SEGMENTATION-AWARE CONVOLUTIONAL NETS

by

Adam W. Harley

BA, Ryerson University, Canada, 2012

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Program of

Computer Science

Toronto, Ontario, Canada, 2016

SEGMENTATION-AWARE CONVOLUTIONAL NETS

Master of Science in Computer Science, 2016

Adam W. Harley

Ryerson University

Abstract

This thesis introduces a method to both obtain segmentation information and integrate it uniformly within a convolutional neural network (CNN). This counter-acts the tendency of CNNs to produce smooth predictions, which is undesirable for pixel-wise prediction tasks, such as semantic segmentation. The segmentation information is obtained by a form of metric learning, where a CNN learns to compute pixel embeddings that reflect whether any pair of pixels is likely to belong to the same region. This information is then used within a larger network, to replace all convolutions with foreground-focused convolutions, where the foreground is determined adaptively at each image point by local embeddings. The resulting network is called a segmentation-aware CNN, because the network can change its behaviour at each image location according to local segmentation cues. The proposed method yields systematic improvements on a standard semantic segmentation benchmark when compared to a strong baseline.

Acknowledgements

Thanks to my advisor, Kosta Derpanis, for pulling me into the vibrant world of computer vision, and for showing me so many opportunities to make progress toward my goals. Thanks also to Iasonas Kokkinos, who supervised me in an internship, for his trust and generosity in sharing valuable research ideas.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In computer vision, deep learning models (LeCun, Bengio, & Hinton, 2015; Schmidhuber, 2015) are at the core of state-of-the-art approaches to a variety of high-level tasks, including object recognition (Krizhevsky, Sutskever, & Hinton, 2012; Simonyan & Zisserman, 2015), object detection (Girshick, Donahue, Darrell, & Malik, 2014), and image annotation (Karpathy & Fei-Fei, 2015).

There exist many different types of deep learning models, but the most popular is likely the convolutional neural network (CNN, or convnet). Like most deep learning models, convolutional nets can be understood as an ordered set of processing stages, which gradually transform the input into a desired output. For example, the input may be an image, and the output may be a word that describes the image. The processing stages are called layers, which can be imagined as forming a tall (or deep) stack, where the bottom layer reads in the input, and the top layer provides the final abstraction.

Inside a convolutional net, the stack of layers generally alternates between two layer types: convolution layers, and subsampling layers. The convolution layers use filters to extract feature-rich versions of their inputs. The definition of a desirable “feature” depends on the convolution layer, but features generally correspond to edges, textons (i.e., texture elements), and object parts. The subsampling layers reduce the spatial resolution of their inputs. Together, these layers allow the network to gradually consolidate its collected features into higher-order representations. Figure 1.1 shows an illustration of the two layer types working together. This is often interpreted as a hierarchical process of perception: lower layers of the network detect contours, middle layers find shapes, and higher layers perform (for instance) object recognition.

Many tasks require concise representations of whole images, and repeated subsampling aligns well with this demand. Classification is a good example of this: the top output of a convolutional net designed for classification has spatial dimensions 1×1, representing a single prediction for the entire image. There are some tasks, however, for which the typical setup of a convolutional network is ill-suited. Semantic segmentation is one such task. For an input image with dimensions $H \times W$, the goal in semantic segmentation is to produce an $H \times W$ matrix of predictions, indicating the object category present at every pixel in the image. This is illustrated in Figure 1.2. Other tasks requiring dense, spatially-precise predictions include optical flow estimation (Dosovitskiy et al., 2015), surface normal estimation (Wang, Fouhey, & Gupta, 2015), and depth estimation (Eigen, Puhrsch, & Fergus, 2014).

In the task of producing an accurate mapping from $H \times W$ inputs to $H \times W$ outputs, convolutional nets struggle with two issues: *spatial resolution loss*, and *smoothness of predictions*. That is, whereas the ideal prediction map

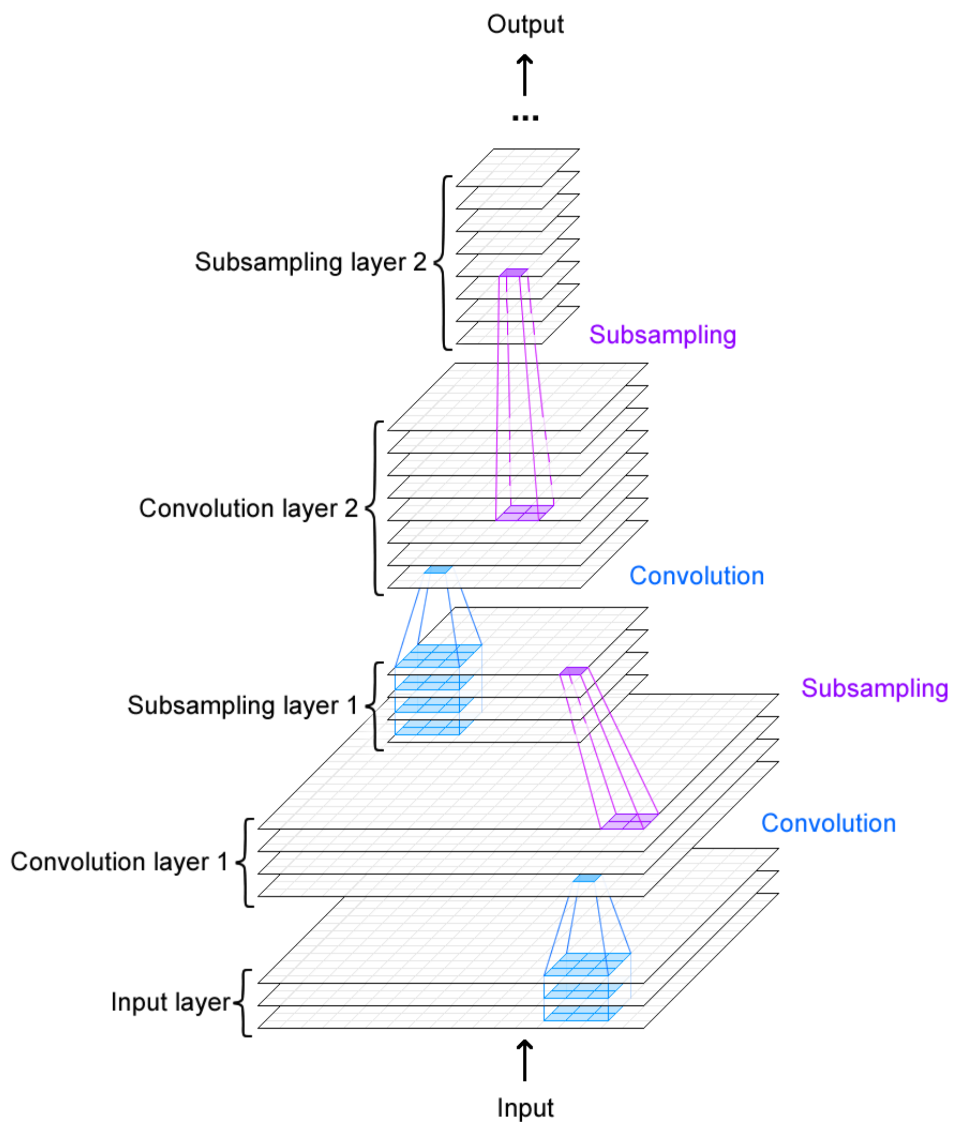


Figure 1.1: Part of a typical convnet for computer vision. The input (a three-channel colour image) acts as the bottom layer of the network. Subsequent layers alternate between convolution and subsampling. Convolution layers filter their inputs to extract features from them. Subsampling layers reduce the spatial resolution of their inputs, to give higher layers a wider field of view. Through many such layer pairs (and much training), the network attempts to achieve a (pre-specified) mapping from inputs to outputs.

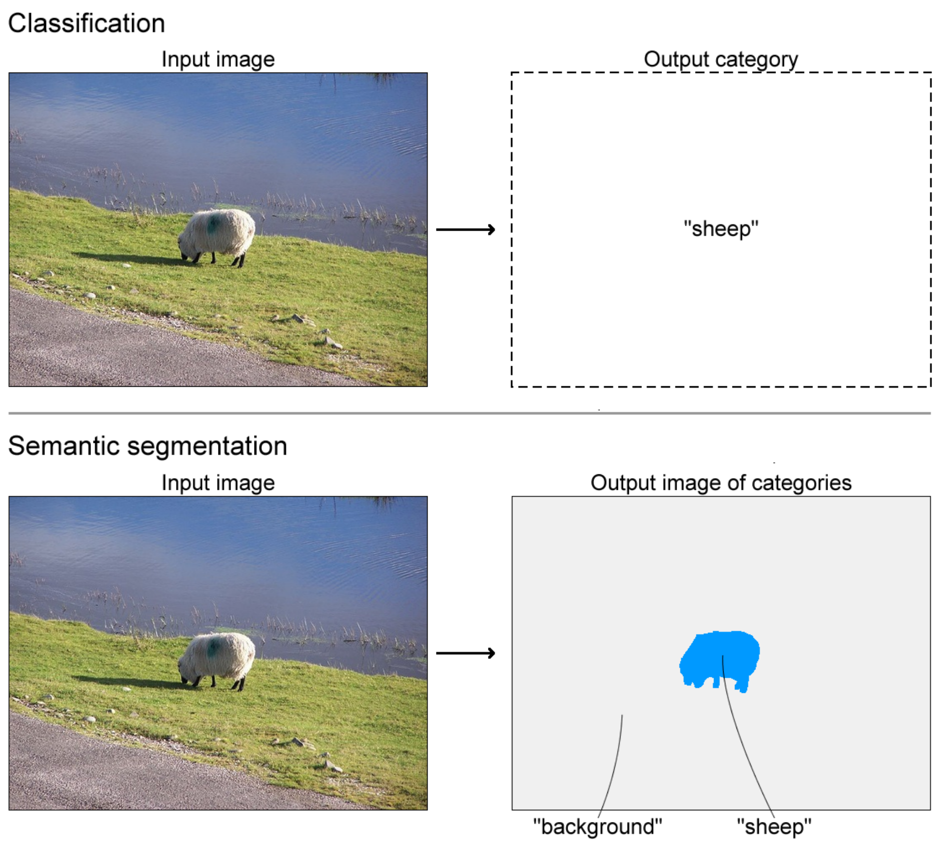


Figure 1.2: Classification vs. semantic segmentation. In classification, the objective is to produce a class label for an entire image. In semantic segmentation, the objective is to produce a class label for every pixel in the image.

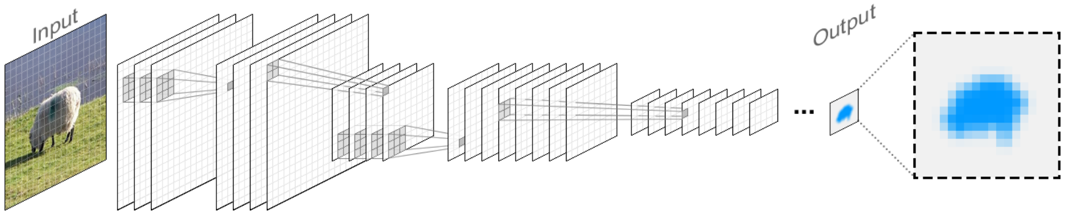


Figure 1.3: Issues with convolutional nets on semantic segmentation. The output from these networks tends to be low-resolution (due to subsampling), and smooth (due to convolution).

would match the input with pixel-perfect accuracy, the predictions produced by convnets are small and blurry. This is illustrated in Figure 1.3.

Spatial resolution loss is caused by the subsampling layers of the network. These stages play an important role in the hierarchical consolidation of features, and widen the higher layers' field of view. Because of these benefits, subsampling cannot be removed in a straightforward manner. There exists substantial research on working around this issue: methods have been proposed for replacing these layers with resolution-preserving alternatives (Chen, Papandreou, Kokkinos, Murphy, & Yuille, 2015; Yu & Koltun, 2016), and restoring the lost resolution via upsampling stages (Noh, Hong, & Han, 2015; Long, Shelhamer, & Darrell, 2014).

While there has been much recent work on the issue of spatial resolution loss, the issue of smoothness has remained relatively unexplored. Considered independently from the spatial resolution issue, smoothness is caused by the convolution layers. As the image of an object moves through the receptive field of a neuron (or set of neurons), the activation levels change gradually, as a function of how well the input matches the neurons' learned templates.

This property of convnets is very useful for full-image understanding tasks, but degrades performance on per-pixel prediction accuracy, where rapid pixel-accurate changes in behaviour are required.

Thus, it is towards resolving convolutional nets' *smooth predictions* that this thesis makes a contribution.

1.1 Motivation

As a motivating example to the key idea of this thesis, consider the case where a convolutional network is attempting to classify a pixel that is just off the boundary of an object. Figure 1.4 (top) shows an example of this. To classify the pixel, the convnet needs to draw in information from a wide area surrounding the pixel. Since the area will contain a mix of "background" pixels and "object" pixels, the network will likely make an unconfident prediction (e.g., output 60% confidence in "background", and 40% confidence in "object"), reflecting the mix of useful and distracting information in the patch. Similarly, when the network attempts to classify a pixel that is just within the boundary of an object, the patch will again contain a mix of information, and the prediction will be similar. Visualizing these predictions as an image, they appear blurred and imprecise. Therefore, the issue of "smooth predictions" relates to a weakness of convnets in dealing with the presence of distracting information near object boundaries.

This thesis proposes a way for convnets to block incoming information that is likely to be distracting. The basic approach is to complement each convolution filter with a local foreground-background segmentation mask, so that the network has a mechanism for allowing only "foreground" pixels to pass through the filters. The "foreground" of each patch is defined in relation to the patch's central pixel. Critically, the masks are generated within the convnet, and are computed "on the fly" during convolution. Additionally, the segmentation cues on which the masks rely, as well as the parameters defining mask usage throughout the network, are learned in the selfsame convnet. Figure 1.4 illustrates the approach, and compares it with standard convolutional nets. The proposed network is called a segmentation-aware convolutional net, because the network adjusts its behaviour on a per-pixel basis according to local segmentation cues derived from the input image.

1.2 Contributions

There are three key components to creating segmentation-aware convolutional nets, corresponding to the three primary contributions of this thesis.

1. **Dense convolutional embeddings.** The first idea is to learn an embedding (i.e., a feature space) that differentiates pixels according to relative foreground-background information. This is done by training features to respect distance thresholds from their neighbours, according to semantic parity.

2. **Segmentation-aware bilateral filtering.** The second idea is to create local segmentation masks from the embeddings, in a new convolution-like layer for convnets that generalizes the bilateral filter (an edge-preserving smoothing technique; J.-S. Lee, 1983; Aurich & Weule, 1995; Smith &

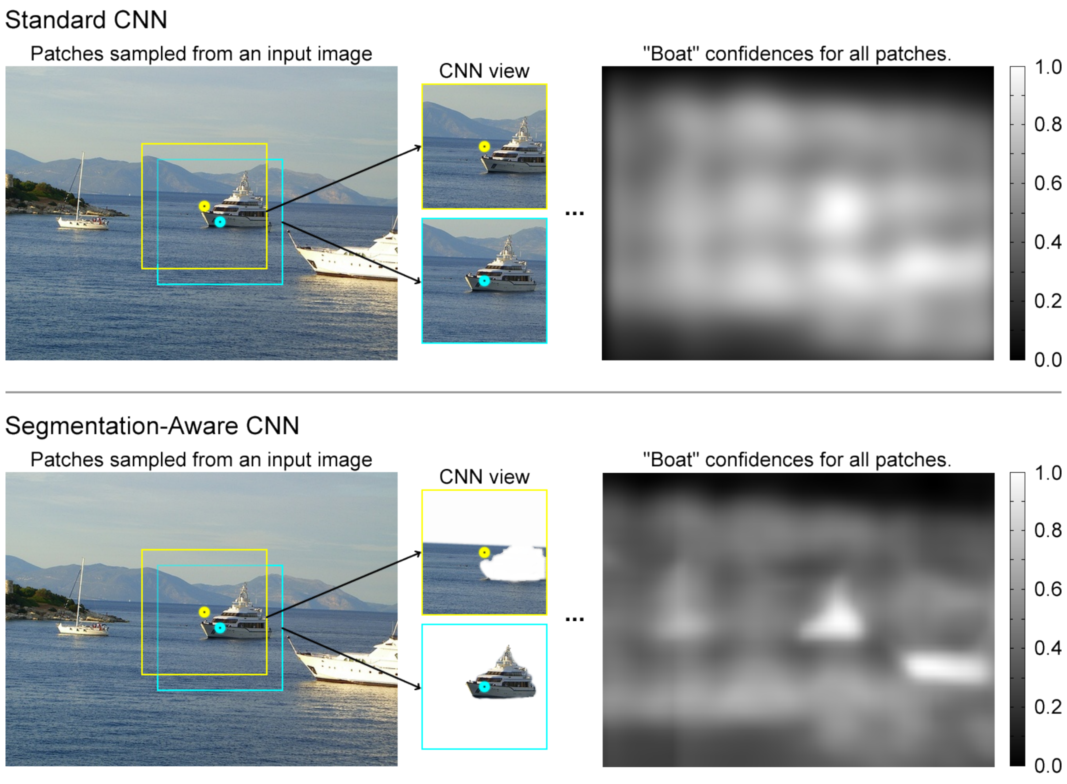

Standard CNN
Patches sampled from an input image
CNN view
"Boat" confidences for all patches.
1.0
0.8
0.6
0.4
0.2
0.0
Segmentation-Aware CNN
Patches sampled from an input image
CNN view
"Boat" confidences for all patches.
1.0
0.8
0.6
0.4
0.2
0.0

Figure 1.4: Standard CNNs versus segmentation-aware CNNs. From an input image (left), two neighbouring patches are highlighted, with their centre pixels marked. The first patch is centered on a water pixel, just off the boat. The second patch is centered on a boat pixel. Since the patches are very similar overall, the standard CNN generates similar predictions for the two patches (e.g., "boat" for both). Visualizing these predictions as a heatmap, they appear blurred and imprecise. In a segmentation-aware CNN, the network internally creates local segmentation masks according to the central pixel of each patch. In the illustration, the masked regions are coloured white; in the network, input from these regions would be ignored. This enables the network to treat each patch differently, and generate predictions with finer spatial precision. The benefits of this are visible in the heatmap generated by the network, which is noticeably superior to the corresponding output of the standard CNN.

Brady, 1997; Tomasi & Manduchi, 1998). The masks can be used to filter the convnet's prediction maps, to sharpen the predictions at object boundaries, while making predictions within boundaries more uniform.

3. **Segmentation-aware convolution.** The third component is a simple modification to convolution layers, allowing the dense non-linear masks to modulate the application of the linear filters of convolution. This achieves a learnable type of normalized convolution (cf. Knutsson & Westin, 1993).

Together, these components make a segmentation-aware convolutional net. Some advantages of the segmentation-aware convnets over standard convnets are discernible in visualizations of segmentation-aware networks' outputs. Furthermore, experimental results show that segmentation-aware convnets achieve a considerable gain in accuracy on a challenging semantic segmentation benchmark. While the evaluation in this work is focused on semantic segmentation, the contributions may prove useful in other tasks requiring dense pixel-wise predictions.

1.3 Outline of thesis

The thesis is organized as follows:

- Chapter 2 provides a basic overview of neural network theory and algorithms, and follows with a summary of the literature most closely related to the current work.

- Chapter 3 presents segmentation-aware convolutional nets, in several parts. The chapter begins with a theoretical overview of the three main ideas: embeddings, bilateral filtering, and segmentation-aware convolution. Next is a description of how each concept is realized in a neural network. Finally, a technical description is provided on efficient implementation details.

- Chapter 4 describes the empirical evaluation used to verify the findings of this work. This includes visualizations that provide qualitative information on the approach's strengths and weaknesses, and a quantitative evaluation on a standard semantic segmentation benchmark. The evaluations explore, in turn, (i) the embeddings, (ii) the bilateral filtering, and (iii) the segmentation-aware convolution.

- Chapter 5 provides a summary of the main contributions and results, and discusses possible directions for future work.

Parts of this work have appeared in "Learning Dense Convolutional Embeddings for Semantic Segmentation" (Harley, Derpanis, & Kokkinos, 2016).

Chapter 2

Literature Review

There are two main sections in this chapter. The first (Sec. 2.1) is dedicated to summarizing the theory and mathematics of neural networks, so as to provide sufficient background information for the subsequent chapters. The second section (Sec. 2.2) attempts to situate the thesis in relation to the active fields of research around it.

2.1 Background

The following subsections provide a brief overview of neural networks. Considering the breadth of neural network theory and literature, this section is only intended to summarize the aspects that are most relevant to the current work; a more thorough overview can be found in review articles (e.g., LeCun et al., 2015; Schmidhuber, 2015) and texts (e.g., Bishop, 1995). The first subsection provides a definition for the "neurons" in neural networks (Sec. 2.1.1). Following this is a subsection on "learning" (Sec. 2.1.1), describing what is meant by the word, and how learning occurs in neural nets. Finally, "architectures" are covered (Sec. 2.1.3), and convolutional nets are introduced.

2.1.1 Neurons

A neural network is a feed-forward graph of nearly-identical processor units called *neurons*. Each unit computes a nonlinear function, typically

$$\sigma(x) = \max(0, x), \tag{2.1}$$

which is called the *rectified linear* unit function (ReLU; Nair & Hinton, 2010). (Earlier work with neural networks used tanh or logistic sigmoid units (e.g., LeCun, Bottou, Bengio, & Haffner, 1998), but the ReLU has since been shown to produce slightly more powerful networks.)

In a ReLU, x is a weighted sum of inputs,

$$x = \sum_n w_n \phi_n + b, \tag{2.2}$$

where w_n is the weight applied to the input ϕ_n, and b is a shift factor called a *bias*. The inputs can either be data, or outputs from preceding neurons in the network. All weights and biases are typically learnable parameters.

2.1.2 Learning

Learning in this context simply refers to advancing toward some desired behaviour. The only apparatus for this is to adjust the weights and biases of the neurons in the network.

The adjustments to the weights and biases are derived from gradient descent on a *loss function* (or *energy function*). The loss represents how poorly the network is performing its task, as evaluated on a labelled dataset. For instance, to train a network to categorize images of hand-drawn digits, one could set up a network that has ten neurons in the final layer, and task the network with the following objective: for an image depicting digit k, neuron k should produce a value of one, and all other neurons should produce zeros. This objective can be phrased as the following loss function:

$$\mathcal{L} = \sum_{e} \sum_{i=0}^{9} (y_{e,i} - d_{e,i})^2, \tag{2.3}$$

where e iterates over all of the examples in the dataset, $y_{e,i}$ is the output of neuron i for example e, and $d_{e,i}$ is the desired output at that neuron. If the network produces the correct output $y_{e,i} = d_{e,i}$ for all digits i for all examples e, the loss will be zero; any deviation from this objective will be accumulated in $\mathcal{L}$.

Moving to a lower loss (i.e., learning) requires computing the partial derivative of the loss function with respect to every parameter of the network, and shifting every parameter along the negative gradient. This can be done efficiently through an algorithm called *backpropagation* (Werbos, 1982; Rumelhart, Hinton, & Williams, 1986), which is a dynamic programming algorithm for iteratively applying the chain rule. Gradient descent is a greedy algorithm, so it is only guaranteed to converge to a local minimum, not necessarily the global one.

Ideally, gradient descent would be done directly on $\mathcal{L}$, but it is prohibitively expensive to consider all examples before performing a parameter update. For this reason, updates are typically computed over a small set of examples at a time. The set of examples is typically called a *batch*. If the examples are chosen randomly, this achieves *stochastic* gradient descent, which is very reliable in practice (LeCun et al., 1998).

In practice, the true gradient (notwithstanding stochasticity) is never used directly to update the weights and biases. Instead, a modified version of the gradient is used, depending typically on two design parameters. The first parameter is the *learning rate*, which is simply a multiplicative scale factor on the gradient. If the learning rate is too high, gradient descent may not find a good minimum; if the learning rate is too low, gradient descent will take an unnecessarily long time. The second design parameter is *momentum*, which replaces the gradient with a weighed average of the current gradient and prior gradients (Sutskever, Martens, Dahl, & Hinton, 2013). The update to a weight w, then, is

$$\begin{aligned} w_{t+1} &= w_t + v_{t+1}, \quad \text{where} \\ v_{t+1} &= \mu v_t - \epsilon \nabla_{w_t} \mathcal{L}. \end{aligned} \tag{2.4}$$

In these equations, t refers to the current iteration, v refers to the "velocity" of the gradient, μ is the momentum coefficient, and ϵ is the learning rate. The design parameters μ, ϵ are sometimes referred to as *hyperparameters*, since they are not optimized within the learning process of the neural network like regular parameters.

2.1.3 Architectures

Many neural network graphs, called *architectures*, are viable. Architectures implicitly comprise hundreds of hyperparameters, but there exist some standard (i.e., well-tested and popular) architectures, which helps simplify the network design process. Figure 2.1 illustrates a *fully-connected network*, which

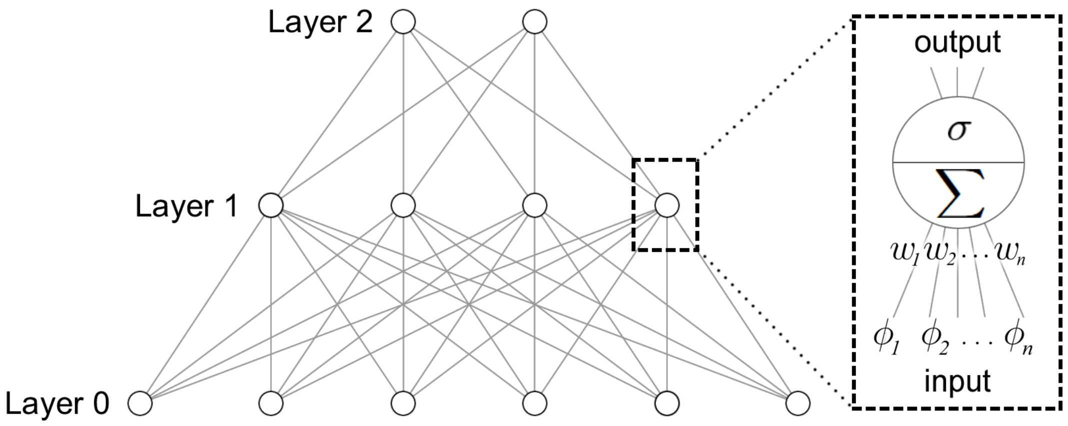


Figure 2.1: A fully-connected network. Each neuron processes the output of the entire layer preceding it, by taking a weighted sum and passing that sum through a nonlinearity.

is a graph with stacked layers of neurons, in which every pair of neighbouring layers is fully connected. Every edge in the graph of a fully-connected network corresponds to a learnable parameter (i.e., a weight).

This work, and other works like it, use a *convolutional network* (Fukushima, 1980; LeCun et al., 1998), which is a graph with far fewer connections, but a similar layered structure. Figure 2.2 illustrates the architecture. Similar to a fully-connected network, each layer of a convolutional net processes the output of the layer preceding it, but each neuron only connects to a local region in the layer below, and all neurons within a layer share their parameters. Interpreting the set of weights for a convolutional layer as a template, t (i.e., $t = [w_1, w_2, \dots, w_n]$), one can write the output for a neuron as

$$y_i = \sigma\left(\sum_{j \in \mathcal{N}_i} x_{i-j} t_j\right), \tag{2.5}$$

where i is the index of the current neuron, and the indices $j \in \mathcal{N}_i$ iterate over

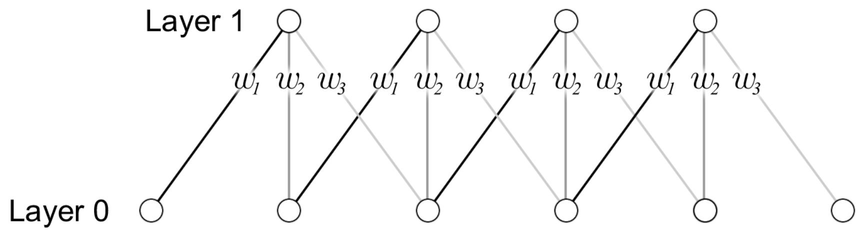
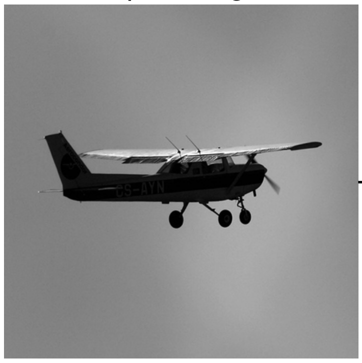

Layer 1
w_1 w_2 w_3 w_1 w_2 w_3 w_1 w_2 w_3 w_1 w_2 w_3
Layer 0

Figure 2.2: A convolutional network. Each neuron processes the output of a local neighbourhood from the layer preceding it. Neurons within a layer share their parameters.

Input image Convolution filter Output image

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} \longrightarrow \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \longrightarrow \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & 3 & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$$

Input image



Convolution filter

$$\longrightarrow \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \longrightarrow$$

Output image

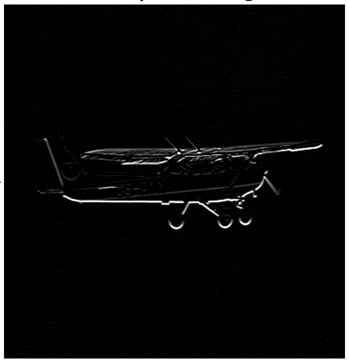


Figure 2.3: The convolution operation, in two examples. At every location in the input, a dot product is taken between the 3×3 filter (or template) and the local 3×3 area in the input. In the first example, the boundary cases are marked with dots; the values to be filled in at those locations depend on the implementation. In the second example, the same filter is applied to a grayscale image. Inspecting the output, one can deduce that the filter is selective for horizontal edges.

offsets in the local spatial neighbourhood. That is, each y_i (before the non-linearity) is produced by pointwise multiplying a template with a local region in the input. This amounts to sliding the template across every location in the input, and computing the dot product at every location; this produces a "filtered" version of the input. "Filter" and "template" are therefore synonymous here. The result of a convolution is called a *feature map*, so called because it marks the locations of the feature targeted by the filter. Figure 2.3 illustrates the operation in 2D, first with a small matrix and then with an image. (The name "convolution" comes from signal processing literature, although in signal processing, the operation strictly requires rotating the filter by $180°$ before the dot product; in neural networks, this step is often skipped, so it is skipped in Figure 2.3 as well.)

The use of convolution layers instead of fully-connected layers is motivated by two important properties of natural images. First, pixels far away from each other are unlikely to be correlated, meaning that long-range connections in the network are unnecessary. This motivates using local connections, rather than full (image-spanning) connections. Second, patterns in images are often repeated in different spatial locations. For example, horizontal edges may appear anywhere in the image. This motivates using the same filters everywhere, i.e., sharing weights between neurons.

Note that it is also within the capacity of fully-connected nets to learn local, spatially-invariant feature detectors. However, since convolutional networks are designed to accomplish this task with far fewer parameters, they are easier to train. This relates to the issue of *overfitting*: with too many parameters, or not enough data, there is a risk of learning (i.e., modelling) noise, rather than the underlying signal. The robustness of convolutional nets, paired with the availability of large datasets of labelled images, has made room for extremely large (and effective) convolutional nets.

For high-level vision tasks, convolutional nets typically have somewhere between 5 and 19 layers, each with somewhere between 64 and 1024 unique filters, with filter dimensions between 3×3 and 7×7. Recently, these networks have grown even further in complexity, reaching upwards of 100 layers (He, Zhang, Ren, & Sun, 2016). It is not unusual for state-of-the-art models to have over 100 million parameters (Krizhevsky et al., 2012; Simonyan & Zisserman, 2015). With massively parallel graphics processing unit (GPU) implementations, these models are relatively fast to evaluate at test time; for example, classifying a 256×256 pixel image takes approximately 2 milliseconds with an 8-layer architecture. Training, however, can take days or weeks.

2.2 Related work

This section focuses on four major research topics in feature learning and computer vision closely linked to the current work: metric learning (Sec. 2.2.1), segmentation-aware descriptor construction (Sec. 2.2.2), the use of convnets for semantic segmentation (Sec. 2.2.3), and finally the use of conditional random fields (in Sec. 2.2.4).

2.2.1 Metric learning

The goal of metric learning is to produce features from which one can estimate the similarity between pixels or regions in the input (Frome, Singer, Sha, & Malik, 2007). Bromley, Guyon, LeCun, Sackinger, and Shah (1994) pioneered learning these descriptors in a convolutional network, for signature verification. Subsequent related work has yielded compelling results for tasks such as wide-baseline stereo correspondence (Han, Leung, Jia, Sukthankar, & Berg, 2015; Zagoruyko & Komodakis, 2015; Žbontar & LeCun, 2014). Recently, the topic of metric learning has been studied extensively in conjunction with handcrafted image descriptors (Trulls, Kokkinos, Sanfeliu, & Moreno-Noguer, 2013; Simo-Serra et al., 2015), improving the applicability of those descriptors to patch-matching problems.

Most prior work in metric learning has been concerned with the task of finding one-to-one correspondences between pixels seen from different viewpoints. The current work, in contrast, will address the task of matching all pairs of points that lie in the same region. This requires a higher degree of invariance than has previously been necessary—not only to rotation, scale, and partial occlusion, but also to objects' interior details.

2.2.2 Segmentation-aware descriptors

The purpose of a segmentation-aware descriptor is to capture the appearance of the foreground while being invariant to changes in the background or occlusions. To date, most work in this domain has relied on handcrafted segmentation-aware descriptors. For instance, soft segmentation masks (Ott &

Everingham, 2009; Leordeanu, Sukthankar, & Sminchisescu, 2012) and boundary cues (Maire, Arbeláez, Fowlkes, & Malik, 2008; Shi & Malik, 2000) have been used to augment handcrafted features, in a way that suppresses contributions from pixels likely to come from the background (Trulls et al., 2013; Trulls, Tsogkas, Kokkinos, Sanfeliu, & Moreno-Noguer, 2014). The current work will do this as well, though for learned features, and in a fashion that allows the suppression function itself to be learned.

Prior work has also investigated the use of affinity cues to improve performance on segmentation tasks (Fowlkes, Martin, & Malik, 2003; Ren & Malik, 2003; Dai, He, & Sun, 2015), but these required handcrafted algorithms for computing the affinity information, and would typically be pre-computed in a separate process. The current work is unique for learning the cues directly from image data, and for computing the affinities densely and "on the fly" within a convnet.

2.2.3 Convolutional nets for semantic segmentation

As expressed in the introduction, convnets typically produce class-accurate but low-resolution and spatially-imprecise predictions. Several methods have been proposed for addressing the spatial resolution problem. A resolution-preserving alternative to pooling is to add "holes" to the convolution filters (Chen et al., 2015), so that they cover a wider field of view with the same number of parameters. A complementary strategy is to add up-sampling stages to the network, via trainable "de-convolution" layers (Noh et al., 2015; Long et al., 2014). These techniques are effective at addressing resolution loss, but only indirectly address the issue of smoothness.

More in line with the current work, there is also research on incorporating segmentation cues into the network, to prevent the loss of spatial precision. For example, Dai et al. (2015) recently used superpixels to generate masks for convolutional feature maps, enforcing sharp contours in their outputs. The current work takes this idea further, by replacing the sparse handcrafted segmentation cues with dense learned variants.

2.2.4 Conditional random fields

The loss of spatial precision has also been addressed as a post-processing step disjoint from the convnet, by attaching a dense conditional random field (CRF) (Krähenbühl & Koltun, 2011) to the final layer of the network (Chen et al., 2015; Yu & Koltun, 2016; Zheng et al., 2015). In these works, the CRF initializes its prediction map with the one provided by the convnet, and iteratively optimizes it (through mean field inference) to maximize the label agreement between all pairs of similar pixels. One of the pixel similarity cues for the CRF is provided by a bilateral filter. The utility of the bilateral filter is explored in this work as well, but without the CRF formulation. Another important difference is that the standard bilateral filter uses Gaussians in colour and position space to compute pixel similarity, whereas the current work computes similarity for arbitrary feature vectors, and allows the features to be learned through training.

Concurrent with this work, Jampani, Kiefel, and Gehler (2016) recently explored the idea of learning high-dimensional filters and implementing them in convolutional nets. The current work takes this idea further, by combining the bilateral filter with arbitrary convolutional filters, allowing any layer (or even all layers) to perform segmentation-aware versions of their original filtering operations.

A final distinction to make from these prior works is in the width of the bilateral filter. Implementations of CRFs typically use a very wide filter, sometimes extending to the full width of the image. This necessitates that the filtering be performed in a permutohedral lattice (Krähenbühl & Koltun, 2011; Chen et al., 2015; Zheng et al., 2015). The current work, in contrast, uses relatively small filter sizes (e.g., 9×9 or smaller), allowing an implementation of bilateral filtering very similar to the efficient convolution already present in publicly available convolutional network frameworks, such as Caffe (Jia et al., 2014).

Chapter 3

Technical Approach

This chapter presents the main theoretical and technical components of segmentation-aware convolutional nets. The first part (Sec. 3.1) is focused on theory, describing the functions that compose segmentation-aware convolution. The second part (Sec. 3.2) is focused on implementation details, describing how the functions form a network, and outlining how each part was implemented efficiently. The intent with this ordering is to provide a coarse-to-fine overview of the technical approach. The chapter ends with a short summary (Sec. 3.3).

3.1 Segmentation-aware convolutional nets

The following subsections describe the three main components of the proposed approach, which build on one another. The first section (Sec. 3.1.1) characterizes the objective function for segmentation embeddings, and describes how these embeddings can be learned from per-pixel labels. The next section (Sec. 3.1.2) uses the embeddings to create per-patch soft segmentation masks, generalizing the bilateral filter. The last section (Sec. 3.1.3) shows how segmentation-aware masks can be merged with convolution filters, achieving

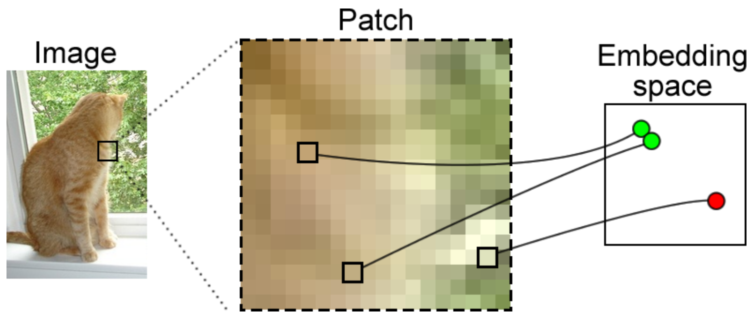


Figure 3.1: Visualization of the goal for pixel embeddings. For any two pixels sampled from the same object, the embeddings should have a small relative distance. For any two pixels sampled from different objects, the embeddings should have a large relative distance. The embedding space is illustrated in 2D, though in principle it can have any dimensionality.

segmentation-aware convolution.

3.1.1 Learning segmentation embeddings

The first goal of the current work is to train a set of convolutional layers to create an embedding function, which maps (i.e., *embeds*) pixels into a feature space $\mathbb{R}^N$, where their semantic similarity can be measured as a distance. Pixel pairs that share a semantic category should produce similar embeddings (i.e., a short distance), and pairs with different categories should produce dissimilar embeddings (i.e., a large distance). Some example 2D embeddings are illustrated in Figure 3.1. (The embeddings will eventually be implemented as 64-dimensional vectors, but the descriptions to follow will remain general.)

The embedding goal is represented in a loss function, $\mathcal{L}$, which accumulates the (inverse) quality of embedding pairs sampled across the image. In this work, pairwise comparisons are made between each pixel i and its spatial neighbours, $j \in \mathcal{N}_i$. Collecting pairs within a fixed window lends simplicity and tractability, although in general the pairs can be collected at any range. Denoting the quality of a particular pair of embeddings with ℓ_{ij}, the overall loss for an image of embeddings is defined as

$$\mathcal{L} = \sum_{i \in I} \sum_{j \in \mathcal{N}_i} \ell_{ij}, \tag{3.1}$$

where I represents the image, and $i \in I$ iterates over all the pixel indices in the image. The network is trained to minimize this loss through stochastic gradient descent.

The inner loss function, ℓ_{ij}, represents how well a pair of embeddings e_i and e_j respect the affinity goal. Pixel-wise labels are a convenient resource for quantifying this loss, since they can provide information on whether or not the pixels belong to the same region. Using this information, the distance between embeddings can be optimized according to their label parity. That is, same-label pairs can be optimized to have a small distance, and different-label pairs can be optimized to have a large distance. Denoting the label of pixel i with l_i, and the embedding at that pixel with e_i, the inner loss is defined as:

$$\ell_{ij} = \begin{cases} \max\left(|e_i - e_j| - \alpha, 0\right) & \text{if } l_i = l_j \\ \max\left(\beta - |e_i - e_j|, 0\right) & \text{if } l_i \neq l_j \end{cases}, \tag{3.2}$$

where α and β are design parameters that specify the “near” and “far” thresholds against which the embedding distances are compared, respectively. In this work, $\alpha = 0.5$, and $\beta = 2$ were used. In practice, the specific values of α and β are unimportant, so long as $\alpha \leq \beta$ and the remainder of the network can learn to compensate for the scale of the resulting embeddings (e.g., through λ

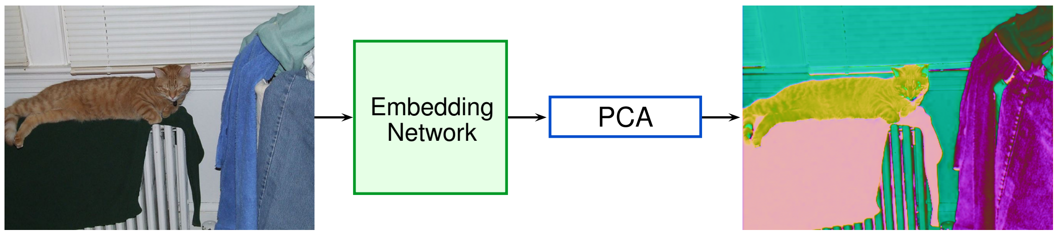


Figure 3.2: Visualization of the learned embeddings. An image (left) is sent through the embedding network, which generates a 64-channel embedding for every pixel. The embeddings were then compressed into a 3-channel image (right). The visualization reveals successful separations between most of the objects in the scene.

in Eq. 3.3).

The embedding distances can be computed with any distance function. In this work, L_1 and L_2 norms were both tried initially. Embeddings learned from the two distances were similar, but the L_1-based embeddings were found to be easier to train. Specifically, it was found that the L_1-based embeddings can safely be trained with a higher learning rate than L_2-based ones, because they are less vulnerable to the problem of exploding gradients. Therefore, the implementation in this work uses an L_1 distance for the embeddings.

Figure 3.2 shows a visualization of the embeddings after training. As will be detailed in Sec. 3.2.1, the embeddings are learned in a set of convolution layers, whose final output is a high-dimensional feature vector (e.g., 64 dimensions) for every pixel. The visualization was generated by compressing the embeddings down to three dimensions by principal component analysis (PCA), and using those three images as the colour channels of a new image.

3.1.2 Segmentation-aware bilateral filtering

The embeddings learned with the above method can provide a measure of how likely two pixels i and j belong to the same region. A soft version of this quantity can be obtained with a monotonically decreasing function of their distance,

$$m_{ij} = \exp(-\lambda|e_i - e_j|), \tag{3.3}$$

where λ is a learnable parameter specifying the hardness of this decision. Given any pixel i, one can use this function to compute soft segmentation masks of its neighbourhood. Figure 3.3 shows examples of the learned embeddings and resulting segmentation masks, and compares them with simpler masks created from photometric colour distances. In general, the learned embeddings successfully generate accurate foreground-background masks; colour-based embeddings are not quite as reliable.

A first application of these masks is to smooth the responses of a layer in a way that respects the underlying image boundaries. Given an input signal, x, one can compute a weighted average, y, as follows:

$$y_i = \frac{\sum_{j \in \mathcal{N}_i} x_{i-j} m_{ij}}{\sum_{j \in \mathcal{N}_i} m_{ij}}. \tag{3.4}$$

where x_i is the input at location i, and j ranges over the neighbourhood of i, $\mathcal{N}_i$.

Equation 3.4 has some interesting special cases, which depend on the underlying indexed embeddings e_k (for an arbitrary index k):

- if $e_k = 0$, the equation yields the average filter, or "average pooling";

- if $e_k = (k)$ the equation yields Gaussian smoothing;
- if $e_k = (k, I_k)$, where I_k denotes the image red-green-blue (RGB) vector at k, the equation yields bilateral filtering (J.-S. Lee, 1983; Aurich & Weule, 1995; Smith & Brady, 1997; Tomasi & Manduchi, 1998), which is a well-known edge-preserving smoothing technique.

Since the embeddings are learned in a convolutional net (and may learn to represent anything), Eq. 3.4 represents a generalization of all these cases.

Interestingly, a current popular algorithm for dense conditional random fields (CRFs; Krähenbühl & Koltun, 2011) also uses the bilateral filter. In the inference step for computing the CRF, the iterative update to the mean field approximation is implemented as a repeated application of two filters, one bilateral and one spatial. Pushing this a step further, Jampani et al. (2016) proposed to learn the kernel used in the bilateral filter, but keep the arguments to the similarity measure (i.e., e_i) fixed. In this work, by training the network to provide convolutional embeddings, even the arguments of the bilateral distance function can be learned.

When the embeddings are integrated into a larger network that uses them for filtering, the embedding loss function (Eq. 3.1) is no longer necessary. Since all of the terms in the normalized mask (Eq. 3.4) are differentiable, the global objective (e.g., classification accuracy) can be used to tune not only the input terms, x_i, but also the mask terms, m_{ij}, and their arguments, e_i and e_j. Therefore, the embeddings can be learned end-to-end in the network when used to create masks. In this work, the embeddings were trained first with a dedicated loss, then fine-tuned in the larger pipeline as masks.

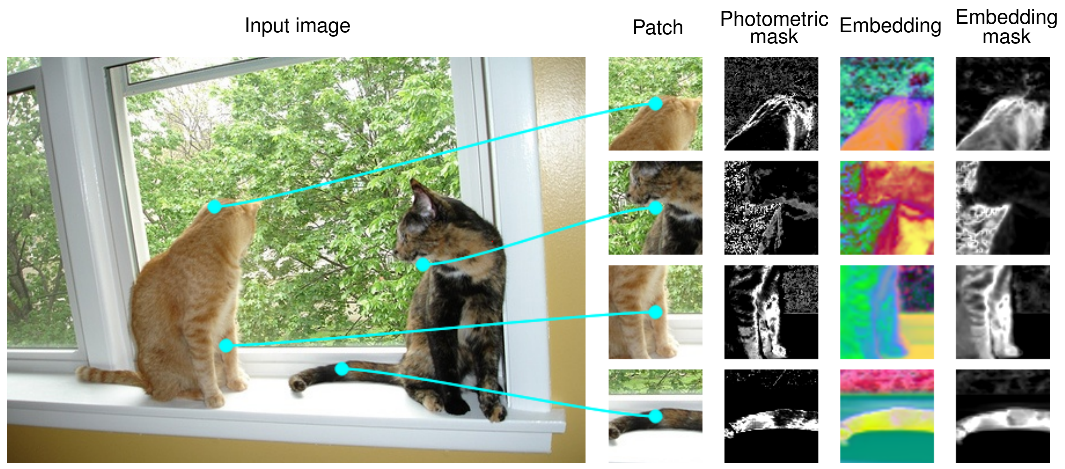


Figure 3.3: Examples of embedding-based segmentation masks, compared with photometric-based masks. For four locations in the image shown on the left, the figure shows (left-to-right): a patch extracted around that location, the mask generated by colour distances in RGB space, a visualization of the embeddings, and the embedding-based mask.

Figure 3.4 shows an overview of how segmentation-aware bilateral filtering sharpens predictions in practice.

3.1.3 Segmentation-aware convolution

The averaging described in the previous subsection can be understood as a very special case of convolution with a constant filter. The edge-preserving averaging has been repeatedly shown to yield improved spatial accuracy (J.-S. Lee, 1983; Aurich & Weule, 1995; Smith & Brady, 1997; Tomasi & Manduchi, 1998), which leads to the question of what the counterpart would be for general convolution.

Extending directly the idea of the previous section, if one writes discrete convolution of the signal, x, with a template, t, as $y_i = \sum_{j \in \mathcal{N}_i} x_{i-j} t_j$,

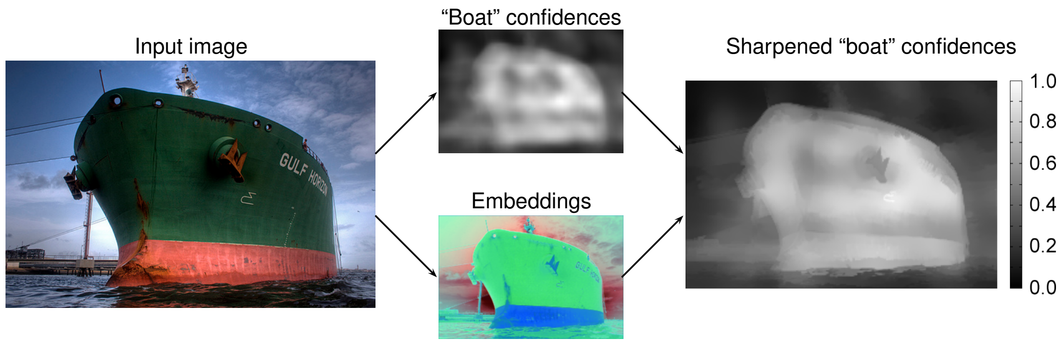


Figure 3.4: Overview of segmentation-aware bilateral filtering. Given an input image (left), a CNN typically produces a smooth prediction map (middle top). With learned pixel embeddings (middle bottom), the prediction map can be sharpened via segmentation-aware bilateral filtering (right).

segmentation-aware convolution is given by:

$$y_i = \frac{\sum_{j \in \mathcal{N}_i} x_{i-j} m_{ij} t_j}{\sum_{j \in \mathcal{N}_i} m_{ij}}, \tag{3.5}$$

i.e., a non-linear convolution, where the input signal is multiplied pointwise by the normalized local mask at every location, before forming the inner product.

Note that the mask acts as an applicability function for the signal, which makes segmentation-aware convolution a special case of *normalized convolution* (Knutsson & Westin, 1993). In general, the idea of normalized convolution is to "focus" the convolution operator on the part of the input that truly describes the input signal, avoiding the interpolation of noise or missing information. In this case, "noise" corresponds to information coming from regions distinct from the one to which node i belongs; this type of information may adversely affect the features because it comes from other objects and the background.

Any convolutional filter can be made segmentation-aware. The advantage of segmentation awareness depends on the filter. For instance, a center-surround filter might be rendered useless by the effect of the mask (since it would block input from the "surround"), whereas a filter selective to a particular shape might benefit from invariance to context. The basic intuition is that the information masked out needs to be distracting rather than helping—which calls for learning the masking functions. In this work, backpropagation is used to learn both the arguments and the softness of each layer's masking operation, so the network can always converge to the fallback option of a standard CNN.

3.2 Implementation details

This section first describes how the basic ideas of the technical approach are integrated in a CNN architecture, and then establishes details on how the individual components are implemented efficiently as convolution-like CNN layers.

3.2.1 Network architecture

There are two main parts to the architecture: one part produces embeddings, and the other does semantic segmentation (i.e., assigns object class labels to the pixels). The semantic segmentation part is a re-implementation of the popular "DeepLab" semantic segmentation network (Chen et al., 2015; specifically, the publicly released "large field-of-view" architecture). Both DeepLab and the embeddings net are based on the VGG-16 network (Simonyan & Zisserman, 2015), which is a powerful object recognition model. The VGG-16

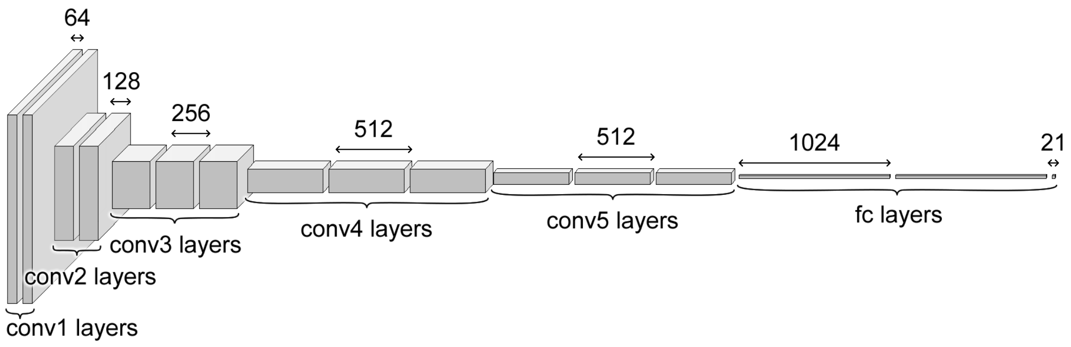


Figure 3.5: The VGG-16 network. The layer dimensionalities are annotated above the network, and the layer names are annotated below, with "conv" for convolutional, and "fc" for fully-connected. The convolution layers are grouped by the resolution of their feature maps. In all, the network has 16 layers. In the DeepLab variant of this network, the fully-connected layers are converted into convolutional ones.

network is illustrated in Figure 3.5. An overview of the full two-part architecture is shown in Figure 3.6.

The embeddings network has the following architecture. The first five layers share the design of the earliest convolutional layers in VGG-16. There is a subsampling layer after the second convolution layer and also after the fourth convolution layer, so the five convolution layers capture information at different scales. The output from each layer is sent to a pairwise distance computation (*im2dist*, detailed in Sec. 3.2.2) followed by a loss (as in Eq. 3.2), so that each layer develops embedding-like representations. The idea of using a loss at each intermediate layer is inspired by Xie and Tu (2015), who used this strategy to learn boundary cues in a CNN. The motivation behind this strategy is to provide early layers a stronger signal of the network's end goal, reducing the burden on backpropagation to carry the signal through multiple

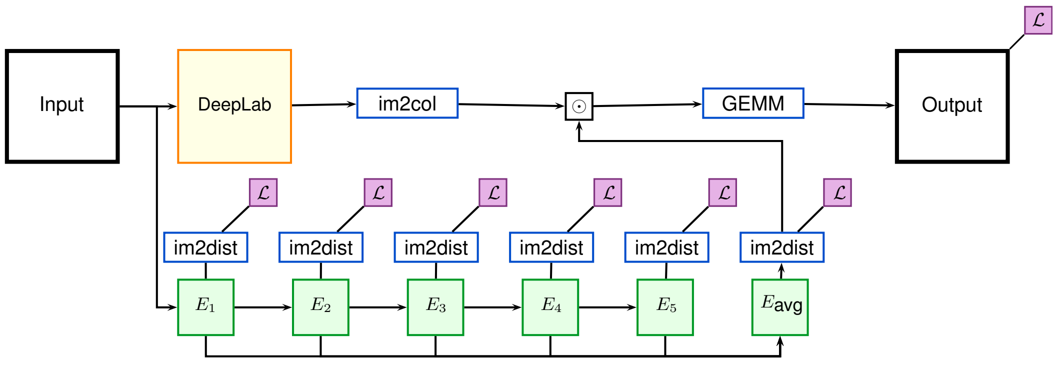


Figure 3.6: Schematic for the CNN featured in this work. The CNN combines a semantic segmentation network (DeepLab) with an embedding network. Embedding layers are indicated with boxes labelled E; the final embedding layer computes a weighted average of the other embeddings. Loss layers, indicated with $\mathcal{L}$ boxes, provide gradients for each embedding layer, and also for the final output. Outputs from DeepLab are merged with segmentation information from the embedding network, by pointwise multiplication of an *im2col* output from DeepLab, and an *im2dist* output from the embeddings; this result is sent to a general matrix multiplication (GEMM), which can achieve either segmentation-aware convolution (by multiplication with learned filter weights), or bilateral-like filtering (by multiplication with ones). This merging and multiplication can be applied to any layer of DeepLab.

layers (C.-Y. Lee, Xie, Gallagher, Zhang, & Tu, 2015). The benefits of the multi-layer loss will be investigated in the evaluation (Sec. 4.2.1).

The outputs from the intermediate embedding layers are upsampled to a common resolution, concatenated, and sent to a randomly-initialized convolutional layer with 1×1 filters. This layer learns a weighted average of the first five convolutional layers' outputs, and creates the final embedding for each pixel. The layer is trained in the same way as the intermediate layers (with *im2dist* and a loss), and the output is used as the final set of embeddings. The final embeddings are used to mask and sharpen DeepLab's intermediate representations. The most basic version of this masking involves performing segmentation-aware bilateral filtering on DeepLab's final layer outputs, just before the softmax (i.e., "FC8"); this achieves the sharpening effect illustrated in Figure 3.4. The most intrusive version of the masking involves masking the inputs to every convolutional layer in the network, converting all convolutions into segmentation-aware convolutions.

3.2.2 Efficient convolutional implementation details

This section provides the implementation details that are required to efficiently integrate the embeddings, masks, and segmentation-aware convolutions with CNNs. Source code for this work will be made available online. All new layers are implemented both for CPU and GPU in the popular deep learning framework Caffe (Jia et al., 2014).

Computing distances

Computing pairwise distances densely across the image is a computationally expensive process. The current work implements this efficiently by solving it in the same way Caffe (Jia et al., 2014) realizes convolution: via an image-to-column transformation, followed by matrix multiplication. The image-to-column transformation in convolution simply involves re-organizing the elements of each (potentially overlapping) patch into a column—this process is named *im2col*. The image-to-column transformation for distances involves computing distances within each patch, and organizing those distances into a column (ordered the same way as in *im2col*)—this new process is named *im2dist*. The two processes are illustrated and compared in Figure 3.7.

More precisely, the distance computation works as follows. For every position i in the feature-map provided by the layer below, a patch of features is extracted from the neighbourhood $j \in \mathcal{N}_i$, and local distances are computed between the central feature and its neighbours. These distances are arranged into a row vector of length K, where K is the total dimensionality of the patch. This process turns an $H \times W$ feature-map into an $H \cdot W \times K$ matrix, where each element in the K dimension holds a distance relating that pixel to the central pixel at that spatial index.

Computing masks

To convert the distances into masks, the $H \cdot W \times K$ matrix is passed through an exponential function with a specified hardness, λ. This operation realizes the mask term (Eq. 3.3). In this work, the hardness of the exponential is learned as a parameter of the CNN.

The mask matrix is then normalized (Eq. 3.4). To find the normalizing coefficients, i.e., the denominator in the normalized mask (Eq. 3.4), the mask matrix is summed across the K dimension, creating a mask sum for every spatial location. The mask values are then pointwise divided by the sums, creating the final normalized masks.

A potentially convenient alternative to the two previous steps (corresponding to Eq. 3.3 and Eq. 3.4) is to scale the $H \cdot W \times K$ matrix of distances by λ, and send the result through the softmax function

$$\sigma(z_k) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } k = 1, \ldots, K. \tag{3.6}$$

The reason this may be convenient is that many neural network frameworks

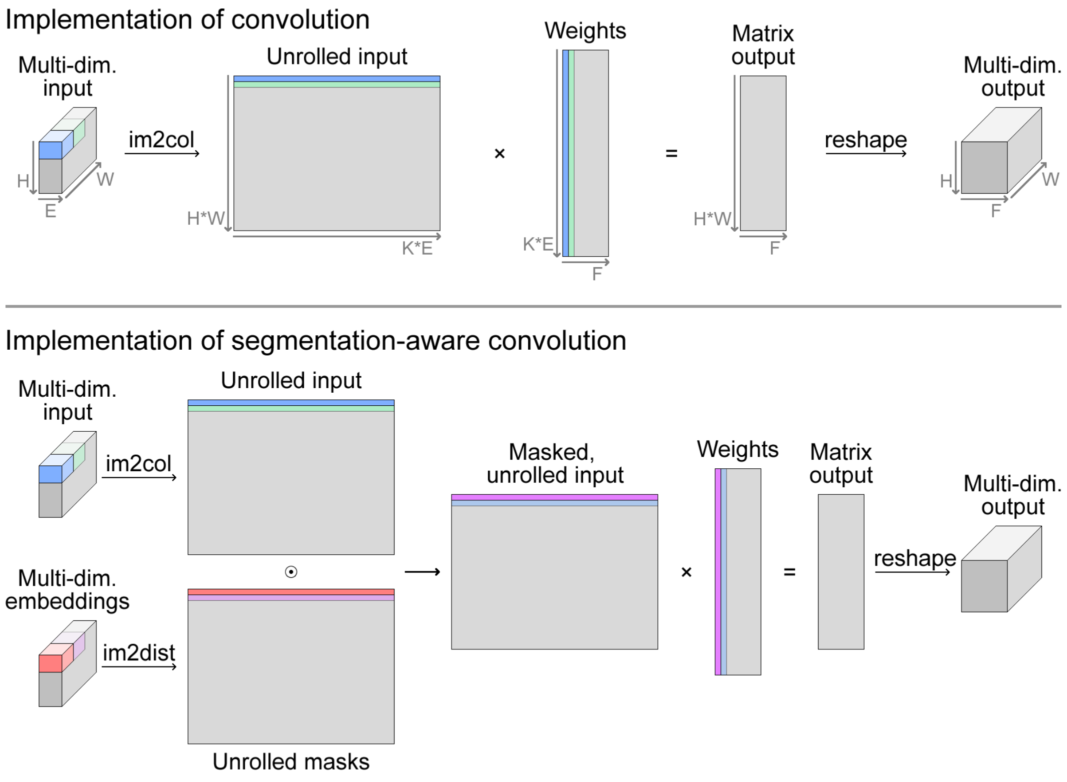


Figure 3.7: Implementation of convolution in Caffe, compared with the implementation of segmentation-aware convolution. The variables H, W denote the height and width of the input; E denotes the number of channels in the input (i.e., the "depth" of the input); K denotes the dimensionality of a patch (e.g., $K = 9$ in convolution with a 3×3 filter); F denotes the number of filters (and the dimensionality of the output). In both cases, an $H \times W \times E$ input is transformed into an $H \times W \times F$ output. Convolution works by an image-to-column transformation (i.e., "unrolling" the input), and a matrix multiplication with weights. Segmentation-aware convolution works similarly, with an image-to-column transformation on the input, an image-to-distance transformation on the embeddings, a pointwise multiplication of those two matrices, and then a matrix multiplication with weights.

(e.g., *Caffe*) already have implementations for the softmax function.

Applying masks

To perform the actual masking, the input to be masked is simply processed by *im2col* (producing another $H \cdot W \times K$ matrix), then multiplied pointwise with the normalized mask matrix. Since *im2dist* is modelled after *im2col*, the values in the matrices are aligned so that each pixel of a patch gets multiplied by its similarity to the central pixel.

From the product, segmentation-aware bilateral filtering is merely a matter of summing across the K dimension, or equivalently, multiplying with a $K \times 1$ matrix of ones, producing an $H \cdot W \times 1$ matrix that can be reshaped into dimensions $H \times W$.

Segmentation-aware convolution (Eq. 3.5) simply requires multiplying the $H \cdot W \times K$ masked values with a $K \times F$ matrix of weights, where F is the number of convolution filters. The result of this multiplication can be reshaped into F different $H \times W$ feature maps.

Computing loss

The loss function for the embeddings (3.1) is implemented using similar computational techniques. First, the label image is processed with a new layer named *im2parity*. This creates an $H \cdot W \times K$ matrix, where for each pixel i, the K-dimensional row vector specifies (with $\{0, 1\}$) whether or not each local neighbour $j \in \mathcal{N}_i$ has the same label as the central pixel. The result of this process can then be straightforwardly combined with the $H \cdot W \times K$ result of *im2dist*, to penalize each distance according to the correct loss case (as in

Eq. 3.2).

3.3 Summary

This chapter presented the technical approach, which consists of three main components: embeddings, segmentation-aware bilateral filtering, and segmentation-aware convolution. The embeddings are pixel-wise features, which map pixels into a space where their relative distance can be used to infer a local foreground-background segmentation. The embeddings are used to create a mask for every patch in the image, which highlights the “foreground” relative to the central pixel of each patch. Using the masks as filters (i.e., taking a dot product of the mask and the patch at each location) accomplishes segmentation-aware bilateral filtering. This can be used to sharpen the outputs of a convnet. Using the masks together with convolution (so that each convolution filter is first multiplied pointwise with the local mask) achieves segmentation-aware convolution. This has the effect of spatially “focusing” the convolution filters on the foreground at every location, which is expected to improve the spatial precision of the resulting outputs.

The embeddings are learned in a standard convolutional architecture (a simplified VGG-16 architecture). The segmentation-aware bilateral filtering and convolution are integrated into a strong baseline semantic segmentation CNN (DeepLab), with no other changes to the architecture. These networks are implemented in a popular deep learning framework (Caffe). The low-level implementation details are based closely on the optimized implementation of convolution from that framework.

Chapter 4

Experiments

This chapter presents an empirical evaluation of the proposed approach. The first section provides an overview of the datasets and metrics used (Sec. 4.1). The second section presents the evaluation of the proposed approach against a standard baseline (Sec. 4.2). The third section discusses the significance of these results (Sec. 4.3).

4.1 Overview

This section describes the dataset (Sec. 4.1.1), and the metrics used in the evaluation (Sec. 4.1.2), to clarify and motivate the interpretations to follow in the evaluation.

4.1.1 Datasets

The evaluation in this thesis is centered on one standard benchmark, the PASCAL VOC 2012 semantic segmentation challenge (Everingham, Van-Gool, Williams, Winn, & Zisserman, 2012). Since the PASCAL dataset is too small to train a large CNN without overfitting, two additional datasets are used for their training data. This selection and usage is consistent with many works in semantic segmentation, including this work's baseline, Chen et al. (2015).

PASCAL VOC 2012

The dataset used for evaluation is the PASCAL Visual Object Classes (VOC) 2012 dataset (Everingham et al., 2012), augmented with additional images from Hariharan, Arbeláez, Bourdev, Maji, and Malik (2011). The complete dataset consists of 10,582 training images, 1,449 validation images, and 1,456 testing images. All images are annotated with object class labels, on a per-pixel basis. The annotations are stored in uncompressed PNG images, using the pixel values to denote the object class identification number at each location. These labels are typically visualized as images, with a colourmap provided in the VOC development kit. Figure 4.1 shows a sample of the images in the dataset, and corresponding coloured label images. There are twenty object classes, and one "background" class (which contains other objects). The object classes are: *airplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, dining table, dog, horse, motorbike, person, potted plant, sheep, sofa, train,* and *tv/monitor.* The dataset allows some slack for hard-to-categorize pixels, e.g., at object boundaries; these hard pixels are marked as "void", and do not get scored in the evaluation.

Annotations for the training and validation sets are available for use and inspection, but annotations for the test set are held out, and stored on a secure server. This is done to standardize the evaluation across all works that use the dataset. Evaluation on the test set is only possible by submitting results in a pre-specified format to the server. The frequency of submissions is strictly

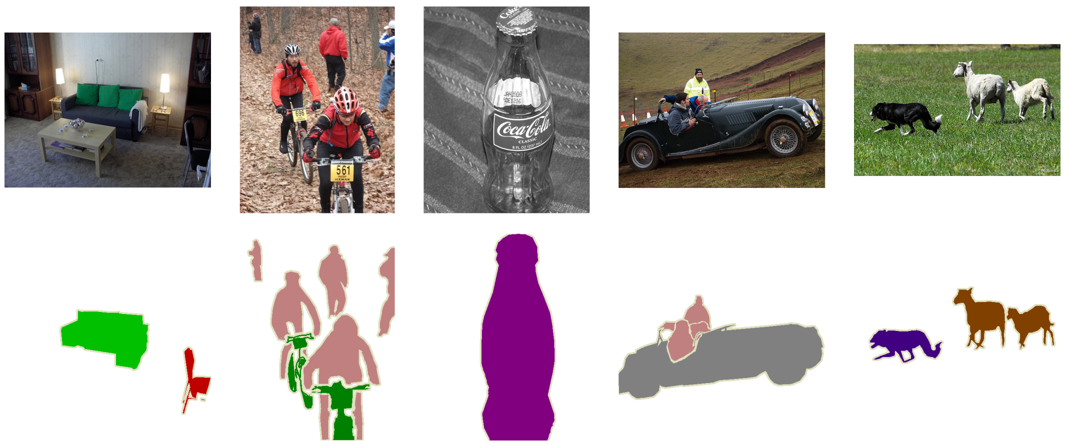


Figure 4.1: Overview of the PASCAL VOC 2012 dataset. Example images are shown in the top row, and the corresponding colour-mapped label images are shown in the bottom row. The white areas in the label images are categorized as "background"; the light edges noticeable on some object contours are "void" regions, which are not scored in the evaluation.

limited to two submissions per week, to discourage tuning of the algorithms based on test performance. This is currently the standard benchmark for semantic segmentation algorithms (Long et al., 2014; Chen et al., 2015; Zheng et al., 2015; Yu & Koltun, 2016; Noh et al., 2015).

MS COCO

One of the two datasets used for training only (i.e., no evaluation), is the Microsoft Common Objects in Context (MS COCO) dataset (Lin et al., 2014). The dataset consists of 82,783 training, 40,504 validation, and 40,775 testing images, with per-pixel labels for 80 object classes and one background class. The classes cover all of the PASCAL VOC object classes, plus a variety of additional animals, vehicles, sports equipment, kitchenware, food, furniture, appliances, and electronics. In this work, the MS COCO training and validation sets were simply concatenated (making a total of 123,287 images), and used as a “pre-training” set. All CNNs in this work were trained on MS COCO before they were trained on PASCAL VOC.

ImageNet

The second dataset used only for training is ImageNet (Russakovsky et al., 2015). This dataset consists of 1.3 million training images, 50,000 validation images, and 100,000 testing images. This dataset is labelled on a per-image basis, with a single object label per image. A total of 1000 object classes are represented in the labels. This dataset was used to train the VGG-16 network (Simonyan & Zisserman, 2015). The learned weights from the fully-trained VGG-16 were used to initialize the weights of both the baseline (DeepLab) and the proposed approach. This sort of initialization is equivalent to pre-training the networks on ImageNet for object recognition. Training a VGG-16 network on ImageNet takes two to three weeks (on four GPUs working in parallel), but trained models have been publicly released online.

4.1.2 Metrics

Performance on the PASCAL VOC 2012 semantic segmentation challenge is assessed on the pixel-wise *intersection over union* (IOU) for each class separately (Everingham, Van Gool, Williams, Winn, & Zisserman, 2010). The sets in the IOU are the predicted labels and the correct labels for a single class; the complete calculation, therefore, requires finding the size of the intersection (i.e., the number of pixels that were correctly predicted to belong to the class), then calculating the size of the union (i.e., the number of correct predictions, the number of incorrect predictions, and the number of missed pixels of the class), and dividing the first quantity by the second. This can be written as follows:

$$\text{accuracy} = \frac{\text{true positives}}{\text{true positives} + \text{false positives} + \text{false negatives}}, \tag{4.1}$$

where “positives” and “negatives” depend on the class. The mean accuracy over all classes is the standard measure of overall performance.

For reference, the first row of Table 4.1 shows the accuracy of the replicated baseline DeepLab CNN, at 66.33% IOU.

4.1.3 Training details

Following the implementation of Chen et al. (2015), and additional details released in follow-up work (Papandreou, Chen, Murphy, & Yuille, 2015), the baseline CNN was initialized from a fully-convolutional version of VGG-16, then trained in two stages. In the first stage, it was trained for 24,000 iterations on MS COCO, starting with a learning rate of 0.01 on the final classifier, and 0.001 on all other layers, and multiplying these rates by 0.1 every 8,000 iterations. In the second stage, the network was trained for 6,000 iterations on PASCAL VOC, beginning with the same learning rates as the first stage, but multiplying by 0.1 every 2000 iterations. Both stages used a momentum factor of 0.9 on the learning rate, and a batch size of 30 images for the gradient descent. The data was augmented by taking random 321×321 crops, and performing left-right mirror flipping at random for every image. Additional training details can be found in Chen et al. (2015).

The embeddings were initialized from VGG-16, then trained for 20,000 iterations on MS COCO, starting with a learning rate of 1e-5 on the final embedding, and 1e-6 on all other layers. The learning rate was gradually decreased to 0 following the polynomial decay function $(1 - \text{iter}/\text{max\_iter})^{0.9}$. The batch size in this case was 10 images. All other aspects of training matched the baseline training procedure.

The segmentation-aware convolution network was initialized from the MS COCO-trained baseline (DeepLab) network, then infused with embeddings, and trained on the PASCAL VOC dataset for 12,000 iterations, multiplying the learning rate by 0.1 every 4,000 iterations. All other aspects of training matched the baseline training procedure.

4.2 Evaluation

This section steps through an evaluation of the proposed approach on the Pascal VOC 2012 dataset. Parallel to the initial presentation of the technical approach (Sec. 3.1), the evaluation first focuses on the embeddings (Sec. 4.2.1), then segmentation-aware bilateral filtering (Sec. 4.2.2), and finally on segmentation-aware convolution (Sec. 4.2.3).

4.2.1 Embeddings

As noted earlier, any neural network architecture implicitly comprises hundreds of hyperparameters—design parameters that control the complexity and capacity of the network. For this reason, it is important to validate any major choices that are not supported by prior work. Since the embeddings network is based on a standard architecture (VGG-16), the evaluation here will focus on the dimensionality of the final embeddings, and the supervision technique.

Since the main interest is the quality of the embeddings in the context of segmentation-aware masking and filtering, the evaluation of this section will center on the simplest application in that context: sharpening the prediction maps produced by DeepLab through bilateral filtering, as a post-processing step. Specifically, the embeddings will be used to generate bilateral filtering masks, which will replace each prediction with a weighted average of a local 9×9 region from the prediction map.

As a baseline for the embeddings, the accuracy obtainable with RGB-based bilateral filtering is shown in Table 4.1, at 66.97%. Note that the performance boost from RGB-based bilateral filtering ($\approx 0.6\%$) is itself a minor contribution of this work, since the bilateral filtering is made possible through the proposed implementation.

Next, Table 4.1 shows mean IOU performance across embeddings of various dimensions. The results show a large difference between 32-dimensional embeddings and 64-dimensional embeddings, and slight improvements as the dimensionality increases further. Although the best embedding is found at 256 dimensions, the remainder of this work will use the 64-dimensional embeddings, as they require substantially less memory (especially in *im2dist*), and offer nearly the same performance. Compared to RGB, the learned 64-dimensional embeddings are almost equal in quality (66.98% vs. 66.97%).

Training the embeddings with top-down supervision would be faster and simpler than using the multi-layer supervision suggested earlier (in Sec. 3.2.1, motivated by Xie & Tu, 2015). However, Table 4.1 shows that the multi-layer supervision yields a slight improvement in performance, bringing the accuracy of the learned embedding to 67.00%. Since this strategy only affects training time, this is likely a worthwhile investment.

Note that the learned embeddings are still not substantially better than RGB (which had an accuracy of 66.97%). This attests to the strength of colour as a region embedding. However, whereas the RGB vector is fixed, the embeddings can be improved further, through fine-tuning within the larger network where they are used. The results with segmentation-aware convolution (in Sec. 4.2.3) make use of this fine-tuning process.

4.2.2 Segmentation-aware bilateral filtering

Although segmentation-aware bilateral filtering is only a special case of segmentation-aware convolution, it is interesting to explore how much of an improvement in accuracy is possible by tuning the hyperparameters that define how (and to what extent) the technique is applied. Experimentation on the technique as a post-processing step (as considered here) is especially appealing, because the effects of various modifications can be observed without requiring new models to be trained. The main design parameter to consider in this context is the number of times to apply the filter.

Once the embeddings and masks are computed, it is trivial to run the masking process repeatedly, though this is limited by memory constraints. Applying the process multiple times is expected to improve performance, by strengthening the contribution from similar neighbours in the radius. This

Table 4.1: PASCAL VOC 2012 validation results for the various considered approaches, compared against the baseline. The baseline is shown at the top; subsequent methods are grouped by the aspect they investigate. In order, the table summarizes the independent effects of: embedding dimensionality, supervision style, repeated application of the filter, segmentation-aware layers, and the combination of segmentation-aware layers and bilateral filtering.

Method	IOU (%)
Baseline (DeepLab with no modifications)	66.33
with 9×9 RGB bilateral filter	66.97
with 32-dim 9×9 seg.-aware bilateral filter	66.84
with 64-dim 9×9 seg.-aware bilateral filter	66.98
with 128-dim 9×9 seg.-aware bilateral filter	67.01
with 256-dim 9×9 seg.-aware bilateral filter	67.02
with 64-dim 9×9 filter and multi-layer supervision	67.00
with 64-dim 9×9 segmentation-aware bilateral filter $\times 2$	67.36
with 64-dim 9×9 segmentation-aware bilateral filter $\times 4$	67.68
with FC6 segmentation-aware	67.40
with all layers segmentation-aware	67.94
with all layers seg.-aware + 9×9 seg.-aware bilateral filter	68.00
with all layers seg.-aware + 7×7 seg.-aware bilateral filter $\times 2$	**68.57**
with all layers seg.-aware + 5×5 seg.-aware bilateral filter $\times 4$	68.52

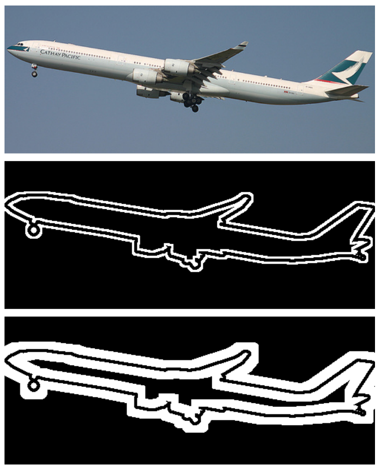
parameter also effectively allows information from a wider radius to contribute to each prediction: after filtering with 9×9 filters, each new 9×9 region effectively contains information from an 17×17 area in the original input. Table 4.1 shows the results of applying a 9×9 filter once, twice, and four times; four is the maximum possible number of times given a 12-gigabyte (NVIDIA Tesla K-40) GPU. As expected, repeating the application of the filter improves performance substantially.

4.2.3 Segmentation-aware convolution

For segmentation-aware convolution, the main question is: which stages of the network should be made segmentation-aware? Guided by the intuition that foreground-background segmentation is only useful when the receptive field of a neuron covers a relatively wide region, one might choose to only make the higher convolution layers segmentation-aware. The FC6 layer is a good candidate layer for this strategy, since it is the highest convolution layer in the network (excluding layers with filters of size 1×1). On the other hand, one can make every layer segmentation-aware, and allow learning to determine the layer-wise usage of the technique. Table 4.1 shows the results from both strategies: making FC6 segmentation-aware yields a substantial improvement over the baseline ($\approx$ 1.1%); making all layers segmentation-aware yields an even stronger improvement ($\approx$ 1.6%).

Finally, since segmentation-aware bilateral filtering and segmentation-aware convolution are not mutually exclusive, one can combine the two techniques for still better results. Table 4.1 shows the effect of sharpening the predictions produced by the segmentation-aware network. As before, memory constraints limit the application of this technique, to such an extent in this case that the filter size must be decreased to 7×7 for two repeated applications, and to 5×5 for four repeated applications. The best performance is found in the 7×7 case, at 68.57% ($\approx$ 2.2% over the baseline).

It was hypothesized earlier that these techniques would especially provide gains in accuracy near object boundaries. To see whether this is the case, one can compute each method's accuracy within "trimaps" that extend from the



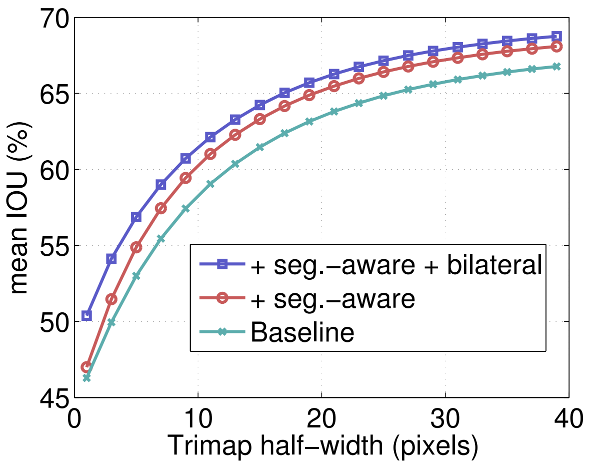


Figure 4.2: Performance near object boundaries ("trimaps"). Example trimaps are visualized (in white) for the image in the top left; the trimap of half-width three is shown in the middle left, and the trimap of half-width ten is shown on the bottom left. Note the thin black contour within each trimap is the actual object boundary, which is marked "void" in PASCAL VOC and not evaluated for accuracy (since precise boundaries are ambiguous). Mean IOU performance of the baseline and two segmentation-aware variants are plotted (right) for trimap half-widths 1-40.

objects' boundaries. A trimap is a narrow band (of a specified half-width) that surrounds a boundary on either side; measuring accuracy exclusively within this band can help separate within-object accuracy from on-boundary accuracy (Chen et al., 2015). Figure 4.2 (left) shows visualizations of trimaps, and (right) plots accuracies as a function of trimap width. The results show that segmentation-aware convolution offers its main improvement slightly away from the boundaries (i.e., beyond 10 pixels), while bilateral filtering offers its largest improvement very near the boundary (i.e., within 5 pixels).

The best technique arrived at on the validation set (i.e., all layers segmentation-aware, plus 7×7 segmentation-aware bilateral filtering applied twice) was sub-

Table 4.2: PASCAL VOC 2012 test results for the baseline approach and the final proposed segmentation-aware approach.

Method	IOU (%)
Baseline (DeepLab with no modifications)	66.96
with all layers seg.-aware + 7×7 seg.-aware bilateral filter $\times 2$	**69.01**

mitted to the PASCAL VOC server for evaluation on the official test set. As shown in Table 4.2, the proposed approach achieved approximately a 2.1% gain in mean IOU accuracy over the baseline (achieving 69.01%, compared with 66.96%).

Visualizations of results on the PASCAL VOC validation set are shown in Figure 4.3. Qualitatively, the baseline results appear blurred and blob-like, though the class detections are accurate. The results produced by the best proposed approach fit the contours of the underlying objects slightly better, and discard some extraneous off-object detections.

4.3 Discussion

Overall, the results attained from the segmentation-aware modifications show a substantial improvement over the baseline CNN. In terms of training embeddings, it was shown that higher dimensionality on the final embedding improves results slightly, though not far beyond the results achievable with RGB embeddings. Multi-layer supervision was also found to improve the embeddings slightly, which suggests that training embeddings is more akin to training a boundary detector (like Xie & Tu, 2015, which showed multi-layer supervision is critical) than training a semantic segmentation net (like Chen et al., 2015, which only used top-down supervision). This suggests that perhaps additional training strategies could be adopted from boundary detection literature.

More importantly, the results showed that both bilateral filtering and segmentation-aware convolution are effective ways of improving accuracy in semantic segmentation. Furthermore, the two techniques can be combined to improve performance further still. Overall, the segmentation-aware modifications achieved a 2.1% gain in accuracy over the baseline, as evaluated on the standard PASCAL VOC semantic segmentation challenge. Other works have attained similar-magnitude improvements with, for example, multi-scale context aggregation (Yu & Koltun, 2016), learned de-convolution layers (Long et al., 2014), and CRFs (Chen et al., 2015). The especially promising aspect of the improvement presented in this work is that it is complementary to most other currently-pursued approaches. That is, almost all semantic segmentation pipelines use convolution layers, and any convolution layer can be made segmentation-aware, meaning that the improvements in this work could potentially transfer to a wide variety of state-of-the-art semantic segmentation pipelines.

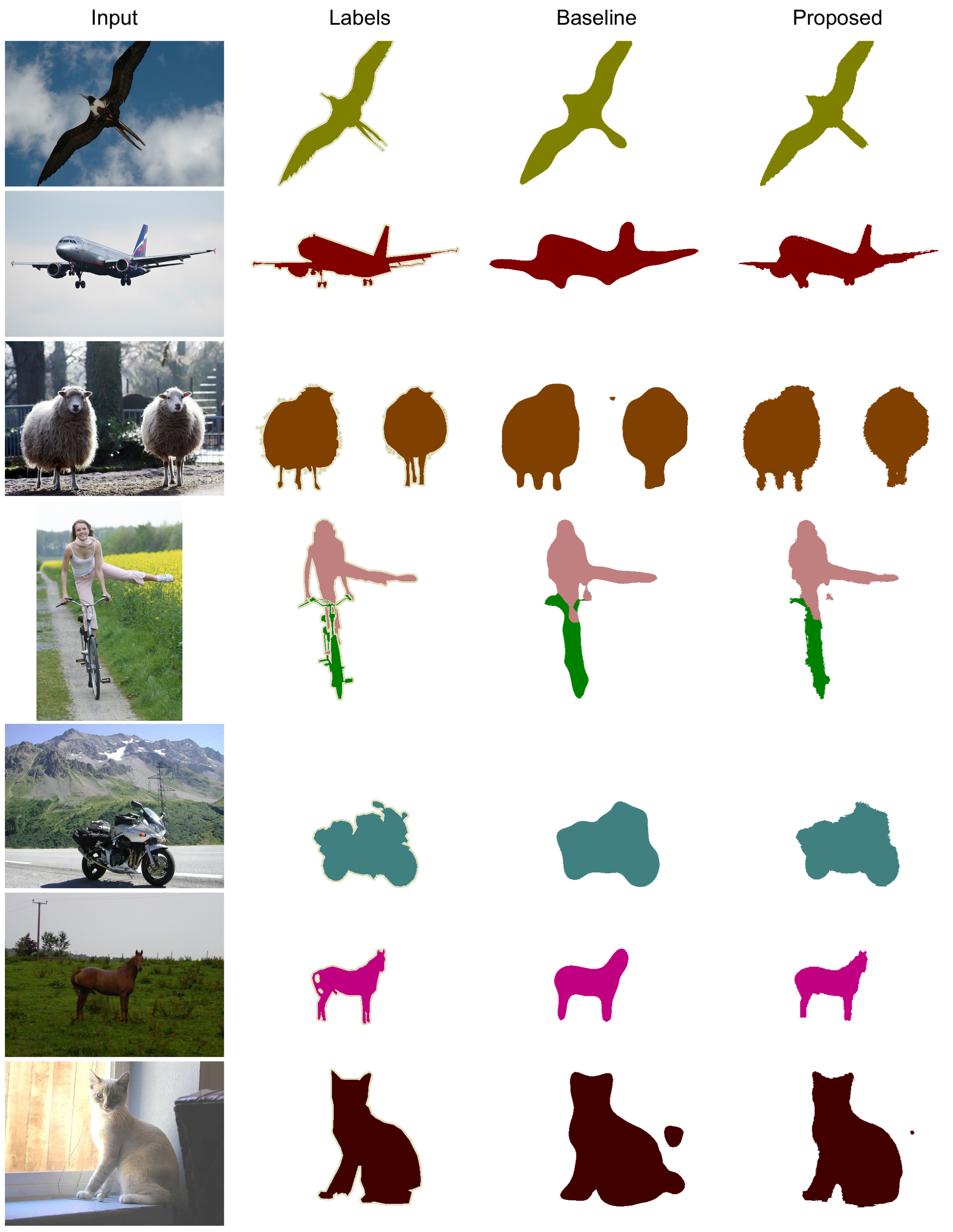


Figure 4.3: Visualizations of the results. These are colour-mapped semantic segmentations for the PASCAL VOC 2012 validation set, produced by the baseline (DeepLab), the proposed approach (using segmentation-aware convolution and bilateral filtering), and finally the provided labels.

Chapter 5

Conclusion

This chapter summarizes the main contributions of the thesis (Sec. 5.1), and lays out some interesting directions for future work (Sec. 5.2).

5.1 Thesis summary

This thesis presented segmentation-aware CNNs as the culmination of three closely-related contributions. First, it was shown that convolutional layers can be trained to produce class-invariant segmentation cues, providing dense embeddings of local foreground/background information. Second, it was shown that efficient pairwise distance computation between the learned embeddings provides a route to constructing a segmentation-aware bilateral filter. Third, the bilateral filter was combined with convolutional filters, achieving a learnable variant of normalized convolution in the CNN.

The complete architecture is named a segmentation-aware CNN, because it can adjust its behaviour on a per-pixel basis according to local segmentation cues. This technique directly addresses the issue of *smooth predictions* in CNNs. Compared to the outputs of standard CNNs, segmentation-aware

CNNs produce prediction maps with substantially better spatial accuracy.

Empirical results demonstrated that modifying a strong baseline semantic segmentation CNN with segmentation-aware convolutions systematically improves performance on a standard benchmark, as does segmentation-aware bilateral filtering. The best results are obtained by combining the two techniques.

The contributions are implemented efficiently in the popular Caffe deep-learning framework, making it straightforward to augment existing dense prediction techniques with segmentation-aware convolutions.

5.2 Future work

There are a variety of ways to expand on the current work, both in applications and in the technical approach. This section will first outline some potentially interesting applications and experiments enabled by the current work, then describe some extensions to the core technical approach.

Many potential applications of segmentation-aware convolution are fairly obvious: the success of the technique in semantic segmentation suggests that it may yield improvements in other pixel-wise prediction tasks, such as optical flow estimation, surface normal estimation, and depth estimation. Some of these tasks may even be a better fit for the technique than semantic segmentation. In particular, discarding information from the context is occasionally counter-productive for object recognition (e.g., ignoring the pasture around a cow may add difficulty to recognizing the cow); this is perhaps less frequently the case in calculating, for example, surface normals. A segmentation-aware convnet may therefore use its masks to a greater extent in these other dense prediction tasks, and potentially demonstrate an even clearer improvement over standard CNN baselines.

Another promising direction for experimentation concerns finding the optimal use of the segmentation-aware bilateral filter. The bilateral filter was primarily presented here as a special case of segmentation-aware convolution, but it is a powerful technique in its own right. A useful advantage of the technique is that it is parameter-free (given the embeddings). In this work, the bilateral filter was only used experimentally to sharpen the final prediction map produced by a CNN; an equally interesting direction would be to sharpen the intermediate feature maps produced within the CNN. It is plausible that every layer would benefit from sharper input feature maps. This might provide an additional route to countering the "smooth predictions" issue. Since the bilateral filter in this work is differentiable, the network can be trained end-to-end with this sharpening technique implemented throughout.

One extension to the technical approach, which has clear promise, is to allow the mask scaling coefficient, λ in Eq. 3.3, to vary across filters within a convolutional layer. This is likely a valuable point of flexibility, because the filters within a layer are responsible for a wide variety of tasks, and these tasks may understandably place different requirements on the presence of context. By using only a single scaling coefficient per layer, the current implementation essentially takes the average of all these demands. Unfortunately, this extension does not appear to be a trivial modification. As detailed earlier (in Sec. 3.2.2), the proposed implementation involves *im2col* on the image, an *im2dist* on the embeddings, a pointwise product, and a matrix multiplication with weights. A critical point in the efficiency of this implementation is that only a single matrix multiplication is required, no matter how many convolution filters are involved. That is, convolution is achieved by the multiplication of the $H \cdot W \times K$ matrix of masked patches with a $K \times F$ matrix of weights, where H, W are height and width, K is the size of the filter, and F is the number of filters. Allowing a separate scaling coefficient for each filter requires F separate versions of the $H \cdot W \times K$ matrix, and F matrix multiplications. Besides the additional computational cost, this quickly multiplies the memory requirements beyond current hardware limits. For instance, F in the VGG-16 "conv4" layer is 512, meaning the memory requirements of that layer would be multiplied by 512.

A second extension to the technical approach is to use segmentation-aware bilateral filtering to generalize conditional random fields. As mentioned in the introduction (Sec. 2.2.4), CRFs have recently been used in semantic segmentation as a post-processing step, to sharpen the predictions produced by the convnet. To achieve this, the CRF iteratively refines the label agreement between pairs of similar pixels, primarily via repeated applications a wide bilateral filter (Krähenbühl & Koltun, 2011). There exists some work on framing the steps of a CRF as differentiable modules of a recurrent neural network (Zheng et al., 2015); the contributions of this thesis could help to take that work several steps further. In particular, there are two components of the basic CRF, which are normally held fixed, that the current work could render into learnable parameters.

The first component of the CRF to replace is the similarity metric used in the bilateral filter. As demonstrated earlier (in Sec. 3.1.2), the similarity metric (normally distance in position and colour space) can be an arbitrary embedding, learned through backpropagation. Assuming that the learned embeddings are superior to RGB in that context, carrying out this replacement should improve the performance of the CRF. The second replaceable component is the filter itself. It was shown (in Sec. 3.1.3) that segmentation-aware bilateral filtering is merely a special case of segmentation-aware convolution, in which the filter is all ones. Converting the bilateral filter in CRFs into a learnable segmentation-aware convolution should provide an additional improvement in performance.

There are, however, technical challenges associated with these modifications. The foremost challenge arises from the width of the bilateral filter typically used in CRFs, which sometimes extends to the full width of the image (e.g., on the range of 300×300, while this work considered 9×9 filters and smaller). Although achieving segmentation-aware convolution with a dense filter of that size is likely infeasible, it would be interesting to investigate the performance gains attainable with smaller filters.

References

Aurich, V., & Weule, J. (1995). Non-linear gaussian filters performing edge preserving diffusion. In *Proceedings of the DAGM Symposium* (pp. 538–545).

Bishop, C. M. (1995). *Neural networks for pattern recognition.* Oxford university press.

Bromley, J., Guyon, I., LeCun, Y., Sackinger, E., & Shah, R. (1994). Signature verification using a "siamese" time delay neural network. In *Proceedings of Neural Information Processing Systems.*

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2015). Semantic image segmentation with deep convolutional nets and fully connected CRFs. In *International Conference on Learning Representations.*

Dai, J., He, K., & Sun, J. (2015). Convolutional feature masking for joint object and stuff segmentation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition.*

Dosovitskiy, A., Fischer, P., Ilg, E., Häusser, P., Hazırbaş, C., Golkov, V., ... Brox, T. (2015). FlowNet: Learning optical flow with convolutional networks. In *Proceedings of IEEE International Conference on Computer Vision.*

Eigen, D., Puhrsch, C., & Fergus, R. (2014). Depth map prediction from a single image using a multi-scale deep network. In *Proceedings of Neural Information Processing Systems.*

Everingham, M., Van Gool, L., Williams, C. K., Winn, J., & Zisserman, A. (2010). The PASCAL Visual Object Classes (VOC) challenge. *International Journal of Computer Vision*, *88*(2), 303–338.

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

Fowlkes, C., Martin, D., & Malik, J. (2003). Learning affinity functions for image segmentation: Combining patch-based and gradient-based approaches. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition.*

Frome, A., Singer, Y., Sha, F., & Malik, J. (2007). Learning globally-consistent local distance functions for shape-based image retrieval and classification. In *Proceedings of IEEE International Conference on Computer Vision.*

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, *36*(4), 193–202.

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition.*

Han, X., Leung, T., Jia, Y., Sukthankar, R., & Berg, A. (2015). MatchNet: Unifying feature and metric learning for patch-based matching. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition.*

Hariharan, B., Arbeláez, P., Bourdev, L., Maji, S., & Malik, J. (2011). Semantic contours from inverse detectors. In *Proceedings of IEEE International Conference on Computer Vision.*

Harley, A. W., Derpanis, K. G., & Kokkinos, I. (2016). Learning dense convolutional embeddings for semantic segmentation. In *International Conference on Learning Representations.*

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition.*

Jampani, V., Kiefel, M., & Gehler, P. V. (2016). Learning sparse high dimensional filters: Image filtering, dense CRFs and bilateral neural networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition.*

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., ... Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on multimedia* (pp. 675–678).

Karpathy, A., & Fei-Fei, L. (2015). Deep visual-semantic alignments for generating image descriptions. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition.*

Knutsson, H., & Westin, C.-F. (1993). Normalized and differential convolution. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition.*

Krähenbühl, P., & Koltun, V. (2011). Efficient inference in fully connected CRFs with Gaussian edge potentials. In *Proceedings of Neural Information Processing Systems.*

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Proceedings of Neural Information Processing Systems.*

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*, 436–444.

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

Lee, C.-Y., Xie, S., Gallagher, P., Zhang, Z., & Tu, Z. (2015). Deeply-supervised nets. *Proceedings of AAAI Conference on Artificial Intelligence*, *2*(3), 6.

Lee, J.-S. (1983). Digital image smoothing and the sigma filter. *Computer Vision, Graphics, and Image Processing*, *24*(2), 255–269.

Leordeanu, M., Sukthankar, R., & Sminchisescu, C. (2012). Efficient closed-form solution to generalized boundary detection. In *Proceedings of European Conference on Computer Vision* (pp. 516–529).

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. In *Proceedings of European Conference on Computer Vision* (pp. 740–755).

Long, J., Shelhamer, E., & Darrell, T. (2014). Fully convolutional networks for semantic segmentation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition.*

Maire, M., Arbeláez, P., Fowlkes, C., & Malik, J. (2008). Using contours to detect and localize junctions in natural images. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition.*

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *Proceedings of International Conference on Machine Learning* (pp. 807–814).

Noh, H., Hong, S., & Han, B. (2015). Learning deconvolution network for semantic segmentation. In *Proceedings of IEEE International Conference on Computer Vision.*

Ott, P., & Everingham, M. (2009). Implicit color segmentation features for pedestrian and object detection. In *Proceedings of IEEE International Conference on Computer Vision.*

Papandreou, G., Chen, L.-C., Murphy, K., & Yuille, A. L. (2015). Weakly- and semi-supervised learning of a a deep convolutional network for semantic image segmentation. In *Proceedings of IEEE International Conference* *on Computer Vision.*

Ren, X., & Malik, J. (2003). Learning a classification model for segmentation. In *Proceedings of IEEE International Conference on Computer Vision.*

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. *Nature*, *323*, 533–536.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, *115*(3), 211-252.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, *61*, 85–117.

Shi, J., & Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*(8), 888–905.

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations.*

Simo-Serra, E., Trulls, E., Ferraz, L., Kokkinos, I., Fua, P., & Moreno-Noguer, F. (2015). Discriminative learning of deep convolutional feature point descriptors. In *Proceedings of IEEE International Conference on Computer Vision.*

Smith, S. M., & Brady, J. M. (1997). SUSAN – A new approach to low level image processing. *International Journal of Computer Vision*, *23*(1), 45–78.

Sutskever, I., Martens, J., Dahl, G. E., & Hinton, G. E. (2013). On the importance of initialization and momentum in deep learning. In *Proceedings of International Conference on Machine Learning* (Vol. 28, pp. 1139–1147).

Tomasi, C., & Manduchi, R. (1998). Bilateral filtering for gray and color images. In *Proceedings of IEEE International Conference on Computer Vision.*

Trulls, E., Kokkinos, I., Sanfeliu, A., & Moreno-Noguer, F. (2013). Dense segmentation-aware descriptors. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition.*

Trulls, E., Tsogkas, S., Kokkinos, I., Sanfeliu, A., & Moreno-Noguer, F. (2014). Segmentation-aware deformable part models. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition.*

Wang, X., Fouhey, D., & Gupta, A. (2015). Designing deep networks for surface normal estimation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition.*

Werbos, P. J. (1982). Applications of advances in nonlinear sensitivity analysis. In *System modeling and optimization: Proceedings of the 10th IFIP conference* (pp. 762–770). New York: Springer-Verlag.

Xie, S., & Tu, Z. (2015). Holistically-nested edge detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition.*

Yu, F., & Koltun, V. (2016). Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representations.*

Zagoruyko, S., & Komodakis, N. (2015). Learning to compare image patches via convolutional neural networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition.*

Žbontar, J., & LeCun, Y. (2014). Computing the stereo matching cost with a convolutional neural network. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition.*

Zheng, S., Jayasumana, S., Romera-Paredes, B., Vineet, V., Su, Z., Du, D., … Torr, P. H. (2015). Conditional random fields as recurrent neural networks. In *Proceedings of IEEE International Conference on Computer Vision.*