Adaptive Deconvolutional Networks for Mid and High Level Feature Learning

Matthew D. Zeiler, Graham W. Taylor and Rob Fergus
Dept. of Computer Science, Courant Institute, New York University
{zeiler,gwtaylor,fergus}@cs.nyu.edu

Abstract

We present a hierarchical model that learns image decompositions via alternating layers of convolutional sparse coding and max pooling. When trained on natural images, the layers of our model capture image information in a variety of forms: low-level edges, mid-level edge junctions, high-level object parts and complete objects. To build our model we rely on a novel inference scheme that ensures each layer reconstructs the input, rather than just the output of the layer directly beneath, as is common with existing hierarchical approaches. This makes it possible to learn multiple layers of representation and we show models with 4 layers, trained on images from the Caltech-101 and 256 datasets. When combined with a standard classifier, features extracted from these models outperform SIFT, as well as representations from other feature learning methods.

1. Introduction

For many tasks in vision, the critical problem is discovering good image representations. For example, the advent of local image descriptors such as SIFT and HOG has precipitated dramatic progress in matching and object recognition. Interestingly, many of the successful representations are quite similar [18], essentially involving the calculation of edge gradients, followed by some histogram or pooling operation. While this is effective at capturing low-level image structure, the challenge is to find representations appropriate for mid and high-level structures, i.e. corners, junctions, and object parts, which are surely important for understanding images.

In this paper we propose a way of learning image representations that capture structure at all scales, from low-level edges to high-level object parts, in an unsupervised manner. In building our model, we propose novel solutions to two fundamental problems associated with feature hierarchies. The first relates to invariance: while edges only vary in orientation and scale, larger-scale structures are more variable. Trying to explicitly record all possible shapes of t-junction or corners, for example, would lead to a model that is exponential in the number of primitives. Hence invariance is crucial for modeling mid and high-level structure.

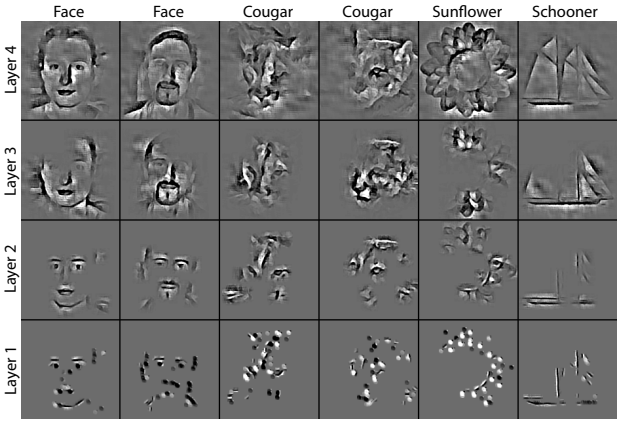

Figure 1. Top-down parts-based image decomposition with an adaptive deconvolutional network. Each column corresponds to a different input image under the same model. Row 1 shows a single activation of a 4^{th} layer feature map projected into image space. Conditional on the activations in the layer above, we also take a subset of 5, 25 and 125 active features in layers 3, 2 and 1 respectively and visualize them in image space (rows 2-4). The activations reveal mid and high level primitives learned by our model. In practice there are many more activations such that the complete set sharply reconstructs the *entire* image from each layer.

The second problem relates to the layer-by-layer training scheme employed in hierarchical models, such as deep belief networks [6, 11] and convolutional sparse coding [3, 8, 20]. Lacking a method to efficiently train all layers with respect to the input, these models are trained greedily from the bottom up, using the output of the previous layer as input for the next. The major drawback to this paradigm is that the image pixels are discarded after the first layer, thus higher layers of the model have an increasingly diluted connection to the input. This makes learning fragile and impractical for models beyond a few layers.

Our solution to both these issues is to introduce a set of latent switch variables, computed for each image, that locally adapt the model's filters to the observed data. Hence, a relatively simple model can capture wide variability in image structure. The switches also provide a direct path to the input, even from high layers in the model, allowing each layer to be trained with respect to the image, rather than the

2011 IEEE International Conference on Computer Vision 2018
978-1-4577-1102-2/11/$26.00

output of the previous layer. As we demonstrate, this makes learning far more robust. Additionally, the switches enable the use of an efficient training method, allowing us to learn models with many layers and hundreds of feature maps on thousands of images.

1.1. Related Work

Convolutional Networks (ConvNets) [10], like our approach, produce a hierarchy of latent feature maps via learned filters. However, they process images bottom-up and are trained discriminatively and purely supervised, while our approach is top-down (generative) and unsupervised. Predictive Sparse Decomposition (PSD) [7] adds a sparse coding component to ConvNets that allows unsupervised training. In contrast to our model, each layer only reconstructs the layer below. Additional shift invariance can be incorporated as in [12] by recording transformation parameters for use in reconstruction.

This limitation is shared by Deep Belief Networks (DBNs) [6, 11] which are comprised of layers of Restricted Boltzmann Machines. Each RBM layer, conditional on its input, has a factored representation that does not directly perform explaining away. Also, training is relatively slow.

The closest approaches to ours are those based on convolutional sparse coding [3, 8, 20]. Like PSD and DBNs, each layer only attempts to reconstruct the output of the layer below. Additionally, they manually impose sparse connectivity between the feature maps of different layers, thus limiting the complexity of the learned representation. In contrast, our model has full connectivity between layers, which allows us to learn more complex structures. Additional differences include: the lack of pooling layers [20] and inefficient inference schemes [3, 20] that do not scale.

Our model performs a decomposition of the full image, in the spirit of Zhu and Mumford [22] and Tu and Zhu [16]. This differs from other hierarchical models, such as Fidler and Leonardis [4] and Zhu *et al.* [21], that only model a stable sub-set of image structures at each level rather than all pixels. Another key aspect of our approach is that we learn the decomposition from natural images. Several other hierarchical models such as Serre *et al.*'s HMax [13, 15] and Guo *et al.* [5] use hand-crafted features at each layer.

2. Approach

Our model produces an over-complete image representation that can be used as input to standard object classifiers. Unlike many image representations, ours is learned from natural images and, given a new image, requires inference to compute. The model decomposes an image in a hierarchical fashion using multiple alternating layers of convolutional sparse coding (deconvolution [20]) and max-pooling. Each of the deconvolution layers attempts to directly minimize the reconstruction error of the input image under a sparsity constraint on an over-complete set of feature maps.

The cost function $C_l(y)$ for layer l comprises two terms: (i) a likelihood term that keeps the reconstruction of the input $\hat{y}_l$ close to the original input image y; (ii) a regularization term that penalizes the ℓ_1 norm of the 2D feature maps $z_{k,l}$ on which the reconstruction $\hat{y}_l$ depends. The relative weighting of the two terms is controlled by λ_l:

$$C_l(y) = \frac{\lambda_l}{2}\|\hat{y}_l - y\|_2^2 + \sum_{k=1}^{K_l} |z_{k,l}|_1 \qquad (1)$$

Unlike existing approaches [3, 8, 20], our convolutional sparse coding layers attempt to directly minimize the reconstruction error of the input image, rather than the output of the layer below.

Deconvolution: Consider the first layer of the model, as shown in Fig. 2(a). The reconstruction $\hat{y}_1$ (comprised of c color channels) is formed by convolving each of the 2D feature maps $z_{k,1}$ with filters $f_{k,1}^c$ and summing them:

$$\hat{y}_1^c = \sum_{k=1}^{K_1} z_{k,1} * f_{k,1}^c \qquad (2)$$

where $*$ is the 2D convolution operator. The filters f are the parameters of the model common to all images. The feature maps z are latent variables, specific to each image. Since $K_1 > 1$ the model is over-complete, but the regularization term in Eqn. 1 above ensures that there is a unique solution. We describe the inference scheme used to discover an optimal z_1 and the closely related learning approach for estimating f_1 in Sections 2.1 and 2.2 respectively. For notational brevity, we combine the convolution and summing operations of layer l into a single convolution matrix F_l and convert the multiple 2D maps $z_{k,l}$ into a single vector z_l:

$$\hat{y}_1 = F_1 z_1 \qquad (3)$$

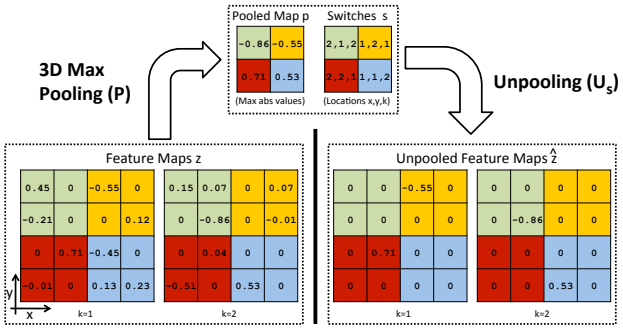

Figure 3. An example of 3D max pooling using a neighborhood of size $2 \times 2 \times 2$, as indicated by the colors. The pooling operation P is applied to the feature maps z, yielding pooled maps p and switches s that record the location of the maximum (irrespective of sign). Given the pooled maps and switches, we can also perform an unpooling operation U_s which inserts the pooled values in the appropriate locations in the feature maps, with the remaining elements being set to zero.

Pooling: On top of each deconvolutional layer, we perform a 3D max-pooling operation on the feature maps z.

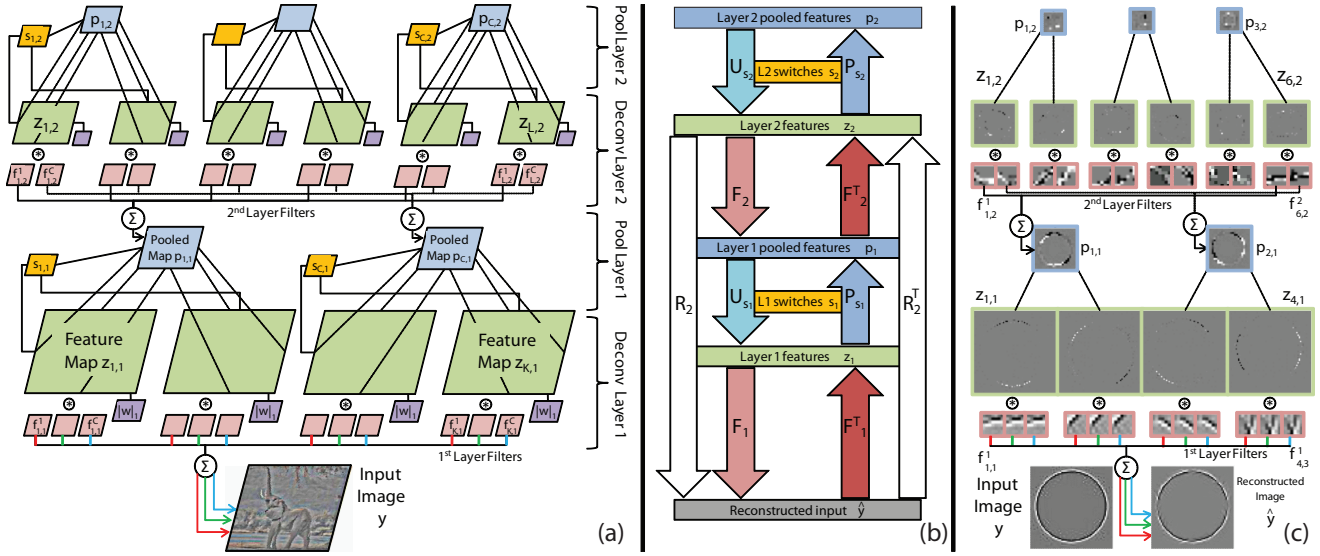

Figure 2. **(a):** A visualization of two layers of our model. Each layer consists of a deconvolution and a max-pooling. The deconvolution layer is a convolutional form of sparse coding that decomposes input image y into feature maps z_1 (green) and learned filters f_1 (red), which convolve together and sum to reconstruct y. The filters have c planes, each used to reconstruct a different channel of the input image. Each z map is penalized by a per-element ℓ_1 sparsity term (purple). The max-pooling layer pools within and between feature maps, reducing them in size and number to give pooled maps p (blue). The locations of the maxima in each pooling region are recorded in switches s (yellow). The second deconvolution/pooling layer is conceptually identical to the first, but now has two input channels rather than three. In practice, we have many more feature maps per layer and have up to 4 layers in total. **(b):** A block diagram view of the inference operations within the model for layer 2. See Section 2.1 for an explanation. **(c):** A toy instantiation of the model on the left, trained using a single input image of a (contrast-normalized) circle. The switches and sparsity terms are not shown. Note the sparse feature maps (green) and the effect of the pooling operations (blue). Since the input is grayscale, the planes of the 1st layer filters are identical.

This allows the feature maps of the layer above to capture structure at a larger scale than the current layer. The pooling is 3D in that it occurs both spatially (within each 2D z map) and also between adjacent maps, as shown in Fig. 3. Within each neighborhood of z we record both the value and location of the maximum (irrespective of sign). *Pooled maps* p store the values, while *switches* s record the locations.

Our model uses two distinct forms of pooling operation on the feature maps z. The first, shown in Fig. 3, treats the switches s as an output: $[p, s] = P(z)$. The second takes the switches s as an input, where they specify which elements in z are copied into p. If s is fixed, then this is a linear operation which can be written as $p = P_s z$, with P_s being a binary selection matrix, set by switches s.

The corresponding unpooling operation U_s, shown in Fig. 3, takes the elements in p and places them in z at the locations specified by s, the remaining elements being set to zero: $\hat{z} = U_s p$. Note that this is also a linear operation for fixed s and that $U_s = P_s^T$.

Multiple Layers: The architecture remains the same for higher layers in the model but the number of feature maps K_l may vary. At each layer we reconstruct the input through the filters and switches of the layers below. We define a *reconstruction* operator R_l that takes feature maps z_l from layer l and alternately convolves (F) and unpools them (U_s) down to the input:

$$\hat{y}_l = F_1 U_{s_1} F_2 U_{s_2} \ldots F_l z_l = R_l z_l \tag{4}$$

Note that $\hat{y}_l$ depends on the feature maps z_l from the current layer but not those beneath[1]. However, the reconstruction operator R_l does depend on the pooling switches in the intermediate layers ($s_{l-1} \ldots s_1$) since they determine the unpooling operations $U_{s_{l-1}} \ldots U_{s_1}$. These switches are configured by the values of $z_{l-1} \ldots z_1$ from previous iterations.

We also define a *projection* operator R_l^T that takes a signal at the input and projects it back up to the feature maps of layer l, given previously determined switches $s_1 \ldots s_{l-1}$:

$$R_l^T = F_l^T P_{s_{l-1}} F_{l-1}^T P_{s_{l-2}} \ldots P_{s_1} F_1^T \tag{5}$$

A crucial property of our model is that *given the switches* s, both the reconstruction R_l and projection operators R_l^T are linear, thus allowing the gradients to be easily computed, even in models with many layers, making inference and learning straightforward. Fig. 2(a) illustrates two layers of deconvolution and pooling within our model. Fig. 2(b) shows how the reconstruction and projection operators are made up of the filtering, pooling and unpooling operations.

[1] In other words, when we project down to the image, we do not impose sparsity on any of the intermediate layer reconstructions $\hat{z}_{l-1}, \ldots, \hat{z}_1$

2.1. Inference

For a given layer l, inference involves finding the feature maps z_l that minimize $C_l(y)$, given an input image y and filters f. For each layer we need to solve a large ℓ_1 convolutional sparse coding problem and we adapt the ISTA scheme of Beck and Teboulle [1]. This uses an iterative framework of gradient and shrinkage steps.

Gradient step: This involves taking a step in the direction of the gradient g_l of the reconstruction term of Eqn. 1, with respect to z_l: $g_l = R_l^T(R_l z_l - y)$.

To compute the gradient, we take feature maps z_l and, using the filters and switch settings of the layers below, reconstruct the input $\hat{y} = R_l z_l$. We then compute the reconstruction error $\hat{y} - y$. This is then propagated back up the network using R_l^T which alternately filters (F^T) and pools it (P_s) up to layer l, yielding the gradient g_l. This process is visualized in Fig. 2(middle) for a two layer model.

Once we have the gradient g_l, we then can update z_l:

$$z_l = z_l - \lambda_l \beta_l g_l \tag{6}$$

where the β_l parameter sets the size of the gradient step.

Shrinkage step: Following the gradient step, we perform a per-element shrinkage operation that clamps small elements in z_l to zero, thus increasing its sparsity:

$$z_l = \max(|z_l| - \beta_l, 0)\text{sign}(z_l) \tag{7}$$

Pooling/unpooling: We then update the switches s_l of the current layer by performing a pooling operation[2] $[p_l, s_l] = P(z_l)$, immediately followed by an unpooling operation $z_l = U_{s_l} p_l$. This fulfills two functions: (i) it ensures that we can accurately reconstruct the input through the pooling operation, when building additional layers on top and (ii) it updates the switches to reflect the revised values of the feature maps. Once inference has converged, the switches will be fixed, ready for training the layer above. Hence, a secondary goal of inference is to determine the optimal switch settings in the current layer.

Overall iteration: A single ISTA iteration consists of each of the three steps above: gradient, shrinkage and pooling/unpooling. During inference we perform 10 ISTA iterations per image for each layer.

Both the reconstruction R and propagation R^T operations are very quick, just consisting of convolutions, summations, pooling and unpooling operations, all of which are amenable to parallelization. This makes it possible to efficiently solve the system in Eqn. 1, even with massively over-complete layers where z_l may be up to 10^5 in length.

Note that while the gradient step is linear, the model as a whole is not. The non-linearity arises from two sources: (i) sparsity, as induced by the shrinkage Eqn. 7, and (ii) the settings of the switches s which alter the pooling/unpooling within R_l.

[2] This is the form of pooling shown in Fig. 3 that treats the switches as an output. It is not the same as the form of pooling used in projection operator R^T, where they are an input

Algorithm 1 Adaptive Deconvolutional Networks

Require: Training set Y, # layers L, # epochs E, # ISTA steps T
Require: Regularization weights λ_l, # feature maps K_l
Require: Shrinkage parameters β_l
1: **for** $l = 1 : L$ **do** %% Loop over layers
2: Init. features/filters: $z_l^i \sim \mathcal{N}(0, \epsilon)$, $f_l \sim \mathcal{N}(0, \epsilon)$
3: **for** epoch $= 1 : E$ **do** %% Epoch iteration
4: **for** $i = 1 : N$ **do** %% Loop over images
5: **for** $t = 1 : T$ **do** %% ISTA iteration
6: Reconstruct input: $\hat{y_l}^i = R_l z_l^i$
7: Compute reconstruction error: $e = \hat{y_l}^i - y^i$
8: Propagate error up to layer l: $g_l = R_l^T e$
9: Take gradient step: $z_l^i = z_l^i - \lambda_l \beta_l g_l$
10: Perform shrinkage: $z_l^i = \max(|z_l^i| - \beta_l, 0)\text{sign}(z_l^i)$
11: Pool z_l^i, updating the switches s_l^i: $[p_l^i, s_l^i] = P(z_l^i)$
12: Unpool p_l^i, using s_l^i to give z_l^i: $z_l^i = U_{s_l^i} p_l^i$
13: **end for**
14: **end for**
15: Update f_l by solving Eqn. 8 using CG
16: **end for**
17: **end for**
18: Output: filters f, feature maps z and switches s.

2.2. Learning

In learning the goal is to estimate the filters f in the model, which are shared across all images $Y = \{y^1, \ldots y^i, \ldots, y^N\}$. For a given layer l, we perform inference to compute z_l^i. Taking derivatives of Eqn. 1 with respect to f_l and setting to zero, we obtain the following linear system in f_l[3]:

$$\sum_{i=1}^{N} \left({z_l^i}^T P_{s_{l-1}}^i {R^i}_{l-1}^T \right) \hat{y}_l^i = \sum_{i=1}^{N} \left({z_l^i}^T P_{s_{l-1}}^i {R^i}_{l-1}^T \right) y^i \tag{8}$$

where $\hat{y}^i$ is the reconstruction of the input using the current value of f_l. We solve this system using linear conjugate gradients (CG). The matrix-vector product of the left-hand side is computed efficiently by mapping down to the input and back up again using the R_l and R_l^T operations. After solving Eqn. 8, we normalize f_l to have unit length.The overall algorithm for learning all layers of the model is given in Algorithm 1. We alternate small steps in z_l and f_l, using a single ISTA step per epoch to infer z_l and two CG iterations for f_l, repeated over 10 epochs. The procedure for inference is identical, except the f_l update on line 15 is not performed and we use a single epoch with 10 ISTA steps.

3. Application to object recognition

Our model is purely unsupervised and so must be combined with a classifier to perform object recognition. In view of its simplicity and performance, we use the Spatial Pyramid Matching (SPM) of Lazebnik *et al.* [9].

Given a new image, performing inference with our model decomposes it into multiple layers of feature maps

[3] f_l appears in the reconstruction: $\hat{y}_l = R_l z_l = R_{l-1} U_{s_{l-1}} F_l z_l$.

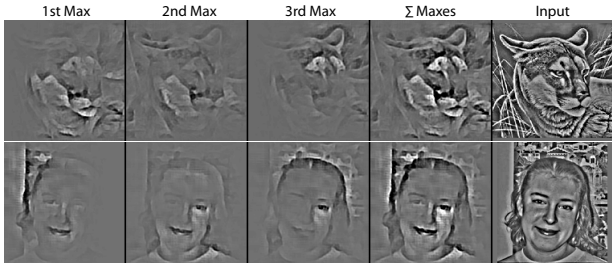

Figure 4. Col 1–3: The largest 3 absolute activations in the 4th layer projected down to pixel space for 2 different images. Note how distinct structures are reconstructed, despite the model being entirely unsupervised. See Section 3 for details on their use for recognition. Col 4–5: sum of first 3 columns; original input image.

and switch configurations. We now describe a novel approach for using this decomposition in conjunction with the SPM classifier.

While the filters are shared between images, the switch settings are not, thus the feature maps of two images are not directly comparable since they use different bases R_l. For example, in Fig. 1 each 4th layer top-down decomposition begins from the same feature map, yet gives quite different reconstructions. This shows two key aspects of the model: (i) within a class or between similar classes, the decompositions share similar parts and focus on particular regions of the image; and (ii) the adaptability of the switch settings allows the model to learn complex interactions with other classes. However, this makes direct use of the higher-level feature maps problematic for classification and we propose a different approach.

For each image i, we take the set of the M largest absolute activations from the top layer feature maps and project each one *separately* down to the input to create M different images $(\hat{y}^{i,1}, \ldots, \hat{y}^{i,M})$, each containing various image parts generated by our model. This only makes sense for high layers with large receptive fields. In Fig. 4 we show the pixel space reconstructions of the top $M = 3$ 4th layer activations inferred for 2 different images. Note how they contain good reconstructions of select image structures, as extracted by the model, while neighboring content is suppressed, providing a soft decomposition. For example, the 2nd max for the face reconstructs the left eye, mouth, and left shoulder, but little else. Conversely, the 3rd max focuses on reconstructing the hair. The structures within each max reconstruction consist of textured regions (e.g. shading of the cougar), as well as edge structures. They also tend to reconstruct the object better than the background.

Instead of directly inputting $\hat{y}^{i,1}, \ldots, \hat{y}^{i,M}$ to the SPM, we instead use the corresponding reconstructions of the 1st layer feature maps (i.e. $\hat{z}_1^{i,1}, \ldots, \hat{z}_1^{i,M}$), since activations at this layer are roughly equivalent to unnormalized SIFT features (the standard SPM input [9]). After computing separate pyramids for each $\hat{z}_1^{i,m}$, we average all M of them to give a single pyramid for each image. We can also apply

Property	Layer 1	Layer 2	Layer 3	Layer 4
# Feature maps K_l	15	50	100	150
Pooling size	3x3x3	3x3x2	3x3x2	3x3x2
λ_l	2	0.1	0.005	0.001
β_l	10^{-3}	10^{-4}	10^{-6}	10^{-8}
CPU Inference time	0.12 s	0.21 s	0.38 s	0.54 s
GPU Inference time	0.006 s	0.01 s	0.03 s	0.05 s
z pixel field	7x7	21x21	63x63	189x189
Feature map dims	156x156	58x58	26x26	15x15
# Filter Params	735	7,350	122,500	367,500
Total # z & s	378,560	178,200	71,650	37,500

Table 1. Parameter settings (top 4 rows) and statistics (lower 5 rows) of our model.

SPM to the actual 1st layer feature maps z_1^i, which are far denser and have even coverage of the image[4]. The pyramids of the two can be combined to boost performance.

4. Experiments

We train our model on the entire training set of 3060 images from the Caltech-101 dataset (30 images per class).

Pre-processing: Each image is converted to gray-scale and resized to 150×150 (zero padding to preserve the aspect ratio). Local subtractive and divisive normalization (i.e. the patch around each pixel should have zero mean and unit norm) is applied using a 13×13 Gaussian filter with $\sigma = 5$.

Model architecture: We use a 4 layer model, with 7×7 filters, and $E = 10$ epochs of training. Various parameters, timings and statistics are shown in Table 1. Due to the efficient inference scheme, we are able to train with many more feature maps and more data than other approaches, such as [3, 8, 11]. By the 4th layer, the receptive field of each feature map element (z pixel field) covers the entire image, making it suitable for the novel feature extraction process described in Section 3. At lower layers of the model, the representation has many latent variables (i.e. z's and s's) but as we ascend, the number drops. Counterbalancing this trend, the number of filter parameters grows dramatically as we ascend and the top layers of the model are able to learn object specific structures.

Timings: With 3060 training images and $E = 10$ epochs, it takes around 5 hours to train the entire 4 layer model using a MATLAB implementation on a six-core CPU. For inference, a single epoch suffices with 10 ISTA iterations at each layer. The total inference time per image is 1.25 secs (see Table 1 for each layer). Additionally, the algorithm can easily be parallelized, thus when using a Nvidia GTX 480 GPU these timings improve to 30 mins for training the entire model and 0.1 secs to infer each image.

4.1. Model visualization

The top-down nature of our model makes it easy to inspect what it has learned. In Fig. 5 we visualize the filters in the model by taking each feature map separately and

[4] Specific details: pixel spacing=2, patch size=16, codebook size=2000.

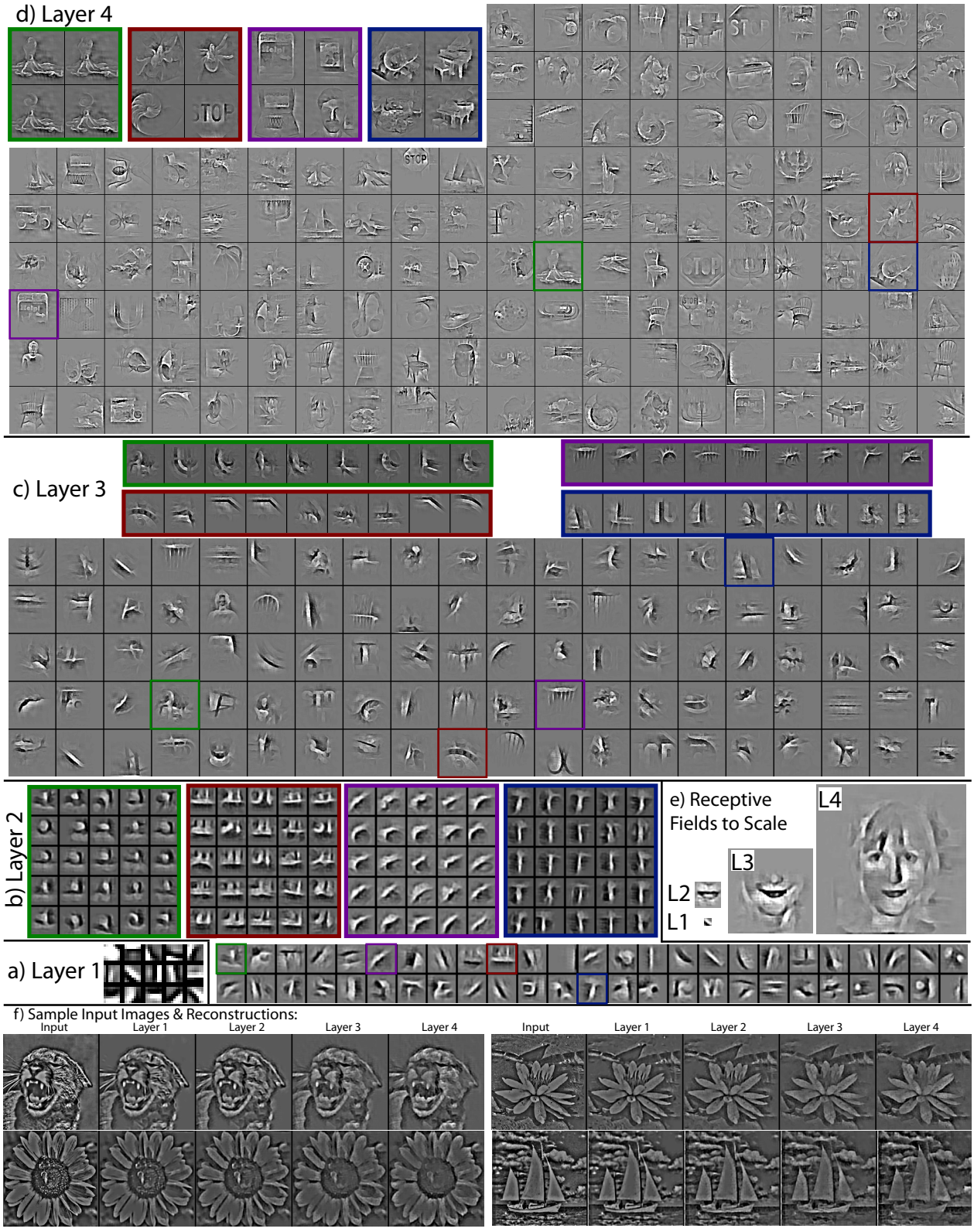

Figure 5. a-d) Visualizations of the filters learned in each layer of our model with zoom-ins showing the variability of select features. e) An illustration of the relative receptive field sizes. f) Image reconstructions for each layer. See Section 4.1 for explanation. This figure is best viewed in electronic form.

Our model - layer 1	$67.8 \pm 1.2\%$
Our model - layer 4	$69.8 \pm 1.2\%$
Our model - layer 1 + 4	$\mathbf{71.0 \pm 1.0\%}$
Chen *et al.* [3] layer-1+2 (ConvFA)	$65.7 \pm 0.7\%$
Kavukcuoglu *et al.* [8] (ConvSC)	$65.7 \pm 0.7\%$
Zeiler *et al.* [20] layer-1+2 (DN)	$66.9 \pm 1.1\%$
Boureau *et al.* [2] (Macrofeatures)	$70.9 \pm 1.0\%$
Jarrett *et al.* [7] (PSD)	$65.6 \pm 1.0\%$
Lazebnik *et al.* [9] (SPM)	$64.6 \pm 0.7\%$
Lee *et al.* [11] layer-1+2 (CDBN)	$65.4 \pm 0.5\%$
Our Caltech-256 Model - layer 1+4	$\mathbf{70.5 \pm 1.1\%}$

Table 2. Recognition performance on Caltech-101 compared to other approaches grouped by similarity (from top). Group 1: our approach; Group 2: related convolutional sparse coding methods with SPM classifier; Group 3: other methods using SPM classifier; group 4: our model with filters trained on Caltech-256 images.

picking the single largest absolute activation over the entire training set. Using the switch settings particular to that activation we project it down to the input pixel space. At layer 1 (Fig. 5(a)), we see a range of oriented Gabors of differing frequencies and some DC filters. In layer 2 (Fig. 5(b)), a range of edge junctions and curves can be seen, built from combinations of the 1st layer filters. For select filters (highlighted in color), we expand to show the 25 strongest activations across all images. Each group shows clustering with a certain degree of variation produced by the specific switch settings for that particular activation. See, for example, the sliding configuration of the T-junction (blue box). Reflecting their large receptive field, the filters in layer 3 (Fig. 5(c)) show a range of complex compositions. The highlighted boxes show that the model is able to cluster quite complex structures. Note that the groupings produced are quite different to a pixel-space clustering of image patches since they are: (i) far from rectangular in shape; (ii) utilize the adaptable geometric transformations offered by the switches below. The 4th layer filters (Fig. 5(d)) show fairly complete reconstructions of entire objects with groupings amongst objects of the same class or of similar shape.

To understand the relative sizes of each projection we also show the receptive fields for layers 1-4 in Fig. 5(e). Finally, reconstructions from each layer of the model of 4 example input images are shown in Fig. 5(f). Note that unlike related models, such as Lee *et al.* [11], sharp image edges are preserved in the reconstructions, even from layer 4.

4.2. Evaluation on Caltech-101

We use $M = 50$ decompositions from our model to produce input for training the Spatial Pyramid Match (SPM) classifier of Lazebnik *et al.* [9]. The classification results on the Caltech-101 test set are shown in Table 2.[5]

Applying the SPM classifier to layer 1 features z_1 from our model produces similar results (67.8%) to many other approaches, including those using convolutional sparse coding (2nd group in Table 2). However, using the 50 max decompositions from layer 4 in the SPM classifier, as detailed in Section 3, we obtain a significant performance improvement of 2%, surpassing the majority of hierarchical and sparse coding approaches that also use the same SPM classifier (middle two groups in Table 2). Summing the SVM kernels resulting from the max activations from layer 4 and the layer 1 features, we achieve 71.0%. The only approach based on the SPM with comparable performance is that of Boureau *et al.* [2], based on Macrofeatures. Current state-of-the-art techniques [2, 19, 17] use forms of soft quantization of the descriptors, in place of the hard k-means quantization used in the SPM (e.g. Wang *et al.* [17] obtain 73.4% using a sparse-coding based classifier).

[5] In Table 2 and Table 3, we only consider approaches based on a single feature type. Approaches that combine hundreds of different features with multiple kernel learning methods outperform the methods listed.

Our model - layer 1	$31.2 \pm 1.0\%$
Our model - layer 4	$30.1 \pm 0.9\%$
Our model - layer 1 + 4	$\mathbf{33.2 \pm 0.8\%}$
Yang *et al.* [19] (SPM)	$29.5 \pm 0.5\%$
Our Caltech-101 Model - layer 1+4	$\mathbf{33.9 \pm 1.1\%}$

Table 3. Caltech-256 recognition performance of our model and a similar SPM method. Our Caltech-101 model was also evaluated.

4.3. Evaluation on Caltech-256

Using the same training and evaluation parameters as for Caltech-101, we evaluated our model on the more difficult Caltech-256 dataset (see Table 3). By training our model on 30 images in each of the 256 categories and using $M = 50$ decompositions as before, we show a gain of 3.7% over SIFT features with the SPM classifier [19].

4.4. Transfer learning

By using filters trained on Caltech-101, then classifying Caltech-256 and vice versa we can test how well our model generalizes to new images. In both cases, classification performance remains within errors of the original results (see Table 2 and Table 3) showing the adaptability of our model to generalize to new instances and entirely new classes.

4.5. Classification and reconstruction relationship

While we have shown sparsity to be useful for learning high level features and decomposing images into a hierarchy of parts, it is not necessarily a strong cue for classification as Rigamonti *et al.* [14] note. By varying the λ_l pa-

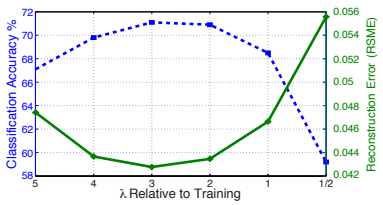

Figure 6. Relationship between reconstruction (solid line) and classification rates (dotted line) for varied amounts of sparsity.

rameter at each layer for inference, holding the filters fixed, we analyzed the tradeoff of sparsity and classification performance on Caltech-101 in Fig. 6. With higher λ_l values, sparsity is reduced, reconstructions improve, and recognition rates increase. The optimum appears around $3\times$ the λ_l used for training, after which the increased λ_l results in larger ISTA steps that introduce instability in optimization.[6]

4.6. Analysis of switch settings

Other deep models lack explicit pooling switches, thus during reconstruction either place a single activation in the center of each pool [3], or distribute it equally over all locations [11]. Fig. 7 demonstrates the utility of switches: we (i) reconstruct using the top 25 activations in layers 2,3 and 4 for different forms of switch behavior; (ii) sum the resulting reconstructions and (iii) classify Caltech-101 using the layer 1 features (as before).

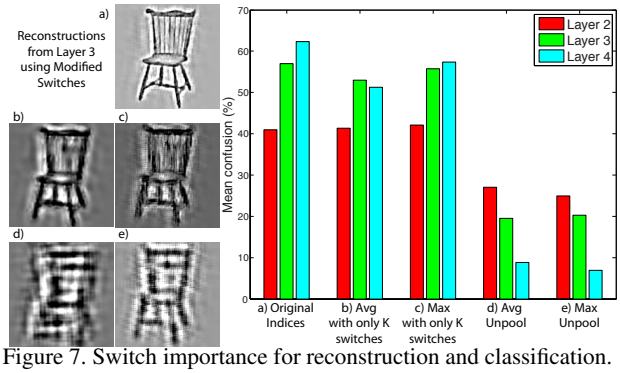

Figure 7. Switch importance for reconstruction and classification.

It is evident from the layer 3 reconstructions shown in Fig. 7(a) that retaining all max locations allows for sharp reconstructions while average unpooling Fig. 7(b,d) causes blur and using the center indices in max unpooling Fig. 7(c,e) causes jitter with corresponding drops in recognition performance When reconstruction, maintaining the proper k switches Fig. 7(b,c) is crucial for selecting the proper feature maps in lower layers, so preventing extreme deformations of the objects (see Fig. 7(d,e)) which leads to severely reduced recognition performance.

5. Discussion

The novel methods introduced in this paper allow us to reliably learn models with many layers. As we ascend the layers, the switches in our model allow the filters to adapt to increasingly variable input patterns. The model is thus able to capture mid and high-level features that generalize between classes. Using these features with standard classifiers gives highly competitive rates on Caltech-101 and Caltech-256. The generality of our learned representation is demonstrated by its ability to generalize to datasets on which it was not trained, while maintaining a comparable performance. Matlab code for our algorithm is available at www.matthewzeiler.com/pubs/iccv2011/.

[6]The results in Table 2 and Table 3 used $2\times$ λ_l.

References

[1] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009. 4

[2] Y. Boureau, F. Bach, Y. LeCun, and J. Ponce. Learning mid-level features for recognition. In *CVPR*. IEEE, 2010. 7

[3] B. Chen, G. Sapiro, D. Dunson, and L. Carin. Deep learning with hierarichal convolutional factor analysis. *JMLR*, page Submitted, 2010. 1, 2, 5, 7, 8

[4] S. Fidler, M. Boben, and A. Leonardis. Similarity-based cross-layered hierarchical representation for object categorization. In *CVPR*, 2008. 2

[5] C. E. Guo, S. C. Zhu, and Y. N. Wu. Primal sketch: Integrating texture and structure. *CVIU*, 106:5–19, 2007. 2

[6] G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, 2006. 1, 2

[7] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *ICCV*, 2009. 2, 7

[8] K. Kavukcuoglu, P. Sermanet, Y. Boureau, K. Gregor, M. Mathieu, and Y. LeCun. Learning convolutional feature hierachies for visual recognition. In *NIPS*, 2010. 1, 2, 5, 7

[9] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, 2006. 4, 5, 7

[10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *IEEE*, 86(11):2278–24, 1998. 2

[11] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *ICML*, pages 609–616, 2009. 1, 2, 5, 7, 8

[12] M. Ranzato, F. Huang, Y. Boureau, and Y. LeCun. Unsupervised learning of invariant feature hierarchies with applications to object reocgnition. In *CVPR*, 2007. 2

[13] M. Riesenhuber and T. Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2(11):1019–1025, 1999. 2

[14] R. Rigamonti, M. Brown, and V. Lepetit. Is sparsity really relevant for image classification? Technical Report 152499, Ecole Polytechnique Federale de Lausanne (EPFL), 2010. 7

[15] T. Serre, L. Wolf, and T. Poggio. Object recognition with features inspired by visual cortex. In *CVPR*, 2005. 2

[16] Z. W. Tu and S. C. Zhu. Parsing images into regions, curves, and curve groups. *IJCV*, 69(2):223–249, August 2006. 2

[17] J. Wang, J. Yang, K. Yu, T. Huang, and Y. Gong. Locality-constrained linear coding for image classification. In *CVPR*, 2010. 7

[18] S. Winder, G. Hua, and M. Brown. Picking the best daisy. In *CVPR*, 2009. 1

[19] J. Yang, K. Yu, Y. Gong, and T. Huang. Linear spatial pyramid matching using sparse coding for image classification. In *CVPR*, 2009. 7

[20] M. Zeiler, D. Krishnan, G. Taylor, and R. Fergus. Deconvolutional networks. In *CVPR*, 2010. 1, 2, 7

[21] L. Zhu, Y. Chen, and A. L. Yuille. Learning a hierarchical deformable template for rapid deformable object parsing. *PAMI*, March 2009. 2

[22] S. Zhu and D. Mumford. A stochastic grammar of images. *Foundations and Trends in Comp. Graphics and Vision*, 2(4):259–362, 2006. 2