUC	25%	50%	75%	Recall (%)	Time (s)
BING[5]	0.20	292	-	-	29	**0.2**
Ranta[24]	0.23	184	584	-	68	10
Objectness[1]	0.27	27	-	-	39	3
Rand. P.[21]	0.35	42	349	3023	80	1
Rahtu [23]	0.37	29	307	-	70	3
SelSearch [29]	0.40	28	199	1434	**87**	10
CPMC [3]	0.41	15	111	-	65	250
MCG [2]	0.48	10	81	871	83	34
E.B [30]	0.47	12	108	800	**87**	0.25
DeepBox	**0.60**	**3**	**20**	**183**	**87**	2.5

Table 1. Comparison of DeepBox with existing object proposal techniques. All numbers are for an IoU threshold of 0.7. The metrics are the number of proposals needed to achieve 25%, 50% and 75% recall and the maximum recall using 5000 boxes. DeepBox outperforms in all metrics. Note that the timing numbers for DeepBox include computation on the GPU.

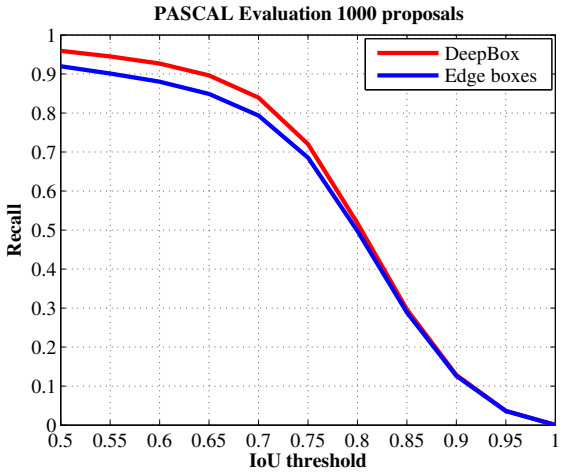


Figure 4. Variation of recall with IoU threshold at 1000 proposals. DeepBox (average recall: 57.0%) is better than Edge boxes (average recall: 54.4%) in all regimes. Comparisons to other proposal methods is shown in the supplementary.

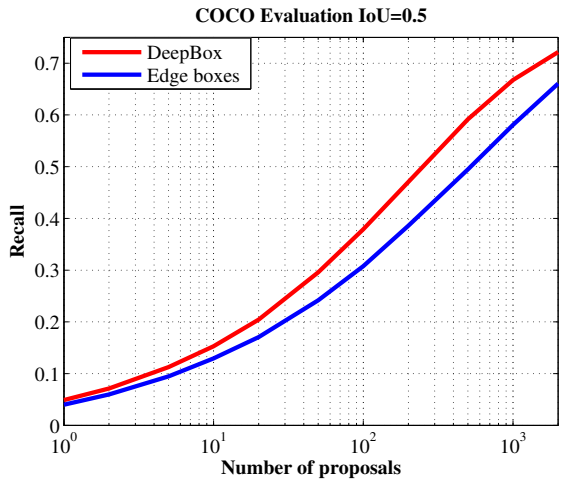


Figure 7. Evaluation on unseen categories, when ranking all proposals, at IoU=0.5.

ing on a dataset where the 36 training categories have not been labeled at all.

All other experimental settings remain the same. As before, we use Edge boxes for the initial pool of proposals which we rerank, and compare the ranking we obtain to the ranking output by edge boxes.

When reranking the top 2048 proposals, DeepBox achieved $15.7\%(15.3\%)$ AUC improvement over Edge boxes for IoU=0.5 (0.7). When reranking *all* proposals output by Edge boxes, DeepBox achieved $18.5\%(16.2\%)$ improvement over Edge boxes for IoU=0.5 (0.7) (Figure 7). In this setting, DeepBox outperforms Edge box in all regimes on unseen categories for both IoUs.

In Figure 8, we plot the ratio of the AUC obtained by DeepBox to that obtained by Edge box for all the 44 testing categories. In more than half of the testing categories, we obtain an improvement greater than 20%, suggesting that the gains provided by DeepBox are spread over multiple categories. This suggests that DeepBox is actually learning a class-agnostic notion of objectness. Note that DeepBox performs especially well for the animal super category because all animal categories have similar geometric structure and training the net on some animals helps it recognize other animals. This validates our intuition that there are high-level semantic cues that can help with objectness. In the sports super category, DeepBox performs worse than Edge boxes, perhaps because most objects of this category have salient contours that favor the Edge boxes algorithm.

These results on COCO demonstrate that our network has learnt a notion of objectness that generalizes beyond training categories.

	Vanilla	DeepBox (Trained on [30])	DeepBox (Finetuned)	Total Time (s)
Sel. Se.	0.27/0.17	0.27/0.19	0.32/0.22	12.5
MCG	**0.38/0.25**	0.30/0.22	0.37/0.26	36.5
Edge Box	0.28/0.20	**0.38/0.27**	**0.38/0.27**	**2.5**

Table 2. DeepBox on top of other proposers. For each method we show the AUC at IoU=0.5/0.7 of (left to right) the original ranking, the reranking produced by DeepBox trained on Edge boxes, and that produced after finetuning on the corresponding proposals.

4.5. DeepBox using Large Networks

We also experimented with larger networks such as "Alexnet" [18] and "VGG" [27]. Unsurprisingly, large networks capture objectness better than our small network. However, the difference is quite small: With VGG, the AUCs on PASCAL are $0.78(0.65)$ for IoU=0.5(0.7). The numbers for Alexnet are $0.76(0.62)$. In comparison our network achieves $0.74(0.60)$.

For evaluation on COCO, we randomly selected 5000 images and computed AUCs using the VGG and Alexnet architecture. All networks re-rank just top 2048 Edge box proposals. At IoU=0.5, VGG gets 0.43 and Alexnet gets 0.42, compared to the 0.38 obtained by our network. At IoU=0.7, VGG gets 0.31 and Alexnet gets 0.30, compared to the 0.27 obtained by our network. When re-ranking all proposals, our small net gets $0.40(0.28)$ for IoU=0.5(0.7).

These experiments suggest that our network architecture is sufficient to capture the semantics of objectness, while also being much more efficient to evaluate compared to the more massive VGG and Alexnet architectures.

4.6. DeepBox with Other Proposers

It is a natural question to ask whether DeepBox framework applies to other bottom-up proposers as well. We experimented with MCG and Selective Search by just reranking top 2048 proposals. (For computational reasons, we did this experiment on a smaller set of 5000 COCO images.) We experimented with two kinds of DeepBox models: a single model trained on Edge boxes, and separate models trained on each proposal method (Table 2).

A model trained on Edge boxes does not provide much gains, and indeed sometimes hurts, when run on top of Selective Search or MCG proposals. However, if we retrain DeepBox separately on each proposal method, this effect goes away. In particular, we get large gains on Selective Search for both IoU thresholds (as with Edge boxes). On MCG, DeepBox does not hurt, but does not help much either. Nevertheless, the gains we get on Edge boxes and Selective Search suggest that our approach is general and can work with any proposal method, or even ensembles of proposal methods (a possibility we leave for future work).

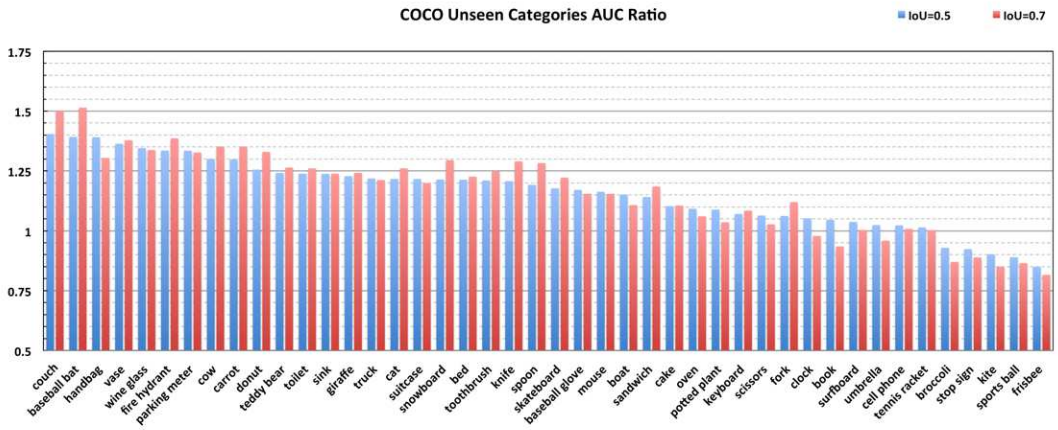


Figure 8. Evaluation on unseen categories: category-wise breakdown. This demonstrates that DeepBox network has learnt a notion of objectness that generalizes beyond training categories.

4.7. Fast DeepBox

We experimented with Fast DeepBox on COCO. The AUCs for multiscale Fast DeepBox are $0.40(0.29)$ vs Edge box's $0.28(0.20)$ for IoU=0.5(0.7), a gain of $41\%(44\%)$ over Edge boxes. When we re-ranked top 2048 proposals instead, the AUCs are $0.37(0.27)$. Compared with DeepBox's multi-thread runtime of $2.5s$, Fast DeepBox (multiscale) is an order of magnitude faster: it takes 0.26s to re-rank all proposals or 0.12s to re-rank the top-2000. This compares favorably to other bottom-up proposals. In terms of average recall with 1000 proposals, our performance (0.39) is better than GOP (0.36) [17], Selective Search (0.36) [29] and Edge boxes (0.34) [30], and is about the same as MCG (0.40) [2] while being almost 70 times faster. In contrast, Deepmask achieves 0.45 with a much deeper network at the expense of being 3 times slower (1.6 s) [22].

With a small decrease in performance, Fast DeepBox can be made much faster. With a single scale, AUC drops by about 0.005 when re-ranking top-2000 proposals and 0.01 when re-ranking all proposals. However, it only takes 0.11s to re-rank all proposals or 0.060s for the top-2000. One can also make training faster by removing the scanning-window stage and using a single scale. This speedup comes with a drop in performance of 0.015 when reranking all proposals compared to the multiscale two-stage version.

4.8. Impact on Object Detection

The final metric for any proposal method is its impact on object detection. Good proposals not only reduce the computational complexity but can also make object detection easier by reducing the number of candidates that the detector has to choose from [14, 10]. We found that this is indeed true: when using 500 DeepBox proposals, Fast-RCNN (with the VGG-16 network) gives a mAP of **37.8**% on COCO Test at IoU=0.5, compared to only **33.3**% when using 500 Edge Box proposals. Even when using 2000 Edge Box proposals, the mAP is still lower (35.9%). For comparison, Fast R-CNN using 2000 Selective Search proposals gets a mean AP of 35.8%, indicating that with just 500 DeepBox proposals we get a 2 point jump in performance.

5. Discussion and Conclusion

We have presented an efficient CNN architecture that learns a semantic notion of objectness that generalizes to unseen categories. We conclude by discussing other applications of our objectness model.

First, as the number of object categories increases, the computational complexity of the detector increases and it becomes more and more useful to have a generic objectness system to reduce the number of locations the detector looks at. Objectness can also help take the burden of localization off the detector, which then has an easier task.

Second, AI agents navigating the world cannot expect to be trained on labeled data like COCO for every object category they see. For some categories the agent will have to collect data and build detectors on the fly. In this case, objectness allows the agent to pick a few candidate locations in a scene that look like objects and track them over time, thus collecting data for training a detector. Objectness can thus be useful for *object discovery* [16], especially when it captures semantic properties as in our approach.

6. Acknowledgement

This work is supported by a Berkeley Graduate Fellowship and a Microsoft Research Fellowship. We thank NVIDIA for giving GPUs through their academic program too.

References

[1] B. Alexe, T. Deselaers, and V. Ferrari. Measuring the objectness of image windows. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(11):2189–2202, 2012. 1, 2, 6

[2] P. Arbeláez, J. Pont-Tuset, J. Barron, F. Marques, and J. Malik. Multiscale combinatorial grouping. In *CVPR*, 2014. 2, 3, 6, 8

[3] J. Carreira, R. Caseiro, J. Batista, and C. Sminchisescu. Semantic segmentation with second-order pooling. In *ECCV*, 2012. 6

[4] J. Carreira and C. Sminchisescu. Cpmc: Automatic object segmentation using constrained parametric min-cuts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(7):1312–1328, 2012. 2

[5] M. Cheng, Z. Zhang, W. Lin, , and P. Torr. Bing: Binarized normed gradients for objectness estimation at 300fps. In *CVPR*, 2014. 6

[6] I. Endres and D. Hoiem. Category independent object proposals. In *Computer Vision–ECCV 2010*, pages 575–588. Springer, 2010. 2

[7] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. Scalable object detection using deep neural networks. In *CVPR*, 2014. 2

[8] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010. 1, 4

[9] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *TPAMI*, 32(9), 2010. 1

[10] R. Girshick. Fast R-CNN. In *ICCV*, 2015. 1, 2, 3, 8

[11] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 1, 2

[12] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Simultaneous detection and segmentation. In *ECCV*, 2014. 1

[13] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*, 2014. 1, 2, 3

[14] J. Hosang, R. Benenson, P. Dollár, and B. Schiele. What makes for effective detection proposals? *arXiv preprint arXiv:1502.05082*, 2015. 1, 2, 8

[15] J. Hosang, R. Benenson, and B. Schiele. How good are detection proposals, really? In *BMVC*, 2014. 2

[16] H. Kang, M. Hebert, A. A. Efros, and T. Kanade. Connecting missing links: Object discovery from sparse observations using 5 million product images. In *ECCV*. 2012. 8

[17] P. Krähenbühl and V. Koltun. Geodesic object proposals. In *ECCV*, 2014. 2, 8

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1, 2, 3, 7

[19] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989. 1

[20] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollr, and C. L. Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014. 1, 2, 3, 4

[21] S. Manen, M. Guillaumin, and L. Van Gool. Prime object proposals with randomized prim's algorithm. In *ICCV*, 2013. 6

[22] P. O. Pinheiro, R. Collobert, and P. Dollár. Learning to segment object candidates. In *NIPS*, 2015 (To appear). 2, 8

[23] E. Rahtu, K. Juho, and B. Matthew. Learning a category independent object detection cascade. In *ICCV*, 2011. 6

[24] R. E. Rantalankila P., Kannala J. Generating object segmentation proposals using global and local search. In *CVPR*, 2014. 6

[25] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In *NIPS*, 2015 (To appear). 2

[26] B. C. Russell, W. T. Freeman, A. A. Efros, J. Sivic, and A. Zisserman. Using multiple segmentations to discover objects and their extent in image collections. In *CVPR*, 2006. 1

[27] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 7

[28] C. Szegedy, S. Reed, D. Erhan, and D. Anguelov. Scalable, high-quality object detection. *arXiv preprint arXiv:1412.1441*, 2014. 2

[29] J. R. Uijlings, K. E. van de Sande, T. Gevers, and A. W. Smeulders. Selective search for object recognition. *IJCV*, 104(2), 2013. 2, 3, 6, 8

[30] C. L. Zitnick and P. Dollár. Edge boxes: Locating object proposals from edges. In *ECCV*, 2014. 1, 2, 3, 4, 6, 7, 8

Flowing ConvNets for Human Pose Estimation in Videos

Tomas Pfister
Dept. of Engineering Science
University of Oxford
tp@robots.ox.ac.uk

James Charles
School of Computing
University of Leeds
j.charles@leeds.ac.uk

Andrew Zisserman
Dept. of Engineering Science
University of Oxford
az@robots.ox.ac.uk

Abstract

The objective of this work is human pose estimation in videos, where multiple frames are available. We investigate a ConvNet architecture that is able to benefit from temporal context by combining information across the multiple frames using optical flow.

To this end we propose a network architecture with the following novelties: (i) a deeper network than previously investigated for regressing heatmaps; (ii) spatial fusion layers that learn an implicit spatial model; (iii) optical flow is used to align heatmap predictions from neighbouring frames; and (iv) a final parametric pooling layer which learns to combine the aligned heatmaps into a pooled confidence map.

We show that this architecture outperforms a number of others, including one that uses optical flow solely at the input layers, one that regresses joint coordinates directly, and one that predicts heatmaps without spatial fusion.

The new architecture outperforms the state of the art by a large margin on three video pose estimation datasets, including the very challenging Poses in the Wild dataset, and outperforms other deep methods that don't use a graphical model on the single-image FLIC benchmark (and also [5, 35] in the high precision region).

1. Introduction

Despite a long history of research, human pose estimation in videos remains a very challenging task in computer vision. Compared to still image pose estimation, the temporal component of videos provides an additional (and important) cue for recognition, as strong dependencies of pose positions exist between temporally close video frames.

In this work we propose a new approach for using optical flow for part localisation in deep Convolutional Networks (ConvNets), and demonstrate its performance for human pose estimation in videos. The key insight is that, since for localisation the prediction targets are positions in the image space (*e.g.* (x, y) coordinates of joints), one can use dense optical flow vectors to *warp predicted positions* onto a target image. In particular, we show that when regressing a *heatmap* of positions (in our application for human joints), the heatmaps from neighbouring frames can be warped and aligned using optical flow, effectively propagating position confidences temporally, as illustrated in Fig 1.

We also propose a deeper architecture that has additional convolutional layers beyond the initial heatmaps to enable learning an *implicit* spatial model of human layout. These layers are able to learn dependencies between human body parts. We show that these 'spatial fusion' layers remove pose estimation failures that are kinematically impossible.

Related work. Traditional methods for pose estimation have often used pictorial structure models [2, 8, 10, 27, 39], which optimise a configuration of parts as a function of local image evidence for a part, and a prior for the relative positions of parts in the human kinematic chain. An alternative approach uses poselets [1, 13]. More recent work has tackled pose estimation holistically: initially with Random Forests on depth data [12, 29, 31, 34] and RGB [3, 4, 24], and most recently with convolutional neural networks.

The power of ConvNets has been demonstrated in a wide variety of vision tasks – object classification and detection [11, 21, 28, 40], face recognition [32], text recognition [15, 16], video action recognition [20, 30] and many more [7, 22, 25].

For pose estimation, there were early examples of using ConvNets for pose comparisons [33]. More recently, [37] used an AlexNet-like ConvNet to directly regress *joint coordinates*, with a cascade of ConvNet regressors to improve accuracy over a single pose regressor network. Chen and Yuille [5] combine a parts-based model with ConvNets (by using a ConvNet to learn conditional probabilities for the presence of parts and their spatial relationship with image patches). In a series of papers, Tompson, Jain *et al*. developed ConvNet architectures to directly regress *heatmaps* for each joint, with subsequent layers to add an Markov Random Field (MRF)-based spatial model [17, 36], and a pose refinement model [35] (based on a Siamese network with shared weights) upon a rougher pose estimator ConvNet.

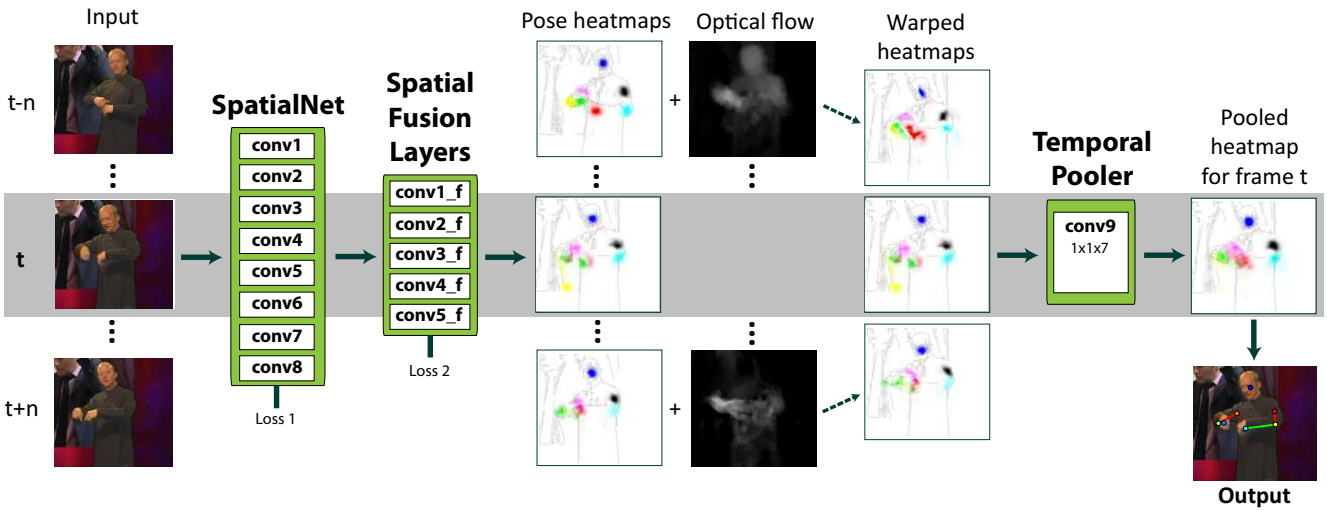


Figure 1. **Deep expert pooling architecture for pose estimation.** The network takes as an input all RGB frames within a n-frame neighbourhood of the current frame t. The fully convolutional network (consisting of a heatmap net with an implicit spatial model) predicts a confidence heatmap for each body joint in these frames (shown here with a different colour per joint). These heatmaps are then temporally *warped* to the current frame t using optical flow. The warped heatmaps (from multiple frames) are then *pooled* with another convolutional layer (the temporal pooler), which learns how to weigh the warped heatmaps from nearby frames. The final body joints are selected as the maximum of the pooled heatmap (illustrated here with a skeleton overlaid on top of the person).

Temporal information in videos was initially used with ConvNets for action recognition [30], where optical flow was used as an input *motion feature* to the network. Following this work, [18, 24] investigated the use of temporal information for pose estimation in a similar manner, by inputting flow or RGB from multiple nearby frames into the network, and predicting joint positions in the current frame.

However, pose estimation differs from action recognition in a key respect which warrants a different approach to using optical flow: in action recognition the prediction target is a class label, whereas in pose estimation the target is a set of (x, y) positions onto the image. Since the targets are positions in the image space, one can use dense optical flow vectors not only as an input feature but also to *warp predicted positions* in the image, as done in [4] for random forest estimators. To this end, our work explicitly predicts joint positions for *all* neighbouring frames, and temporally aligns them to frame t by warping them backwards or forwards in time using tracks from dense optical flow. This effectively reinforces the confidence in frame t with a strong set of 'expert opinions' (with corresponding confidences) from neighbouring frames, from which joint positions can be more precisely estimated. Unlike [4] who average the expert opinions, we learn the expert pooling weights with backpropagation in an end-to-end ConvNet.

Our ConvNet outperforms the state of the art on three challenging video pose estimation datasets (BBC Pose, ChaLearn and Poses in the Wild) – the heatmap regressor alone surpasses the state of the art on these datasets, and the pooling from neighbouring frames using optical flow gives a further significant boost. We have released the models and code at http://www.robots.ox.ac.uk/~vgg.

2. Temporal Pose Estimation Networks

Fig 1 shows an overview of the ConvNet architecture. Given a set of input frames within a temporal neighbourhood of n frames from a frame t, a spatial ConvNet regresses joint confidence maps ('heatmaps') for each input frame separately. These heatmaps are then individually *warped* to frame t using dense optical flow. The warped heatmaps (which are effectively 'expert opinions' about joint positions from the past and future) are then pooled into a single heatmap for each joint, from which the pose is estimated as the maximum.

We next discuss the architecture of the ConvNets in detail. This is followed by a description of how optical flow is used to warp and pool the output from the Spatial ConvNet.

2.1. Spatial ConvNet

The network is trained to regress the location of the human joint positions. However, instead of regressing the joint (x, y) positions directly [24, 37], we regress a *heatmap* of the joint positions, separately for each joint in an input image. This heatmap (the output of last convolutional layer, conv8) is a fixed-size $i \times j \times k$-dimensional cube (here $64 \times 64 \times 7$ for $k = 7$ upper-body joints). At training time, the ground truth label are heatmaps synthesised for each joint separately by placing a Gaussian with fixed variance at the ground truth joint position (see Fig 2). We then

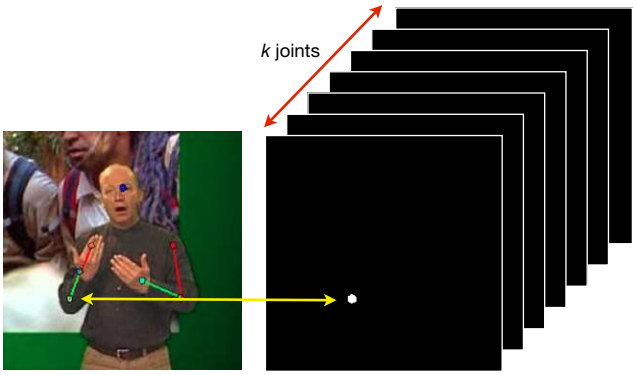


Figure 2. **Regression target for learning the Spatial ConvNet.** The learning target for the convolutional network is (for each of k joints) a heatmap with a synthesised Gaussian with a fixed variance centred at the ground truth joint position. The loss is l_2 between this target and the output of the last convolutional layer.

use an l_2 loss, which penalises the squared pixel-wise differences between the predicted heatmap and the synthesised ground truth heatmap.

We denote the training example as $(\mathbf{X}, \mathbf{y})$, where $\mathbf{y}$ stands for the coordinates of the k joints in the image $\mathbf{X}$. Given training data $N = \{\mathbf{X}, \mathbf{y}\}$ and the ConvNet regressor ϕ (output from conv8), the training objective becomes the task of estimating the network weights λ:

$$\arg\min_{\lambda} \sum_{(\mathbf{X},\mathbf{y})\in N} \sum_{i,j,k} \|G_{i,j,k}(\mathbf{y}_k) - \phi_{i,j,k}(\mathbf{X}, \lambda)\|^2 \quad (1)$$

where $G_{i,j,k}(\mathbf{y}_i) = \frac{1}{2\pi\sigma^2} e^{-[(y_k^1 - i)^2 + (y_k^2 - j)^2]/2\sigma^2}$ is a Gaussian centred at joint y_k with fixed σ.

Discussion. As noted by [36], regressing coordinates directly is a highly non-linear and more difficult to learn mapping, which we also confirm here (Sect 5). The benefits of regressing a heatmap rather than (x, y) coordinates are twofold: first, one can understand failures and visualise the 'thinking process' of the network (see Figs 3 and 5); second, since by design, the output of the network can be multimodal, *i.e.* allowed to have confidence at multiple spatial locations, learning becomes easier: early on in training (as shown in Fig 3), multiple locations may fire for a given joint; the incorrect ones are then slowly suppressed as training proceeds. In contrast, if the output were only the wrist (x, y) coordinate, the net would only have a lower loss if it gets its prediction right (even if it was 'growing confidence' in the correct position).

Architecture. The network architecture is shown in Fig 1, and sample activations for the layers are shown in Fig 5. To maximise the spatial resolution of the heatmap we make two important design choices: (i) minimal pooling is used (only two 2×2 max-pooling layers), and (ii) all strides are

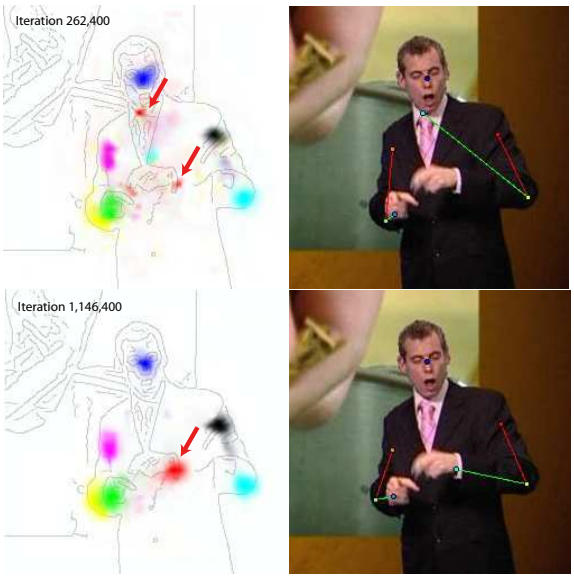


Figure 3. **Multiple peaks are possible with the Spatial ConvNet.** Early on in training (top), multiple locations may fire for a given joint. These are then suppressed as training proceeds (bottom). The arrows identify two modes for the wrist; one correct, one erroneous. As the training proceeds the erroneous one is diminished.

unity (so that the resolution is not reduced). All layers are followed by ReLUs except conv9 (the pooling layer). In contrast to AlexNet [21], our network is fully convolutional (no fully-connected layers) with the fully-connected layers of [21] replaced by 1×1 convolutions. In contrast to both AlexNet and [35], our network is deeper, does not use local contrast normalisation (as we did not find this beneficial), and utilises less max-pooling.

2.2. Spatial fusion layers

Vanilla heatmap pose nets do not learn spatial dependencies of joints, and thus often predict kinematically impossible poses (see examples in the extended arXiv version of this paper). To address this, we add what we term 'spatial fusion layers' to the network. These spatial fusion layers (normal convolutional layers) take as an input pre-heatmap activations (conv7), and learn dependencies between the human body parts locations represented by these activations. In detail, these layers take as an input a concatenation of conv7 and conv3 (a skip layer), and feed these through five more convolutional layers with ReLUs (see Fig 4). Large kernels are used to inflate the receptive field of the network. We attach a separate loss layer to the end of this network and backpropagate through the whole network.

2.3. Optical flow for pose estimation

Given the heatmaps from the Spatial ConvNet from multiple frames, the heatmaps are reinforced with optical flow. This is done in three steps: (1) the confidences from nearby

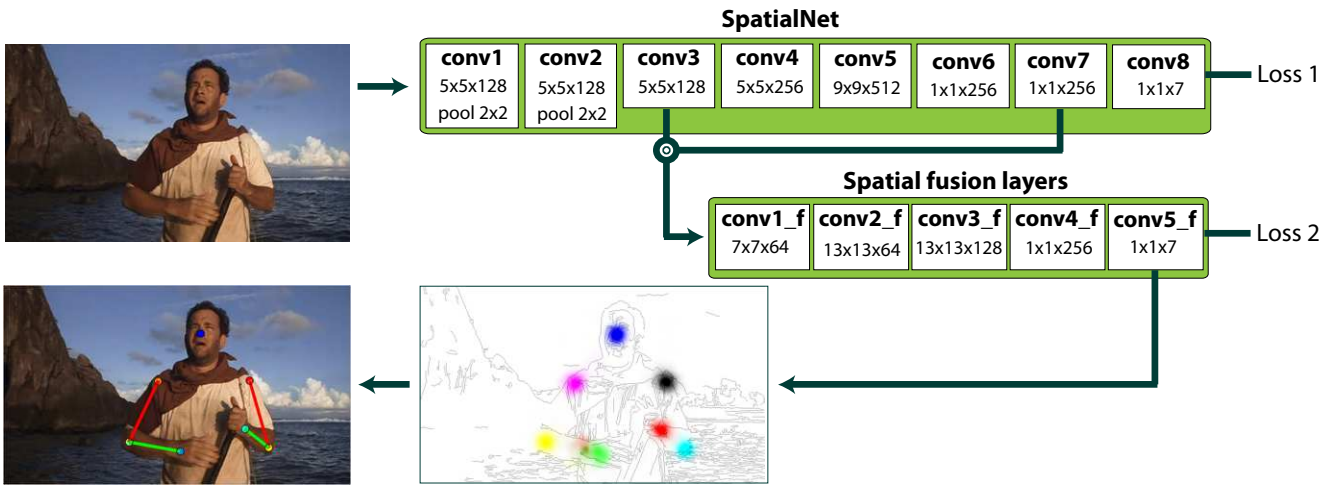


Figure 4. **Spatial fusion layers.** The fusion layers learn to encode dependencies between human body parts locations, learning an implicit spatial model.

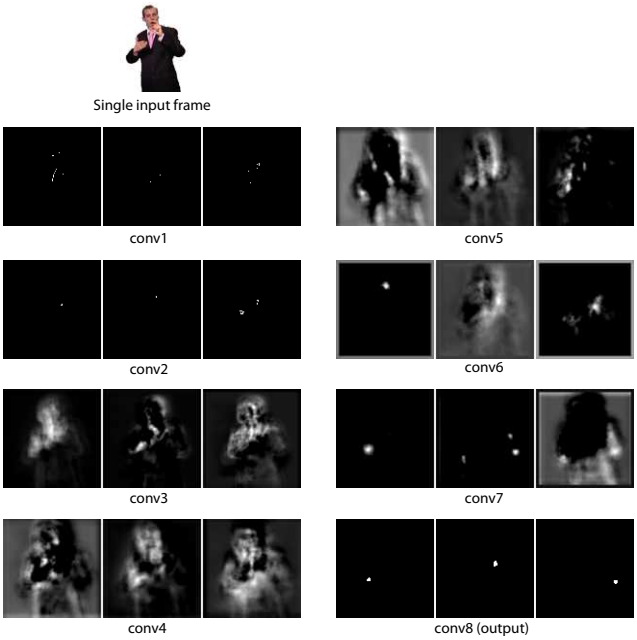


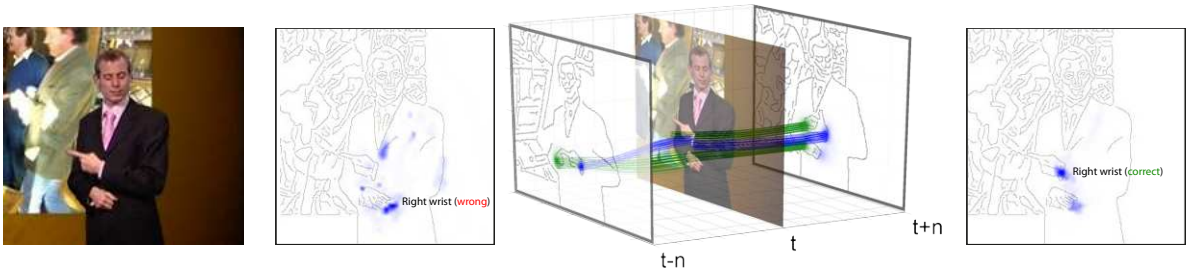
Figure 5. **Sample activations for convolutional layers.** Neuron activations are shown for three randomly selected channels for each convolutional layer (resized here to the same size), with the input (pre-segmented for visualisation purposes) shown above. Low down in the net, neurons are activated at edges in the image (*e.g.* conv1 and conv2); higher up, they start responding more clearly to body parts (conv6 onwards). The outputs in conv8 are shown for the right elbow, left shoulder and left elbow.

frames are aligned to the current frame using dense optical flow; (2) these confidences are then *pooled* into a composite confidence map using an additional convolutional layer; and (3) the final upper body pose estimate for a frame is then simply the positions of maximum confidence from the composite map. Below we discuss the first two steps.

Step 1: Warping confidence maps with optical flow. For a given frame t, pixel-wise temporal tracks are computed from all neighbouring frames within n frames from $((t-n)$ to $(t+n))$ to frame t using dense optical flow [38]. These optical flow tracks are used to warp confidence values in neighbouring confidence maps to align them to frame t by effectively shifting confidences along the tracks [4]. Example tracks and the warping of wrist confidence values are shown in Fig 6.

Step 2: Pooling the confidence maps. The output of Step 1 is a set of confidence maps that are warped to frame t. From these 'expert opinions' about the joint positions, the task is first to select a confidence for each pixel for each joint, and then to select one position for each joint. One solution would be to simply average the warped confidence maps. However, not all experts should be treated equally: intuitively, frames further away (thus with more space for optical flow errors) should be given lower weight.

To this end we learn a parametric cross-channel pooling layer that takes as an input a set of warped heatmaps for a given joint, and as an output predicts a single 'composite heatmap'. The input to this layer is a $i \times j \times t$ heatmap volume, where t is the number of warped heatmaps (*e.g.* 31 for a neighbourhood of $n = 15$). As the pooling layer, we train a 1×1 kernel size convolutional layer for each joint. This is equivalent to cross-channel weighted sum-pooling, where we learn a single weight for each input channel (which correspond to the warped heatmaps). In total, we therefore learn $t \times k$ weights (for k joints).



(a) Frame t (b) Confidence at t (c) Warping (d) Pooled at t

Figure 6. **Warping neighbouring heatmaps for improving pose estimates.** (a) RGB input at frame t. (b) Heatmap at frame t for right hand in the image. (c) Heatmaps from frames $(t-n)$ and $(t+n)$ warped to frame t using tracks from optical flow (green & blue lines). (d) Pooled confidence map with corrected modes.

	BBC	Ext. BBC	ChaLearn	PiW
Train frames	1.5M	7M	1M	4.5K (FLIC)
Test frames	1,000	1,000	3,200	830
Train labels	[2]	[2, 3]	Kinect	Manual
Test labels	Manual	Manual	Kinect	Manual
Train videos	13	85	393	-
Val videos	2	2	287	-
Test videos	5	5	275	30

Table 1. **Dataset overview, including train/val/test splits.**

3. Implementation Details

Training. The input frames are rescaled to height 256. A 248×248 sub-image (of the $N \times 256$ input image) is randomly cropped, randomly horizontally flipped, randomly rotated between $-40°$ and $-40°$, and resized to 256×256. Momentum is set to 0.95. The variance of the Gaussian is set to $\sigma = 1.5$ with an output heatmap size of 64×64. A temporal neighbourhood of $n = 15$ is input into the parametric pooling layer. The learning rate is set to 10^{-4}, and decreased to 10^{-5} at 80K iterations, to 10^{-6} after 100K iterations and stopped at 120K iterations. We use Caffe [19].

Training time. Training was performed on four NVIDIA GTX Titan GPUs using a modified version of the Caffe framework [19] with multi-GPU support. Training SpatialNet on FLIC took 3 days & SpatialNet Fusion 6 days.

Optical flow. Optical flow is computed using FastDeepFlow [38] with Middlebury parameters.

4. Datasets

Experiments in this work are conducted on four large video pose estimation datasets, two from signed TV broadcasts, one of Italian gestures, and the third of Hollywood movies. An overview of the datasets is given in Table 1. The first three datasets are available at `http://www.robots.ox.ac.uk/~vgg/data/pose`.

BBC Pose dataset. This dataset [3] consists of 20 videos (each 0.5h–1.5h in length) recorded from the BBC with an overlaid sign language interpreter. Each frame has been assigned pose estimates using the semi-automatic but reliable pose estimator of Buehler *et al*. [2] (used as training labels). 1,000 frames in the dataset have been manually annotated with upper-body pose (used as testing labels).

Extended BBC Pose dataset. This dataset [24] contains 72 additional training videos which, combined with the original BBC TV dataset, yields in total 85 training videos. The frames of these new videos have been assigned poses using the automatic tracker of Charles *et al*. [3]. The output of this tracker is noisier than the semi-automatic tracker of Buehler *et al*., which results in partially noisy annotations.

ChaLearn dataset. The ChaLearn 2013 Multi-modal gesture dataset [9] contains 23 hours of Kinect data of 27 people. The data includes RGB, depth, foreground segmentations and full body skeletons. In this dataset, both the training and testing labels are noisy (from Kinect). The large variation in clothing across videos poses a challenging task for pose estimation methods.

Poses in the Wild (PiW) and FLIC datasets. The Poses in the Wild dataset [6] contains 30 sequences (total 830 frames) extracted from Hollywood movies. The frames are annotated with upper-body poses. It contains realistic poses in indoor and outdoor scenes, with background clutter, severe camera motion and occlusions. For training, we follow [6] and use all the images annotated with upper-body parts (about 4.5K) in the FLIC dataset [26].

5. Experiments

We first describe the evaluation protocol, then present comparisons to alternative network architectures, and finally give a comparison to state of the art. A demo video is online at `http://youtu.be/pj2N5DqBOgQ`.

5.1. Evaluation protocol and details

Evaluation protocol. In all pose estimation experiments we compare the estimated joints against frames with manual

ground truth (except ChaLearn, where we compare against output from Kinect). We present results as graphs that plot accuracy vs distance from ground truth in pixels, where a joint is deemed correctly located if it is within a set distance of d pixels from a marked joint centre in ground truth.

Experimental details. All frames of the training videos are used for training (with each frame randomly augmented as detailed above). The frames are randomly shuffled prior to training to present maximally varying input data to the network. The hyperparameters (early stopping, variance σ *etc*.) are estimated using the validation set.

Baseline method. As a baseline method we include a CoordinateNet (described in [23]). This is a network with similar architecture to [28], but trained for regressing the joint positions directly (instead of a heatmap) [24].

Computation time. Our method is real-time (50fps on 1 GPU without optical flow, 5fps with optical flow).

5.2. Component evaluation

For these experiments the SpatialNet and baseline are trained and tested on the BBC Pose and Extended BBC Pose datasets. Fig 7 shows the results for wrists

With the SpatialNet, we observe a significant boost in performance (an additional 6.6%, from 79.6% to 86.1% at $d = 6$) when training on the larger Extended BBC dataset compared to the BBC Pose dataset. As noted in Sect 4, this larger dataset is somewhat noisy. In contrast, the CoordinateNet is unable to make effective use of this additional noisy training data. We believe this is because its target (joint coordinates) does not allow for multi-modal output, which makes learning from noisy annotation challenging.

We observe a further boost in performance from using optical flow to warp heatmaps from neighbouring frames (an improvement of 2.6%, from 86.1% to 88.7% at $d = 6$). Fig 9 shows the automatically learnt pooling weights. We see that for this dataset, as expected, the network learns to weigh frames temporally close to the current frame higher (because they contain less errors in optical flow).

Fig 8 shows a comparison of different pooling types (for cross-channel pooling). We compare learning a parametric pooling function to sum-pooling and to max-pooling (maxout [14]) across channels. As expected, parametric pooling performs best, and improves as the neighbourhood n increases. In contrast, results with both sum-pooling and max-pooling deteriorate as the neighbourhood size is increased further, as they are not able to down-weigh predictions that are further away in time (and thus more prone to errors in optical flow). As expected, this effect is particularly noticeable for max-pooling.

Failure modes. The main failure mode for the vanilla heatmap network (conv1-conv8) occurs when multiple modes are predicted and the wrong one is selected (and the resulting poses are often kinematically impossible for a human to perform). Examples of these failures are included in the extended arXiv version of this paper. The spatial fusion layers resolve these failures.

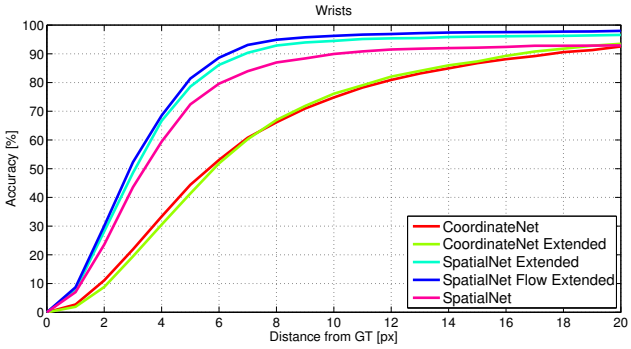


Figure 7. **Comparison of the performance of our nets for wrists on BBC Pose.** Plots show accuracy as the allowed distance from manual ground truth is increased. CoordinateNet is the network in [23]; SpatialNet is the heatmap network; and SpatialNet Flow is the heatmap network with the parametric pooling layer. 'Extended' indicates that the network is trained on Extended BBC Pose instead of BBC Pose. We observe a significant gain for the SpatialNet from using the additional training data in the Extended BBC dataset (automatically labelled – see Sect 4) training data, and a further boost from using optical flow information (and selecting the warping weights with the parametric pooling layer).

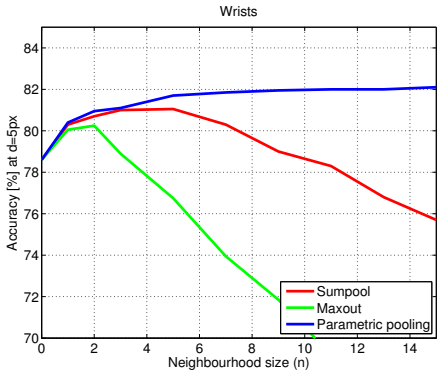


Figure 8. **Comparison of pooling types.** Results are shown for wrists in BBC Pose at threshold $d = 5$px. Parametric pooling (learnt cross-channel pooling weights) performs best.

5.3. Comparison to state of the art

Training. We investigated a number of strategies for training on these datasets including training from scratch (using only the training data provided with the dataset), or training on one (*i.e*. BBC Pose) and fine-tuning on the others. We found that provided the first and last layers of the Spatial Net are initialized from (any) trained heatmap network, the rest can be trained either from scratch or fine-tuned with similar performance. We hypothesise this is because the datasets are very different – BBC Pose contains long-sleeved persons, ChaLearn short-sleeved persons and

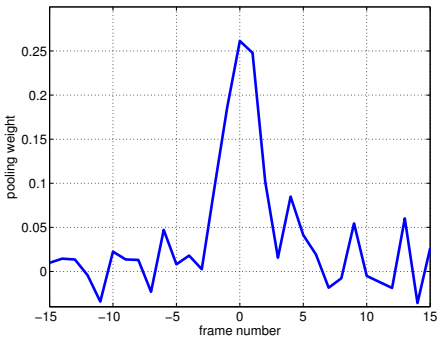


Figure 9. **Learnt pooling weights for BBC Pose with $n = 15$.** Weights shown for the right wrist. The centre frame receives highest weight. The jitter in weights is due to errors in optical flow computation (caused by the moving background in the video) – the errors become larger further away from the central frame (hence low or even negative weights far away).

Poses in the Wild contains non-frontal poses with unusual viewing angles. For all the results reported here we train BBC Pose from scratch, initialize the first and last layer from this, and fine-tune on training data of other datasets.

BBC Pose. Fig 10 shows a comparison to the state of the art on the BBC Pose dataset. We compare against all previous reported results on the dataset. These include Buehler *et al*. [2], whose pose estimator is based on a pictorial structure model; Charles *et al*. (2013) [3] who uses a Random Forest; Charles *et al*. (2014) [4] who predict joints sequentially with a Random Forest; Pfister *et al*. (2014) [24] who use a deep network similar to our CoordinateNet (with multiple input frames); and the deformable part-based model of Yang & Ramanan (2013) [39].

We outperform all previous work by a large margin, with a particularly noticeable gap for wrists (an addition of 10% compared to the best competing method at $d = 6$).

Chalearn. Fig 11 shows a comparison to the state of the art on ChaLearn. We again outperform the state of the art even without optical flow (an improvement of 3.5% at $d = 6$), and observe a further boost by using optical flow (beating state of the art by an addition of 5.5% at $d = 6$), and a significant further improvement from using a deeper network (an additional 13% at $d = 6$).

Poses in the Wild. Figs 12 & 14 show a comparison to the state of the art on Poses in the Wild. We replicate the results of the previous state of the art method using code provided by the authors [6]. We outperform the state of the art on this dataset by a large margin (an addition of 30% for wrists and 24% for elbows at $d = 8$). Using optical flow yields a significant 10% improvement for wrists and 13% for elbows at $d = 8$. Fig 15 shows example predictions.

FLIC. Fig 13 shows a comparison to the state of the art on FLIC. We outperform all pose estimation methods that

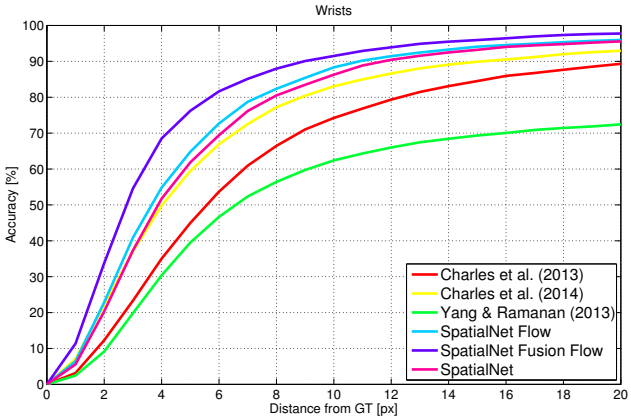


Figure 11. **Comparison to the state of the art on ChaLearn.** Our method outperforms state of the art by a large margin (an addition of 19% at $d = 4$). See arXiv version for elbows & shoulders.

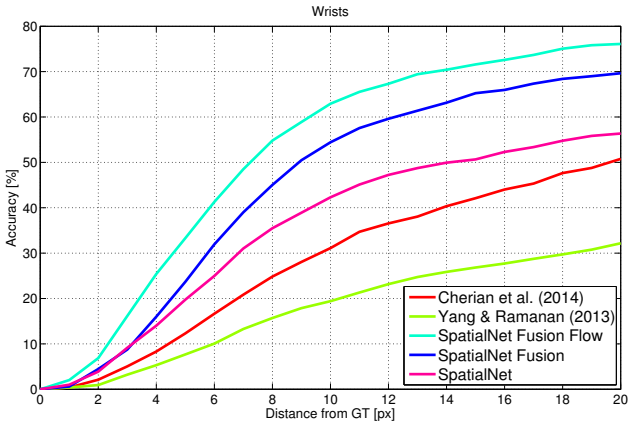


Figure 12. **Comparison to state of the art on Poses in the Wild.** Our method outperforms state of the art by a large margin (an addition of 30% at $d = 8$, with 10% from flow).

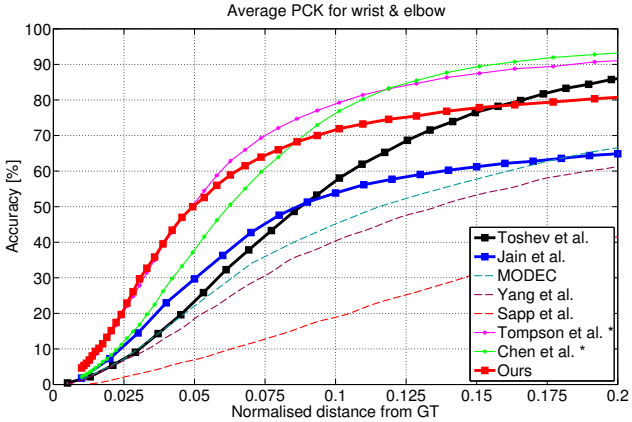


Figure 13. **Comparison to state of the art on FLIC.** Solid lines represent deep models; methods with a square (■) are without a graphical model; methods with an asterisk (*) are with a graphical model. Our method outperforms competing methods without a graphical model by a large margin in the high precision area (an addition of 20% at $d = 0.05$).

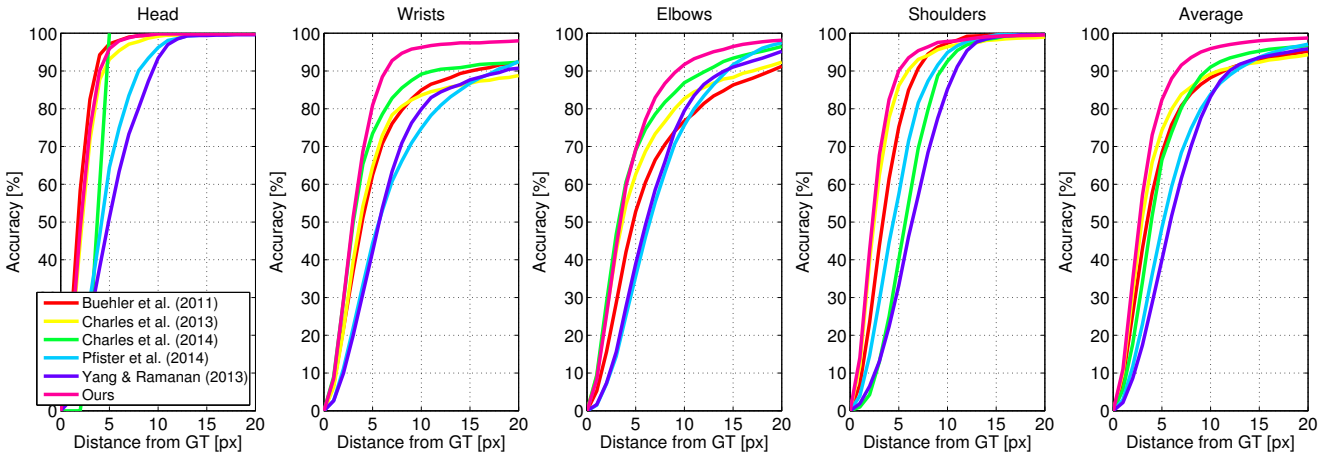


Figure 10. **Comparison to the state of the art on BBC Pose.** Plots show accuracy per joint type (average over left and right body parts) as the allowed distance from manual ground truth is increased. We outperform all previous work by a large margin; notice particularly the performance for wrists, where we outperform the best competing method with an addition of 10% at $d = 6$. Our method uses SpatialNet Flow Extended. Pfister *et al*. (2014) uses Extended BBC Pose; Buehler *et al*., Charles *et al*. and Yang & Ramanan use BBC Pose.

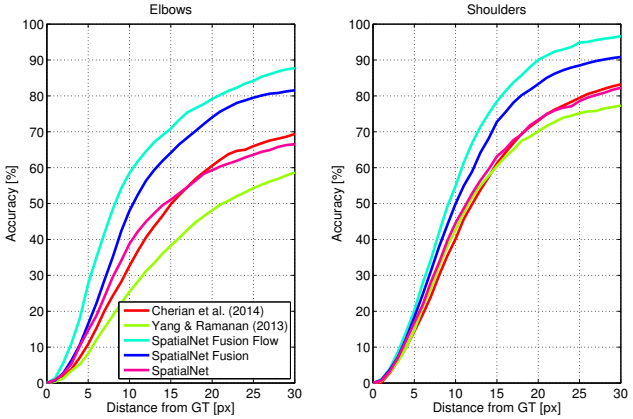


Figure 14. **Poses in the Wild: elbows & shoulders.**

Figure 15. **Example predictions on Poses in the Wild.**

don't use a graphical model, and match or even slightly outperform graphical model-based methods [5, 35] in the very high precision region (< 0.05 from GT). The increase in accuracy at $d = 0.05$ is 20% compared to methods not using a graphical model, and 12% compared to [5] who use a graphical model. Tompson *et al*. is [35]; Jain *et al*. is [17]. Predictions are provided by the authors of [5, 35] and evaluation code by the authors of [35].

6. Conclusion

We have presented a new architecture for pose estimation in videos that is able to utilizes appearances across multiple frames. The proposed ConvNet is a simple, direct method for regressing heatmaps, and its performance is improved by combining it with optical flow and spatial fusion layers. We have also shown that our method outperforms the state of the art on three large video pose estimation datasets.

Further improvements may be obtained by using additional inputs for the spatial ConvNet, for example multiple RGB frames [24] or optical flow [18] – although prior work has shown little benefit from this so far.

The benefits of aligning pose estimates from multiple frames using optical flow, as presented here, are complementary to architectures that explicitly add spatial MRF and refinement layers [35, 36].

Finally, we have demonstrated the architecture for human pose estimation, but a similar optical flow-mediated combination of information could be used for other tasks in video, including classification and segmentation.

Acknowledgements: Financial support was provided by Osk. Huttunen Foundation and EPSRC grant EP/I012001/1.

References

[1] L. Bourdev and J. Malik. Poselets: Body part detectors trained using 3d human pose annotations. In *Proc. CVPR*, 2009. 1

[2] P. Buehler, M. Everingham, D. P. Huttenlocher, and A. Zisserman. Upper body detection and tracking in extended signing sequences. *IJCV*, 2011. 1, 5, 7

[3] J. Charles, T. Pfister, M. Everingham, and A. Zisserman. Automatic and efficient human pose estimation for sign language videos. *IJCV*, 2013. 1, 5, 7

[4] J. Charles, T. Pfister, D. Magee, D. Hogg, and A. Zisserman. Upper body pose estimation with temporal sequential forests. *Proc. BMVC*, 2014. 1, 2, 4, 7

[5] X. Chen and A. Yuille. Articulated pose estimation with image-dependent preference on pairwise relations. In *Proc. NIPS*, 2014. 1, 8

[6] A. Cherian, J. Mairal, K. Alahari, and C. Schmid. Mixing body-part sequences for human pose estimation. In *Proc. CVPR*, 2014. 5, 7

[7] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *Proc. ICML*, 2014. 1

[8] M. Eichner, M. Marin-Jimenez, A. Zisserman, and V. Ferrari. 2d articulated human pose estimation and retrieval in (almost) unconstrained still images. *IJCV*, 2012. 1

[9] S. Escalera, J. Gonzalez, X. Baro, M. Reyes, O. Lopes, I. Guyon, V. Athistos, and H. Escalante. Multi-modal gesture recognition challenge 2013: Dataset and results. In *Proc. ICMI*, 2013. 5

[10] P. Felzenszwalb and D. Huttenlocher. Pictorial structures for object recognition. *IJCV*, 2005. 1

[11] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proc. CVPR*, 2014. 1

[12] R. Girshick, J. Shotton, P. Kohli, A. Criminisi, and A. Fitzgibbon. Efficient regression of general-activity human poses from depth images. In *Proc. ICCV*, 2011. 1

[13] G. Gkioxari, B. Hariharan, R. Girshick, and J. Malik. Using k-poselets for detecting people and localizing their keypoints. In *Proc. CVPR*, 2014. 1

[14] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. *J. Machine Learning Research*, 2013. 6

[15] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. In *Proc. ICLR*, 2014. 1

[16] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman. Synthetic data and artificial neural networks for natural scene text recognition. *Proc. NIPS Workshops*, 2014. 1

[17] A. Jain, J. Tompson, M. Andriluka, G. Taylor, and C. Bregler. Learning human pose estimation features with convolutional networks. *Proc. ICLR*, 2014. 1, 8

[18] A. Jain, J. Tompson, Y. LeCun, and C. Bregler. MoDeep: A deep learning framework using motion features for human pose estimation. *Proc. ACCV*, 2014. 2, 8

[19] Y. Jia. Caffe: An open source convolutional architecture for fast feature embedding. http://caffe.berkeleyvision.org/, 2013. 5

[20] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proc. CVPR*, 2014. 1

[21] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. NIPS*, 2012. 1, 3

[22] M. Osadchy, Y. LeCun, and M. Miller. Synergistic face detection and pose estimation with energy-based models. *JMLR*, 2007. 1

[23] T. Pfister. *Advancing Human Pose and Gesture Recognition*. PhD thesis, University of Oxford, 2015. 6

[24] T. Pfister, K. Simonyan, J. Charles, and A. Zisserman. Deep convolutional neural networks for efficient pose estimation in gesture videos. *Proc. ACCV*, 2014. 1, 2, 5, 6, 7, 8

[25] S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. *CVPR Workshops*, 2014. 1

[26] B. Sapp and B. Taskar. Modec: Multimodal decomposable models for human pose estimation. In *Proc. CVPR*, 2013. 5

[27] B. Sapp, D. Weiss, and B. Taskar. Parsing human motion with stretchable models. In *Proc. CVPR*, 2011. 1

[28] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *Proc. ICLR*, 2014. 1, 6

[29] J. Shotton, R. Girshick, A. Fitzgibbon, T. Sharp, M. Cook, M. Finocchio, R. Moore, P. Kohli, A. Criminisi, A. Kipman, and A. Blake. Efficient human pose estimation from single depth images. *PAMI*, 2013. 1

[30] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. *Proc. NIPS*, 2014. 1, 2

[31] M. Sun, P. Kohli, and J. Shotton. Conditional regression forests for human pose estimation. In *Proc. CVPR*, 2012. 1

[32] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proc. CVPR*, 2014. 1

[33] G. Taylor, R. Fergus, G. Williams, I. Spiro, and C. Bregler. Pose-sensitive embedding by nonlinear nca regression. In *Proc. NIPS*, 2010. 1

[34] J. Taylor, J. Shotton, T. Sharp, and A. Fitzgibbon. The vitruvian manifold: Inferring dense correspondences for one-shot human pose estimation. In *Proc. CVPR*, 2012. 1

[35] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler. Efficient object localization using convolutional networks. *Proc. CVPR*, 2015. 1, 3, 8

[36] J. Tompson, A. Jain, Y. LeCun, and C. Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. *Proc. NIPS*, 2014. 1, 3, 8

[37] A. Toshev and C. Szegedy. DeepPose: Human pose estimation via deep neural networks. *CVPR*, 2014. 1, 2

[38] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid. Deepflow: Large displacement optical flow with deep matching. In *Proc. ICCV*, 2013. 4, 5

[39] Y. Yang and D. Ramanan. Articulated human detection with flexible mixtures of parts. *PAMI*, 2013. 1, 7

[40] M. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *Proc. ECCV*, 2014. 1

Understanding deep features with computer-generated imagery

Mathieu Aubry
École des Ponts ParisTech UC Berkeley
mathieu.aubry@imagine.enpc.fr

Bryan C. Russell
Adobe Research
brussell@adobe.com

Abstract

We introduce an approach for analyzing the variation of features generated by convolutional neural networks (CNNs) with respect to scene factors that occur in natural images. Such factors may include object style, 3D viewpoint, color, and scene lighting configuration. Our approach analyzes CNN feature responses corresponding to different scene factors by controlling for them via rendering using a large database of 3D CAD models. The rendered images are presented to a trained CNN and responses for different layers are studied with respect to the input scene factors. We perform a decomposition of the responses based on knowledge of the input scene factors and analyze the resulting components. In particular, we quantify their relative importance in the CNN responses and visualize them using principal component analysis. We show qualitative and quantitative results of our study on three CNNs trained on large image datasets: AlexNet [18], Places [40], and Oxford VGG [8]. We observe important differences across the networks and CNN layers for different scene factors and object categories. Finally, we demonstrate that our analysis based on computer-generated imagery translates to the network representation of natural images.

1. Introduction

The success of convolutional neural networks (CNNs) [18, 21] raises fundamental questions on how their learned representations encode variations in visual data. For example, how are different layers in a deep network influenced by different scene factors, the task for which the network was trained for, or the choice in network architecture? These questions are important as CNNs with different architectures and trained/fine tuned for different tasks have shown to perform differently [17, 40] or have different feature response characteristics [39]. An analysis of the features may help with understanding the tradeoffs across different trained networks and may inform the design of new architectures. It may also help the choice of CNN features for tasks where training or fine tuning a network is not possible, e.g. due to lack of labeled data.

Prior work has focused on a part-based analysis of the learned convolutional filters. Examples include associating filters with input image patches having maximal response [12], deconvolution starting from a given filter response [38], or by masking the input to recover the receptive field of a given filter [39] to generate "simplified images" [6, 31]. Such visualizations typically reveal the parts of an object [38] (e.g. "eye" of a cat) or scene [39] (e.g. "toilet" in bathroom). While these visualizations reveal the nature of learned filters, they largely ignore the question of the dependence of the CNN representation on continuous factors that may influence the depicted scene, such as 3D viewpoint, scene lighting configuration, and object style.

In this paper, we study systematically how different scene factors that arise in natural images are represented in a trained CNN. Example factors may include those intrinsic to an object or scene, such as category, style, and color, and extrinsic ones, such as 3D viewpoint and scene lighting configuration. Studying the variations associated with such factors is a nontrivial task as it requires (i) input data where the factors can be independently controlled and (ii) a procedure for detecting, visualizing, and quantifying each factor in a trained CNN.

To overcome the challenges associated with obtaining input data, we leverage computer-generated (CG) imagery to study trained CNNs. CG images offer several benefits. First, there are stores of 3D content online (e.g. Trimble 3D Warehouse), with ongoing efforts to curate and organize the data for research purposes (e.g. ModelNet [35]). Such data spans many different object categories and styles. Moreover, in generating CG images we have control over all rendering parameters, which allows us to systematically and densely sample images for any given factor. A database of natural images captured in controlled conditions and spanning different factors of variations, e.g. the NORB [22], ETH-80 [23] and RGB-D object [20] datasets, where different objects are rotated on a turntable and lighting is varied during image capture, are difficult and costly to collect. Moreover they do not offer the same variety of object styles present in 3D model collections, nor the flexibility given by rendering.

Given a set of rendered images generated by varying one or more factors, we analyze the responses of a layer for a trained CNN (e.g. "pool5" of AlexNet [18]). We perform a decomposition of the responses based on knowledge of the input scene factors, which allows us to quantify the relative importance of each factor in the representation. Moreover, we visualize the responses via principal component analysis (PCA).

Contributions. Our technical contribution is an analysis of contemporary CNNs trained on large-scale image datasets via computer generated images, particularly rendered from 3D models. From our study we observe:

- Features computed from image collections that vary along two different factors, such as style and viewpoint, can often be approximated by a linear combination of features corresponding to the factors, especially in the higher layers of a CNN.
- Sensitivity to viewpoint decreases progressively in the last layers of the CNNs. Moreover, the VGG fc7 layer appears to be less sensitive to viewpoint than AlexNet and Places.
- Relative to object style, color is more important for the Places CNN than for AlexNet and VGG. This difference is more pronounced for the background color than for foreground.
- The analysis we perform on deep features extracted from rendered views of 3D models is related to understanding their representation in natural images.

1.1. Related work

In addition to the prior work to visualize learned CNN filters [5, 11, 12, 30, 38, 39], there has been work to visualize hand-designed [34] and deep [24] features. Also related are recent work to understand the quantitative tradeoffs across different CNN layers for networks trained on large image databases [1, 37] and designing CNN layers manually [7].

Our use of a large CAD model dataset can be seen in the context of leveraging such data for computer vision tasks, e.g. object detection [2]. Contemporary approaches have used synthetic data with CNNs to render images for particular scene factors, e.g. style, pose, lighting [10, 19].

Our PCA feature analysis is related to prior work on studying visual embeddings. The classic Eigenfaces paper [33] performed PCA on faces. Later work studied nonlinear embeddings, such as LLE [27] and IsoMap [32]. Most related to us is the study of nonlinear CNN feature embeddings with the NORB dataset [14]. In contrast we study large-scale, contemporary CNNs trained on large image datasets. Finally, our multiple factor study is related to intrinsic image decomposition [4]. We note a contemporary approach for separating style and content via autoencoders [9].

1.2. Overview

Our deep feature analysis begins by rendering a set of stimuli images by varying one or more scene factors. We present the stimuli images to a trained CNN as input and record the feature responses for a desired layer. Given the feature responses for the stimuli images we analyze the principal modes of variation in the feature space via PCA (section 2.1). When more than one factor is present we linearly decompose the feature space with respect to the factors and perform PCA on the feature decomposition (section 2.2). We give details of our experimental setup in section 3 and show qualitative and quantitative results over a variety of synthetic and natural images in section 4.

2. Approach for deep feature analysis

In this section we describe our approach for analyzing the image representation learned by a CNN. We seek to study how the higher levels of the CNN encodes the diversity present in a set of images. The minimal input for our analysis is a set of related images. We first describe our approach for analyzing jointly their features. What we can learn with such an approach is however limited since it cannot identify the origin of the variations of the input images. The factors of variation can be, e.g., variations in the style of an object, changes in its position, scaling, 3D rotation, lighting, or color. For this reason, we then focus on the case when the images are computer generated and we have full control of the different factors. In this case, we seek to separate the influence of the different factors on the representation, analyze them separately, and compare their relative importance.

2.1. Image collection analysis

We seek to characterize how a CNN encodes a collection of related images, Ω, e.g. images depicting a "car" or a black rectangle on white background. We sample images $r_\theta \in \Omega$ indexed by $\theta \in \Theta$. In the case of natural images Θ is an integer index set over the collection Ω. In the case of computer-generated images Θ is a set of parameters corresponding to a scene factor we wish to study (e.g. azimuth and elevation angles for 3D viewpoint, 3D model instances for object style, or position of the object in the image for 2D translation). Given a trained CNN, let $\tilde{F}^L(r_\theta)$ be a column vector of CNN responses for layer L (e.g. "pool5", "fc6", or "fc7" in AlexNet [18]) to the input image r_θ.

The CNN responses $\tilde{F}^L$ are high-dimensional feature vectors that represent the image information. However, since Ω contains related images, we expect the features to be close to each other and their intrinsic dimension to be smaller than the actual feature dimension. For this reason, we use principal component analysis (PCA) [25] to identify a set of orthonormal basis vectors that capture the principal modes

of variation of the features.

Given centered features $F^L(\theta) = \tilde{F}^L(r_\theta) - \frac{1}{|\Theta|}\sum_{t\in\Theta}\tilde{F}^L(r_t)$, where $|X|$ is the number of elements in set X, we compute the eigenvectors associated with the largest eigenvalues of the covariance matrix $\frac{1}{|\Theta|}\sum_{\theta\in\Theta}F^L(\theta)(F^L(\theta))^T$. The projection of the features onto the subspace defined by the D components with maximal eigenvalues corresponds to an optimal D-dimensional linear approximation of the features. We evaluate the intrinsic dimensionality of the features by computing the number of dimensions necessary to explain 95% of the variance. Moreover, we can visualize the embedding of the images by projecting onto the components with high variance.

2.2. Multiple factor analysis

In the case when we have control of the variation parameters, we can go further and attempt to decompose the features as a linear combination of uncorrelated components associated to the different factors of variation. Features decomposing linearly into different factors would be powerful and allow to perform image transformations directly in feature space and, e.g., to compare easily images taken under different viewpoints.

Let $\Theta_1, \ldots, \Theta_N$ be sets of parameters for N factors of variation we want to study. We consider an image of the scene with parameters $\theta = (\theta_1, \ldots, \theta_N)$, where $\theta \in \Theta = \Theta_1 \times \cdots \times \Theta_N$. We assume the θ_k are sampled independently. We define marginal features $F_k^L(\theta_k)$ for scene factor k by marginalizing over the parameters for all factors except k:

$$F_k^L(t) = \mathbb{E}(F^L(\theta)|\theta_k = t) \quad (1)$$

$$= \frac{|\Theta_k|}{|\Theta|}\sum_{\theta\in\Theta|\theta_k=t} F^L(\theta) \quad (2)$$

Similar to section 2.1, we can study via PCA the principal modes of variation over the marginal features F_k^L for each factor k.

Finally, we define a residual feature $\Delta^L(\theta)$, which is the difference of the centered CNN features $F^L(\theta)$ and the sum of all the marginal features $F_k^L(\theta_k)$. This results in the following decomposition:

$$F^L(\theta) = \sum_{k=1}^{N} F_k^L(\theta_k) + \Delta^L(\theta) \quad (3)$$

Using computer-generated images, we can easily compute this decomposition by rendering the images with all the rendering parameters corresponding to the sum in equation (2). Direct computation shows that all the terms in decomposition (3) have zero mean and are uncorrelated. This implies:

$$\text{var}(F^L) = \sum_{k=1}^{N} \text{var}(F_k^L) + \text{var}(\Delta^L) \quad (4)$$

We can thus decompose the variance of the features F^L as the sum of the variances associated to the different factors and a residual. When analyzing the decomposed features we report the relative variance $R_k^L = \text{var}(F_k^L)/\text{var}(F^L)$. We also report the relative variance of the residual $R_\Delta^L = \text{var}(\Delta^L)/\text{var}(F^L)$. A factor's relative variance provides an indication of how much the factor is represented in the CNN layer compared to the others. A high value indicates the factor is dominant in the layer and conversely a low value indicates the factor is largely negligible compared to the others. Moreover, a low value of the residual relative variance indicates the factors are largely separated in the layer. Note that $R_\Delta^L + \sum_{k=1}^{N} R_k^L = 1$ and the values of R_k^L and R_Δ^L do not depend on the relative sampling of the different factors.

3. Experimental setup

In this section we describe details of our experimental setup. In particular we describe the details of the CNN features we extract, our rendering pipeline, and the set of factors we seek to study.

3.1. CNN features

We study three trained CNN models: AlexNet [18], winner of the 2012 ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [28], Places [40], which has the same architecture as AlexNet but trained on a large image database depicting scenes, and Oxford VGG [8] CNN-S network. In particular, we study the features of the higher layers "pool5", "fc6", and "fc7" of these networks. Note that the Oxford VGG architecture is different and the dimension of its "pool5" layer is two times larger than AlexNet and Places. We use the publicly-available CNN implementation of Caffe [16] to extract features for the different layers and pre-trained models for AlexNet, Places, and Oxford VGG from their model zoo.

3.2. Computer-generated imagery

We present two types of image stimuli as input: (i) 2D abstract stimuli consisting of constant color images or rectangular patches on constant background, which are described in detail in section 4.1, and (ii) rendered views from 3D models. For the latter we seek to render different object categories spanning many different styles from a variety of viewpoints and under different illumination conditions. We used as input CAD 3D models from the ModelNet database [35], which contains many models having different styles for a variety of object classes. We downloaded the CAD models in Collada file format for the following object classes: chair (1261 models), car (485 models), sofa (701 models), toilet (191 models) and bed (258 models). We adapted the publicly-available OpenGL renderer from [3], which renders a textured CAD model with matte surfaces under fixed lighting configuration and allows the viewpoint to be specified by a 3×4 camera

(a) AlexNet dims. 1, 2 (b) Places dims. 1, 2 (c) VGG dims. 1, 2 (d) VGG dims. 2, 3

Figure 1: PCA embeddings of constant-color images for the fc7 layer of different CNNs. The AlexNet and Places embeddings are similar to a hue color wheel, with more variation visible for the green and blue channels.

matrix [15]. We render the models under different lighting conditions and with different uniform colors.

We show results on two categories that have received the most attention in 3D-based image analysis: cars [13, 26], and chairs [2, 10]. Detailed results for three other categories are presented in the supplementary material and our quantitative results are averages over the 5 categories.

3.3. 3D scene factors

We study two types of factors affecting the appearance of a scene: (i) intrinsic factors – object category, style, and color, and (ii) extrinsic factors – 2D position, 2D scale, 3D viewpoint, and scene lighting configuration. The object category and style factors are specified by the CAD models from ModelNet. For object color we study grayscale matte surfaces (specified by Lambertian surface model), and constant-colored matte surfaces with the color uniformly sampled on a grid in RGB colorspace. For the 2D extrinsic factors, we uniformly sample 2D positions along a grid in the image plane and vary the 2D scale linearly. For 3D viewpoint we manually aligned all the 3D models to a canonical coordinate frame, i.e. all models are consistently oriented with respect to gravity and face the same direction, orbit the object at constant 3D distance and uniformly sample the azimuth and elevation angles with respect to the object's coordinate frame. Finally, we vary the scene lighting such that the source light varies from left to right and front to back on a uniform grid.

For the quantitative experiments we sampled 36 azimuth angles (keeping the elevation fixed at 10 degrees) for rotation, 36 positions for translation, 40 scales, 36 light positions, and 125 colors. For the visualizations we sampled 120 azimuth angles, 121 light positions, and 400 positions to make the embeddings easier to interpret. We checked that the different sampling did not change our quantitative results, which was expected since our method is not sensitive to the relative number of samples for each factor.

Table 1: Relative variance of the aspect ratio, 2D position, and residual feature for our synthetic rectangle experiment with AlexNet. Notice that the relative variance of the aspect ratio increases with the higher layers while 2D position decreases, which indicates that the features focus more on the shape and less on the 2D location in the image.

	2D position	Aspect ratio	Δ^L
AlexNet, pool5	49.8 %	9.5 %	40.8 %
AlexNet, fc6	45.1 %	22.3 %	32.6 %
AlexNet, fc7	33.9 %	37.0 %	29.1 %

4. Results

In this section we highlight a few results from our experiments on CNN feature analysis. The supplementary material reports our detailed quantitative results for all object categories and provides a visualization tool to interactively select and compare embeddings for the first ten PCA components.

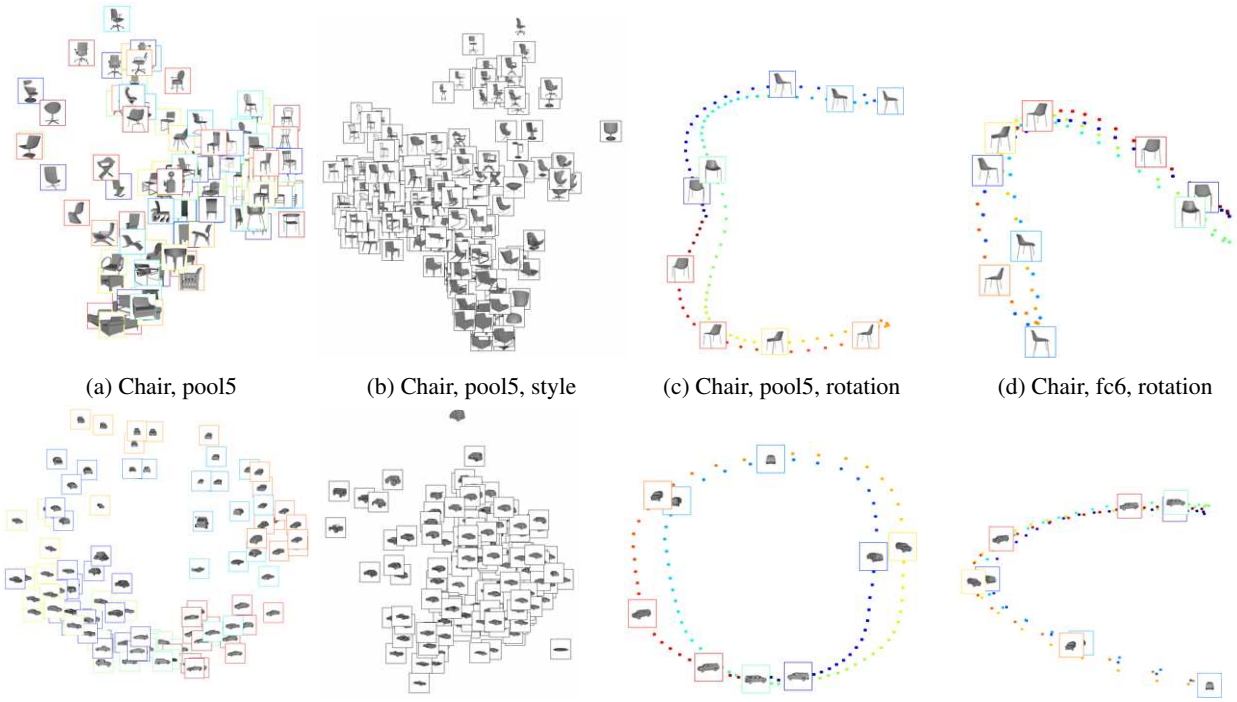
We first report results for manually-designed 2D stimuli in section 4.1 and then for rendered views of 3D models from several object categories in section 4.2. We finally show in section 4.3 that our results obtained with computer-generated images are related to natural images.

4.1. 2D abstract stimuli

In this section we apply the deep feature analysis of section 2 on manually-designed 2D abstract stimuli presented to a trained CNN. We first perform a PCA analysis on a single factor, color. Next, we perform two-factor quantitative analyses on the aspect ratio/2D position of a rectangle and on the foreground/background colors of a centered square.

Uniform color. We perform PCA on a set of images with constant color, as described in section 2.1. We sampled 1331 colors uniformly on a grid in RGB color space. The resulting embedding for the fc7 layers of the three CNNs are shown in figure 1. The resulting embeddings for the AlexNet and Places CNNs are surprisingly similar to a hue color wheel, with more variation visible for blues and greens and less for reds and violets. VGG has a different behavior, with the first dimension similar to saturation. The embedding corresponding to the second and third dimensions is similar to that of the other CNNs. The number of PCA dimensions necessary to explain 95% of the variance is approximately 20 for all three networks and all three layers, which is higher than the three dimensions of the input data.

Position and aspect ratio. We used the methodology of section 2.2 to study the features representing a small black rectangle located at different positions with different aspect ratios on a white background. The rectangle area was kept



(a) Chair, pool5 (b) Chair, pool5, style (c) Chair, pool5, rotation (d) Chair, fc6, rotation

(e) Car, pool5 (f) Car, pool5, style (g) Car, pool5, rotation (h) Car, fc6, rotation

Figure 2: **Best viewed in the electronic version.** PCA embeddings (dims. 1,2) of AlexNet features for "chairs" (first row) and "cars" (second row). Column 1 – Direct embedding of the rendered images without viewpoint-style separation. Columns 2,3 – Embeddings associated with style (for all rotations) and rotation (for all styles). Column 4 – Rotation embedding for fc6, which is qualitatively different than pool5. Colors correspond to orientation and can be interpreted via the example images in columns 3,4. Similar results for other categories and PCA dimensions are available in the supplementary material.

constant at 0.26^2 of the image area. We consider the position and aspect ratio as two factors of variation and sample 36 positions on a grid and 12 aspect ratios on a log scale. The variance explained by each factor for the different layers of AlexNet is presented in table 1. For all three networks the relative variance associated to the position decreases, which quantitatively supports the idea that the higher layers have more translation invariance. In contrast the relative variance associated to the aspect ratio increases for the higher CNN layers. For AlexNet, less than 10% of the relative variance for pool5 is explained by the aspect ratio alone, while it explains 37% of the relative variance for fc7. Also, the relative variance associated with the residual decreases for the higher CNN layers, which indicates the two factors are more easily linearly separated in the higher layers.

Center and surrounding color. Similar to the experiments presented in the previous paragraph, we considered a square of one color on a background of a different color. We chose the size of the central square to be half the image size. Quantitative results for the fc7 features of the different networks are presented in table 2. Results for the other layers are in the supplementary material. A first observation is that the features do not separate as well the foreground and background colors in the representation as the aspect ratio and position in the previous experiment. We also observe that for all the networks the variance associated to the background color is higher than the variance associated to the foreground. The difference is more striking for the Places fc7 layer (3.8x versus 2x for AlexNet fc7 and 1.8x for VGG fc7). Future work could determine if the background color of an image is especially important for scene classification, while the foreground color is less important.

Remarks. As the CNNs were not trained on the 2D artificial stimuli presented in this section, we find it somewhat surprising that the embeddings resulting from the above feature analysis is meaningful. From our experiments we saw that the CNNs learn a rich representation of colors, identifying in particular variations similar to hue and saturation. Moreover, the last layers of the network better encode translation invariance, focusing on shape. These results will be confirmed and generalized on more realistic stimuli in the next sections.

Table 2: Relative variance and intrinsic dimensionality of a foreground square of one color on a background color. Each cell: top – rel. variance; bottom – intrinsic dim.

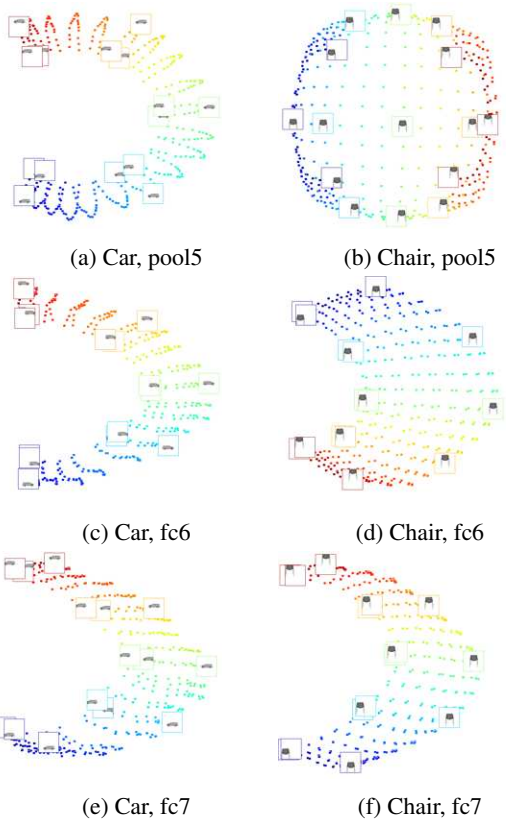
	Foreground	Background	Δ^L
Places, fc7	13.4% 13	51.1% 14	35.5% 216
AlexNet, fc7	19.2% 14	39.9% 16	40.8% 315
VGG, fc7	20.2% 11	36.9% 15	42.9% 216

4.2. Object categories

In this section we want to explore the embedding generated by the networks for image sets and factors related to the tasks for which they are trained, namely object category classification in the case of AlexNet and VGG. We also compare against the CNN trained on Places. We thus select an object category and, using rendered views of 3D models, we analyze how the CNN features are influenced by the style of the specific instances as well as different transformations and rendering parameters. The parameter sampling for each experiment is described in section 3.3.

Model–orientation separation. The first variation we study jointly with style is the rotation of the 3D model. The first column of figure 2 visualizes the PCA embedding of the resulting pool5 features. This embedding is hard to interpret because it mixes information about viewpoint (important for cars) and instance style (important for chairs). To separate this information, we perform the decomposition presented in section 2. The decomposition provides us with embedding spaces for style and viewpoint and associates to each model and viewpoint its own descriptor. We visualize the embeddings in figure 2; the second column corresponds to style and the third to viewpoint. Note that the different geometries of the two categories lead to different embeddings of rotation in pool5. While a left-facing car typically looks similar to a right-facing car and is close in the feature space (figure 2g), a right-facing chair is usually different from left-facing chairs and is far in the embedding (figure 2c). The last column shows the viewpoint embedding for fc6. The comparison of the last two columns indicates that much viewpoint information is lost between pool5 and fc6 and that fc6 is largely left-right flip invariant. A potential interesting future direction could be to interpret the viewpoint embeddings relative to classic work on mental rotation [29].

Translation, scale, lighting, color. We repeated the same experiment for the following factors: 2D translation, scale, light direction, background color, and object color. For simplicity and computational efficiency, we considered in all experiments a frontal view of all the instances of the objects. The framework allows the same analysis using the object orientation as an additional factor. The embeddings associated with AlexNet features for translation of cars and chairs are shown in figure 3. Note that similar to rotations, the embedding corresponding to cars and chairs are different, and that the first two components of the fc6 features indicate a left-right flip-invariant representation. The embeddings for the pool5 layer of the car category for the other factors are shown figure 4.



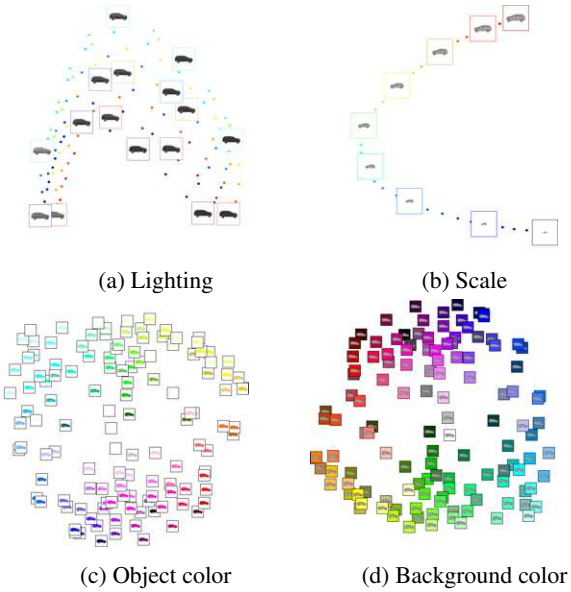
(a) Car, pool5 (b) Chair, pool5

(c) Car, fc6 (d) Chair, fc6

(e) Car, fc7 (f) Chair, fc7

Figure 3: PCA embeddings for 2D position on AlexNet.

Quantitative analysis: viewpoint. We analyze the relative variance explained by the 3D rotation, translation, and scale experiments. While the variance was different for each factor and category, the variation across the layers and networks was consistent in all cases. For this reason we report in table 3 an average of the variance across all five categories and all three factors. We refer the reader to the supplementary material for detailed results. The analysis of table 3 reveals several observations. First, the proportion of the variance of deeper layers corresponding to viewpoint information is less important, while the proportion corresponding to style is more important. This corresponds to the



(a) Lighting (b) Scale

(c) Object color (d) Background color

Figure 4: PCA embeddings for different factors using AlexNet pool5 features on "car" images. Colors in (a) correspond to location of the light source (green – center).

intuition that higher layers are more invariant to viewpoint. We also note that the residual feature Δ^L is less important in higher layers, indicating style and viewpoint are more easily separable in those layers. These observations are consistent with our results of section 4.1. Second, the part of the variance associated with style is more important in the fc7 layer for VGG than in AlexNet and Places. Also, the part associated with the viewpoint and residual is smaller. Note that this does not hold in pool5, where the residual is important for the VGG network. This effect may be related to the difference in the real and intrinsic dimension of the features. The intrinsic dimension of the style component of VGG pool5 features is larger and decreases from pool5 to fc7. On the contrary, the intrinsic dimensionality of AlexNet has smaller variation across layers. Finally, we note that the intrinsic dimensionality of the fc7 style feature of Places is smaller than the other networks. This may indicate that it is less rich, and may be related to the fact that identifying the style of an object is less crucial for scene classification. We believe it would be an interesting direction for future work to study how the improved performance of VGG for object classification is related to the observed reduced sensitivity to viewpoint.

Quantitative analysis: color. We report in table 4 the average across categories of our quantitative study for object and background color. The results are different from those of viewpoint. First, we observe that a larger part of the variance of the features of the Places network is explained by the color in all layers. This may be related to the fact that color is a stronger indicator of the scene type than it is of an object category. Second, while the part of the variance explained by foreground and background color is similar in the fc7 feature of the Places network, it is much larger for the foreground object than for the background object in AlexNet and VGG. Once again, one can hypothesize that it is related to the fact that the color of an object is more informative than the color of its background for object classification. Finally, we note that similarly to our previous experiments, the difference between networks is present in pool5 and increases in the higher layers, indicating that the features become more tuned to the target task in the higher layers of the networks.

Table 3: Relative variance and intrinsic dimensionality averaged over experiments for different object categories and viewpoints (3D orientation, translation, and scale). Each cell: top – rel. variance; bottom – intrinsic dim. We do not report the intrinsic dim. of Δ^L since it is typically larger than 1K across the experiments and expensive to compute.

		pool5	fc6	fc7
Viewpoint	Places	26.8 % 8.5	21.4 % 7.0	17.8 % 5.9
	AlexNet	26.4 % 8.3	19.4 % 7.2	15.6 % 6.0
	VGG	21.2 % 10.0	16.4 % 7.7	12.3 % 6.2
Style	Places	26.8 % 136.3	39.1 % 105.5	49.4 % 54.6
	AlexNet	28.2 % 121.1	40.3 % 125.5	49.4 % 96.7
	VGG	26.4 % 181.9	44.3 % 136.3	56.2 % 94.2
Δ^L	Places	46.8 %	39.5 %	32.9 %
	AlexNet	45.0 %	40.3 %	35.0 %
	VGG	52.4 %	39.3 %	31.5 %

4.3. Natural images

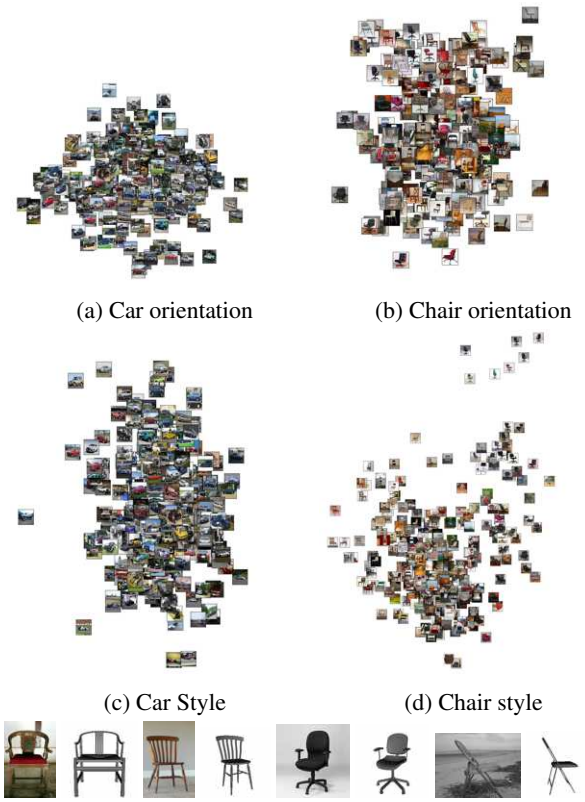
Embedding. We used ImageNet [28] images to study the embeddings of natural images. Since we have no control over the image content, we cannot perform a detailed analysis of the different factors similar to the previous sections. Our only choice is to consider the images altogether. The direct embedding of natural images is possible but hard to interpret. We can however project the images in the spaces discovered in section 4.2. The resulting embeddings for style and viewpoint are shown in figure 5 and are similar to the embeddings obtained with the CAD models.

2D-3D instance recognition. The observed similarity of the embeddings for natural and rendered images motivates

Table 4: Average relative variance over five classes for color/style separation.

Foreground/Style		pool5	fc6	fc7
FG color	Places	23.4 %	29.4 %	34.9 %
	AlexNet	23.2 %	24.0 %	24.0 %
	VGG	15.0 %	22.6 %	25.0 %
Style	Places	48.9 %	41.3 %	40.3 %
	AlexNet	56.6 %	52.1 %	52.5 %
	VGG	59.0 %	51.4 %	51.3 %
Δ^L	Places	27.7 %	29.3 %	24.8 %
	AlexNet	20.3 %	24.0 %	23.5 %
	VGG	26.0 %	25.9 %	23.6 %

Background/Style		pool5	fc6	fc7
BG color	Places	24.3 %	29.6 %	35.1 %
	AlexNet	17.3 %	16.2 %	14.4 %
	VGG	9.1 %	13.8 %	14.3 %
Style	Places	51.5 %	40.7 %	40.9 %
	AlexNet	63.7 %	59.4 %	61.8 %
	VGG	71.4 %	64.3 %	65.3 %
Δ^L	Places	24.2 %	29.7 %	24.0 %
	AlexNet	19.0 %	24.4 %	23.9 %
	VGG	19.5 %	22.0 %	20.4 %



(a) Car orientation (b) Chair orientation

(c) Car Style (d) Chair style

(e) 2D-3D retrieval examples

Figure 5: PCA embeddings over AlexNet pool5 features for cars and chairs with orientation and style separated.

an application to retrieve a 3D model of an object present in an image without explicitly performing alignment or detection. The approach is different from approaches in the 2D-3D matching community that find explicit correspondences between the models and the image. We tested this idea using the chair category and computing similarity as dot product between AlexNet features. We rendered 36 azimuth and 4 elevation angles to span typical viewpoints depicted in the natural images. To improve efficiency, we reduced the dimension of our features to 1000 via PCA. This allows us to perform nearest-neighbor retrieval between 1000 natural images and the 144 rendered views of 1261 3D models in approximately 22 seconds total using a Python implementation with pre-computed features. We visualize results in figure 5e and in the supplementary material. We evaluated the viewpoint accuracy using the annotations of [36] and found that the orientation error was below 20 degrees for 60% of the images using pool5 features, 39% using fc6, and 26% using fc7. This is consistent with our earlier finding that orientation is not as well represented in the higher layers. We conducted a user study on Mechanical Turk to evaluate the quality of the style matching for the images where the orientation was correctly estimated with the pool5 features. The workers were presented with a pair of images and asked to judge if the style of the chairs and their orientation were similar or different, similar to [2]. Each pair was evaluated 5 times. There was agreement in 75% of the cases that the match was fair and in 18% that it was exact. While the above results could probably be beaten by state-of-the-art 2D-3D matching techniques or simply by adding position and scale to our database, they show that our analysis of rendered 3D models is pertinent for understanding the CNN representation of natural images.

Real images with pose variation. We applied the methodology of section 4.2 to natural images in the ETH-80 dataset [23]. The dataset spans 8 object categories with 10 instances each captured under 41 viewpoints. As the dataset is significantly smaller than the ModelNet database, this limits considerably the number of variations we can explore. Results using AlexNet features are shown in table 5. The high-level conclusions are the same as those of section 4.2, but the differences are less obvious. The variance is explained more by the style and less by the viewpoint and residual as one progresses toward the higher layers in the network. Please see detailed results for each category in the supplementary material.

Table 5: Average relative variance over the 8 categories of the ETH-80 dataset [23].

	Rotation	Style	Δ^L
AlexNet, pool5	35.4 %	21.6 %	43.0 %
AlexNet, fc6	30.2 %	27.7 %	42.0 %
AlexNet, fc7	29.5 %	30.5 %	40.0 %

5. Conclusion

We have introduced a method to qualitatively and quantitatively analyze deep features by varying the network stimuli according to factors of interest. Utilizing large collections of 3D models, we applied this method to compare the relative importance of different factors and have highlighted the difference in sensitivity between the networks and layers to object style, viewpoint and color. We believe our analysis gives new intuitions and opens new perspectives for the design and use of convolutional neural networks.

Acknowledgments. Mathieu Aubry was partly supported by ANR project Semapolis ANR-13-CORD-0003, Intel, a gift from Adobe, and hardware donation from Nvidia.

References

[1] P. Agrawal, R. Girshick, and J. Malik. Analyzing the performance of multilayer neural networks for object recognition. In *ECCV*, 2014. 2

[2] M. Aubry, D. Maturana, A. A. Efros, B. C. Russell, and J. Sivic. Seeing 3D chairs: Exemplar part-based 2D-3D alignment using a large dataset of CAD models. In *CVPR*, 2014. 2, 4, 8

[3] M. Aubry, B. C. Russell, and J. Sivic. Painting-to-3D model alignment via discriminative visual elements. *ACM Transactions on Graphics*, 33(2), 2014. 3

[4] H. Barrow and J. Tenenbaum. Recovering intrinsic scene characteristics from images. In A. Hanson and E. Riseman, editors, *Computer Vision Systems*, pages 3–26. Academic Press, N.Y., 1978. 2

[5] P. Berkes and L. Wiskott. On the analysis and interpretation of inhomogeneous quadratic forms as receptive fields. *Neural Computation*, 2006. 2

[6] I. Biederman. *Visual object recognition*, volume 2. MIT press Cambridge, 1995. 1

[7] J. Bruna and S. Mallat. Invariant scattering convolution networks. *IEEE PAMI*, 35(8):1872–1886, 2013. 2

[8] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *Proc. BMVC.*, 2014. 1, 3

[9] B. Cheung, J. Livezey, A. Bansal, and B. Olshausen. Discovering hidden factors of variation in deep networks. In *ICLR workshop*, 2015. 2

[10] A. Dosovitskiy, J. T. Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. In *CVPR*, 2015. 2, 4

[11] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. Technical report, University of Montreal, 2009. 2

[12] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 1, 2

[13] D. Glasner, M. Galun, S. Alpert, R. Basri, and G. Shakhnarovich. Viewpoint-aware object detection and pose estimation. In *ICCV*, 2011. 4

[14] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *CVPR*, 2006. 2

[15] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004. 4

[16] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. 3

[17] S. Karayev, M. Trentacoste, H. Han, A. Agarwala, T. Darrell, A. Hertzmann, and H. Winnemöller. Recognizing image style. In *Proc. BMVC.*, 2014. 1

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012. 1, 2, 3

[19] T. D. Kulkarni, W. Whitney, P. Kohli, and J. B. Tenenbaum. Deep convolutional inverse graphics network. *arXiv preprint arXiv:1503.03167*, 2015. 2

[20] K. Lai, L. Bo, X. Ren, and D. Fox. A large-scale hierarchical multi-view RGB-D object dataset. In *Proc. Intl. Conf. on Robotics and Automation*, 2011. 1

[21] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551, 1989. 1

[22] Y. LeCun, F.-J. Huang, and L. Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *CVPR*, 2004. 1

[23] B. Leibe and B. Schiele. Analyzing appearance and contour based methods for object categorization. In *CVPR*, volume 2, pages II–409. IEEE, 2003. 1, 8, 9

[24] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *CVPR*, 2015. 2

[25] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(11):559–572, 1901. 2

[26] B. Pepik, M. Stark, P. Gehler, and B. Schiele. Teaching 3D geometry to deformable part models. In *CVPR*, 2012. 4

[27] S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000. 2

[28] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *arXiv:1409.0575*, 2014. 3, 7

[29] R. Shepard and J. Metzler. Mental rotation of three dimensional objects. *Science*, 171(972):701–3, 1971. 6

[30] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR workshop*, 2014. 2

[31] K. Tanaka. Neuronal mechanisms of object recognition. *Science*, 262(5134):685–688, 1993. 1

[32] J. Tenenbaum, V. de Silva, and J. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(550):2319–2323, December 2000. 2

[33] M. Turk and A. Pentland. Eigenfaces for recognition. *J. of Cognitive Neuroscience*, 3(1):71–86, 1991. 2

[34] C. Vondrick, A. Khosla, T. Malisiewicz, and A. Torralba. HOGgles: Visualizing object detection features. In *ICCV*, 2013. 2

[35] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A deep representation for volumetric shape modeling. In *CVPR*, 2015. 1, 3

[36] Y. Xiang, R. Mottaghi, and S. Savarese. Beyond PASCAL: A benchmark for 3D object detection in the wild. In *WACV*, 2014. 8

[37] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *NIPS*, 2014. 2

[38] M. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014. 1, 2

[39] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Object detectors emerge in deep scene CNNs. In *ICLR*, 2015. 1, 2

[40] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using Places database. In *NIPS*, 2014. 1, 3

Visual Tracking with Fully Convolutional Networks

Lijun Wang[1,2], Wanli Ouyang[2], Xiaogang Wang[2], and Huchuan Lu[1]
[1]Dalian University of Technology, China
[2]The Chinese University of Hong Kong, Hong Kong, China

Abstract

We propose a new approach for general object tracking with fully convolutional neural network. Instead of treating convolutional neural network (CNN) as a black-box feature extractor, we conduct in-depth study on the properties of CNN features offline pre-trained on massive image data and classification task on ImageNet. The discoveries motivate the design of our tracking system. It is found that convolutional layers in different levels characterize the target from different perspectives. A top layer encodes more semantic features and serves as a category detector, while a lower layer carries more discriminative information and can better separate the target from distracters with similar appearance. Both layers are jointly used with a switch mechanism during tracking. It is also found that for a tracking target, only a subset of neurons are relevant. A feature map selection method is developed to remove noisy and irrelevant feature maps, which can reduce computation redundancy and improve tracking accuracy. Extensive evaluation on the widely used tracking benchmark [36] shows that the proposed tacker outperforms the state-of-the-art significantly.

1. Introduction

Visual tracking, as a fundamental problem in computer vision, has found wide applications. Although much progress [7, 29, 38] has been made in the past decade, tremendous challenges still exist in designing a robust tracker that can well handle significant appearance changes, pose variations, severe occlusions, and background clutters.

Existing appearance-based tracking methods adopt either generative or discriminative models to separate the foreground from background and distinct co-occurring objects. One major drawback is that they rely on low-level hand-crafted features which are incapable to capture semantic information of targets, not robust to significant appearance changes, and only have limited discriminative power.

Driven by the emergence of large-scale visual data sets and fast development of computation power, Deep Neural Networks (DNNs), especially convolutional neural networks [17] (CNNs), with their strong capabilities of learning feature representations, have demonstrated record breaking performance in computer vision tasks, *e.g.*, image classification [16, 27], object detection [8, 23, 22], and saliency detection [33, 39]. Different from hand-crafted features, those learned by CNNs from massive annotated visual data and a large number of object classes (such as ImageNet [4]) carry rich high-level semantic information and are strong at distinguishing objects of different categories. These features have good generalization capability across data sets. Recent studies [28, 1] have also shown that such features are robust to data corruption. Their neuron respons-

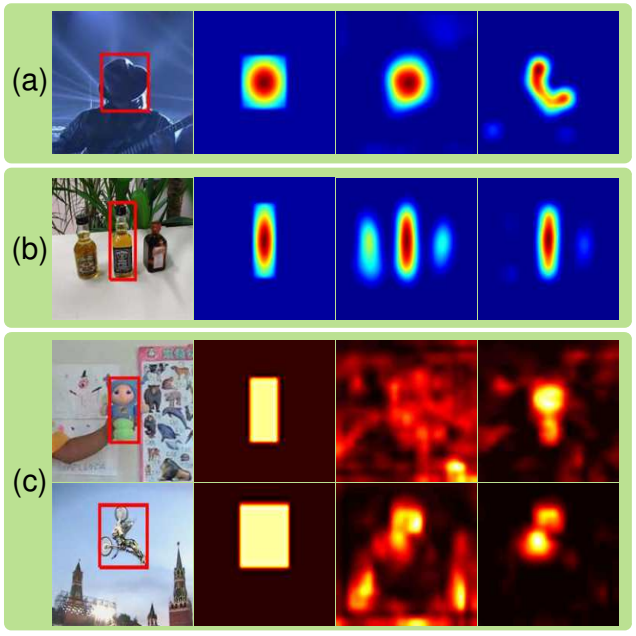


Figure 1. Feature maps for target localization. (a)(b) From left to right: input image, the ground truth target heat map, the predicted heat maps using feature maps of conv5-3 and conv4-3 layers of VGG network [27] (See Section 4.2 for the regression method). (c) From left to right: input image, ground truth foreground mask, average feature maps of conv5-3 (top) and conv4-3 (bottom) layers, average selected feature maps conv5-3 (top) and conv4-3 (bottom) layers (See Section 4.1 for the feature map selection method).

es have strong selectiveness on object identities, *i.e*., for a particular object only a subset of neurons are responded and different objects have different responding neurons.

All these motivate us to apply CNNs to address above challenges faced by tracking. Given the limited number of training samples in online tracking and the complexity of deep models, it is inferior to directly apply CNNs to tracking, since the power of CNNs relies on large-scale training. Prior works [6, 35, 12] attempted to transfer offline learned DNN features (*e.g*. from ImageNet) for online tracking and achieved state-of-the-art performance. However, DNN was treated as a black-box classifier in these works. In contrast, we conduct in-depth study on the properties of CNN features from the perspective of online visual tracking and in order to make better use of them in terms of both efficiency and accuracy. Two such properties are discovered and they motivate the design of our tracking system.

First, CNN features at different levels/depths have different properties that fit the tracking problem. A top convolutional layer captures more abstract and high-level semantic features. They are strong at distinguishing objects of different classes and are very robust to deformation and occlusion as shown in Figure 1 (a). However, they are less discriminative to objects of the same category as shown by the examples in Figure 1 (b). A lower layer provides more detailed local features which help to separate the target from distracters (*e.g*. other objects in the same class) with similar appearance as shown in Figure 1 (b). But they are less robust to dramatic change of appearance, as shown in Figure 1 (a). Based on these observations, we propose to automatically switch the usage of these two layers during tracking depending on the occurrence of distracters.

Second, the CNN features pre-trained on ImageNet are for distinguishing generic objects. However, for a particular target, not all the features are useful for robust tracking. Some feature responses may serve as noise. As shown in Figure 1 (c), it is hard to distinguish the target object from background if all the feature maps are used. In contrast, through proper feature selection, the noisy feature maps not related to the representation of the target are cleared out and the remaining ones can more accurately highlight the target and suppress responses from background. We propose a principled method to select discriminative feature maps and discard noisy or unrelated ones for the tracking target.

The contributions of this work are three folds:

i) We analyze CNN features learned from the large-scale image classification task and find important properties for online tracking. It facilitates further understanding of CNN features and helps to design effective CNN-based trackers.

ii) We propose a new tracking method which jointly considers two convolutional layers of different levels so that they complement each other in handling drastic appearance change and distinguishing target object from its similar distracters. This design significantly mitigate drifts.

iii) We develop a principled method which automatically selects discriminative feature maps and discards noisy or unrelated ones, further improving tracking accuracy.

Evaluation on the widely used tracking benchmark [36] shows that the proposed method well handles a variety of challenging problems and outperforms state-of-the-art methods.

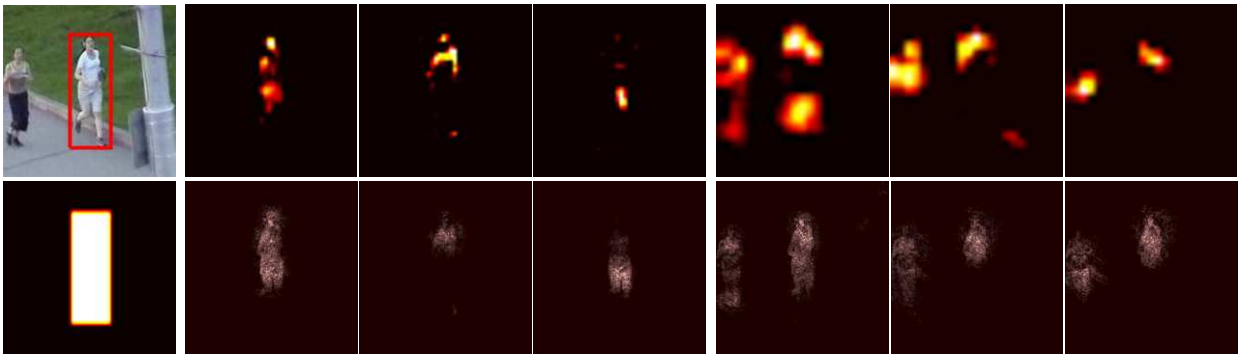
2. Related Work

A tracker contains two components: an appearance model updated online and a search strategy to find the most likely target locations. Most recent works [2, 41, 11, 20] focus on the design of appearance models. In generative models, candidates are searched to minimize reconstruction errors. For example, Ross *et al*. [25] learned subspace online to model target appearance. Recently, sparse coding has been exploited for tracking [21, 3, 32, 31, 30], where the target is reconstructed by a sparse linear combination of target templates. In discriminative models, tracking is cast as a foreground and background separation problem [24, 37, 34]. Online learning algorithms based on CRFs [24], boosting [9], multiple instance learning [2] and structured SVM [10] were applied in tracking and achieved good performance. In [40], the generative and discriminative models were incorporated for more accurate online tracking. All these methods used hand-crafted features.

The application of DNNs in online tracking is under fully explored. In [35], a stacked denoising autoencoder (SDAE) was offline trained on an auxiliary tiny image data set to learn generic features and then used for online tracking. In [19], tracking was performed as foreground-background classification with CNN trained online without offline pretraining. Fan *et al*. [6] used fully convolutional network for human tracking. It took the whole frame as input and predicted the foreground heat map by one-pass forward propagation. Redundant computation was saved. Whereas [35] and [19] operated in a patch-by-by scanning manner. Given N patches cropped from the frame, DNNs had to be evaluated for N times. The overlap between patches leads to a lot of redundant computation. In [12], pre-trained CNN features were used to construct target-specific saliency maps for online tracking. Existing works treated DNNs as black-box feature extractors. Our contributions summarized in Section 1 were not explored in these works.

3. Deep Feature Analysis for Visual Tracking

Analysis on deep representations is important to understand the mechanism of deep learning. However, it is still very rare for the purpose of visual tracking. In this section, we present some important properties of CNN features which can better facilitate visual tracking. Our feature analysis is conducted based on the 16-layer VGG network [27]



(a) (b) (c)

Figure 2. CNNs trained on image classification task carry spatial configuration information. (a) input image (top) and ground truth foreground mask. (b) feature maps (top row) of conv4-3 layer which are activated within the target region and are discriminative to the background distracter. Their associated saliency maps (bottom row) are mainly focused on the target region. (c) feature maps (top row) of conv5-3 layer which are activated within the target region and capture more semantic information of the category (both the target and background distracter). Their saliency maps (bottom row) present spatial information of the category.

pre-trained on the ImageNet image classification task [4], which consists of 13 convolutional layers followed by 3 fully connected layers. We mainly focus on the conv4-3 layer (the 10-th convolutional layer) and the conv5-3 layer (the 13-th convolutional layer), both of which generate 512 feature maps.

Observation 1 *Although the receptive field* [1] *of CNN feature maps is large, the activated feature maps are sparse and localized. The activated regions are highly correlated to the regions of semantic objects .*

Due to pooling and convolutional layers, the receptive fields of the conv4-3 and conv5-3 layers are very large (92×92 and 196×196 pixels, respectively). Figure 2 shows some feature maps with the maximum activation values in the object region. It can be seen that the feature maps have only small regions with nonzero values. These nonzero values are localized and mainly correspond to the image region of foreground objects. We also use the approach in [26] to obtain the saliency maps of CNN features. The saliency maps in Figure 2 (bottom row) show that the change of input that results in the largest increase of the selected feature maps are located within the object regions. Therefore, the feature maps are capturing the visual representation related to the objects. These evidences indicate that DNN features learned from image classification are localized and correlated to the object visual cues. Thus, these CNN features can be used for target localization.

Observation 2 *Many CNN feature maps are noisy or unrelated for the task of discriminating a particular target from its background.*

The CNN features pre-trained on ImageNet can describe a large variety of generic objects and therefore they are supposed to detect abundant visual patterns with a large number of neurons. However, when tracking a particular target object, it should focuses on a much smaller subset of visual patterns which well separate the target from background. As illustrated in Figure 1 (c), the average of all the feature maps is cluttered with background noise. And we should discard feature maps that have high response in both target region and background region so that the tracker does not drift to the background regions. Figure 3 shows the histograms of the activation values for all the feature maps within the object region. The activation value of a feature map is defined as the sum of its responses in the object region. As demonstrated in Figure 3, most of the feature maps have small or zero values within the object region. Therefore, there are lots of feature maps that are not related to the target object. This property provides us the possibility in selecting only a small number of feature maps with small degradation in tracking performance.

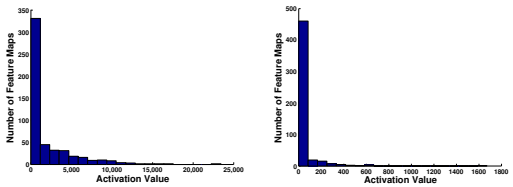


Figure 3. Activation value histograms of feature maps in conv4-3 (left) and conv5-3 (right).

Observation 3 *Different layers encode different types of features. Higher layers capture semantic concepts on object categories, whereas lower layers encode more discriminative features to capture intra class variations.*

Because of the redundancy of feature maps, we employ a sparse representation scheme to facilitate better visualization. By feeding forward an image of an object through the VGG network, we obtain the feature maps $\mathbf{F} \in \mathbb{R}^{d\times n}$ of a convolutional layer (either the conv4-3 or conv5-3 layer),

[1] We use the term *receptive field* to denote the input image region that are connected to a particular neuron in the feature maps

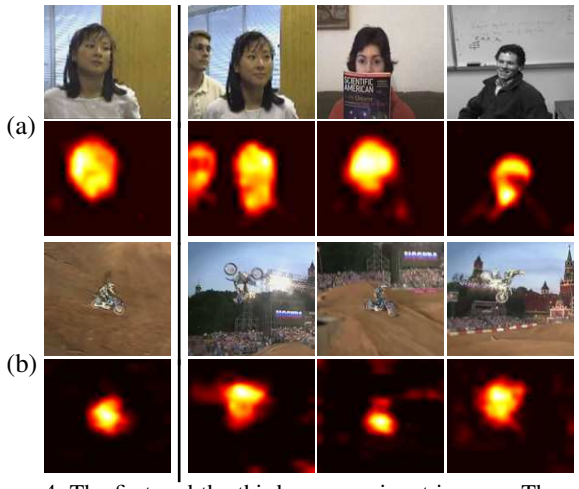


Figure 4. The first and the third rows are input images. The second and the fourth rows are reconstructed foreground masks using conv5-3 feature maps. The sparse coefficients are computed using the images in the first column and directly applied to the other columns without change.

where each feature map is reshaped into a d-dimensional vector and n denotes the number of feature maps. We further associate the image with a foreground mask $\boldsymbol{\pi} \in \mathbb{R}^{d\times 1}$, where the i-th element $\pi_i = 1$ if the i-th neuron of each feature map is located within the foreground object, and $\pi_i = 0$, otherwise. We reconstruct the foreground mask using a subset of the feature maps by solving

$$\begin{aligned} \min_{\mathbf{c}} \|\boldsymbol{\pi} - \mathbf{F}\mathbf{c}\|_2^2 + \lambda\|\mathbf{c}\|_1, \\ \text{s.t. } \mathbf{c} \succeq 0, \end{aligned} \tag{1}$$

where $\mathbf{c} \in \mathbb{R}^{n\times 1}$ is the sparse coefficient vector, and λ the parameter to balance the reconstruction error and sparsity.

Figure 4 shows some of the reconstructed foreground masks using the feature maps of conv5-3. For the two examples (face and motorcycle) in Figure 4, we only compute the sparse coefficients for images in the first column and use the coefficients to reconstruct foreground masks for the rest of the columns. The selected feature maps in Figure 4 (a) capture the semantic concepts of human faces and are robust to faces with appearance variation and even identity change. The selected feature maps in Figure 4 (b) accurately separate the target from cluttered background and are invariant to pose variation and rotation. Although trained on the image classification task, the high-level semantic representation of object categories encoded by the conv5-3 layer enables object localization. However, these features are not discriminative enough to different objects of the same category, thus they can not be directly applied to visual tracking.

Compared with the conv5-3 feature maps, the features captured by conv4-3 are more sensitive to intra-class appearance variation. In Figure 2, the selected feature maps of conv4-3 can well separate the target person from the other non-target person. Besides, different feature maps focus on different object parts.

Table 1. Face classification accuracy using different feature maps. Experiment 1 is to classify face and non-face. Experiment 2 is classify face identities.

Feature map	Experiment 1	Experiment 2
conv4-3	76.42%	83.83%
conv5-3	89.56%	57.28%

To further verify this, we conduct two quantitative experiments. 1800 human face images belonging to six identities and 2000 images containing non-face objects are collected from the benchmark sequences [36]. Each image is associated with a foreground mask to indicate the region of the foreground object. In the first experiment, we evaluate the accuracy in classifying the images into face and non-face using the conv4-3 and conv5-3 layers separately. Three face images belonging to three identities are selected as positive training samples to compute a set of sparse coefficients $\{\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_2\}$ via (1). At the test stage, given the feature maps $\mathbf{F}$ and the foreground mask $\boldsymbol{\pi}$ of an input image, the reconstruction error e for the foreground map is computed by

$$e = \min_i \|\boldsymbol{\pi} - \mathbf{F}\mathbf{c}_i\|_2^2. \tag{2}$$

The image is classified as a face image if its reconstruction error e is less than a predefined threshold. Otherwise, it is classified as a non-face image.

In the second experiment, our task is to classify all the face images into different identities. For each identity, 20 images are used as the training samples to learn the sparse coefficients $\mathbf{c}_i,\ i = 1, 2, \ldots, 6$ using (1). At the test stage, the foreground mask reconstruction error for each identity is calculated, and the test image is classified as the identity that has the minimum error as follows:

$$id = \arg\min_i \|\boldsymbol{\pi} - \mathbf{F}\mathbf{c}_i\|_2^2. \tag{3}$$

The classification accuracy using the feature maps of conv4-3 and conv5-3 for the two experiments are demonstrated in Table 1. The feature maps of conv5-3 encode high level semantic information and can better separate face from non-face objects. But they achieve lower accuracy than the features maps of conv4-3 in discriminating one identity from another. The feature maps of conv4-3 preserve more middle-level information and enables more accurate classification of different images belonging to the same category (human faces). But they are worse than the feature maps of conv5-3 in discriminating face from non-face. These results motivate us to consider these two layers jointly for more robust tracking.

4. Proposed Algorithm

An overview (Figure 5) of the proposed fully convolutional network based tracker (FCNT) is as follows:

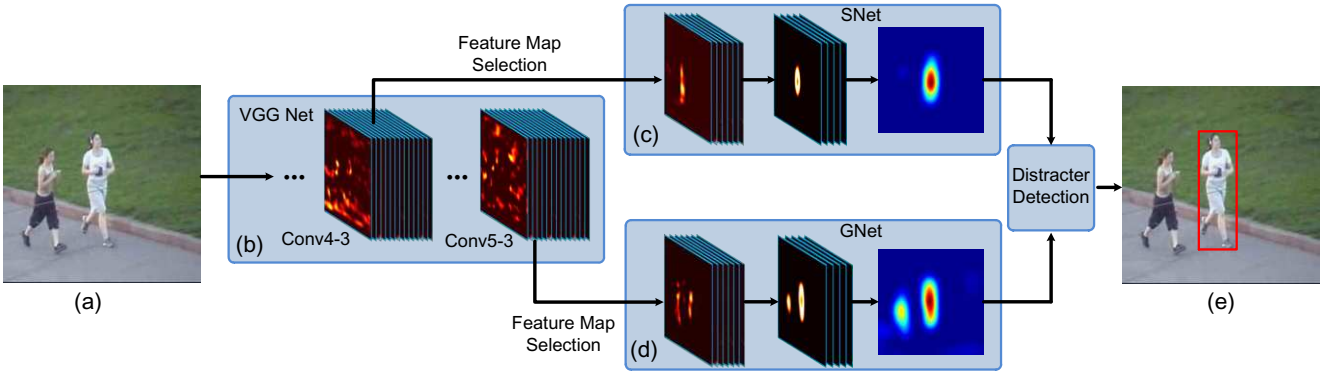


Figure 5. Pipeline of our algorithm. (a) Input ROI region. (b) VGG network. (c) SNet. (d) GNet. (e) Tracking results.

1. For a given target, a feature map selection process is performed on the conv4-3 and conv5-3 layers of the VGG network to select the most relevant feature maps and avoid overfitting on noisy ones.

2. A general network (GNet) that captures the category information of the target is built on top of the selected feature maps of the conv5-3 layer.

3. A specific network (SNet) that discriminates the target from background with similar appearance is built on top of the selected feature maps of the conv4-3 layer.

4. Both GNet and SNet are initialized in the first frame to perform foreground heat map regression for the target and adopt different online update strategies.

5. For a new frame, a region of interest (ROI) centered at the last target location containing both target and background context is cropped and propagated through the fully convolutional network.

6. Two foreground heat maps are generated by GNet and SNet, respectively. Target localization is performed independently based on the two heat maps.

7. The final target is determined by a distracter detection scheme that decides which heat map in step 6 to be used.

4.1. Feature Map Selection

The proposed feature map selection method is based on a target heat map regression model, named as sel-CNN, and is conducted independently on the conv4-3 and conv5-3 layers of VGG. The sel-CNN model consists of a dropout layer followed by a convolutional layer without any nonlinear transformation. It takes the feature maps (conv4-3 or con5-3) to be selected as input to predict the target heat map $\mathbf{M}$, which is a 2-dimensional Gaussian centered at the ground truth target location with variance proportional to the target size (See Figure 1 (a) and (b) for an example). The model is trained by minimizing the square loss between the predicted foreground heat map $\hat{\mathbf{M}}$ and the target heat map $\mathbf{M}$:

$$L_{sel} = \|\hat{\mathbf{M}} - \mathbf{M}\|^2. \tag{4}$$

After parameter learning using back-propagation converges, we fix the model parameters and select the feature maps according to their impacts on the loss function. The input feature maps $\mathbf{F}$ are vectorized into a vector denoted by $vec(\mathbf{F})$. Denote f_i as the i-th element of $vec(\mathbf{F})$. The change of the loss function caused by the perturbation of the feature map $\delta\mathbf{F}$ can be computed by a two-order Taylor expansion as follows:

$$\delta L_{sel} = \sum_i g_i \delta f_i + \frac{1}{2}\sum_i h_{ii}(\delta f_i)^2 + \frac{1}{2}\sum_{i\neq j} h_{ij}\delta f_i \delta f_j, \tag{5}$$

where $g_i = \frac{\partial L_{sel}}{\partial f_i}$ and $h_{ij} = \frac{\partial^2 L_{sel}}{\partial f_i \partial f_j}$ are, respectively, the first and second order derivatives of the objective function with respect to the input feature maps. The number of elements in the feature maps is very large ($> 270,000$). The complexity for computing all the second order derivatives h_{ij} is $O(270,000^2)$, which is too time consuming. We approximate the Hessian matrix with a diagonal matrix, in which the third term of the right hand side of (5) is neglected. Both the first derivatives g_i and the second derivatives h_{ii} can be efficiently computed via back-propagation.

We define the significance of the element f_i as the change of the objective function after setting f_i to zero, *i.e.*, $\delta f_i = 0 - f_i$. According to (5), the significance of f_i can then be computed as

$$s_i = -g_i f_i + \frac{1}{2} h_{ii} f_i^2. \tag{6}$$

The significance of the k-th feature map are further defined as the summation of significance of all its elements $S_k = \sum_{x,y} s(x,y,k)$, where $s(x,y,k)$ is the significance of the element indexed by location (x,y) on the k-th feature map. All the feature maps are sorted in the descending order by their significance, and the top K feature maps are selected. These selected feature maps have significant impact on the objective function and thus are most relevant to the tracking task. Our feature map selection method can be conducted in an online fashion. In our experiments, we only conduct feature selection at the first frame and have achieved good performance. This should be partially attributed to the robustness of CNN features.

The idea of using quadratic approximation of the cost function to remove connections in networks can be traced back to 1989 [18]. The aim was to reduce the number of parameters and improve speed, while we target on removing noisy feature maps and improving tracking accuracy.

4.2. Target Localization

Figure 5 (c) and (d) show the design of the CNNs for target localization. After feature map selection in the first frame, we build the SNet and the GNet on top of the selected conv4-3 and conv5-3 feature maps, respectively. Both networks share the same architecture that consists of two additional convolutional layers. The first additional convolutional layer has convolutional kernels of size 9×9 and outputs 36 feature maps as the input to the next layer. The second additional convolutional layer has kernels of size 5×5 and outputs the foreground heat map of the input image. ReLU is chosen as the nonlinearity for these two layers.

SNet and GNet are initialized in the first frame by minimizing the following square loss function:

$$\begin{aligned} L &= L_S + L_G, \\ L_U &= \|\hat{\mathbf{M}}_U - \mathbf{M}\|_F^2 + \beta\|\mathbf{W}_U\|_F^2, \end{aligned} \tag{7}$$

where the subscript $U \in \{S, G\}$ indicates SNet and GNet, respectively; $\hat{\mathbf{M}}_U$ represents the foreground heat map predicted by the network; $\mathbf{M}$ is the target heat map, $\mathbf{W}_U$ is the weight parameter of the convolutional layers; β is a trade off parameter for weight decay.

Note that the sel-CNN for selecting features and the S-Net and GNet for localization are different in CNN structures. The sel-CNN architecture is very simple to avoid using noisy feature maps to overfit the objective function, whereas the SNet and GNet are more complex. Since the noisy feature maps have been discarded by the feature map selection, more complex models facilitate more accurate tracking. Detailed experimental results and analysis on the selection of different model architectures for localization and feature map selection are provided in the supplementary materials.

In a new frame, we crop a rectangle ROI region centered at the last target location. By forward propagating the ROI region through the networks, the foreground heat maps are predicted by both GNet and SNet. Target localization is first performed on the heat map produced by GNet. Denote the target location as $\hat{\mathbf{X}} = (x, y, \sigma)$, where x, y and σ represent the center coordinates and scale of the target bounding box, respectively. Given the target location $\hat{\mathbf{X}}^{t-1}$ in the last frame, we assume the locations of target candidates in the current frame are subject to a Gaussian distribution

$$p(\mathbf{X}^t|\hat{\mathbf{X}}^{t-1}) = \mathcal{N}(\mathbf{X}^t; \hat{\mathbf{X}}^{t-1}, \mathbf{\Sigma}), \tag{8}$$

where $\mathbf{\Sigma}$ is a diagonal covariance matrix that indicates the variances of the location parameters. The confidence of the i-th candidate is computed as the summation of all the heat map values within the candidate region $conf_i = \sum_{j\in\mathbf{R}_i} \hat{\mathbf{M}}_G(j)$, where $\hat{\mathbf{M}}_G$ denotes the heat map generated by GNet; $\mathbf{R}_i$ is the region of the i-th target candidate according to its location parameter $\mathbf{X}_i^t$ (8); j denotes the coordinate index. The candidate with the highest confidence is predicted as the target by GNet.

According to the analysis in Section 3, GNet based on the conv5-3 layer captures semantic features and is highly invariant to intra class variation. Hence, the foreground heat map generated by GNet highlights both the target and background distracters with similar appearances.

To prevent the tracker from drifting to background, we further promote a distracter detection scheme to determine the final target location. Denote the target location predicted by GNet as $\hat{\mathbf{X}}_G$, the corresponding target region in the heat map as $\mathbf{R}_G$. The probability of distracter occurring in background is evaluated by the proportion between the confidence values outside and inside the target region

$$P_d = \frac{\sum_{j\in\hat{\mathbf{M}}_G-\mathbf{R}_G} \hat{\mathbf{M}}_G(j)}{\sum_{k\in\mathbf{R}_G} \hat{\mathbf{M}}_G(k)}, \tag{9}$$

where $\hat{\mathbf{M}}_G - \mathbf{R}_G$ represents the background region on heat map $\hat{\mathbf{M}}_G$. When the proportion P_d is less than a threshold (0.2 in all the experiments), we assume no co-occurring distracter and use the target location predicted by GNet as the final result. Otherwise, the same target localization procedure described above is performed on the heat map $\hat{\mathbf{M}}_S$ predicted by SNet to determine the final target location.

4.3. Online Update

To avoid the background noise introduced by online update, we fix GNet and only update SNet after the initialization in the first frame. SNet is updated following two different rules: the adaptation rule and the discrimination rule, which aim to adapt SNet to target appearance variation and improve the discriminative power for foreground and background, respectively. According to the adaptation rule, we finetune SNet every 20 frames using the most confident tracking result within the intervening frames. Based on the discrimination rule, when distracters are detected using (9), SNet is further updated using the tracking results in the first frame and the current frame by minimizing

$$\begin{aligned} \min \beta\|\mathbf{W}_S\|_F^2 + \sum_{x,y} \Big\{ &\left[\hat{\mathbf{M}}_S^1(x,y) - \mathbf{M}^1(x,y)\right]^2 \\ &+ \left[1 - \mathbf{\Phi}^t(x,y)\right] \left[\hat{\mathbf{M}}_S^t(x,y) - \mathbf{M}^t(x,y)\right]^2 \Big\}, \end{aligned} \tag{10}$$

where $\mathbf{W}_S$ denotes the convolutional weight of SNet; (x, y) are spatial coordinates; $\hat{\mathbf{M}}_S^t$ and $\mathbf{M}^t$ represent the heat map for the t-th frame predicted by SNet and the heat map

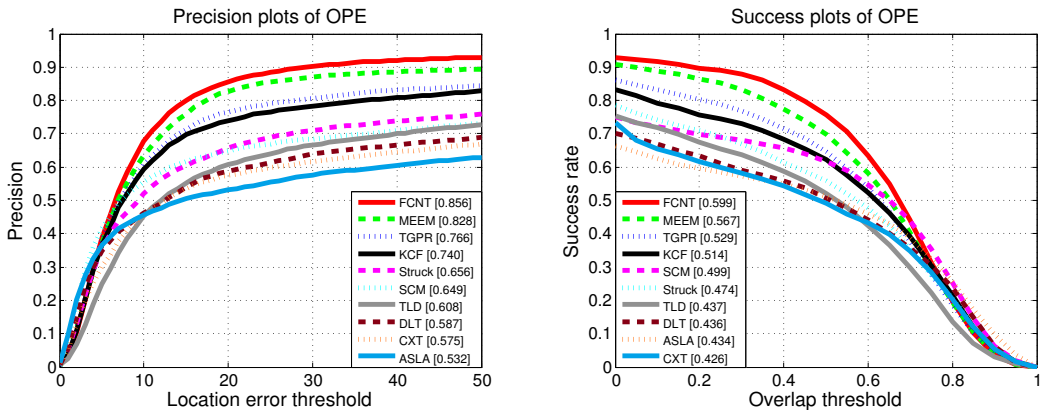


Figure 6. The precision plots and success plots of OPE for the top 10 trackers. The performance score for each tracker is shown in the legend. The performance score of precession plot is at error threshold of 20 pixels while the performance score of success plot is the AUC value.

Table 2. Average precision scores on different attributes: illumination variation (IV), out-of-plane rotation (OPR), scale variation (SV), occlusion (OCC), deformation (DEF), motion blur (MB), fast motion (FM), in-plane rotation (IPR), out-of-view (OV), background cluttered (BC) and low resolution (LR). The best and the second best results are in **red** and **green** colors, respectively.

	SCM	Struck	TLD	DLT	ASLA	CXT	MEEM	KCF	TGPR	FCNT	FCNT_G	FCNT_S	FCNT_{512}	FCNT_{256}	FCNT_{128}	FCNT_{64}
IV	0.594	0.558	0.537	0.534	0.517	0.501	0.778	0.728	0.687	**0.830**	0.776	0.731	**0.791**	0.758	0.760	0.787
OPR	0.618	0.597	0.596	0.561	0.518	0.574	**0.838**	0.729	0.741	**0.831**	0.820	0.775	0.789	0.779	0.758	0.751
SV	0.672	0.639	0.606	0.590	0.552	0.550	0.787	0.679	0.703	**0.830**	**0.806**	0.762	0.799	0.787	0.755	0.785
OCC	0.640	0.564	0.563	0.574	0.460	0.491	**0.801**	0.749	0.708	**0.797**	0.745	0.764	0.740	0.730	0.740	0.707
DEF	0.586	0.521	0.512	0.563	0.445	0.422	0.859	0.740	0.768	**0.917**	**0.917**	0.884	**0.919**	0.857	0.896	0.850
MB	0.339	0.551	0.518	0.453	0.278	0.509	0.740	0.650	0.578	**0.789**	0.666	0.737	0.734	0.712	**0.743**	0.729
FM	0.333	0.604	0.551	0.446	0.253	0.515	**0.757**	0.602	0.575	**0.767**	0.686	0.727	0.725	0.671	0.731	0.713
IPR	0.597	0.617	0.584	0.548	0.511	0.610	0.790	0.725	0.705	**0.811**	0.797	0.749	0.778	**0.824**	0.750	0.751
OV	0.429	0.539	0.576	0.444	0.333	0.510	**0.730**	0.650	0.576	**0.741**	0.636	0.641	0.648	0.520	0.650	0.554
BC	0.578	0.585	0.428	0.495	0.496	0.443	**0.807**	0.753	0.761	**0.799**	0.722	0.679	0.722	0.741	0.674	0.727
LR	0.305	0.545	0.349	0.396	0.156	0.371	0.494	0.381	0.539	**0.765**	0.577	0.633	0.747	0.740	**0.761**	0.750
Overall	0.649	0.656	0.608	0.587	0.532	0.575	**0.828**	0.740	0.766	**0.856**	0.794	0.801	0.824	0.817	0.800	0.798

generated according to the predicted target location (a 2-dimensional Gaussian centered at the target location), respectively; the foreground mask $\mathbf{\Phi}^t$ indicates the predicted target bounding box, *i.e.*, $\mathbf{\Phi}^t(x, y) = 1$ if the location (x, y) belongs to the target region and $\mathbf{\Phi}^t(x, y) = 0$, otherwise.

The second term in (10) corresponds to loss for locating the target in the first frame. When distracters appear or the target undergoes severe occlusion in the current frame, the estimated target region is not reliable for learning target appearance. Therefore, we choose a conservative scheme by adding the first frame to supervise update so that the learned model still captures the appearance in the first frame. Meanwhile, the third term in (10) removes the loss in the unreliable target region and only considers those within the background region in the current frame. It enforces the model to put more efforts on assigning the co-occurring distracters as background. The combination of the second term and the third term in (10) can help SNet to better separate the target from background and alleviate the model degradation caused by occlusion or distracters.

5. Experiments

Setup. The proposed FCNT tracker is implemented in MATLAB based on the wrapper of Caffe framework [14], and runs at 3 fps on a PC with a 3.4GHz CPU and a TITAN GPU. The source code is publicly available[2]. Both the sel-CNN for feature map selection and the GNet and SNet for target localization are trained in the first frame using back-propagation for 50 iterations. Afterwards, SNet are finetuned for 3 iterations at each update step. The learning rates are set to $1e-9$ for the sel-CNN and $1e-7$ for the GNet and SNet. The number of feature maps selected by the proposed feature selection method is set to $K = 384$ for both the conv4-3 and conv5-3 layers. The size of the input ROI region centered at target location is 386×386 pixels. The weight decay parameter β in (7) and (10) is set to 0.005. At each frame, 600 target candidates are randomly sampled. The variance of location parameters in (8) are set to $\{10, 10, 0.004\}$ for x, y translation and scale, respectively. All the parameters are fixed through the experiment.

Evaluation Methodology. We evaluate the proposed FCNT tracker on the benchmark data set [36] which includes 50 sequences and the results of 29 trackers. In addition, we also compare our method with three recent state-of-the-art methods MEEM [38], TGPR [7] and KCF [11] for a more thorough comparison. The sequences are further tagged with 11 attributes according to different challenging factors. We use the precision plot and the success plot to evaluate all the trackers. The precision plot demonstrates the per-

[2] http://ice.dlut.edu.cn/lu/index.html

Table 3. Average success scores on different attributes: illumination variation (IV), out-of-plane rotation (OPR), scale variation (SV), occlusion (OCC), deformation (DEF), motion blur (MB), fast motion (FM), in-plane rotation (IPR), out-of-view (OV), background cluttered (BC) and low resolution (LR).The best and the second best results are in **red** and **green** colors, respectively.

	SCM	Struck	TLD	DLT	ASLA	CXT	MEEM	KCF	TGPR	FCNT	$FCNT_G$	$FCNT_S$	$FCNT_{512}$	$FCNT_{256}$	$FCNT_{128}$	$FCNT_{64}$
IV	0.473	0.428	0.399	0.405	0.429	0.368	0.548	0.493	0.486	**0.598**	0.519	0.546	**0.565**	0.546	0.558	**0.565**
OPR	0.470	0.432	0.420	0.412	0.422	0.418	**0.562**	0.495	0.507	**0.581**	0.532	0.548	0.550	0.544	0.539	0.529
SV	0.518	0.425	0.421	0.455	0.452	0.389	0.503	0.427	0.443	**0.558**	0.507	0.527	**0.531**	0.525	0.515	0.519
OCC	0.487	0.413	0.402	0.423	0.376	0.372	**0.559**	0.514	0.494	**0.571**	0.491	0.557	0.528	0.519	0.540	0.515
DEF	0.448	0.393	0.378	0.394	0.372	0.324	0.582	0.534	0.556	**0.644**	0.605	0.638	**0.649**	0.618	0.643	0.614
MB	0.293	0.433	0.404	0.363	0.258	0.369	**0.565**	0.497	0.440	**0.580**	0.452	0.563	0.524	0.518	0.539	0.532
FM	0.296	0.462	0.417	0.360	0.247	0.388	**0.568**	0.459	0.441	**0.565**	0.496	0.545	0.523	0.493	0.534	0.513
IPR	0.458	0.444	0.416	0.411	0.425	0.452	0.526	0.497	0.487	**0.555**	0.510	0.519	0.532	**0.561**	0.520	0.520
OV	0.361	0.459	0.457	0.367	0.312	0.427	**0.597**	0.550	0.431	**0.592**	0.478	0.498	0.502	0.419	0.514	0.430
BC	0.450	0.458	0.345	0.339	0.408	0.338	**0.574**	0.535	0.543	**0.564**	0.494	0.510	0.514	0.527	0.498	0.517
LR	0.279	0.372	0.309	0.346	0.157	0.312	0.367	0.312	0.351	**0.514**	0.416	0.452	0.495	0.497	**0.520**	0.483
Overall	0.499	0.474	0.437	0.436	0.434	0.426	0.567	0.514	0.529	**0.599**	0.524	0.574	**0.575**	0.570	0.568	0.559

centage of frames where the distance between the predicted target location and the ground truth location is within a given threshold. All the trackers are ranked according to the precision scores at the threshold of 20 pixels. Whereas the success plot illustrates the percentage of frames where the overlap ratio between the predicted bounding box and the ground truth bounding box is higher than a threshold $\tau \in [0, 1]$. The area under curve (AUC) of each success plot are used to rank the tracking algorithms. We report the results of one pass evaluation (OPE) [36] for the proposed FCNT tracker and the top 10 algorithms in each plot, including MEEM [38], TGPR [7], KCF [11], SCM [40], Struck [10], TLD [15], DLT [35], ASLA [13] and CXT [5].

Results. The precision plot and the success plot of the compared trackers on 50 sequences are demonstrated in Figure 6. The proposed FCNT tracker outperforms all the other trackers in terms of both average precision score and success score. To facilitate better analysis on the tracking performance, we further evaluate all the trackers on sequences with 11 attributes. Table 2 and Table 3 demonstrate that the proposed FCNT can well handle a variety of challenging factors and consistently outperform state-of-the-art methods in almost all the challenges. To demonstrate the robustness of the features learned by DNNs from large scale image classification, we also compare with [19] which use a convolutional network for tracking without pre-training. On the 16 adopted test sequences, the proposed FCNT achieves a precision score of 0.88 and a success score of 0.85, whereas the reported results in [19] are 0.83 and 0.83, respectively. Our pre-trained network outperforms the method in [19] with a large margin. We also note that the proposed FCNT tracker has some failure cases in handling the low resolution (LR) challenge and achieves a relatively lower success score on the LR attribute (Table 3). One reason is that the VGG network is pre-trained on the Imagenet data set with training images of high resolutions.

Ablation Study. To further investigate the effectiveness of tracking by considering multiple layers and the proposed feature map selection method, we report the performance of different variants of the proposed algorithm in Table 2 and 3, where $FCNT_G$ and $FCNT_S$ denote the proposed method using only GNet and SNet for tracking; $FCNT_K$ represents the proposed method using K selected feature maps from both the cnov4-3 and conv5-3 layers of VGG network, and no feature map selection is conducted for $K = 512$. $FCNT_S$ exploits more discriminative feature and with proper online update it has higher performance than $FCNT_G$. By considering both the higher layer and the middle layer, the proposed FCNT can better deal with different challenging factors and outperform both $FCNT_G$ and $FCNT_S$ which only use the feature maps of a single layer. The proposed FCNT tracker uses less feature maps (386) achieves higher performance than $FCNT_{512}$. The $FCNT_{64}$ tracker uses only 1/8 of all the feature maps and can still compare favorably against state-of-the-art methods. This further demonstrate that the proposed feature map selection method can effectively remove unreliable and noisy feature maps and retain relevant ones, which effectively avoids overfitting and improves training convergence rate.

6. Conclusion

In this paper, we empirically present some important properties of CNN features under the viewpoint of visual tracking. Based on these properties, we propose a tracking algorithm using fully convolutional networks pre-trained on image classification task. We observe that convolutional layers at different levels have different properties. And we jointly consider these properties in order to capture the semantic information of the target and discriminate the target from background distracters. We further develop a principled feature map selection method to select discriminative features and discard noisy or unrelated ones. Our approach is shown to effectively improve the tracking performance on challenging scenarios.

Acknowledgements. This work is supported by the Natural Science Foundation of China (NSFC) #61472060, the Fundamental Research Funds for the Central Universities under Grant DUT14YQ101, Hong Kong Innovation and Technology Support Programme (ITS/221/13FP), and the Research Grants Council of Hong Kong (CUHK 417011, CUHK 419412, CUHK 14207814).

References

[1] P. Agrawal, R. B. Girshick, and J. Malik. Analyzing the performance of multilayer neural networks for object recognition. In *ECCV*, 2014. 1

[2] B. Babenko, M.-H. Yang, and S. Belongie. Robust object tracking with online multiple instance learning. *TPAMI*, 33(8):1619–1632, 2011. 2

[3] C. Bao, Y. Wu, H. Ling, and H. Ji. Real time robust l1 tracker using accelerated proximal gradient approach. In *CVPR*, 2012. 2

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 1, 3

[5] T. B. Dinh, N. Vo, and G. Medioni. Context tracker: Exploring supporters and distracters in unconstrained environments. In *CVPR*, 2011. 8

[6] J. Fan, W. Xu, Y. Wu, and Y. Gong. Human tracking using convolutional neural networks. *TNN*, 21(10):1610–1623, 2010. 2

[7] J. Gao, H. Ling, W. Hu, and J. Xing. Transfer learning based visual tracking with gaussian processes regression. In *ECCV*. 2014. 1, 7, 8

[8] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 1

[9] H. Grabner, C. Leistner, and H. Bischof. Semi-supervised on-line boosting for robust tracking. In *ECCV*, 2008. 2

[10] S. Hare, A. Saffari, and P. H. Torr. Struck: Structured output tracking with kernels. In *ICCV*, 2011. 2, 8

[11] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. *TPAMI*, 37(3):583–596, 2015. 2, 7, 8

[12] S. Hong, T. You, S. Kwak, and B. Han. Online tracking by learning discriminative saliency map with convolutional neural network. In *ICML*, 2015. 2

[13] X. Jia, H. Lu, and M.-H. Yang. Visual tracking via adaptive structural local sparse appearance model. In *CVPR*, 2012. 8

[14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM Multimedia*, pages 675–678, 2014. 7

[15] Z. Kalal, K. Mikolajczyk, and J. Matas. Tracking-learning-detection. *TPAMI*, 34(7):1409–1422, 2012. 8

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1

[18] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel. Optimal brain damage. In *NIPS*, 1989. 6

[19] H. Li, Y. Li, and F. M. Porikli. Robust online visual tracking with a single convolutional neural network. In *ACCV*, 2014. 2, 8

[20] X. Li, Z. Han, L. Wang, and H. Lu. Visual tracking via random walks on graph model. *IEEE Transactions on Cybernetics*, PP(99):1–1, 2015. 2

[21] X. Mei and H. Ling. Robust visual tracking using l1 minimization. In *CVPR*, 2009. 2

[22] W. Ouyang and X. Wang. Joint deep learning for pedestrian detection. In *ICCV*, 2013. 1

[23] W. Ouyang, X. Wang, X. Zeng, S. Qiu, P. Luo, Y. Tian, H. Li, S. Yang, Z. Wang, C.-C. Loy, et al. Deepid-net: Deformable deep convolutional neural networks for object detection. In *CVPR*, 2015. 1

[24] X. Ren and J. Malik. Tracking as repeated figure/ground segmentation. In *CVPR*, 2007. 2

[25] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang. Incremental learning for robust visual tracking. *IJCV*, 77(1-3):125–141, 2008. 2

[26] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2013. 3

[27] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 1, 2

[28] Y. Sun, X. Wang, and X. Tang. Deeply learned face representations are sparse, selective, and robust. *CoRR*, abs/1412.1265, 2014. 1

[29] D. Wang and H. Lu. Visual tracking via probability continuous outlier model. In *CVPR*, 2014. 1

[30] D. Wang, H. Lu, Z. Xiao, and M.-H. Yang. Inverse sparse tracker with a locally weighted distance metric. *TIP*, 24(9):2646–2657, 2015. 2

[31] D. Wang, H. Lu, and M. Yang. Robust visual tracking via least soft-threshold squares. *TCSVT*, PP(99):1–1, 2015. 2

[32] D. Wang, H. Lu, and M.-H. Yang. Online object tracking with sparse prototypes. *TIP*, 22(1):314–325, 2013. 2

[33] L. Wang, H. Lu, X. Ruan, and M.-H. Yang. Deep networks for saliency detection via local estimation and global search. In *CVPR*, 2015. 1

[34] L. Wang, H. Lu, and D. Wang. Visual tracking via structure constrained grouping. *Signal Processing Letters*, 22(7):794–798, 2015. 2

[35] N. Wang and D. Yeung. Learning a deep compact image representation for visual tracking. In *NIPS*, 2013. 2, 8

[36] Y. Wu, J. Lim, and M.-H. Yang. Online object tracking: A benchmark. In *CVPR*, 2013. 1, 2, 4, 7, 8

[37] F. Yang, H. Lu, and M.-H. Yang. Robust superpixel tracking. *TIP*, 23(4):1639–1651, 2014. 2

[38] J. Zhang, S. Ma, and S. Sclaroff. Meem: Robust tracking via multiple experts using entropy minimization. In *ECCV*. 2014. 1, 7, 8

[39] R. Zhao, W. Ouyang, H. Li, and X. Wang. Saliency detection by multi-context deep learning. In *CVPR*, 2015. 1

[40] W. Zhong, H. Lu, and M. Yang. Robust object tracking via a sparse collaborative appearance model. *TIP*, 23(5):2356–2368, 2014. 2, 8

[41] B. Zhuang, H. Lu, Z. Xiao, and D. Wang. Visual tracking via discriminative sparse similarity map. *TIP*, 23(4):1872–1881, 2014. 2