# UberNet: Training a Universal Convolutional Neural Network for Low-, Mid-, and High-Level Vision using Diverse Datasets and Limited Memory

Iasonas Kokkinos

University College London & Facebook Artificial Intelligence Research

i.kokkinos@cs.ucl.ac.uk

## Abstract

*In this work we train in an end-to-end manner a convolutional neural network (CNN) that jointly handles low-, mid-, and high-level vision tasks in a unified architecture. Such a network can act like a 'swiss knife' for vision tasks; we call it an "UberNet" to indicate its overarching nature.*

*The main contribution of this work consists in handling challenges that emerge when scaling up to many tasks. We introduce techniques that facilitate (i) training a deep architecture while relying on diverse training sets and (ii) training many (potentially unlimited) tasks with a limited memory budget.*

*This allows us to train in an end-to-end manner a unified CNN architecture that jointly handles (a) boundary detection (b) normal estimation (c) saliency estimation (d) semantic segmentation (e) human part segmentation (f) semantic boundary detection, (g) region proposal generation and object detection. We obtain competitive performance while jointly addressing all tasks in 0.7 seconds on a GPU. Our system will be made publicly available.*

## 1. Introduction

Computer vision involves a host of tasks, such as boundary detection, semantic segmentation, surface estimation, object detection, image classification, to name a few. While Convolutional Neural Networks (CNNs) [32] have been shown to be successful at effectively handling most vision tasks, in the current literature most works focus on individual tasks and devote all of a CNN's power to maximizing task-specific performance. In our understanding a joint treatment of multiple problems can result not only in simpler and faster models, but will also be a catalyst for reaching out to other fields. One can expect that such all-in-one, "swiss knife" architectures will become indispensable for general AI, involving, for instance, robots that will be able to recognize the scene they are in, identify objects, navigate towards them, and manipulate them.

The problem of using a single network to solve multiple tasks has been recently pursued in the context of deep
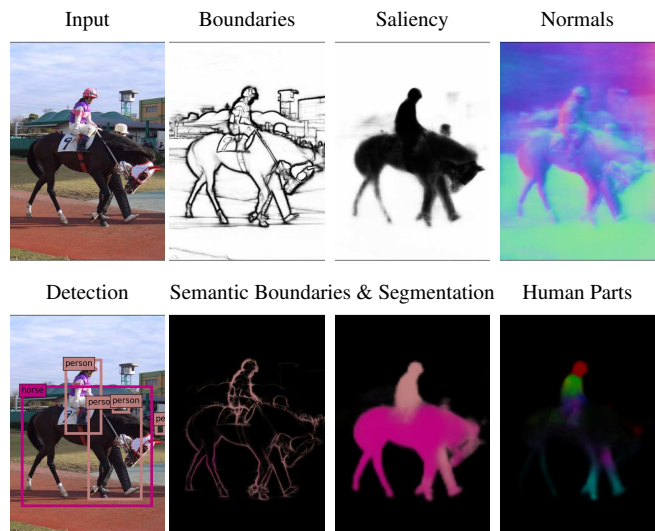


Figure 1: We train in an end-to-end manner a CNN that jointly performs tasks spanning low-, mid- and high- level vision; all results are obtained in 0.7 seconds per frame.

learning for computer vision. In [50] a CNN is used for joint localization, detection and classification, [17] propose a network that jointly solves surface normal estimation, depth estimation and semantic segmentation, while [20] train a system for joint detection, pose estimation and region proposal generation. More recently [41] study the effects of sharing information across networks trained for complementary tasks, [6] propose the introduction of inter-task connections that improves performance through task synergy and [47] propose an architecture for a host of face-related tasks.

Inspired by these works, in Sec. 2 we introduce a CNN architecture that jointly handles multiple tasks by using a shared trunk which feeds into many task-specific branches. Our contribution consists in introducing techniques that enable training to scale up to a large number of tasks.

Our first contribution enables us to train a CNN from *diverse datasets* that contain annotations for distinct tasks.

This problem emerges once we aim at breadth, since no single dataset contains ground truth for all possible tasks. As described in Sec. 3, we use a sample-dependent loss that only penalizes deviations from the ground truth available per training sample. We combine this loss function with Stochastic Gradient Descent, and propose an asynchronous variant of back-propagation that moves away from the idea of a fixed minibatch for all tasks, and instead updates task-specific network parameters only once we have observed sufficient training samples that pertain to that task. This allows us to perform end-to-end CNN training, while using the union of the diverse datasets as a single training set.

Our second contribution addresses the *limited memory* currently available on the Graphics Processing Units (GPUs) used for deep learning. As the number of tasks increases, the memory demands of back-propagation can increase linearly in the number of tasks, placing a limit on the number of tasks that one can handle. We build on recent developments in deep learning with low-memory complexity [12, 21] and develop an algorithm that allows us to perform end-to-end network training with a memory complexity that is *independent of the number of tasks*.

These techniques make it particularly easy to append tasks to a CNN, requiring only the specification of an additional dataset and a loss function per task. Our network is systematically evaluated on the following tasks: (a) boundary detection (b) normal estimation (c) saliency estimation (d) semantic segmentation (e) semantic part segmentation (f) semantic boundary detection and (g) proposal generation and object detection.

Our present, VGG-based [51] system operates in 0.7 seconds per frame on a Titan-X GPU and delivers results that are comparable, and often even better than the state-of-the-art on these tasks. This system is being extended to exploit more recent models, such as ResNet [24] and ResNeXt [55] architectures; code, models, and supplemental materials with more extensive evaluations will become available at `http://www0.cs.ucl.ac.uk/staff/I.Kokkinos/ubernet/`

## 2. UberNet architecture

In Fig. 2 we show the architecture of our network. As in [32, 39, 45, 50], we use a 'fully-convolutional' network, namely a CNN that provides a field of responses, rather than a single classification at its output. For all experiments reported in this paper we use the VGG-16 network [51] and pool features from layers conv1_2, conv2_2, conv3_3, conv4_3, conv5_3, fc7, which are shown as $\mathbf{C}_1, \ldots, \mathbf{C}_6$ in Fig. 2 (please consult the project website for results with more architectures). Modifying slightly [3, 38], we process these intermediate layers with batch normalization [25] so as to bring the intermediate neuron responses in a common range.

For every task we use separate skip layers [23,29,50,56]
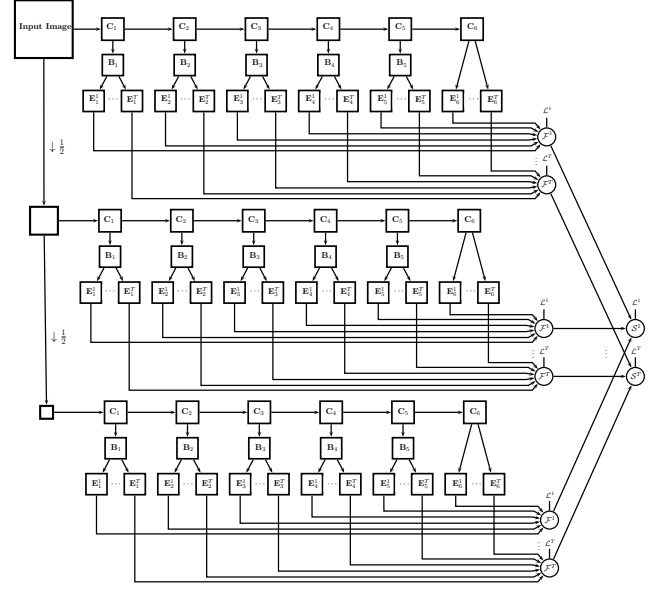


Figure 2: UberNet architecture: an image pyramid is formed by successive down-sampling operations, and each image is processed by a CNN with tied weights; the responses of the network at consecutive layers ($\mathbf{C}_i$) are processed with Batch Normalization ($\mathbf{B}_i$) and then fed to task-specific skip layers ($\mathbf{E}_i^t$); these are combined across network layers ($\mathcal{F}^t$) and resolutions ($\mathcal{S}^t$) and trained using task-specific loss functions ($\mathcal{L}^t$), while the whole architecture is jointly trained end-to-end. For simplicity we omit the interpolation and detection layers mentioned in the text.

that combine the responses of multiple intermediate layers to form the network output. As in [29,56] we keep the task-specific memory and computation budget low by applying linear operations within these skip layers, and fuse skip-layer results through additive fusion with learned weights. We leave for future work the inclusion of additional task-specific convolutional filters [18, 42, 43, 49] or structured prediction operations [8, 10, 30, 37, 57]. We approriately place interpolation layers to ensure that results from different skip layers have commensurate dimensions and use atrous convolution [10, 45] to evaluate high-level neurons more densely.

As in [11, 26, 29, 45] we construct an image pyramid and pass the multi-resolution versions of the image through CNNs with shared weights - the highest resolution image is set to have the smallest side equal to 621 pixels, as in [19], while the other layers are successively downsampled by a factor of two. This multi-resolution processing is incorporated in the network definition, allowing for end-to-end training. As in [11, 29] we use loss layers both at the outputs of the individual scales and the final responses.

For each task we use a task-specific loss, including the cross-entropy loss for discrete labelling tasks (semantic segmentation, human parts, saliency), the Multiple Instance Learning loss of [29] for (semantic) boundary detection, and a combination of $\ell_2$ normalization prior to an $\ell_1$ loss penalty for normal estimation. For object detection as in [48] we use a 'Region Proposal Network' and a 'Faster-RCNN' branch, omitted form the diagram for simplicity. We provide additional details on architecture alongside with an analysis of task-specific choices on the project website.

## 3. Multi-Task Training using Diverse Datasets

Having outlined our network architecture, we now turn to jointly training in an end-to-end manner the shared CNN trunk and the task-specific layers.

Even though earlier works on multi-task CNNs for vision [15, 17, 20] directly minimize the sum of task-specific losses with backpropagation, there is no dataset with common annotations for tasks as diverse as human part segmentation, normal estimation, and saliency estimation.

One approach to addressing this consists in imputing missing data by exploiting domain-specific knowledge, e.g. by using bounding box information to constrain semantic segmentation [14, 44]. This however may not be possible for arbitrary tasks, e.g. normal estimation.

Instead, we simply set to zero the loss of tasks for images that have no task-specific ground truth - which amounts to interleaving tasks, as in [7]. Our training objective is the sum of per-task losses and regularization terms applied to task-specific, as well as shared layers:

$$\mathcal{L}(\mathbf{w}_{0,1,\ldots,T}) = \mathcal{R}(\mathbf{w}_0) + \sum_{t=1}^{T} \gamma_t (\mathcal{R}(\mathbf{w}_t) + \mathcal{L}_t(\mathbf{w}_0, \mathbf{w}_t)), \quad (1)$$

where $t$ indexes tasks, $\mathbf{w}_0$ denotes shared CNN weights, $\mathbf{w}_t$ are task-specific weights, $\gamma_t$ determines the relative importance of task $t$, $\mathcal{R}(\mathbf{w}_*) = \frac{\lambda}{2}\|\mathbf{w}_*\|^2$ is an $\ell_2$ regularization, and $\mathcal{L}_t(\mathbf{w}_0, \mathbf{w}_t)$ is the task-specific loss:

$$\mathcal{L}_t(\mathbf{w}_0, \mathbf{w}_t) = \frac{1}{N} \sum_{i=1}^{N} \delta_{t,i} L_t\left(\mathbf{f}_t^i(\mathbf{w}_0, \mathbf{w}_t), \mathbf{y}_t^i\right). \quad (2)$$

In Eq. 2 we use $i$ to index training samples, $L_t$ for the resulting task-specific loss between the network prediction $\mathbf{f}_t^i$ and ground truth $\mathbf{y}_t^i$ for the $i$-th example, $\mathbf{w}_t$ to indicate the task-specific network parameters, and $\delta_{t,i} \in \{0,1\}$ to indicate whether example $i$ has ground truth for task $t$.

The advantage of using this objective is that it allows us to simply take the union of datasets constructed for different tasks and train a single network that jointly solves all tasks - setting $\delta_{t,i} = 0$ allows an image $i$ to 'not care' about any task $i$ for which it does not contain ground truth, ensuring that we only penalize the network's predictions for

---

Asynchronous SGD
$\mathbf{dw}_0 \leftarrow 0, \mathbf{dw}_1 \leftarrow 0, \ldots, \mathbf{dw}_T \leftarrow 0$
$\mathbf{c}_0 \leftarrow 0, \mathbf{c}_1 \leftarrow 0, \ldots, \mathbf{c}_T \leftarrow 0$
**for** $m = 1$ to $N \cdot$ #epochs **do**
   Sample $i \sim U[1,N]$
   {Pick tasks for which sample $i$ contains ground truth}
   $\mathcal{T} = \{0\} \cup \{t : \delta_{t,i} = 1\}$
   **for** $t \in \mathcal{T}$ **do**
      **if** $t = 0$ {Shared Parameters} **then**
         $\mathbf{gw}_0 = \sum_t \delta_{t,i} \gamma_t \nabla_{\mathbf{w}_0} L_t\left(\mathbf{f}_t^i(\mathbf{w}_0, \mathbf{w}_t), \mathbf{y}_t^i\right)$
      **else**
         $\mathbf{gw}_t = \gamma_t \nabla_{\mathbf{w}_t} L_t\left(\mathbf{f}_t^i(\mathbf{w}_0, \mathbf{w}_t), \mathbf{y}_t^i\right)$
      **end if**
      $\mathbf{c}_t \leftarrow \mathbf{c}_t + 1,$
      $\mathbf{dw}_t \leftarrow \mathbf{dw}_t + \mathbf{gw}_t$
      **if** $\mathbf{c}_t = B_t$ **then**
         {update parameters if we have seen enough}
         $\mathbf{w}_t \leftarrow \mathbf{w}_t - \epsilon\left(\lambda \mathbf{w}_t + \frac{1}{B_t}\mathbf{dw}_t\right)$
         $\mathbf{c}_t \leftarrow 0, \mathbf{dw}_t \leftarrow 0$
      **end if**
   **end for**
**end for**

Table 1: Pseudocode for asynchronous SGD: for any task we update its parameters only after observing sufficiently many samples that contain ground truth for it. Highlighted in blue are the main differences with respect to SGD.

tasks where we do have ground truth. The disadvantage is that Stochastic Gradient Descent (SGD) with standard minibatching can become problematic, as we now describe.

Considering that we use a minibatch $\mathcal{B}$ of size $B$, plain SGD for task $k$ would lead to the following update rules:

$$\mathbf{w}_p' = \mathbf{w}_p - \epsilon(\lambda \mathbf{w}_p + \mathbf{dw}_p), \quad p \in \{0, 1, \ldots, T\} \quad (3)$$

$$\mathbf{dw}_0 = \frac{1}{B} \sum_{i \in \mathcal{B}} \sum_{t=1}^{T} \gamma_t \delta_{t,i} \nabla_{\mathbf{w}_0} L_t\left(\mathbf{f}_t^i(\mathbf{w}_0, \mathbf{w}_t), \mathbf{y}_t^i\right), \quad (4)$$

$$\mathbf{dw}_t = \frac{1}{B} \sum_{i \in \mathcal{B}} \gamma_t \delta_{t,i} \nabla_{\mathbf{w}_t} L_t\left(\mathbf{f}_t^i(\mathbf{w}_0, \mathbf{w}_t), \mathbf{y}_t^i\right), \quad (5)$$

where the weight decay term results from $\ell_2$ regularization and $\nabla_{\mathbf{w}_*} L_t(\hat{y}, y)$ denotes the gradient of the loss for task $t$ with respect to the parameter vector $\mathbf{w}_*$. The difference between the two update terms is that the common, trunk parameters, $\mathbf{w}_0$ affect all tasks, and as such accumulate the gradients over all tasks. By contrast, task-specific parameters $\mathbf{w}_t$ are only affected by those images where $\delta_{t,i} = 1$.

A problem of this scheme is that if for a given minibatch $\sum_{i \in \mathcal{B}} \delta_{t,i}$ is small, the update for $\mathbf{w}_t$ will use a noisy gradient, which in turn leads to poor convergence behavior. We originally handled this heuristically by increasing the minibatch size to 50 images rather that 10, which partially mitigates the problem, but is highly inefficient timewise. More

importantly, this does not allow us to scale up to many tasks - as the number of tasks increases, the minibatch size would need to accordingly increase.

We propose instead to give up the idea of having a common minibatch for all tasks and instead update task-specific parameters only once sufficiently many task-related images have been observed. This can be accomplished using the algorithm described in Table 1 which treats images in a streaming mode, rather than in batches. Whenever we process a training sample that contains ground truth for a task we increment a counter particular to the task, and add the current task-specific gradient to a cumulative gradient sum. Once the task counter exceeds a threshold we update the task parameters and then reset the counter and cumulative gradient to zero. The task parameter updates become decoupled, resulting in an asynchronous update scheme.

We also note that in the pseudocode we use different 'effective batchsizes', $B_p$, which can be useful for efficient training. In particular, for detection tasks it is reported in [48] that a batchsize of 2 suffices, since every image contains hundreds of examples, while for dense labelling tasks such as semantic segmentation a batchsize of 10, 20 or even 30 is often used [10, 56]. In our training we use an effective batchsize $B_p$ of 2 for detection, 10 for all other task-specific parameters, and 30 for the shared CNN features, $\mathbf{w}_0$. Using a larger batch size for the shared CNN features allows their updates to absorb information from more images, containing multiple tasks, so that task-specific idiosyncrasies will cancel out by averaging. This avoids the 'moving target' problem, where every task quickly changes the shared representation of the other tasks.

## 4. Memory-Bound Multi-Task Training

We now turn to handling memory limitations, which become a major problem when training for many tasks. We build on recent advances in memory-efficient backpropagation [12,21], where one trades off computation for memory, but without sacrificing accuracy; we adapt these advances to multi-task learning and develop an algorithm with a memory complexity that is independent of the number of tasks.

As shown in Fig. 3, the common implementation of back-propagation maintains in memory all intermediate layer activations computed during the forward pass; in the backward pass each layer can then combine its activations with the back-propagated gradients and send gradients to its own parameters and to the layer below. Storing intermediate activations saves computation by reusing the computed activation signals, but requires memory. If we consider that every layer requires $N$ bytes of memory for its activation and gradient signals, and we have $L_C$ layers for a shared CNN trunk, $T$ tasks, and $L_T$ layers per task, the memory complexity of a naive implementation would be $2N(L_C + TL_T)$, as shown in Fig. 3 for $L_C = 6, L_T = 3$.
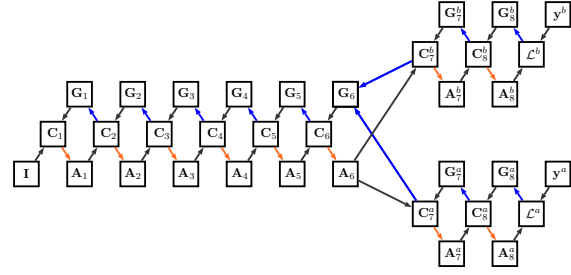


Figure 3: Memory usage in "vanilla" backpropagation for multi-task training: lookup operations are indicated by black arrows, storage operations are indicated by orange and blue arrows for the forward and backward pass, respectively. During the forward pass each layer stores its activation signals in the bottom boxes. During the backward pass these activation signals are combined with the recursively computed gradient signals (top boxes).

We propose instead an algorithm that trades off computation time with memory complexity, adapting the work of [12, 21] to exploit the particularities of our multi-task setup.

In a first stage, shown in Fig. 4(a), we perform a forward pass through the common trunk of the network and store the activations of only a subset of the layers - for a common trunk of depth $L_C$, $\sqrt{L_C}$ activations are stored, lying $\sqrt{L_C}$ layers apart, while the other intermediate activations, shown in grey, are discarded as soon as they are used. These stored activations help start the backpropagation at a deeper layer of the network, acting like anchor points for the computation: as shown in Fig. 4(d,e), back-propagation for any subnetwork requires the activation of its lowest level, and the gradient signal at its highest layer. Since the subnetworks are of length $\sqrt{L_C}$ themselves, we need a total of $2\sqrt{L_C}$ memory units for the trunk.

So far we are entirely along the lines of [12, 21] - however these algorithms would treat the whole multi-task network as a single processing pipeline, meaning that for $T$ tasks of length $L_T$ each the complexity would grow as $\sqrt{L_C + TL_T}$. While substantially lower, this can still become unmanageable as the number of tasks $T$ grows.

We observe however that after the branching point of the different tasks (layer 6 in our figure), the computations decouple: each task-specific branch can work on its own, as shown in Fig. 4(b,c) and return a gradient signal to layer 6. These gradient signals are accumulated over tasks, since our cost is additive over the task-specific losses. This means that each task can be removed from memory once it has communicated its gradient signals to the shared CNN trunk. For a task-specific network depth of $L_T$, the memory complexity is reduced from $\sqrt{L_C + TL_T}$ to $\sqrt{L_C + L_T}$, becoming independent of the number of tasks.

(a) Low-memory forward pass



(b) Low-memory backpropagation - task a



(c) Low-memory backpropagation - task b



(d) Low-memory backpropagation (4-6)
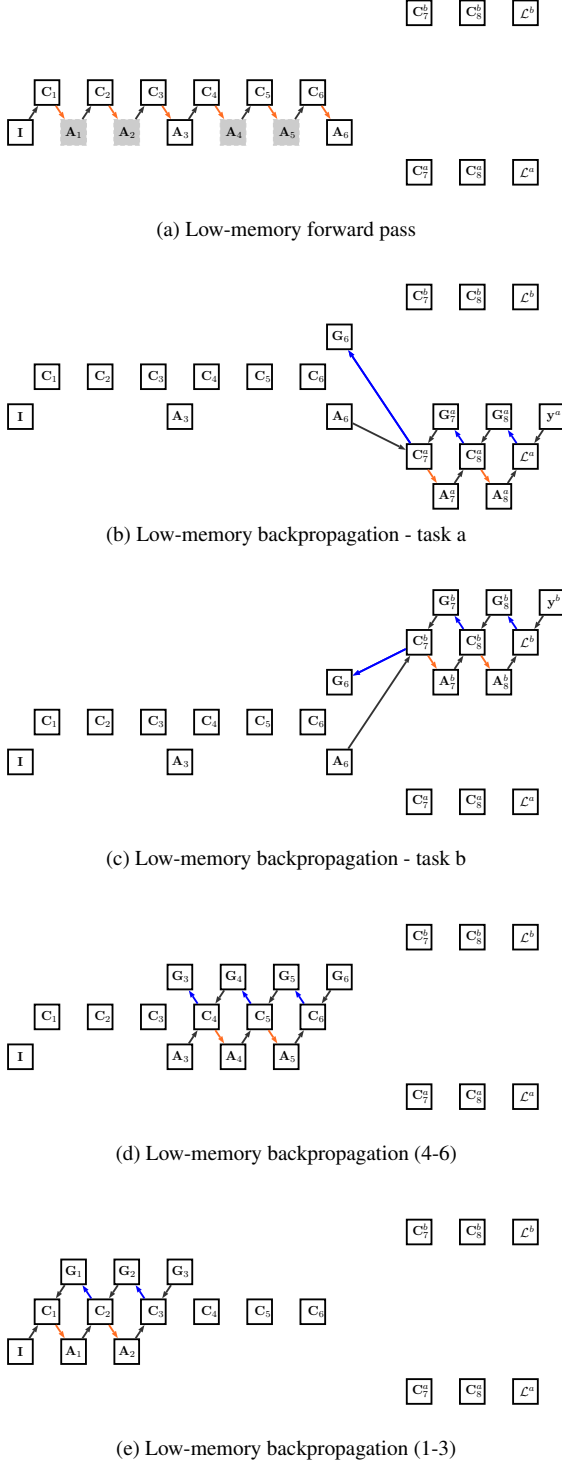


(e) Low-memory backpropagation (1-3)

Figure 4: Low-memory multi-task backpropagation: For the common trunk we store a subset of activations in memory, which serve as 'anchor' points for backpropagation on smaller networks. Each task 'cleans up' its branch once visited, resulting in a memory complexity that is independent of the number of tasks.

This modification of the algorithm has allowed us to tackle an increasing number of tasks with our network without encountering any memory issues. Using a 12GB GPU card we have been able to use a three-layer pyramid, with the largest image size being 921x621, while using skip-layer connections for all network layers, pyramid levels, and tasks. The largest dimension that would be possible without the memory-efficient option for our present number of tasks (7) would be 321x321 - and that would only decrease as more tasks are used. We had originally run several experiments with lower-resolution images, or cropped frames and witnessed in all cases substantial deterioration of detection performance - namely a drop in mean Average Precision from $78\%$ to $67 - 72\%$, apparently due to the missing context, poor resolution, and bounding box distortion due to object cropping. By contrast, our algorithm avoids any compromises in spatial resolution, and accuracy, while being scalable to an arbitrary number of tasks.

Apart from reducing memory demands, we also reduce computation time by *lazy evaluation*: if a training sample does not contain ground truth for certain tasks, these tasks will not contribute any gradient term to the common CNN trunk. The computation over such task-specific branches is therefore avoided, which results in a substantial acceleration of training, and can help scale up to many tasks.

## 5. Experiments

Our experimental evaluation has two objectives: The first one is to show that the generic UberNet architecture introduced in Sec. 2 successfully addresses a broad range of tasks. The second is to explore the effects of incorporating more tasks on individual task performance.

Regarding *individual task performance*, we compare primarily to results obtained by methods that rely on the VGG network [51]. More recent works e.g. on detection [16] and semantic segmentation [10] have shown improvements through the use of deeper ResNets [24], but we consider the choice of network to be orthogonal to this section's goals. Furthermore, due to the multitude of tasks addressed by our system and limited space it is impossible to provide an in-depth analysis of all results on a per-task level. We provide an extensive presentation of results with different networks and task-specific choices in the project website, and focus here on the main interesting results.

Regarding *multi-task performance* we neet to cater for (a) a common initialization, and (b) a consistent construction of the training set. A common initialization requires having at our disposal parameters for both the convolutional labelling tasks, and the region-based object detection task. For this we form a 'frankenstein' network where we stitch together two distinct variants of the VGG network that have both been pretrained on MS-COCO. In particular we use the network of [10] for semantic segmentation ('COCO-S') and

| Method | mAP |
|---|---|
| F-RCNN, [48] VOC 2007++ | 73.2 |
| F-RCNN, [48] MS-COCO + VOC 2007++ | 78.8 |
| Ours, 1-Task | 78.7 |
| Ours, 2-Task | 80.1 |
| Ours, 7-Task | 77.8 |

Table 2: Mean Average Precision (AP) performance (%) on the PASCAL VOC 2007 test set.

the network of [48] for detection, ('COCO-D'), and subsequently finetune.

Turning to (b), the consistent training set construction, we note that our multi-task network is trained with a union of datasets corresponding to the multiple tasks that we want to solve. Even though using larger task-specific datasets could boost performance for the individual tasks, we train the single-task networks only with the subset of the multi-task dataset that pertains to the particular task. For instance, human part segmentation requires putting aside part of the PASCAL Validation set, since that is used to test part segmentation. This sacrifices some task-specific performance for other tasks (e.g. detection, or semantic segmentation) but facilitates comparison between our single- and multi-task training results. We use a particular proportion of images per dataset, moderately favoring the high-level tasks; we elaborate on datasets in the supplemental material.

## 5.1. Experimental Evaluation

**Object Detection:** A main concern has been to ensure that we have high performance in object detection, since it is one of the dominant computer vision problems.

We start our experiments by verifying that we can replicate the results of [48], while using the UberNet architecture with all its modifications (differences are detailed in the supplement). Here we used the COCO-D initialization described above, and train on the VOC2007++ dataset of [48], composed of the union of the VOC2007 and VOC2012 trainval sets. As shown in the first, 'Ours 1-Task' row of Table 2, we have effectively identical performance with the method of [48] that uses the same initialization.

We then measure the performance of the network obtained by training for the joint segmentation and detection task, which, as mentioned in Sec. 5 will serve as our starting point for all ensuing experiments. We observe that by jointly training on detection and segmentation we get a small boost in performance, which is indicating that the additional supervision signal for semantic segmentation can help improve the performance of the detection sub-network.

However, when moving to seven tasks, performance drops, yet remains comparable to the strong baseline of [48]. This is also observed in the tasks described below.

**Semantic Segmentation:** The next task that we consider is semantic segmentation. Even though a really broad range of techniques have been used for the problem (see e.g. [10] for a recent comparison), we only compare to the methods lying closest to our own, which in turns relies on the 'Deeplab-Large Field of View (FOV)' architecture of [9].

The results of the two-task architecture reported in Table 3 indicate that we get effectively the same performance as Deeplab Large-FOV. This is quite surprising given that for this two-task network, as detailed above, our starting point has been a 'frankenstein' network that uses the detection network parameters up to the fifth convolutional layer, rather than the segmentation parameters.

Turning to the multi-task network performance, we observe that performance drops as the number of tasks increases. Still, even without using CRF post-processing, we fare comparably to a strong baseline, such as [44].

**Remaining dense labelling and regression tasks:** From the results provided for the remaining tasks in Tables 4-8, we observe a similar behavior. Namely, when trained in the single-task setting the UberNet architecture has a performance that can directly compete, or sometimes even outperform comparable state-of-the-art systems recently developped for the individual tasks (similar results for certain of these tasks have also been obtained independently in [1]). This can be attributed to the use of skip layers and multiple resolutions, which were originally shown in [29, 56] to substantially help boundary detection.

However, when turning to the seven-task architecture, we typically have a moderate, yet systematic drop in performance with respect to the respective single-task network. This may seem to be in contrast to the experimental finding for the two-task network, and also the general tenet of multi-task training, according to which learning to solve one task can help performance in others. We argue however that this can be anticipated given (a) the diversity of the tasks and (b) the limited parameter 'budget' of our common CNN trunk. We explore this in some larger depth below and comment on possible remedies in the conclusion.

**Balancing diverse tasks:** The performance of our network on the set of tasks it adresses depends on the weights assigned to the different task losses in Eq. 1. A large weight for one task can skew the network's internal representation in favor of the particular task and neglect the rest.

To study this, we focus in particular on the normal estimation task, which has one of the most interesting behaviors. Starting from Table 8, we report multiple results for the single-task training case, obtained by setting different values to the normal task's weight $\gamma_t$ in Eq. 1. Since this is in the single-task setting, $\gamma$ only sets the tradeoff between the loss and the regularization. We observe that $\gamma$ substantially affects performance. A large value leads to results competitive with the current state-of-the-art, while a low weight

Figure 5: Qualitative results of our network. The first three rows indicate outputs for low- and mid-level, category-agnostic tasks; the bottom four rows indicate performance for high-level tasks developed around the 20 categories of PASCAL VOC.

harms performance.

Turning now to multiple tasks, in Table 9 we report how performance changes when we increase the weight of the normal estimation task ($\gamma = 1$ being the default option). Even though jointly training for semantic segmentation and object detection originally improved detection accuracy, we now observe that the performance measures of the different tasks act like communicating vessels: improving normal estimation harms other tasks and vice versa.

**Discussion of results:** We outline possible interpretations of our results and associated future research directions.

Firstly, what is happening can be understood as an instance of "catastrophic forgetting" - we start from a network that is fine-tuned for detection and semantic segmentation,

| Method | mIoU |
|---|---|
| Deeplab -COCO + CRF [44] | 70.4 |
| Deeplab Multi-Scale [29] | 72.1 |
| Deeplab Multi-Scale -CRF [29] | 74.8 |
| Ours, 1-Task | 72.4 |
| Ours, 2-Task | 72.3 |
| Ours, 7-Task | 68.7 |

Table 3: Semantic segmentation: mean Intersection Over Union (IOU) accuracy on PASCAL VOC 2012 test.

| Method | mean IoU |
|---|---|
| Deeplab L-FOV [54] | 51.78 |
| Deeplab L-FOV-CRF [54] | 52.95 |
| Multi-scale averaging [11] | 54.91 |
| Attention [11] | 55.17 |
| Auto Zoom [54] | 57.54 |
| Ours, 1-Task | 51.98 |
| Ours, 7-Task | 48.82 |

Table 4: Part segmentation: mean IOU accuracy on the dataset of [13].

| Method | mAP | mMF |
|---|---|---|
| Semantic Contours [22] | 20.7 | 28.0 |
| High-for-Low [5] | 47.8 | 58.7 |
| High-for-Low-CRF [5] | 54.6 | 62.5 |
| Ours, 1-Task | 54.3 | 59.7 |
| Ours, 7-Task | 44.3 | 48.2 |

Table 5: Semantic Boundary Detection: Mean AP performance (%) and Mean Max F-Measure Score on the PASCAL VOC 2010 validation set by [22].

| Method | MF |
|---|---|
| MDF [33] | 0.764 |
| FCN [34] | 0.793 |
| DCL [34] | 0.815 |
| DCL + CRF [34] | 0.822 |
| Ours, 1-Task | 0.835 |
| Ours, 7-Task | 0.823 |

Table 6: Saliency estimation: Maximal F-measure (MF) on PASCAL-S [35].

| Method | ODS | OIS | AP |
|---|---|---|---|
| HED-fusion [56] | 0.790 | 0.808 | 0.811 |
| Multi-Scale [29] | 0.809 | 0.827 | 0.861 |
| Multi-Scale +sPb [29] | 0.813 | 0.831 | 0.866 |
| Ours, setup of [29] | 0.815 | 0.835 | 0.862 |
| Ours, 1-Task | 0.791 | 0.809 | 0.849 |
| Ours, 7-Task | 0.785 | 0.805 | 0.837 |

Table 7: Boundary Detection: maximal F meaure at the Optimal Dataset Scale, Optimal Image Scale, and Average Precision on the BSD dataset [40].

| Method | Mean | Median | 11.25° | 22.5° | 30° |
|---|---|---|---|---|---|
| VGG-Cascade [17] | 22.2 | 15.3 | 38.6 | 64.0 | 73.9 |
| VGG-MLP [2] | 19.8 | 12.0 | 47.9 | 70.0 | 77.8 |
| VGG-Design [53] | 26.9 | 14.8 | 42.0 | 61.2 | 68.2 |
| Ours, 1-Task $\gamma = 50$ | 21.4 | 15.6 | 35.3 | 65.9 | 76.9 |
| Ours, 1-Task $\gamma = 5$ | 23.3 | 17.6 | 31.1 | 60.8 | 72.7 |
| Ours, 1-Task $\gamma = 1$ | 23.9 | 18.1 | 29.8 | 59.7 | 71.9 |
| Ours, 7-Task $\gamma = 1$ | 26.7 | 22.0 | 24.2 | 52.0 | 65.9 |

Table 8: Normal Estimation: Mean and Median Angle distance (in radians) and percentage of pixels within 11.25, 22.5, and 30 degrees of the ground truth of [31].

| | Detection | Boundaries | | | Saliency | Parts | Surface Normals | | | S. Boundaries | | S. Seg. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | mAP | ODS | OIS | AP | MF | mIoU | 11.2° | 22.5° | 30.0° | MF | mAP | mIoU |
| $\gamma = 1$ | 77.8 | 0.785 | 0.805 | 0.837 | 0.822 | 48.8 | 24.2 | 52.0 | 65.9 | 44.3 | 48.2 | 68.7 |
| $\gamma = 5$ | 76.2 | 0.779 | 0.805 | 0.836 | 0.820 | 36.7 | 23.1 | 51.0 | 64.9 | 33.6 | 34.2 | 67.2 |
| $\gamma = 50$ | 73.5 | 0.772 | 0.802 | 0.830 | 0.814 | 34.2 | 27.7 | 57.3 | 70.2 | 28.6 | 33.2 | 63.5 |

Table 9: Impact of the weight used for the normal estimation loss, when training for seven tasks: Improving normal estimation comes at the cost of decreasing performance in the remaining tasks (higher is better for all tasks).

and as we re-train to solve additional tasks, performance in the original tasks drops. Remedies to this problem can include using the original model outputs as supervision signals [7, 36, 52], or, as advocated in [28], considering the sensitivity of other tasks when updating shared parameters.

Secondly, our network's common CNN trunk has a limited number of parameters and layers. Apart from using deeper networks, we can add nonlinear layers on top of the skip layers, or "twin networks" as in [41] to provide an additional computation/parameter budget to our network.

Finally, another cause could be the highly diverse nature of the tasks - e.g. normal estimation and human part segmentation have little in common. Multi-task learning requires some task relatedness [4] and it is common to pursue some automated identification of tasks which are related and should be reinforcing each other, e.g. [27, 46].

## 6. Conclusions

In this work we have introduced two techniques that allow us to train in an end-to-end manner a 'universal' CNN that jointly tackles a broad set of computer vision problems in a unified architecture. We have shown that one can effectively scale up to many and diverse tasks, since the memory complexity is independent of the number of tasks, and incoherently annotated datasets can be combined during training. This has allowed us to train a single network that can solve multiple tasks in a fraction of a second with competitive performance.

We hope that these advances will make it possible to fully reap the benefits of multi-task learning for CNNs in vision, potentially along the lines outlined in the discussion. We will be sharing our system's implementation to foster research in this direction.

## Acknowledgements

## References

[1] A. Bansal, X. Chen, B. Russell, A. Gupta, and D. Ramanan. Pixelnet: Towards a general pixel-level architecture. *CoRR*, abs/1609.06694, 2016.

[2] A. Bansal, B. Russell, and A. Gupta. Marr revisited: 2d-3d alignment via surface normal prediction. In *Proc. CVPR*, 2016.

[3] S. Bell, C. L. Zitnick, K. Bala, and R. Girshick. Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. *Proc. CVPR*, 2016.

[4] S. Ben-David. A notion of task relatedness yielding provable multiple-task learning guarantees. *M. Learning*, 2008.

[5] G. Bertasius, J. Shi, and L. Torresani. High-for-low and low-for-high: Efficient boundary detection from deep object features and its applications to high-level vision. In *Proc. ICCV*, 2015.

[6] H. Bilen and A. Vedaldi. Integrated perception with recurrent multi-task neural networks. In *Proc. NIPS*, 2016.

[7] R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[8] S. Chandra and I. Kokkinos. Fast, exact and multi-scale inference for semantic image segmentation with deep gaussian crfs. In *Proc. ECCV*, 2016.

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

[10] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *CoRR*, abs/1606.00915, 2016.

[11] L. Chen, Y. Yang, J. Wang, W. Xu, and A. L. Yuille. Attention to scale: Scale-aware semantic image segmentation. In *Proc. CVPR*, 2015.

[12] T. Chen, B. Xu, C. Zhang, and C. Guestrin. Training deep nets with sublinear memory cost. *CoRR*, abs/1604.06174, 2016.

[13] X. Chen, R. Mottaghi, X. Liu, S. Fidler, R. Urtasun, and A. Yuille. Detect what you can: Detecting and representing objects using holistic models and body parts. In *Proc. CVPR*, 2014.

[14] J. Dai, K. He, and J. Sun. Boxsup: Exploiting bounding boxes to supervise convolutional networks for semantic segmentation. In *Proc. ICCV*, 2015.

[15] J. Dai, K. He, and J. Sun. Instance-aware semantic segmentation via multi-task network cascades. In *Proc. CVPR*, 2016.

[16] J. Dai, Y. Li, K. He, and J. Sun. R-FCN: object detection via region-based fully convolutional networks. In *Proc. NIPS*, 2016.

[17] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proc. ICCV*, 2015.

[18] G. Ghiasi and C. C. Fowlkes. Laplacian reconstruction and refinement for semantic segmentation. In *Proc. ECCV*, 2016.

[19] R. B. Girshick. Fast R-CNN. In *Proc. ICCV*, 2015.

[20] G. Gkioxari, R. B. Girshick, and J. Malik. Contextual action recognition with r*cnn. In *Proc. ICCV*, 2015.

[21] A. Gruslys, R. Munos, I. Danihelka, M. Lanctot, and A. Graves. Memory-efficient backpropagation through time. *CoRR*, abs/1606.03401, 2016.

[22] B. Hariharan, P. Arbeláez, L. Bourdev, S. Maji, and J. Malik. Semantic contours from inverse detectors. In *Proc. ICCV*, 2011.

[23] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Hypercolumns for object segmentation and fine-grained localization. In *Proc. CVPR*, 2015.

[24] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. CVPR*, 2016.

[25] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. ICML*, 2015.

[26] A. Kanazawa, A. Sharma, and D. W. Jacobs. Locally scale-invariant convolutional neural networks. *CoRR*, abs/1412.5104, 2014.

[27] Z. Kang, K. Grauman, and F. Sha. Learning with whom to share in multi-task feature learning. In *ICML*, 2011.

[28] J. Kirkpatrick, R. Pascanu, N. C. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming catastrophic forgetting in neural networks. *PNAS*, 2017.

[29] I. Kokkinos. Pushing the boundaries of boundary detection using deep learning. *ICLR*, 2016.

[30] P. Krähenbühl and V. Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *NIPS*, 2011.

[31] L. Ladicky, B. Zeisl, and M. Pollefeys. Discriminatively trained dense surface normal estimation. In *Proc. ECCV*, 2014.

[32] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proc. IEEE*, 1998.

[33] G. Li and Y. Yu. Visual saliency based on multiscale deep features. In *Proc. CVPR*, 2015.

[34] G. Li and Y. Yu. Deep contrast learning for salient object detection. In *Proc. CVPR*, 2016.

[35] Y. Li, X. Hou, C. Koch, J. M. Rehg, and A. L. Yuille. The secrets of salient object segmentation. In *Proc. CVPR*, 2014.

[36] Z. Li and D. Hoiem. Learning without forgetting. In *European Conference on Computer Vision - ECCV*, pages 614–629, 2016.

[37] G. Lin, C. Shen, I. D. Reid, and A. van den Hengel. Efficient piecewise training of deep structured models for semantic segmentation. *CVPR*, 2016.

[38] W. Liu, A. Rabinovich, and A. C. Berg. Parsenet: Looking wider to see better. *CoRR*, abs/1506.04579, 2015.

[39] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proc. CVPR*, 2015.

[40] 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. ICCV*, 2001.

[41] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert. Cross-stitch networks for multi-task learning. In *Proc. CVPR*, 2016.

[42] A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation. *CoRR*, abs/1603.06937, 2016.

[43] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *Proc. ICCV*, 2015.

[44] G. Papandreou, L. Chen, K. Murphy, and A. L. Yuille. Weakly- and semi-supervised learning of a DCNN for semantic image segmentation. In *Proc. ICCV*, 2015.

[45] G. Papandreou, I. Kokkinos, and P. Savalle. Modeling local and global deformations in deep learning: Epitomic convolution, multiple instance learning, and sliding window detection. In *Proc. CVPR*, 2015.

[46] A. Pentina, V. Sharmanska, and C. H. Lampert. Curriculum learning of multiple tasks. In *Proc. CVPR*, 2015.

[47] R. Ranjan, V. M. Patel, and R. Chellappa. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *CoRR*, abs/1603.01249, 2016.

[48] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In *Proc. NIPS*, 2015.

[49] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Proc. MICCAI*, 2015.

[50] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *Proc. ICLR*, 2014.

[51] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. ICLR*, 2015.

[52] S. Thrun and T. M. Mitchell. Lifelong robot learning. *Robotics and Autonomous Systems*, 15(1-2):25–46, 1995.

[53] X. Wang, D. F. Fouhey, and A. Gupta. Designing deep networks for surface normal estimation. In *Proc. CVPR*, 2015.

[54] F. Xia, P. Wang, L. Chen, and A. L. Yuille. Zoom better to see clearer: Human part segmentation with auto zoom net. In *Proc. ECCV*, 2016.

[55] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017.

[56] S. Xie and Z. Tu. Holistically-nested edge detection. In *Proc. ICCV*, 2015.

[57] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. Torr. Conditional random fields as recurrent neural networks. In *Proc. ICCV*, 2015.