Understanding and Designing Convolutional Networks for Local Recognition Problems

Jonathan Long

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2016-97
http://www.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-97.html

May 13, 2016

Copyright © 2016, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Understanding and Designing Convolutional Networks for Local Recognition Problems

by

Jonathan Leonard Long

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Trevor Darrell, Chair
Professor Jitendra Malik
Professor Alexei Efros
Professor Bruno Olshausen

Spring 2016

Understanding and Designing Convolutional Networks for Local Recognition Problems

Copyright 2016
by
Jonathan Leonard Long

Abstract

Understanding and Designing Convolutional Networks for Local Recognition Problems

by

Jonathan Leonard Long

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Trevor Darrell, Chair

In recent years, convolutional networks have dramatically (re)emerged as the dominant paradigm for solving visual recognition problems. Convnets are effective and appealing machines because they are made of a few simple, efficient building blocks, and are learnable end-to-end with straightforward gradient descent methods. However, convnets have often been construed as black-box classification machines, which receive whole images as input, produce single labels as output, and leave uninterpretable activations in between.

This work addresses the extension of convnets to rich prediction problems requiring localization along with recognition. Given their large pooling regions and training from whole-image labels, it has not been clear that classification convnets derive their success from an accurate correspondence model which could be used for precise localization. In the first part of this work, we'll conduct an empirical study of convnet features, asking the question: do networks designed and trained for classification alone contain enough information to understand the local, fine-scale content of images, such as the locations of parts and keypoints? We'll see that convnet features are indeed effective for tasks requiring correspondence. We present evidence that convnet features localize at a much finer scale than their receptive field sizes, that they can be used to perform intraclass alignment as well as conventional hand-engineered features, and that they outperform conventional features in keypoint prediction on objects from PASCAL VOC 2011 [EVGW$^+$].

Encouraged by this positive result, in the second part, we'll go on to see how convolutional networks by themselves, trained end-to-end, pixels-to-pixels, are state-of-the-art semantic segmentation systems. We achieve this by building "fully convolutional" networks that take input of arbitrary size and produce correspondingly-sized output with efficient inference and learning. We define and detail the design space and history of fully convolutional networks, and explain their application to spatially dense prediction tasks. We adapt contemporary classification networks (AlexNet [KSH12], the VGG net [SZ14a], and GoogLeNet [SLJ$^+$14]) into fully convolutional networks and transfer their learned representations by fine-tuning [DJV$^+$14] to the segmentation task. We then define a skip architecture that combines semantic information from a deep, coarse layer with appearance information from a shallow,

fine layer to produce accurate and detailed segmentations. Our FCNs achieve a 20% relative improvement on PASCAL VOC compared to prior methods, as well as performance improvements on NYUDv2 and SIFT Flow, while inference takes less than one fifth of a second for a typical image.

In the third part, we go beyond pixel labeling to explore generating object regions directly from a convolutional network. The networks we build produce object segments at multiple scales without intermediate bounding boxes, and without building an image pyramid, instead taking advantage of the natural pyramid of features present in a subsampling network. We extend the usual notion of convolutional networks, which are indexed spatially, to a notion of *pairwise networks*, which are *doubly* indexed spatially. We describe a wide set of design choices in this space, and relate existing approaches to models of this type. By carefully examining the structure of learned weights in an existing region generating network (DeepMask [PCD15]), we see that one of the simplest operations on pairs improves performance at negligible cost. Our final pure convnet region generator trains and tests in a fraction of a second per image, and produces competitive output on the COCO dataset [LMB$^+$14].

to a few people I won’t name here who have made *doing things* seem so worthwhile

Contents

Acknowledgments

No personal endeavor makes sense except in the context of others.

So thanks to Trevor Darrell, the advisor, for giving me direction, for encouraging my crazy ideas, and for making this whole thing possible. And thanks to the committee: Jitendra Malik, whose optimism and enthusiasm drew me in and kept me excited about this whole subject; Alyosha Efros, from whom I gained a respect for pixels and a distaste for labels; and Bruno Olshausen, from whom I learned nearly everything I know about human vision (and jumping spiders, and much more).

Thanks to my coauthors, especially Ning Zhang and Evan Shelhamer; all the best things are done in collaboration.

Thanks to fellow developers of the Caffe project, especially Yangqing Jia, Evan, and Jeff Donahue, but also numerous others who have hacked, contributed, and improved. It's been a real joy to be able to not just write and experiment but also to *build*. And that has been made possible only by a great and unexpected convergence of people.

Thanks to my colleagues from Trevor's group, from Jitendra and Alyosha's groups, from the rest of the seventh floor, and to numerous visitors with whom I've had countless interesting and productive discussions, but also simply enjoyed the company of. The state of mind of a social animal is largely a reflection of the states of mind of the surrounding individuals, and I've been pleased to reflect a great deal of passion, engagement, and grit.

Looking back, it's difficult to imagine a more incredible time to do a PhD in recognition. Much to my surprise, between when I entered and now, image recognition seems to have transitioned from a general state of *not working* to a general state of *working*. A dream is becoming true, and I'm delighted to be a part of it.

Chapter 1

Introduction

A hungry lion spots a herd of lechwe antelope grazing in an African swamp. Though the prey are adapted for aquatic survival, our lion notices a young straggler near dry land. A chase ensues. The lion flanks the lechwe, runs it down, and dines. [lio]

Although we will not build lions in this thesis, this story illustrates the motivation for the work herein. Acting in the physical world requires drawing rapid inferences from perceptual input. Real recognition systems must be:

1. Accurate. Incorrect judgements can be the difference between life and death: for lion, for lechwe, and for systems that serve human needs.

2. Fast. Accurate judgements are worthless if they come too late.

3. Rich. When surveying the landscape, it is not enough to know general *whats*, such as the type of terrain, or time of day, or presence of certain types of objects, but rather necessary to know specific combinations of *what* and *where*: there is a herd of antelope *over there*, a specific individual *right there*, favorable terrain *in this specific area.* In combination with the above, once the chase begins, the relevant information must be updated many times per second.

Biological systems serve not only as inspiration for what we might be do, but also as an existence proof: it is possible to achieve remarkable levels of performance by all three of these measures simultaneously.

In this work, we'll look for ways to improve recognition systems along these three axes, without stepping backwards by trading off one desideratum for another. In order to do so, we'll need to follow good principles of engineering: we'll build systems out of a small set of fast, reliable, and reusable components. In particular, we'll exploit recent, dramatic recognition advances achieved with convolutional networks. We achieve accuracy by directly transferring models trained for image classification to other tasks. We'll achieve speed by building systems only out of the same type of elements already used in convnets. And we'll obtain richer output by finding ways to use those elements to do more than classification.

In Chapter 2, we study the features already present in a network trained for classification, and ask whether those features contain information at a finer scale than the whole-image output for which they're trained. Encouraged by a positive answer, we continue on in Chapter 3 to build networks which label every pixel of an input image. Finally, in Chapter 4, we develop a general framework for building networks that can break images into regions—even if those regions overlap, and even if they cover adjacent objects of the same class. Throughout, we'll build systems that are trained end-to-end, and that perform inference in a fraction of a second.

1.1 Background

In the remaining chapters, we'll assume familiarity with convolutional networks and related notions. This section provides a brief reference, including definitions of notation and terminology as used in the following.

1.1.1 Parametric Learning by Gradient Descent

Suppose we have labeled data $\{(x_i, y_i)\}$, where each data point (usually, image) x_i is given a corresponding label (scalar, vector, or other object) y_i. In *supervised* learning, we attempt to find a function f such that

$$y_i \approx f(x_i)$$

for all i. To quantify the accuracy of our approximation, we pick a (task-dependent) *loss function* ℓ and minimize the total loss

$$L = \sum_i \ell(y_i, f(x_i)).$$

In *parametric* learning, f is determined by a vector of parameters θ; $f(x_i) = f(x_i; \theta)$. Thus we arrive at the learning problem

$$\min_\theta \sum_i \ell(y_i, f(x_i; \theta)).$$

For our purposes, θ will always be a vector of real numbers (to be given a discrete floating point implementation), and we will perform this minimization by gradient desent. Specifically, we'll use *(minibatch) stochastic gradient descent with momentum.* At each timestep t we'll take a gradient step g_t, updating parameters according to

$$\theta_t = \theta_{t-1} + g_t.$$

Specifically, our update will be (itself recurrently defined as)

$$g_t = -\eta \sum_{i=0}^{k-1} \nabla_\theta \ell(y_{kt+i}, f(x_{kt+i}; \theta_{t-1})) + p g_{t-1},$$

where

- η is a real-valued hyperparameter (i.e., a tunable learning parameter) called the *learning rate* (η determines the absolute scale of each step);
- k is a positive integer called the *(mini)batch size* (minibatching produces smoother, averaged gradients, and can improve computational efficiency);
- ∇ is the ordinary gradient on $\mathbb{R}^n$;[1]
- ℓ is the loss function defined above; and
- p is a real parameter in the interval $[0, 1)$, called the *momentum* (p smooths the updates between minibtaches).

1.1.2 Momentum vs Batch Size

It's worth noting that the expression above can be expanded into an infinite sum with geometric coefficients, i.e.,

$$g_t = -\eta \sum_{s=0}^{\infty} \sum_{i=0}^{k-1} p^s \nabla_\theta \ell(y_{k(t-s)+i}, f(x_{k(t-s)+i}; \theta_{t-s})).$$

Each example is included in the sum with coefficient $p^{\lfloor j/k \rfloor}$, where the index j orders the examples from most recently considered to least recently considered. Approximating this expression by dropping the floor, we see that learning with momentum p and batch size k is similar to learning with momentum p' and batch size k' if $p^{(1/k)} = p'^{(1/k')}$. Note, however, that using a smaller batch size results in a smoother fall off and more frequent parameter updates (and the learning rate may need to be adjusted accordingly). Figure 1.1 gives a visual comparison of the weighting resulting from two training schemes related by this expression.

1.1.3 Convolutional Networks

How shall we choose a function $f(\cdot; \theta)$ to represent a transformation from input (images) to output? A simple choice is a linear function: $f(x; \theta) = \theta x$, where θ is a matrix and x is a vector. While linear regression is itself a powerful technique with a long history, we will be interested in more complex nonlinear transformations.

The basic tool for building model complexity is composition. Compositional representations naturally appeal to intuiton: we think of scenes as made of objects, objects as made of parts, and parts as made of basic shapes. (In practice, we'll arrive at a much more opaque,

[1] This is not the only choice, nor necessarily the best choice. See the *natural gradient* [PB13, Ama98] for one (but not the only) alternative.

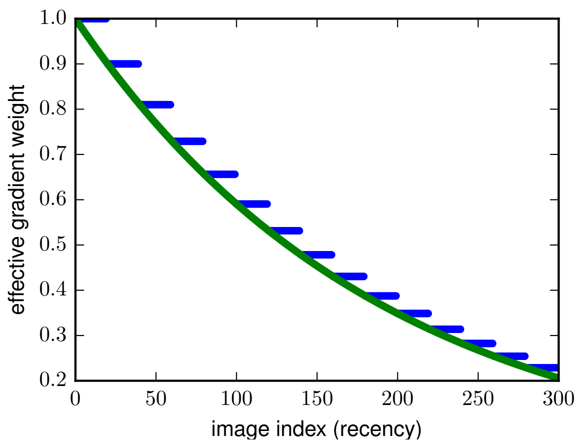


Figure 1.1: Trading off batch size for momentum results in a similar learning procedure which can have smoother behavior and improved convergence. The blue, step-like curve shows the effective weight of each example in a gradient update using momentum 0.9 and batch size 20. The green, smooth curve shows the effective weight using the alternative settings of momentum $0.9^{(1/20)}$, batch size one. Note that in addition to smoothing out these weights, lowering the batch size also increases the frequency of parameter updates.

learned representation.) Because linearity is preserved by composition, nonlinear elements are also needed to build nonlinear functions. Following Occam's razor and standard practice, we therefore build networks[2] that alternate between linear operations and simple *elementwise* nonlinearities, called *activation functions*.

When dealing with images, we can make an additional simplification by insisting that our representations be translation invariant (up to quantization effects). The means that we'll implement our linear transformations as convolutions,[3] rather than unconstrained matrices. (We'll revisit this princple in more detail in Chapter 3.)

Together with a few additional nonlinearities (most notably nonlinear pooling[4]), these are *all* of the ingredients needed to build convolutional networks. The rest is a matter of finding effective architectures and learning parameters.

Convolutional networks are sometimes called *convolutional neural networks* or CNNs. Despite this name, neurons are not necessary in any way to motivate or influence the design of the models that we use. So we'll use the simpler term *convolutional network*, or *convnet* for short.

[2]We'll use the term *network* to mean any function built by composition including linear, parametric components (which will encompass all of the models we learn).

[3]Following standard practice in computer vision, we will use *convolution* to mean *correlation*; the distinction will usually be unimportant.

[4]which, in common cases, may be viewed as convolution in a nonstandard semiring

1.1.4 Network Details

We'll make use of a couple well-known and publicly available benchmark networks (including both their architectures and their weights learned on ImageNet [DDS+09]). These include the eight-layer *AlexNet* model [KSH12], and the sixteen-layer *VGG-16* model [SZ14b] (also called VGG-D). We'll refer to the layers of these networks by commonly-used abbreviations, writing (for example) conv$n$, pool$n$, or fc$n$, for the nth convolutional, pooling, or fully-connected layer, respectively.

(Note that the implementation of AlexNet we use ("CaffeNet") reverses the order of the response normalization and pooling layers.)

We'll use the term *receptive field*, abbreviated rf, to refer to the set of input pixels that are path-connected to a particular unit in a convnet. (Note that other usages of this term exist.)

All experiments are performed using the Caffe [JSD+14] framework, and trained networks (including networks we make use, and many of the networks we produce) are publicly available in the Caffe model zoo [mod]. We'll also use Halide [RKBA+13] for efficient implementations of some of our custom routines.

1.2 Related Work

While each chapter includes a topic-specific discussion of related work, in this section we'll provide a brief (and therefore, necessarily biased and incomplete) overview of the most important overall influences.

A complete history of deep learning is outside the scope of this document; see, for example [Sch15]. This work follows the recent resurgence in popularity of supervised deep learning precipitated by the benchmark ILSVRC [ILS12] results of AlexNet [KSH12]. That network in turn shares overall design principles (with subtle, but important differences) with the classic digit classifier LeNet [LBD+89], which draws inspiration from the Neocognitron [Fuk80]. The idea of extending convolutional networks to produce spatial output dates back to at least the "space displacement neural networks" [MBLD91] that extended LeNet to digit strings.

This work also follows a natural progression in spatial specificity from whole-image classification to per-pixel classification, with benchmark detection results, especially R-CNN [GDDM14b], providing an encouraging intermediate stepping stone. (The ideas presented here have in turn influenced bounding box detection, as seen in the subsequent Fast R-CNN [Gir15] and Faster R-CNN [RHGS15].)

All the while, the use of finetuning means that advances in classification directly improve the performance of methods presented here; we'll especially consider the networks VGG [SZ14b] and GoogLeNet [SLJ+14], and note that ResNet [HZRS15], while too recent to be included in our results, is also showing promise for powerful finetuning (e.g., in [DHS15]).

Chapter 2

Understanding Local Responses: Correspondence and Keypoints[1]

In this chapter we study convnets pretrained for classification, asking questions about the intermediate representations that result, and especially about how useful those representations are for tasks beyond their original purpose of whole-image classification. Note that this is an extremely common mode of usage: a network is first pretrained on a very large dataset for a simple task, after which it is treated as a feature extractor for further applications, much like a traditional hand-engineered feature extractor.

Although backpropagation affords the possibility of *finetuning* the feature extraction stage for a particular new task, we won't be finetuning in this chapter. Instead, we'll intentionally leave the features fixed, and even though we'll examine performance on several tasks, we'll be interested not in building a system with the highest level of performance, but rather in what the tasks tell us about features learned *only from classification*. (In later chapters, we'll continue on to building end-to-end learned systems for acheiving high levels of performance.)

Dramatic improvements in image classification benchmarks [KSH12] were the major event that rekindled interest in convolutional nets for computer vision. Despite the magnitude of these results, many doubted [Yan] that the resulting features had the spatial specificity necessary for localization; after all, whole image classification can rely on context cues and overly large pooling regions to get the job done. For coarse localization, such doubts were alleviated by record breaking results extending the same features to detection on PASCAL [GDDM14b].

However, these results left open the same questions on a finer scale. Are the modern convnets that excel at classification and detection also able to find precise correspondences between object parts? Or do large receptive fields mean that correspondence is effectively pooled away, making this a task better suited for hand-engineered features?

In this chapter, we provide evidence that convnet features perform at least as well as con-

[1]This section is based on joint work with Ning Zhang and Trevor Darrell [LZD14].

Table 2.1: The receptive field sizes and strides for an "AlexNet" network, which takes an input of size 227×227.

layer	rf size	stride
`conv1`	11×11	4×4
`conv2`	51×51	8×8
`conv3`	99×99	16×16
`conv4`	131×131	16×16
`conv5`	163×163	16×16
`pool5`	195×195	32×32

ventional ones, even in the regime of point-to-point correspondence, and achieve considerable performance improvement in certain settings, including category-level keypoint prediction.

2.1 Feature visualization

In this section and Figures 2.1 and 2.2, we conduct a visual investigation of the effective pooling regions of convnet features.

In Figure 2.1, we perform a nonparametric reconstruction of images from features in the spirit of HOGgles [VKMT13]. Rather than paired dictionary learning, however, we simply replace patches with averages of their top-k nearest neighbors in a convnet feature space. To do so, we first compute all features at a particular layer, resulting in an 2d grid of feature vectors. We associate each feature vector with a patch in the original image at the center of the corresponding receptive field and with size equal to the receptive field stride. (Note that the strides of the receptive fields are much smaller than the receptive fields themselves, which overlap. Refer to Table 2.1 above for specific numbers.) We replace each such patch with an average over k nearest neighbor patches using a database of features densely computed on the images of PASCAL VOC 2011. Our database contains at least one million patches for every layer. Features are matched by cosine similarity.

Even though the feature rfs cover large regions of the source images, the specific resemblance of the resulting images shows that information is not spread uniformly throughout those regions. Notable features (e.g., the tires of the bicycle and the facial features of the cat) are replaced in their corresponding locations. Also note that replacement appears to become more semantic and less visually specific as the layer deepens: the eyes and nose of the cat get replaced with differently colored or shaped eyes and noses, and the fur gets replaced with various animal furs, with the diversity increasing with layer number.

Figure 2.2 gives a feature-centric rather than image-centric view of feature locality. For each column, we first pick a random seed feature vector (computed from a PASCAL image), and find k nearest neighbor features, again by cosine similarity. Instead of averaging only the centers, we average the entire receptive fields of the neighbors. The resulting images show that similar features tend to respond to similar colors specifically in the centers of their

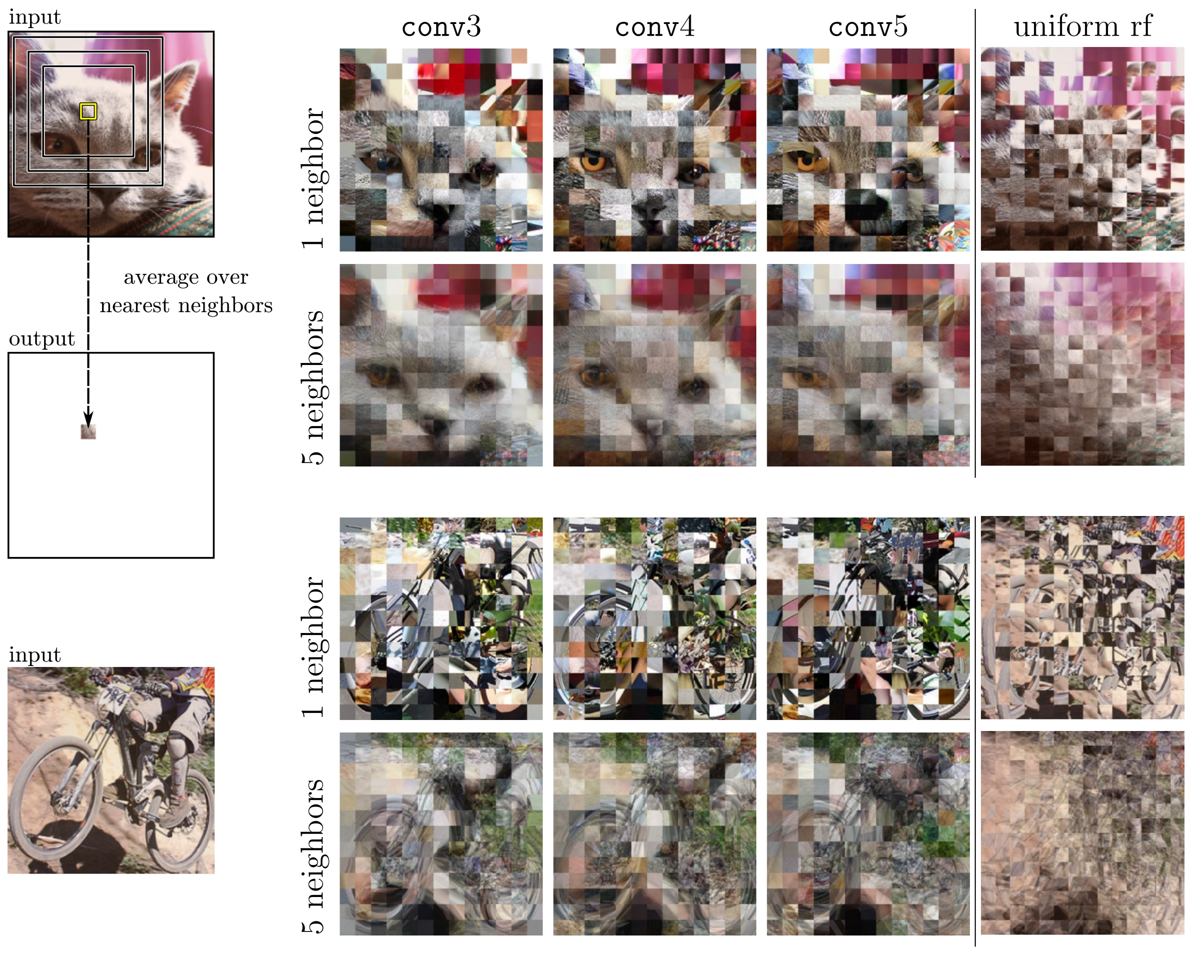


Figure 2.1: Even though they have large receptive fields, convnet features carry local information at a finer scale. Upper left: given an input image, we replaced 16×16 patches with averages over 1 or 5 nearest neighbor patches, computed using convnet features centered at those patches. The yellow square illustrates one input patch, and the black squares show the corresponding rfs for the three layers shown. Right: Notice that the features retrieve reasonable matches for the centers of their receptive fields, even though those rfs extend over large regions of the source image. In the "uniform rf" column, we show the best that could be expected if convnet features discarded all spatial information within their rfs, by choosing input patches uniformly at random from `conv3`-sized neighborhoods. (Best viewed electronically.)

receptive fields.

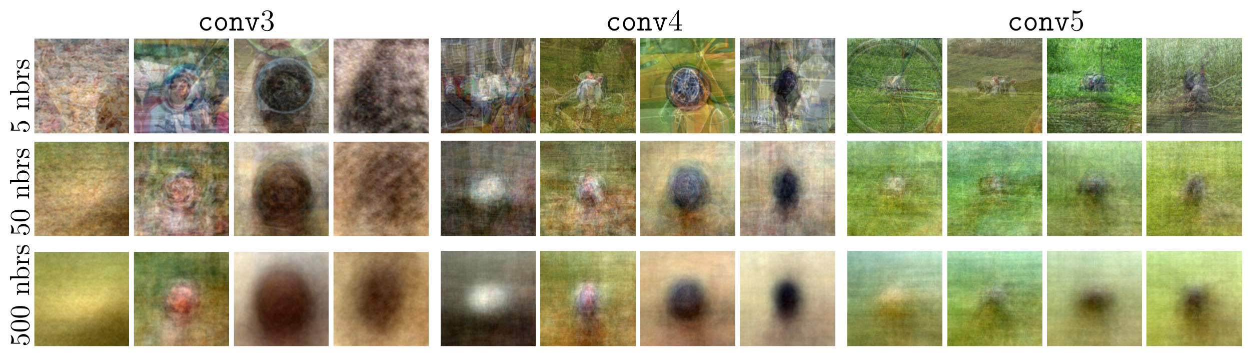


Figure 2.2: Similar convnet features tend to have similar receptive field centers. Starting from a randomly selected seed patch occupying one rf in `conv3`, 4, or 5, we find the nearest k neighbor features computed on a database of natural images, and average together the corresponding receptive fields. The contrast of each image has been expanded after averaging. (Note that since each layer is computed with a stride of 16, there is an upper bound on the quality of alignment that can be witnessed here.)

2.2 Intraclass alignment

We conjecture that category learning implicitly aligns instances by pooling over a discriminative mid-level representation. If this is true, then such features should be useful for post-hoc alignment in a similar fashion to conventional features. To test this, we use convnet features for the task of aligning different instances of the same class. We approach this difficult task in the style of SIFT flow [LYT11]: we retrieve near neighbors using a coarse similarity measure, and then compute dense correspondences on which we impose an MRF smoothness prior which finally allows all images to be warped into alignment.

Nearest neighbors are computed using `fc7` features. Since we are specifically testing the quality of alignment, we use the same nearest neighbors for convnet or conventional features, and we compute both types of features at the same locations, the grid of convnet rf centers in the response to a single image.

Alignment is determined by solving an MRF formulated on this grid of feature locations. Let p be a point on this grid, let $f_s(p)$ be the feature vector of the source image at that point, and let $f_t(p)$ be the feature vector of the target image at that point. For each feature grid location p of the source image, there is a vector $w(p)$ giving the displacement of the corresponding feature in the target image. We use the energy function

$$E(w) = \sum_p \|f_s(p) - f_t(p + w(p))\|_2 + \beta \sum_{(p,q)\in\mathcal{E}} \|w(p) - w(q)\|_2^2,$$

where $\mathcal{E}$ are the edges of a 4-neighborhood graph and β is the regularization parameter. Optimization is performed using belief propagation, with the techniques suggested in [FH06]. Message passing is performed efficiently using the squared Euclidean distance transform

Table 2.2: Keypoint transfer accuracy using convnet flow, SIFT flow, and simple copying from nearest neighbors. Accuracy (PCK) is shown per category using $\alpha = 0.1$ (see text) and means are also shown for the stricter values $\alpha = 0.05$ and 0.025. On average, convnet flow performs as well as SIFT flow, and performs a bit better for stricter tolerances.

	aero	bike	bird	boat	bttl	bus	car	cat	chair	cow	table	dog	horse	mbike	prsn	plant	sheep	sofa	train	tv	mean
`conv4` flow	28.2	34.1	20.4	17.1	50.6	36.7	20.9	19.6	15.7	25.4	12.7	18.7	25.9	23.1	21.4	40.2	21.1	14.5	18.3	33.3	24.9
SIFT flow	27.6	30.8	19.9	17.5	49.4	36.4	20.7	16.0	16.1	25.0	16.1	16.3	27.7	28.3	20.2	36.4	20.5	17.2	19.9	32.9	24.7
NN transfer	18.3	24.8	14.5	15.4	48.1	27.6	16.0	11.1	12.0	16.8	15.7	12.7	20.2	18.5	18.7	33.4	14.0	15.5	14.6	30.0	19.9

mean	$\alpha = 0.1$	$\alpha = 0.05$	$\alpha = 0.025$
`conv4` flow	24.9	11.8	4.08
SIFT flow	24.7	10.9	3.55
NN transfer	19.9	7.8	2.35

[FH04]. (Unlike the L_1 regularization originally used by SIFT flow [LYT11], this formulation maintains rotational invariance of w.)

Based on its performance in the next section, we use `conv4` as our convnet feature, and SIFT with descriptor radius 20 as our conventional feature. From validation experiments, we set $\beta = 3 \cdot 10^{-3}$ for both `conv4` and SIFT features (which have a similar scale).

Given the alignment field w, we warp target to source using bivariate spline interpolation (implemented in SciPy [JOP$^+$]). Figure 2.3 gives examples of alignment quality for a few different seed images, using both SIFT and convnet features. We show five warped nearest neighbors as well as keypoints transferred from those neighbors.

We quantitatively assess the alignment by measuring the accuracy of predicted keypoints. To obtain good predictions, we warp 25 nearest neighbors for each target image, and order them from smallest to greatest deformation energy (we found this method to outperform ordering using the data term). We take the predicted keypoints to be the median points (coordinate-wise) of the top five aligned keypoints according to this ordering.

We assess correctness using mean PCK [FMJZ08]. We consider a ground truth keypoint to be correctly predicted if the prediction lies within a Euclidean distance of α times the maximum of the bounding box width and height, picking some $\alpha \in [0, 1]$. We compute the overall accuracy for each type of keypoint, and report the average over keypoint types. We do not penalize predicted keypoints that are not visible in the target image.

Results are given in Table 2.2. We show per category results using $\alpha = 0.1$, and mean results for $\alpha = 0.1$, 0.05, and 0.025. Indeed, convnet learned features are at least as capable as SIFT at alignment, and better than might have been expected given the size of their receptive fields.

Table 2.3: Keypoint classification accuracies, in percent, on the twenty categories of PASCAL 2011 val, trained with SIFT or convnet features. The best SIFT and convnet scores are bolded in each category.

		aero	bike	bird	boat	bttl	bus	car	cat	chair	cow	table	dog	horse	mbike	prsn	plant	sheep	sofa	train	tv	mean
SIFT	10	36	42	36	32	67	64	40	37	33	37	60	34	39	38	29	63	37	42	64	75	45
(radius)	20	**37**	50	**39**	35	74	67	**47**	**40**	36	**43**	68	**38**	**42**	48	**33**	**70**	**44**	**52**	68	77	50
	40	35	**54**	37	41	**76**	**68**	**47**	37	39	40	69	36	**42**	49	32	69	39	**52**	**74**	**78**	**51**
	80	33	43	37	**42**	75	66	42	30	**43**	36	**70**	31	36	**51**	27	**70**	35	49	69	77	48
	160	27	36	34	38	72	59	35	25	39	30	67	27	32	46	25	**70**	29	48	66	76	44
`conv`	1	16	14	15	19	20	29	15	22	16	17	29	17	14	16	15	33	18	12	27	29	20
(layer)	2	37	43	40	35	69	63	38	44	35	40	61	38	40	44	34	65	39	41	63	72	47
	3	42	50	46	41	76	69	**46**	52	39	45	64	47	48	52	40	74	46	50	71	**77**	54
	4	**44**	**53**	**49**	**42**	**78**	**70**	45	**55**	**41**	**48**	**68**	**51**	**51**	**53**	**41**	**76**	**49**	**52**	**73**	76	**56**
	5	**44**	51	**49**	41	77	68	44	53	39	45	63	50	49	52	39	73	47	47	71	75	54

2.3 Keypoint detection

In this section, we specifically address the ability of convnet features to understand semantic information at the scale of parts. As an initial test, we consider the task of *keypoint classification*: given an image and the coordinates of a keypoint on that image, can we train a classifier to label the keypoint?

For this task we use keypoint data [BM09] on the twenty classes of PASCAL VOC 2011 [EVGW$^+$]. We extract features at each keypoint using SIFT [Low99] and using the column of each convnet layer whose rf center lies closest to the keypoint. (Note that the SIFT features will be more precisely placed as a result of this approximation.) We trained one-vs-all linear SVMs on the train set using SIFT at five different radii and each of the five convolutional layer activations as features (in general, we found pooling and normalization layers to have lower performance). We set the SVM parameter $C = 10^{-6}$ for all experiments based on five-fold cross validation on the training set (see Figure 2.7).

Table 2.3 gives the resulting accuracies on the val set. We find features from convnet layers consistently perform at least as well as and often better than SIFT at this task, with the highest performance coming from layers `conv`4 and `conv`5. Note that we are specifically testing convnet features trained only for classification; the same net could be expected to achieve even higher performance if trained for this task.

Finally, we study the precise location understanding of our classifiers by computing their responses with a single-pixel stride around ground truth keypoint locations. For two example keypoints (cat left eye and nose), we histogram the locations of the maximum responses within a 21 pixel by 21 pixel rectangle around the keypoint, shown in Figure 2.5. We do not include maximum responses that lie on the boundary of this rectangle. While the SIFT classifiers do not seem to be sensitive to the precise locations of the keypoints, in many cases the convnet ones seem to be capable of localization finer than their strides, not just their receptive field sizes. This observation motivates our final experiments to consider detection-based localization performance.

2.4 Keypoint prediction

We have seen that despite their large receptive field sizes, convnets work as well as the hand-engineered feature SIFT for alignment and slightly better than SIFT for keypoint classification. Keypoint prediction provides a natural follow-up test. As in Section 2.2, we use keypoint annotations from PASCAL VOC 2011, and we assume a ground truth bounding box.

Inspired in part by [GDDM14b, SEZ$^+$14, JTA$^+$14], we train sliding window part detectors to predict keypoint locations independently. R-CNN [GDDM14b] and OverFeat [SEZ$^+$14] have both demonstrated the effectiveness of deep convolutional networks on the generic object detection task. However, neither of them have investigated the application of CNNs for keypoint prediction.[2] R-CNN starts from bottom-up region proposal [UvdSGS13], which tends to overlook the signal from small parts. OverFeat, on the other hand, combines convnets trained for classification and for regression and runs in multi-scale sliding window fashion.

We rescale each bounding box to 500 × 500 and compute `conv5` (with a stride of 16 pixels). Each cell of `conv5` contains one 256-dimensional descriptor. We concatenate `conv5` descriptors from a local region of 3×3 cells, giving an overall receptive field size of 195×195 and feature dimension of 2304. For each keypoint, we train a linear SVM with hard negative mining. We consider the ten closest features to each ground truth keypoint as positive examples, and all the features whose rfs do not contain the keypoint as negative examples. We also train using dense SIFT descriptors for comparison. We compute SIFT on a grid of stride eight and bin size of eight using VLFeat [VF08]. For SIFT, we consider features within twice the bin size from the ground truth keypoint to be positives, while samples that are at least four times the bin size away are negatives.

We augment our SVM detectors with a spherical Gaussian prior over candidate locations constructed by nearest neighbor matching. The mean of each Gaussian is taken to be the location of the keypoint in the nearest neighbor in the training set found using cosine similarity on `pool5` features, and we use a fixed standard deviation of 22 pixels. Let $s(X_i)$ be the output score of our local detector for keypoint X_i, and let $p(X_i)$ be the prior score. We combine these to yield a final score $f(X_i) = s(X_i)^{1-\eta} p(X_i)^{\eta}$, where $\eta \in [0, 1]$ is a tradeoff parameter. In our experiments, we set $\eta = 0.1$ by cross validation. At test time, we predict the keypoint location as the highest scoring candidate over all feature locations.

We evaluate the predicted keypoints using the measure PCK introduced in Section 2.2, taking $\alpha = 0.1$. A predicted keypoint is defined as correct if the distance between it and the ground truth keypoint is less than $\alpha \cdot \max(h, w)$ where h and w are the height and width of the bounding box. The results using `conv5` and SIFT with and without the prior are shown in Table 2.4. From the table, we can see that local part detectors trained on the `conv5` feature outperform SIFT by a large margin and that the prior information is helpful in both cases. To our knowledge, these are the first keypoint prediction results reported on this dataset. We

[2]But see works cited in Section 1.1 regarding keypoint localization.

show example results from five different categories in Figure 2.8. Each set consists of rescaled bounding box images with ground truth keypoint annotations and predicted keypoints using SIFT and `conv5` features, where each color corresponds to one keypoint. As the figure shows, `conv5` outperforms SIFT, often managing satisfactory outputs despite the challenge of this task. A small offset can be noticed for some keypoints like eyes and noses, likely due to the limited stride of our scanning windows. A final regression or finer stride could mitigate this issue.

2.5 Conclusion

Through visualization, alignment, and keypoint prediction, we have studied the ability of the intermediate features implicitly learned in a state-of-the-art convnet classifier to understand specific, local correspondence. Despite their large receptive fields and weak label training, we have found in all cases that convnet features are at least as useful (and sometimes considerably more useful) than conventional ones for extracting local visual information.

2.6 Related Work

2.6.1 Image alignment

Image alignment is a key step in many computer vision tasks, including face verification, motion analysis, stereo matching, and object recognition. Alignment results in correspondence across different images by removing intraclass variability and canonicalizing pose. Alignment methods exist on a supervision spectrum from requiring manually labeled fiducial points or landmarks, to requiring class labels, to fully unsupervised joint alignment and clustering models. Congealing [HJLM07] is an unsupervised joint alignment method based on an entropy objective. Deep congealing [HMLLM12] builds on this idea by replacing hand-engineered features with unsupervised feature learning from multiple resolutions. Inspired by optical flow, SIFT flow [LYT11] matches densely sampled SIFT features for correspondence and has been applied to motion prediction and motion transfer. In Section 2.2, we apply SIFT flow using deep features for aligning different instances of the same class.

2.6.2 Keypoint localization

Semantic parts carry important information for object recognition, object detection, and pose estimation. In particular, fine-grained categorization, the subject of many recent works, depends strongly on part localization [LB13, BB13]. Large pose and appearance variation across examples make part localization for generic object categories a challenging task.

Most of the existing works on part localization or keypoint prediction focus on either facial landmark localization [BJKK11] or human pose estimation. Human pose estimation has been approached using tree structured methods to model the spatial relationships between parts

[YR11, SS11, ZR12], and also using poselets [BM09] as an intermediate step to localize human keypoints [GHGM14, GABM13]. Tree structured models and poselets may struggle when applied to generic objects with large articulated deformations and wide shape variance.

2.6.3 Deep learning

Convolutional neural networks have gained much recent attention due to their success in image classification [KSH12]. Convnets trained with backpropagation were initially succesful in digit recognition [LBD$^+$89] and OCR [LBBH98]. The feature representations learned from large data sets have been found to generalize well to other image classification tasks [DJV$^+$14] and even to object detection [GDDM14b, SKCL13]. Recently, Toshev et al. [TS14] trained a cascade of regression-based convnets for human pose estimation and Jain et al. [JTA$^+$14] combine a weak spatial model with deep learning methods.

The latter work trains multiple small, independent convnets on 64×64 patches for binary body-part detection. In contrast, we employ a powerful pretained ImageNet model that shares mid-elvel feature representations among all parts in Section 2.4.

Several recent works have attempted to analyze and explain this overwhelming success. Zeiler and Fergus [ZF14] provide several heuristic visualizations suggesting coarse localization ability. Szegedy et al. [SZS$^+$13] show counterintuitive properties of the convnet representation, and suggest that individual feature channels may not be more semantically meaningful than other bases in feature space. A concurrent work [FDB14a] compares convnet features with SIFT in a standard descriptor matching task. This work illuminates and extends that comparison by providing visual analysis and by moving beyond single instance matching to intraclass correspondence and keypoint prediction.

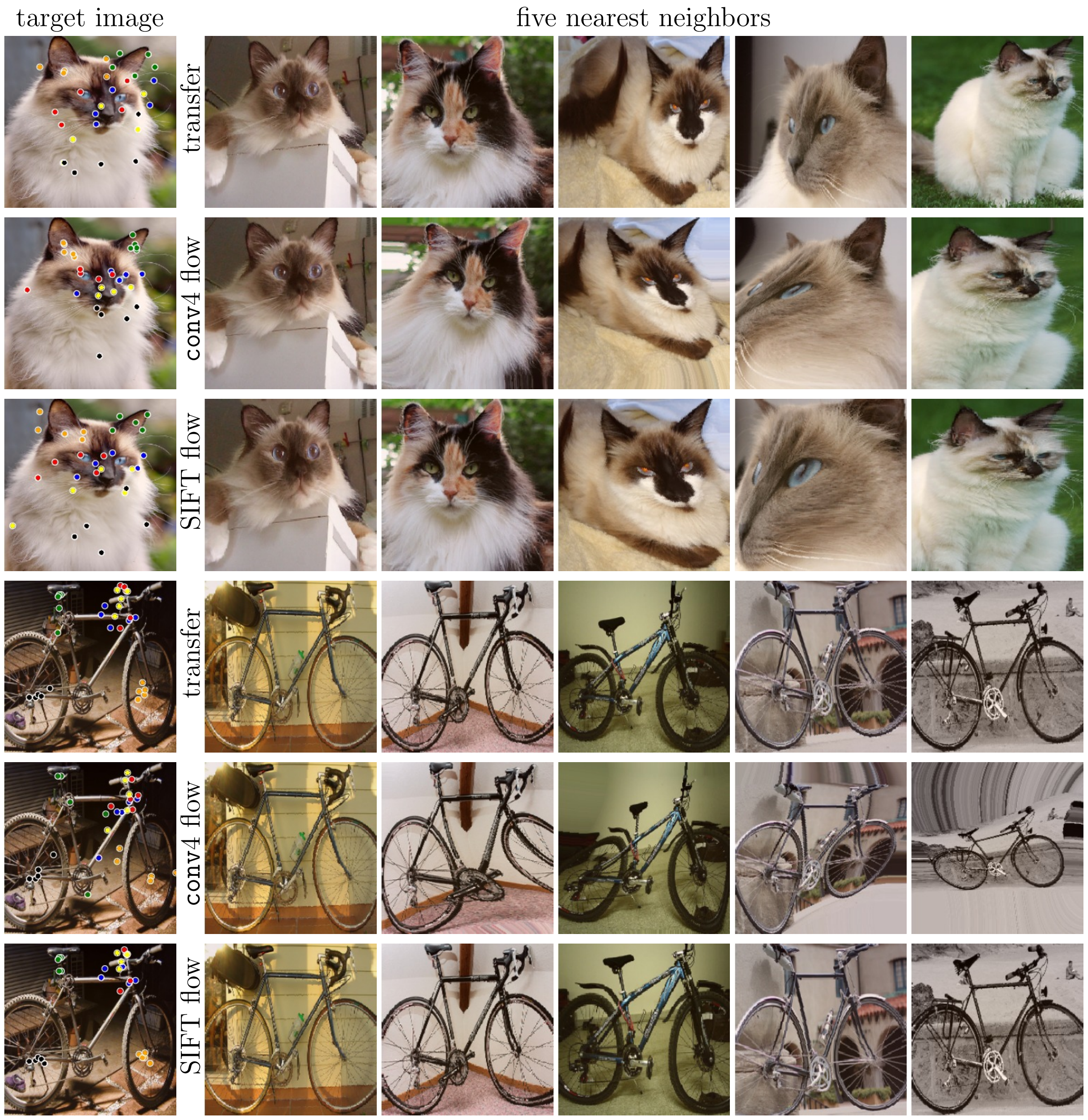
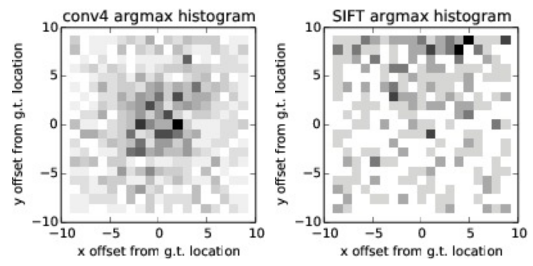
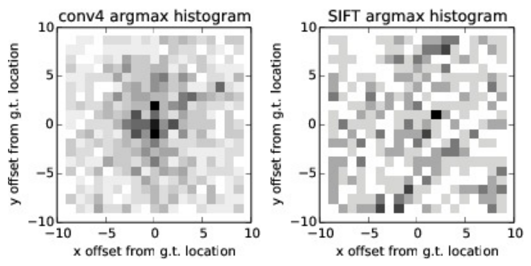


Figure 2.3: Convnet features can bring different instances of the same class into good alignment at least as well (on average) as traditional features. For each target image (left column), we show warped versions of five nearest neighbor images aligned with `conv4` flow (first row), and warped versions aligned with SIFT flow [LYT11] (second row). Keypoints from the warped images are shown copied to the target image. The cat shows a case where convnet features perform better, while the bicycle shows a case where SIFT features perform better. (Note that each instance is warped to a square bounding box before alignment. Best viewed in color.)

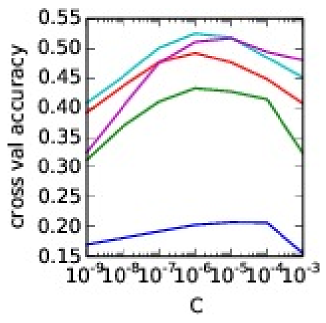


(a) cat left eye

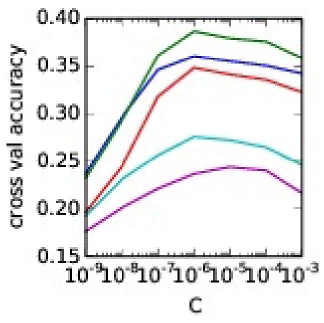


(b) cat nose

Figure 2.5: Convnet features show fine localization ability, even beyond their stride and in cases where SIFT features do not perform as well. Each plot is a 2D histogram of the locations of the maximum responses of a classifer in a 21 by 21 pixel rectangle taken around a ground truth keypoint.



(a)



(b)

Figure 2.7: Cross validation scores for cat keypoint classification as a function of the SVM parameter C. In (a), we plot mean accuracy against C for five different convnet features; in (b) we plot the same for SIFT features of different sizes. We use $C = 10^{-6}$ for all experiments in Table 2.3.

Table 2.4: Keypoint prediction results on PASCAL VOC 2011. The numbers give average accuracy of keypoint prediction using the criterion described in Section 2.2, PCK with $\alpha = 0.1$.

	aero	bike	bird	boat	bttl	bus	car	cat	chair	cow	table	dog	horse	mbike	prsn	plant	sheep	sofa	train	tv	mean
SIFT	17.9	16.5	15.3	15.6	25.7	21.7	22.0	12.6	11.3	7.6	6.5	12.5	18.3	15.1	15.9	21.3	14.7	15.1	9.2	19.9	15.7
SIFT+prior	33.5	36.9	22.7	23.1	44.0	42.6	39.3	22.1	18.5	23.5	11.2	20.6	32.2	33.9	26.7	30.6	25.7	26.5	21.9	32.4	28.4
`conv5`	38.5	37.6	29.6	25.3	54.5	52.1	28.6	31.5	8.9	30.5	24.1	23.7	35.8	29.9	39.3	38.2	30.5	24.5	41.5	42.0	33.3
`conv5`+prior	**50.9**	**48.8**	**35.1**	**32.5**	**66.1**	**62.0**	**45.7**	**34.2**	**21.4**	**41.1**	**27.2**	**29.3**	**46.8**	**45.6**	**47.1**	**42.5**	**38.8**	**37.6**	**50.7**	**45.6**	**42.5**

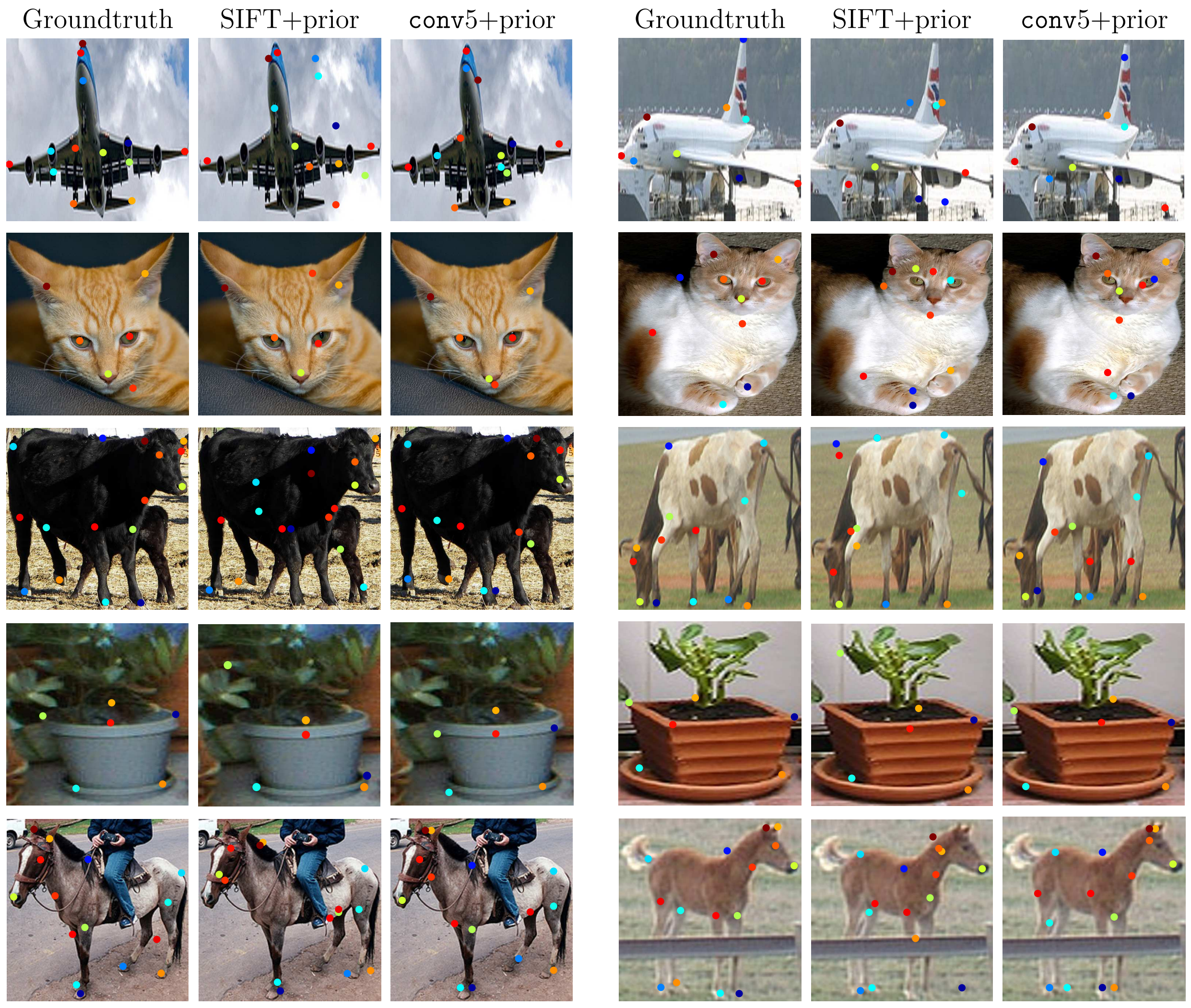


Figure 2.8: Examples of keypoint prediction on five classes of the PASCAL dataset: aeroplane, cat, cow, potted plant, and horse. Each keypoint is associated with one color. The first column is the ground truth annotation, the second column is the prediction result of SIFT+prior and the third column is `conv5`+prior. (Best viewed in color).

Chapter 3

Predicting Pixels Directly: Fully Convolutional Networks[1]

In previous chapter, we've learned that convnet features, even in nets trained only for classification, carry powerful local information. In this chapter, encouraged by those results, we go on to train networks directly for per-pixel prediction.

Convolutional networks have driven advances in recognition. This includes not only whole-image classification [KSH12, SZ14a, SLJ$^+$14], but also local tasks with structured output. These include advances in bounding box object detection [SEZ$^+$14, GDDM14a, HZRS14], but also part and keypoint prediction [ZDGD14, LZD14], and local correspondence [LZD14, FDB14b], as we have seen in the previous chapter.

The natural next step in the progression from coarse to fine inference is to make a prediction at every pixel. Prior approaches have used convnets for semantic segmentation [NDL$^+$05, CGGS12, FCNL13, PC14, HAGM14a, GGAM14, GL14], in which each pixel is labeled with the class of its enclosing object or region, but with shortcomings that this chapter addresses.

We show that a fully convolutional network (FCN) trained end-to-end, pixels-to-pixels on semantic segmentation exceeds the state-of-the-art without further machinery. To our knowledge, this is the first result that trains FCNs end-to-end (1) for pixelwise prediction and (2) from supervised pre-training. Fully convolutional versions of existing networks predict dense outputs from arbitrary-sized inputs. Both learning and inference are performed whole-image-at-a-time by dense feedforward computation and backpropagation. In-network upsampling layers enable pixelwise prediction and learning in nets with subsampled pooling.

This method is efficient, both asymptotically and absolutely, and precludes the need for the complications in other works. Patchwise training is common [NDL$^+$05, CGGS12, FCNL13, PC14, GL14], but lacks the efficiency of fully convolutional training. Our approach does not make use of pre- and post-processing complications, including superpixels [FCNL13, HAGM14a], proposals [HAGM14a, GGAM14], or post-hoc refinement by random fields or

[1]This section is based on joint work with Evan Shelhamer and Trevor Darrell [LSD15].

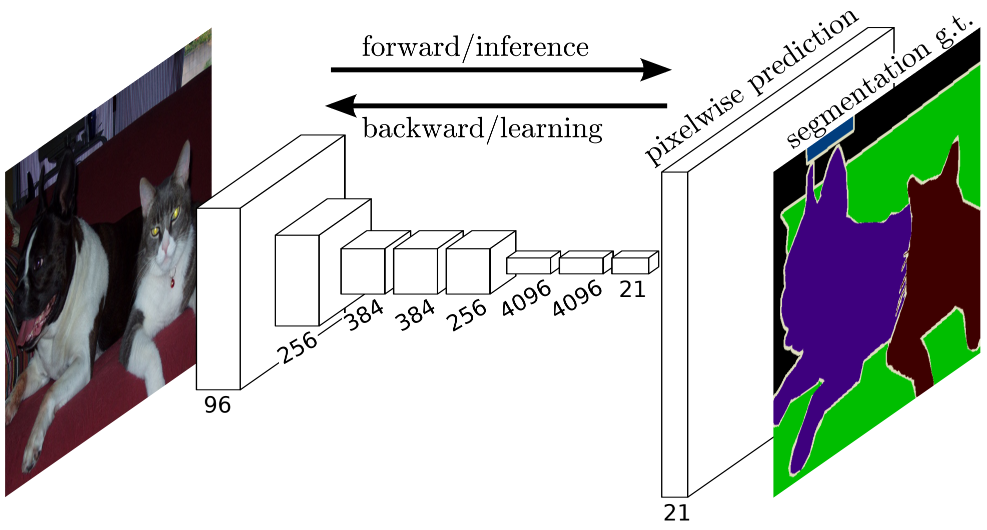


Figure 3.1: Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

local classifiers [FCNL13, HAGM14a]. Our model transfers recent success in classification [KSH12, SZ14a, SLJ$^+$14] to dense prediction by reinterpreting classification nets as fully convolutional and fine-tuning from their learned representations. In contrast, previous works have applied small convnets without supervised pre-training [FCNL13, PC14, NDL$^+$05].

Semantic segmentation faces an inherent tension between semantics and location: global information resolves what while local information resolves where. Deep feature hierarchies encode location and semantics in a nonlinear local-to-global pyramid. We define a skip architecture to take advantage of this feature spectrum that combines deep, coarse, semantic information and shallow, fine, appearance information in Section 3.3.2 (see Figure 3.3).

In the next section, we review related work on deep classification nets, FCNs, and recent approaches to semantic segmentation using convnets. The following sections explain FCN design and dense prediction tradeoffs, introduce our architecture with in-network upsampling and multi-layer combinations, and describe our experimental framework. Finally, we demonstrate state-of-the-art results on PASCAL VOC 2011-2, NYUDv2, and SIFT Flow.

3.1 Related work

Our approach draws on recent successes of deep nets for image classification [KSH12, SZ14a, SLJ$^+$14] and transfer learning [DJV$^+$14, ZF14]. Transfer was first demonstrated on various visual recognition tasks [DJV$^+$14, ZF14], then on detection, and on both instance and semantic segmentation in hybrid proposal-classifier models [GDDM14a, HAGM14a, GGAM14]. We now re-architect and fine-tune classification nets to direct, dense prediction of semantic segmentation. We chart the space of FCNs and situate prior models, both historical and recent, in this framework.

3.1.1 Fully convolutional networks

To our knowledge, the idea of extending a convnet to arbitrary-sized inputs first appeared in Matan *et al.* [MBLD91], which extended the classic LeNet [LBD$^+$89] to recognize strings of digits. Because their net was limited to one-dimensional input strings, Matan *et al.* used Viterbi decoding to obtain their outputs. Wolf and Platt [WP94] expand convnet outputs to 2-dimensional maps of detection scores for the four corners of postal address blocks. Both of these historical works do inference and learning fully convolutionally for detection. Ning *et al.* [NDL$^+$05] define a convnet for coarse multiclass segmentation of *C. elegans* tissues with fully convolutional inference.

Fully convolutional computation has also been exploited in the present era of many-layered nets. Sliding window detection by Sermanet *et al.* [SEZ$^+$14], semantic segmentation by Pinheiro and Collobert [PC14], and image restoration by Eigen *et al.* [EKF13] do fully convolutional inference. Fully convolutional training is rare, but used effectively by Tompson *et al.* [TJLB14] to learn an end-to-end part detector and spatial model for pose estimation, although they do not exposit on or analyze this method.

Alternatively, He *et al.* [HZRS14] discard the non-convolutional portion of classification nets to make a feature extractor. They combine proposals and spatial pyramid pooling to yield a localized, fixed-length feature for classification. While fast and effective, this hybrid model cannot be learned end-to-end.

3.1.2 Dense prediction with convnets

Several recent works have applied convnets to dense prediction problems, including semantic segmentation by Ning *et al.*[NDL$^+$05], Farabet *et al.*[FCNL13], and Pinheiro and Collobert [PC14]; boundary prediction for electron microscopy by Ciresan *et al.*[CGGS12] and for natural images by a hybrid convnet/nearest neighbor model by Ganin and Lempitsky [GL14]; and image restoration and depth estimation by Eigen *et al.*[EKF13, EPF14]. Common elements of these approaches include

- small models restricting capacity and receptive fields;
- patchwise training [NDL$^+$05, CGGS12, FCNL13, PC14, GL14];
- refinement by superpixel projection, random field regularization, filtering, or local classification [FCNL13, CGGS12, GL14];
- "interlacing" to obtain dense output [SEZ$^+$14, PC14, GL14];
- multi-scale pyramid processing [FCNL13, PC14, GL14];
- saturating tanh nonlinearities [FCNL13, EKF13, PC14]; and

- ensembles [CGGS12, GL14],

whereas our method does without this machinery. However, we do study patchwise training (Section 3.2.4) and "shift-and-stitch" dense output (Section 3.2.2) from the perspective of FCNs. We also discuss in-network upsampling (Section 3.2.3), of which the fully connected prediction by Eigen *et al.*[EPF14] is a special case.

Unlike these existing methods, we adapt and extend deep classification architectures, using image classification as supervised pre-training, and fine-tune fully convolutionally to learn simply and efficiently from whole image inputs and whole image ground thruths.

Hariharan *et al.*[HAGM14a] and Gupta *et al.*[GGAM14] likewise adapt deep classification nets to semantic segmentation, but do so in hybrid proposal-classifier models. These approaches fine-tune an R-CNN system [GDDM14a] by sampling bounding boxes and/or region proposals for detection, semantic segmentation, and instance segmentation. Neither method is learned end-to-end. They achieve the previous best segmentation results on PASCAL VOC and NYUDv2 respectively, so we directly compare our standalone, end-to-end FCN to their semantic segmentation results in Section 4.4.5.

We fuse features across layers to define a nonlinear local-to-global representation that we tune end-to-end. In contemporary work Hariharan *et al.* [HAGM15a] also use multiple layers in their hybrid model for semantic segmentation.

3.2 Fully convolutional networks

Each layer of data in a convnet is a three-dimensional array of size $h \times w \times d$, where h and w are spatial dimensions, and d is the feature or channel dimension. The first layer is the image, with pixel size $h \times w$, and d color channels. Locations in higher layers correspond to the locations in the image they are path-connected to, which are called their *receptive fields.*

Convnets are built on translation invariance. Their basic components (convolution, pooling, and activation functions) operate on local input regions, and depend only on *relative* spatial coordinates. Writing $\mathbf{x}_{ij}$ for the data vector at location (i, j) in a particular layer, and $\mathbf{y}_{ij}$ for the following layer, these functions compute outputs $\mathbf{y}_{ij}$ by

$$\mathbf{y}_{ij} = f_{ks}\left(\{\mathbf{x}_{si+\delta i, sj+\delta j}\}_{0\le \delta i, \delta j \le k}\right)$$

where k is called the kernel size, s is the stride or subsampling factor, and f_{ks} determines the layer type: a matrix multiplication for convolution or average pooling, a spatial max for max pooling, or an elementwise nonlinearity for an activation function, and so on for other types of layers.

This functional form is maintained under composition, with kernel size and stride obeying the transformation rule

$$f_{ks} \circ g_{k's'} = (f \circ g)_{k'+(k-1)s', ss'}.$$

While a general deep net computes a general nonlinear function, a net with only layers of this form computes a nonlinear *filter*, which we call a *deep filter* or *fully convolutional network.*

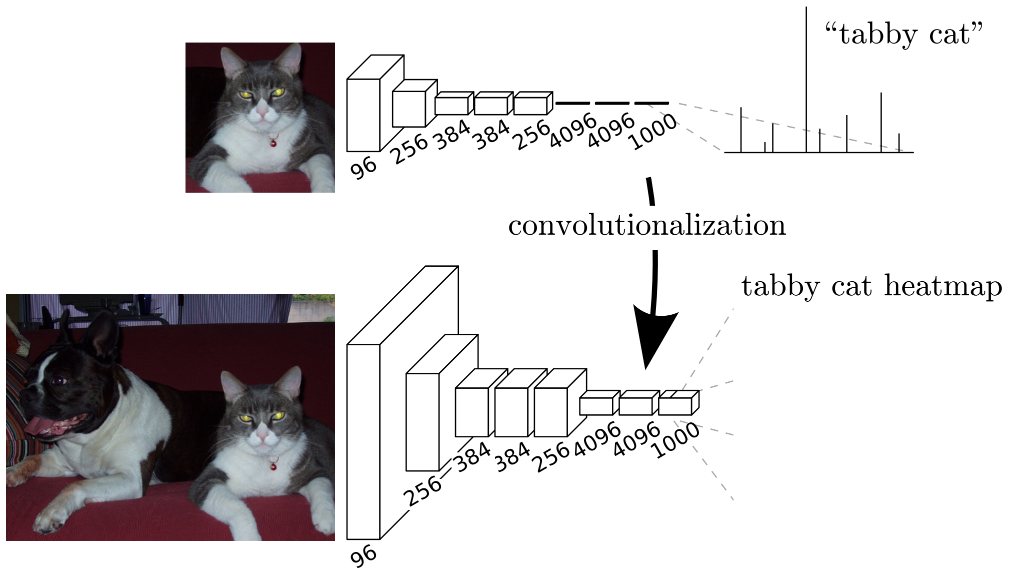


Figure 3.2: Transforming fully connected layers into convolution layers enables a classification net to output a heatmap. Adding layers and a spatial loss (as in Figure 3.1) produces an efficient machine for end-to-end dense learning.

An FCN naturally operates on an input of any size, and produces an output of corresponding (possibly resampled) spatial dimensions.

A real-valued loss function composed with an FCN defines a task. If the loss function is a sum over the spatial dimensions of the final layer, $\ell(\mathbf{x};\theta) = \sum_{ij} \ell'(\mathbf{x}_{ij};\theta)$, its gradient will be a sum over the gradients of each of its spatial components. Thus stochastic gradient descent on ℓ computed on whole images will be the same as stochastic gradient descent on ℓ', taking all of the final layer receptive fields as a minibatch.

When these receptive fields overlap significantly, both feedforward computation *and* backpropagation are much more efficient when computed layer-by-layer over an entire image instead of independently patch-by-patch.

We next explain how to convert classification nets into fully convolutional nets that produce coarse output maps. For pixelwise prediction, we need to connect these coarse outputs back to the pixels. Section 3.2.2 describes a trick, fast scanning [GCM+13], introduced for this purpose. We gain insight into this trick by reinterpreting it as an equivalent network modification. As an efficient, effective alternative, we introduce deconvolution layers for upsampling in Section 3.2.3. In Section 3.2.4 we consider training by patchwise sampling, and give evidence in Section 3.3.4 that our whole image training is faster and equally effective.

3.2.1 Adapting classifiers for dense prediction

Typical recognition nets, including LeNet [LBD+89], AlexNet [KSH12], and its deeper successors [SZ14a, SLJ+14], ostensibly take fixed-sized inputs and produce non-spatial outputs.

The fully connected layers of these nets have fixed dimensions and throw away spatial coordinates. However, these fully connected layers can also be viewed as convolutions with kernels that cover their entire input regions. Doing so casts them into fully convolutional networks that take input of any size and output classification maps. This transformation is illustrated in Figure 3.2.

Furthermore, while the resulting maps are equivalent to the evaluation of the original net on particular input patches, the computation is highly amortized over the overlapping regions of those patches. For example, while AlexNet takes 1.2 ms (on a typical GPU) to infer the classification scores of a 227×227 image, the fully convolutional net takes 22 ms to produce a 10×10 grid of outputs from a 500×500 image, which is more than 5 times faster than the naïve approach[2].

The spatial output maps of these convolutionalized models make them a natural choice for dense problems like semantic segmentation. With ground truth available at every output cell, both the forward and backward passes are straightforward, and both take advantage of the inherent computational efficiency (and aggressive optimization) of convolution. The corresponding backward times for the AlexNet example are 2.4 ms for a single image and 37 ms for a fully convolutional 10×10 output map, resulting in a speedup similar to that of the forward pass.

While our reinterpretation of classification nets as fully convolutional yields output maps for inputs of any size, the output dimensions are typically reduced by subsampling. The classification nets subsample to keep filters small and computational requirements reasonable. This coarsens the output of a fully convolutional version of these nets, reducing it from the size of the input by a factor equal to the pixel stride of the receptive fields of the output units.

3.2.2 Shift-and-stitch is filter dilation

Dense predictions can be obtained from coarse outputs by stitching together outputs from shifted versions of the input. If the output is downsampled by a factor of f, shift the input x pixels to the right and y pixels down, once for every (x, y) such that $0 \leq x, y < f$. Process each of these f^2 inputs, and interlace the outputs so that the predictions correspond to the pixels at the *centers* of their receptive fields.

Although this transformation naïvely increases the cost by a factor of f^2, there is a well-known trick for efficiently producing identical results [GCM+13, SEZ+14]. (This trick is also used in the algorithme à trous [HKMMT89, Mal99] for wavelet transforms and related to the Noble identities [Vai90] from signal processing.)

Consider a layer (convolution or pooling) with input stride s, and a subsequent convolution layer with filter weights f_{ij} (eliding the irrelevant feature dimensions). Setting the earlier layer's input stride to one upsamples its output by a factor of s. However, convolving

[2]Assuming efficient batching of image inputs. The classification scores for a single, unbatched image take 5.4 ms to produce, which is nearly 25 times slower than the fully convolutional version.

the original filter with the upsampled output does not produce the same result as shift-and-stitch, because the original filter only sees a reduced portion of its (now upsampled) input. To produce the same result, dilate (or "rarefy") the filter by forming

$$f'_{ij} = \begin{cases} f_{i/s,j/s} & \text{if } s \text{ divides both } i \text{ and } j; \\ 0 & \text{otherwise,} \end{cases}$$

(with i and j zero-based). Reproducing the full net output of shift-and-stitch involves repeating this filter enlargement layer-by-layer until all subsampling is removed. (In practice, this can be done efficiently by processing subsampled versions of the upsampled input.)

Simply decreasing subsampling within a net is a tradeoff: the filters see finer information, but have smaller receptive fields and take longer to compute. This dilation trick is another kind of tradeoff: the output is denser without decreasing the receptive field sizes of the filters, but the filters are prohibited from accessing information at a finer scale than their original design.

Although we have done preliminary experiments with dilation, we do not use it in our model. We find learning through upsampling, as described in the next section, to be effective and efficient, especially when combined with the skip layer fusion described later on. For further detail regarding dilation, refer to the dilated FCN of [YK16].

3.2.3 Upsampling is (fractionally strided) convolution

Another way to connect coarse outputs to dense pixels is interpolation. For instance, simple bilinear interpolation computes each output y_{ij} from the nearest four inputs by a linear map that depends only on the relative positions of the input and output cells.

In a sense, upsampling with factor f is convolution with a *fractional* input stride of $1/f$. So long as f is integral, it's natural to implement upsampling by convolution with an *output* stride of f. Such an operation is trivial to implement, since it simply reverses the forward and backward passes of more typical input-strided convolution. Thus upsampling is performed in-network for end-to-end learning by backpropagation from the pixelwise loss.

By an abuse of terminology, these layers are sometimes refered to as *deconvolution* layers. Note that the convolution filter in such a layer need not be fixed (e.g., to bilinear upsampling), but can be learned. A stack of deconvolution layers and activation functions can even learn a nonlinear upsampling.

In our experiments, we find that in-network upsampling is fast and effective for learning dense prediction. Our best segmentation architecture uses these layers to learn to upsample for refined prediction in Section 3.3.2.

3.2.4 Patchwise training is loss sampling

In stochastic optimization, gradient computation is driven by the training distribution. Both patchwise training and fully convolutional training can be made to produce any distribution, although their relative computational efficiency depends on overlap and minibatch size.

Whole image fully convolutional training is identical to patchwise training where each batch consists of all the receptive fields of the units below the loss for an image (or collection of images). While this is more efficient than uniform sampling of patches, it reduces the number of possible batches. However, random selection of patches within an image may be recovered simply. Restricting the loss to a randomly sampled subset of its spatial terms (or, equivalently applying a DropConnect mask [WZZ$^+$13] between the output and the loss) excludes patches from the gradient computation.

If the kept patches still have significant overlap, fully convolutional computation will still speed up training. If gradients are accumulated over multiple backward passes, batches can include patches from several images.[3]

Sampling in patchwise training can correct class imbalance [NDL$^+$05, FCNL13, CGGS12] and mitigate the spatial correlation of dense patches [PC14, HAGM14a]. In fully convolutional training, class balance can also be achieved by weighting the loss, and loss sampling can be used to address spatial correlation.

We explore training with sampling in Section 3.3.4, and do not find that it yields faster or better convergence for dense prediction. Whole image training is effective and efficient.

3.3 Segmentation Architecture

We cast ILSVRC classifiers into FCNs and augment them for dense prediction with in-network upsampling and a pixelwise loss. We train for segmentation by fine-tuning. Next, we add skips between layers to fuse coarse, semantic and local, appearance information. This skip architecture is learned end-to-end to refine the semantics and spatial precision of the output.

For this investigation, we train and validate on the PASCAL VOC 2011 segmentation challenge [EVGW$^+$]. We train with a per-pixel multinomial logistic loss and validate with the standard metric of mean pixel intersection over union, with the mean taken over all classes, including background. The training ignores pixels that are masked out (as ambiguous or difficult) in the ground truth.

3.3.1 From classifier to dense FCN

We begin by convolutionalizing proven classification architectures as in Section 3.2. We consider the AlexNet[4] architecture [KSH12] that won ILSVRC12, as well as the VGG nets [SZ14a] and the GoogLeNet[5] [SLJ$^+$14] which did exceptionally well in ILSVRC14. We pick

[3]Note that not every possible patch is included this way, since the receptive fields of the final layer units lie on a fixed, strided grid. However, by shifting the image right and down by a random value up to the stride, random selection from all possible patches may be recovered.

[4]Using the publicly available `CaffeNet` reference model.

[5]Since there is no publicly available version of GoogLeNet, we use our own reimplementation. Our version is trained with less extensive data augmentation, and gets 68.5% top-1 and 88.4% top-5 ILSVRC accuracy.

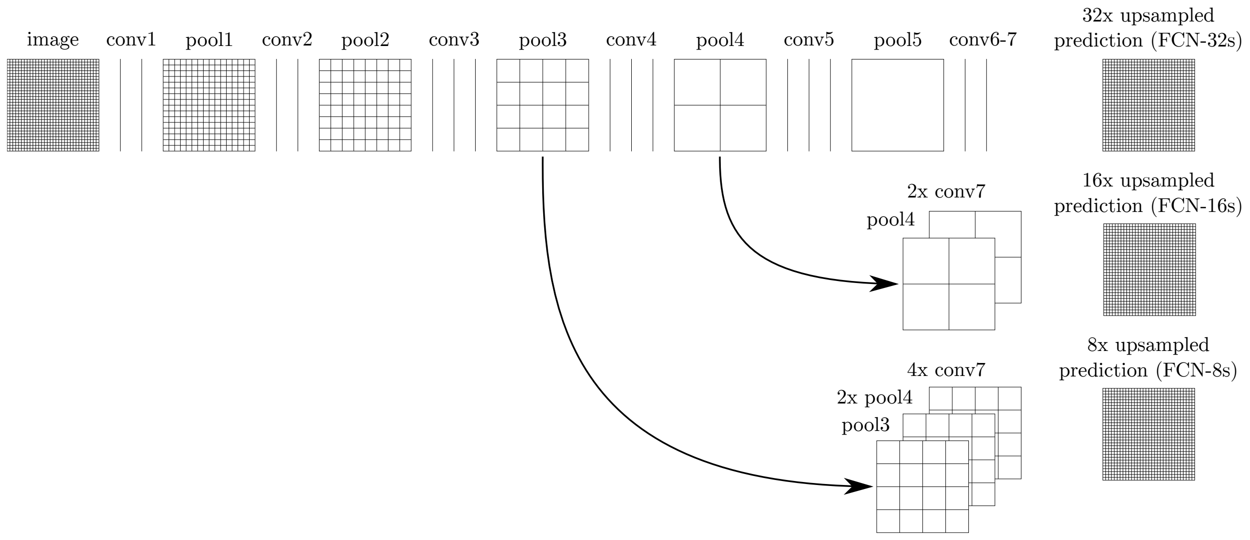


image
conv1
pool1
conv2
pool2
conv3
pool3
conv4
pool4
conv5
pool5
conv6-7
32x upsampled prediction (FCN-32s)
2x conv7
pool4
16x upsampled prediction (FCN-16s)
4x conv7
2x pool4
pool3
8x upsampled prediction (FCN-8s)

Figure 3.3: Our DAG nets learn to combine coarse, high layer information with fine, low layer information. Pooling and prediction layers are shown as grids that reveal relative spatial coarseness, while intermediate layers are shown as vertical lines. First row (FCN-32s): Our single-stream net, described in Section 3.3.1, upsamples stride 32 predictions back to pixels in a single step. Second row (FCN-16s): Combining predictions from both the final layer and the `pool4` layer, at stride 16, lets our net predict finer details, while retaining high-level semantic information. Third row (FCN-8s): Additional predictions from `pool3`, at stride 8, provide further precision.

the VGG 16-layer net[6], which we found to be equivalent to the 19-layer net on this task. For GoogLeNet, we use only the final loss layer, and improve performance by discarding the final average pooling layer. We decapitate each net by discarding the final classifier layer, and convert all fully connected layers to convolutions. We append a 1×1 convolution with channel dimension 21 to predict scores for each of the PASCAL classes (including background) at each of the coarse output locations, followed by a deconvolution layer to bilinearly upsample the coarse outputs to pixel-dense outputs as described in Section 3.2.3. Table 3.1 compares the preliminary validation results along with the basic characteristics of each net. We report the best results achieved after convergence at a fixed learning rate (at least 175 epochs).

Fine-tuning from classification to segmentation gave reasonable predictions for each net. Even the worst model achieved $\sim 75\%$ of state-of-the-art performance. The segmentation-equipped VGG net (FCN-VGG16) already appears to be state-of-the-art at 56.0 mean IU on val, compared to 52.6 on test [HAGM14a]. Training on extra data raises FCN-VGG16 to 59.4 mean IU and FCN-AlexNet to 48.0 mean IU on a subset of val[7]. Despite similar classification accuracy, our implementation of GoogLeNet did not match the VGG16 segmentation result.

[6]Using the publicly available version from the Caffe model zoo.

Table 3.1: We adapt and extend three classification convnets. We compare performance by mean intersection over union on the validation set of PASCAL VOC 2011 and by inference time (averaged over 20 trials for a 500×500 input on an NVIDIA Tesla K40c). We detail the architecture of the adapted nets with regard to dense prediction: number of parameter layers, receptive field size of output units, and the coarsest stride within the net. (These numbers give the best performance obtained at a fixed learning rate, not best performance possible.)

	FCN-AlexNet	FCN-VGG16	FCN-GoogLeNet[4]
mean IU	39.8	**56.0**	42.5
forward time	50 ms	210 ms	59 ms
conv. layers	8	16	22
parameters	57M	134M	6M
rf size	355	404	907
max stride	32	32	32

Table 3.2: Comparison of skip FCNs on a subset[7] of PASCAL VOC 2011 segval. Learning is end-to-end, except for FCN-32s-fixed, where only the last layer is fine-tuned. Note that FCN-32s is FCN-VGG16, renamed to highlight stride.

	pixel acc.	mean acc.	mean IU	f.w. IU
FCN-32s-fixed	83.0	59.7	45.4	72.0
FCN-32s	89.1	73.3	59.4	81.4
FCN-16s	90.0	75.7	62.4	83.0
FCN-8s	**90.3**	**75.9**	**62.7**	**83.2**

3.3.2 Combining what and where

We define a new fully convolutional net (FCN) for segmentation that combines layers of the feature hierarchy and refines the spatial precision of the output. See Figure 3.3.

While fully convolutionalized classifiers can be fine-tuned to segmentation as shown in 3.3.1, and even score highly on the standard metric, their output is dissatisfyingly coarse (see Figure 3.4). The 32 pixel stride at the final prediction layer limits the scale of detail in the upsampled output.

We address this by adding skips [Bis06] that combine the final prediction layer with lower layers with finer strides. This turns a line topology into a DAG, with edges that skip ahead from lower layers to higher ones (Figure 3.3). As they see fewer pixels, the finer scale predictions should need fewer layers, so it makes sense to make them from shallower net outputs. Combining fine layers and coarse layers lets the model make local predictions that respect global structure. This crossing of layers and resolutions is a learned, nonlinear

Table 3.3: Comparison of skip FCNs on a subset[7] of PASCAL VOC 2011 segval. Learning is end-to-end, except for FCN-32s-fixed, where only the last layer is fine-tuned. Note that FCN-32s is FCN-VGG16, renamed to highlight stride.

	pixel acc.	mean acc.	mean IU	f.w. IU
FCN-32s	89.1	73.3	59.4	81.4
FCN-16s	90.0	75.7	62.4	83.0
FCN-8s	**90.3**	**75.9**	**62.7**	**83.2**

counterpart to the multi-scale representation of the Laplacian pyramid [BA83]. By analogy to the jet of Koenderick and van Doorn [KvD87], we call our feature hierarchy the *deep jet*.

We first divide the output stride in half by predicting from a 16 pixel stride layer. We add a 1×1 convolution layer on top of `pool4` to produce additional class predictions. We fuse this output with the predictions computed on top of `conv7` (convolutionalized `fc7`) at stride 32 by adding a $2\times$ upsampling layer and summing[7] both predictions (see Figure 3.3). We initialize the $2\times$ upsampling to bilinear interpolation, but allow the parameters to be learned as described in Section 3.2.3. Finally, the stride 16 predictions are upsampled back to the image. We call this net FCN-16s. FCN-16s is learned end-to-end, initialized with the parameters of the last, coarser net, which we now call FCN-32s. The new parameters acting on `pool4` are zero-initialized so that the net starts with unmodified predictions. The learning rate is decreased by a factor of 100.

Learning this skip net improves performance on the validation set by 3.0 mean IU to 62.4. Figure 3.4 shows improvement in the fine structure of the output. We compared this fusion with learning only from the `pool4` layer, which resulted in poor performance, and simply decreasing the learning rate without adding the skip, which resulted in an insignificant performance improvement without improving the quality of the output.

We continue in this fashion by fusing predictions from `pool3` with a $2\times$ upsampling of predictions fused from `pool4` and `conv7`, building the net FCN-8s. We obtain a minor additional improvement to 62.7 mean IU, and find a slight improvement in the smoothness and detail of our output. At this point our fusion improvements have met diminishing returns, both with respect to the IU metric which emphasizes large-scale correctness, and also in terms of the improvement visible e.g. in Figure 3.4, so we do not continue fusing even lower layers.

3.3.3 Refinement by other means

Decreasing the stride of pooling layers is the most straightforward way to obtain finer predictions. However, doing so is problematic for our VGG16-based net. Setting the `pool5` stride to 1 requires our convolutionalized `fc6` to have kernel size 14×14 to maintain its receptive

[7]Max fusion made learning difficult due to gradient switching.

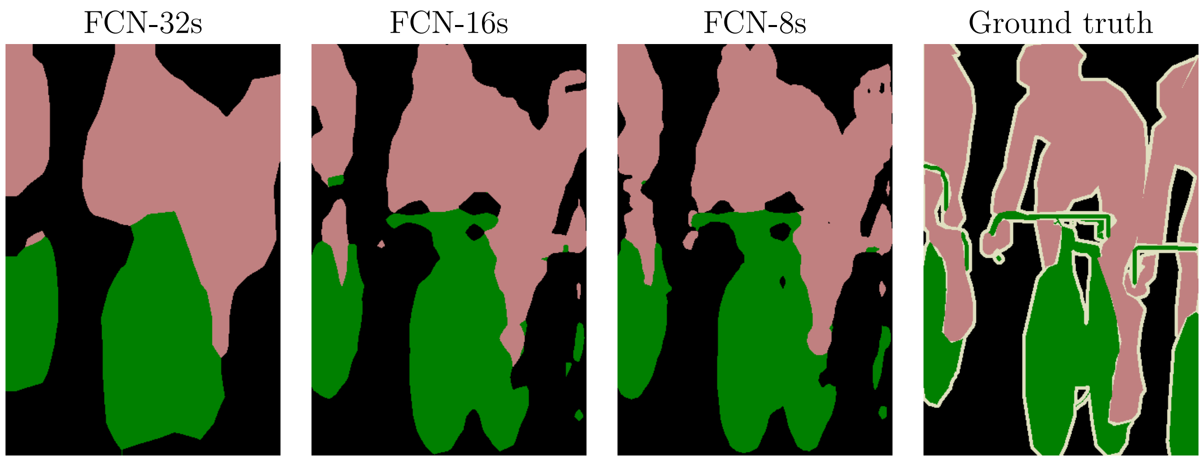


Figure 3.4: Refining fully convolutional nets by fusing information from layers with different strides improves segmentation detail. The first three images show the output from our 32, 16, and 8 pixel stride nets (see Figure 3.3).

field size. In addition to their computational cost, we had difficulty learning such large filters. We attempted to re-architect the layers above `pool5` with smaller filters, but did not achieve comparable performance; one possible explanation is that the ILSVRC initialization of the upper layers is important.

Another way to obtain finer predictions is to use the shift-and-stitch trick described in Section 3.2.2. In limited experiments, we found the cost to improvement ratio from this method to be worse than layer fusion.

3.3.4 Experimental framework

3.3.5 Optimization

We train by SGD with momentum. We use a minibatch size of 20 images and fixed learning rates of 10^{-3}, 10^{-4}, and 5^{-5} for FCN-AlexNet, FCN-VGG16, and FCN-GoogLeNet, respectively, chosen by line search. We use momentum 0.9, weight decay of 5^{-4} or 2^{-4}, and doubled learning rate for biases, although we found training to be sensitive to the learning rate alone. We zero-initialize the class scoring layer, as random initialization yielded neither better performance nor faster convergence. Dropout was included where used in the original classifier nets.

3.3.6 Fine-tuning

We fine-tune all layers by back-propagation through the whole net. Fine-tuning the output classifier alone yields only 70% of the full fine-tuning performance as compared in Table 3.3. Training from scratch is not feasible considering the time required to learn the base classification nets. (Note that the VGG net is trained in stages, while we initialize from the full 16-layer version.) Fine-tuning takes three days on a single GPU for the coarse FCN-32s version, and about one day each to upgrade to the FCN-16s and FCN-8s versions.

3.3.7 More Training Data

The PASCAL VOC 2011 segmentation training set labels 1112 images. Hariharan *et al.* [HAB$^+$11] collected labels for a larger set of 8498 PASCAL training images, which was used to train the previous state-of-the-art system, SDS [HAGM14a]. This training data improves the FCN-VGG16 validation score[8] by 3.4 points to 59.4 mean IU.

3.3.8 Patch Sampling

As explained in Section 3.2.4, our full image training effectively batches each image into a regular grid of large, overlapping patches. By contrast, prior work randomly samples patches over a full dataset [NDL$^+$05, CGGS12, FCNL13, PC14, GL14], potentially resulting in higher variance batches that may accelerate convergence [LBOM98]. We study this tradeoff by spatially sampling the loss in the manner described earlier, making an independent choice to ignore each final layer cell with some probability $1 - p$. To avoid changing the effective batch size, we simultaneously increase the number of images per batch by a factor $1/p$. Note that due to the efficiency of convolution, this form of rejection sampling is still faster than patchwise training for large enough values of p (e.g., at least for $p > 0.2$ according to the numbers in Section 3.2.1). Figure 3.5 shows the effect of this form of sampling on convergence. We find that sampling does not have a significant effect on convergence rate compared to whole image training, but takes significantly more time due to the larger number of images that need to be considered per batch. We therefore choose unsampled, whole image training in our other experiments.

3.3.9 Class Balancing

Fully convolutional training can balance classes by weighting or sampling the loss. Although our labels are mildly unbalanced (about 3/4 are background), we find class balancing unnecessary.

3.3.10 Dense Prediction

The scores are upsampled to the input dimensions by deconvolution layers within the net. Final layer deconvolutional filters are fixed to bilinear interpolation, while intermediate upsampling layers are initialized to bilinear upsampling, and then learned.

[8]There are training images from [HAB$^+$11] included in the PASCAL VOC 2011 val set, so we validate on the non-intersecting set of 736 images.

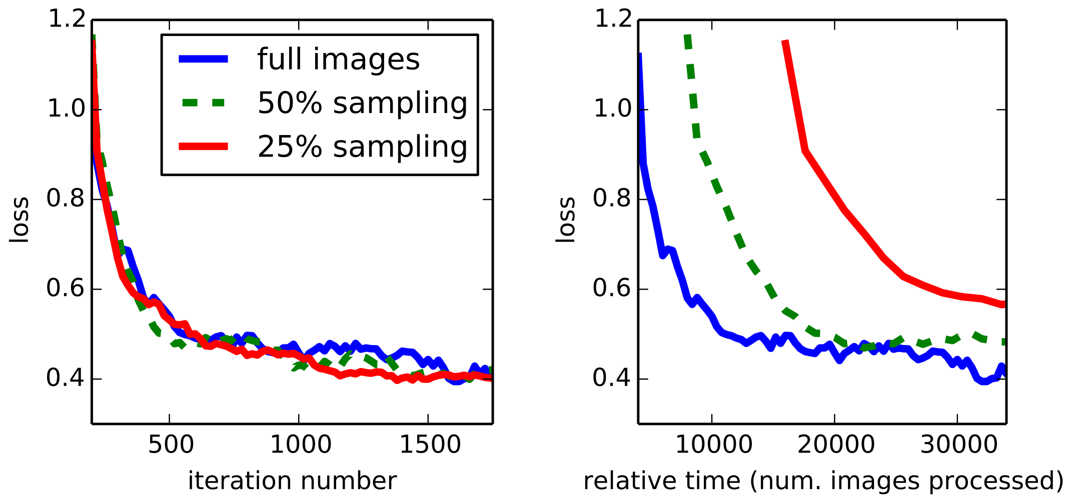


Figure 3.5: Training on whole images is just as effective as sampling patches, but results in faster (wall time) convergence by making more efficient use of data. Left shows the effect of sampling on convergence rate for a fixed expected batch size, while right plots the same by relative wall time.

3.3.11 Augmentation

We tried augmenting the training data by randomly mirroring and "jittering" the images by translating them up to 32 pixels (the coarsest scale of prediction) in each direction. This yielded no noticeable improvement.

3.3.12 Implementation

All models are trained and tested with Caffe [JSD$^+$14] on a single NVIDIA Tesla K40c. Our models and code are publicly available at
`http://fcn.berkeleyvision.org`.

3.4 Results

We test our FCN on semantic segmentation and scene parsing, exploring PASCAL VOC, NYUDv2, and SIFT Flow. Although these tasks have historically distinguished between objects and regions, we treat both uniformly as pixel prediction. We evaluate our FCN skip architecture on each of these datasets, and then extend it to multi-modal input for NYUDv2 and multi-task prediction for the semantic and geometric labels of SIFT Flow.

3.4.1 Metrics

We report four metrics from common semantic segmentation and scene parsing evaluations that are variations on pixel accuracy and region intersection over union (IU). Let n_{ij} be the number of pixels of class i predicted to belong to class j, where there are n_{cl} different classes, and let $t_i = \sum_j n_{ij}$ be the total number of pixels of class i. We compute:

- pixel accuracy: $\sum_i n_{ii} / \sum_i t_i$
- mean accuraccy: $(1/n_{\text{cl}}) \sum_i n_{ii}/t_i$
- mean IU: $(1/n_{\text{cl}}) \sum_i n_{ii} / \left(t_i + \sum_j n_{ji} - n_{ii} \right)$
- frequency weighted IU: $(\sum_k t_k)^{-1} \sum_i t_i n_{ii} / \left(t_i + \sum_j n_{ji} - n_{ii} \right)$

3.4.2 PASCAL VOC

Table 3.4 gives the performance of our FCN-8s on the test sets of PASCAL VOC 2011 and 2012, and compares it to the previous state-of-the-art, SDS [HAGM14a], and the well-known R-CNN [GDDM14a]. We achieve the best results on mean IU[9] by a relative margin of 20%. Inference time is reduced 114× (convnet only, ignoring proposals and refinement) or 286× (overall).

Table 3.4: Our fully convolutional net gives a 20% relative improvement over the state-of-the-art on the PASCAL VOC 2011 and 2012 test sets and reduces inference time.

	mean IU VOC2011 test	mean IU VOC2012 test	inference time
R-CNN [GDDM14a]	47.9	-	-
SDS [HAGM14a]	52.6	51.6	$\sim$ 50 s
FCN-8s	**62.7**	**62.2**	$\sim$ **175 ms**

NYUDv2 [SHKF12] is an RGB-D dataset collected using the Microsoft Kinect. It has 1449 RGB-D images, with pixelwise labels that have been coalesced into a 40 class semantic segmentation task by Gupta *et al.* [GAM13]. We report results on the standard split of 795 training images and 654 testing images. (Note: all model selection is performed on PASCAL 2011 val.) Table 3.5 gives the performance of our model in several variations. First we train our unmodified coarse model (FCN-32s) on RGB images. To add depth information, we train on a model upgraded to take four-channel RGB-D input (early fusion). This provides little benefit, perhaps due to the difficultly of propagating meaningful gradients all the way through the model. Following the success of Gupta *et al.* [GGAM14], we try the three-dimensional HHA encoding of depth, training nets on just this information, as well as a

[9]This is the only metric provided by the test server.

Table 3.5: Results on NYUDv2. *RGBD* is early-fusion of the RGB and depth channels at the input. *HHA* is the depth embedding of [GGAM14] as horizontal disparity, height above ground, and the angle of the local surface normal with the inferred gravity direction. *RGB-HHA* is the jointly trained late fusion model that sums RGB and HHA predictions.

	pixel acc.	mean acc.	mean IU	f.w. IU
Gupta *et al.* [GGAM14]	60.3	-	28.6	47.0
FCN-32s RGB	60.0	42.2	29.2	43.9
FCN-32s RGBD	61.5	42.4	30.5	45.5
FCN-32s HHA	57.1	35.2	24.2	40.4
FCN-32s RGB-HHA	64.3	44.9	32.8	48.0
FCN-16s RGB-HHA	**65.4**	**46.1**	**34.0**	**49.5**

“late fusion” of RGB and HHA where the predictions from both nets are summed at the final layer, and the resulting two-stream net is learned end-to-end. Finally we upgrade this late fusion net to a 16-stride version.

SIFT Flow is a dataset of 2,688 images with pixel labels for 33 semantic categories (“bridge”, “mountain”, “sun”), as well as three geometric categories (“horizontal”, “vertical”, and “sky”). An FCN can naturally learn a joint representation that simultaneously predicts both types of labels. We learn a two-headed version of FCN-16s with semantic and geometric prediction layers and losses. The learned model performs as well on both tasks as two independently trained models, while learning and inference are essentially as fast as each independent model by itself. The results in Table 3.6, computed on the standard split into 2,488 training and 200 test images,[10] show state-of-the-art performance on both tasks.

3.5 Upper Bounds on IU

In this chapter, we have achieved good performance on the mean IU segmentation metric even with coarse semantic prediction. To better understand this metric and the limits of this approach with respect to it, we compute approximate upper bounds on performance with prediction at various scales. We do this by downsampling ground truth images and then upsampling them again to simulate the best results obtainable with a particular downsampling factor. The following table gives the mean IU on a subset of PASCAL 2011 val for various downsampling factors.

[10]Three of the SIFT Flow categories are not present in the test set. We made predictions across all 33 categories, but only included categories actually present in the test set in our evaluation.

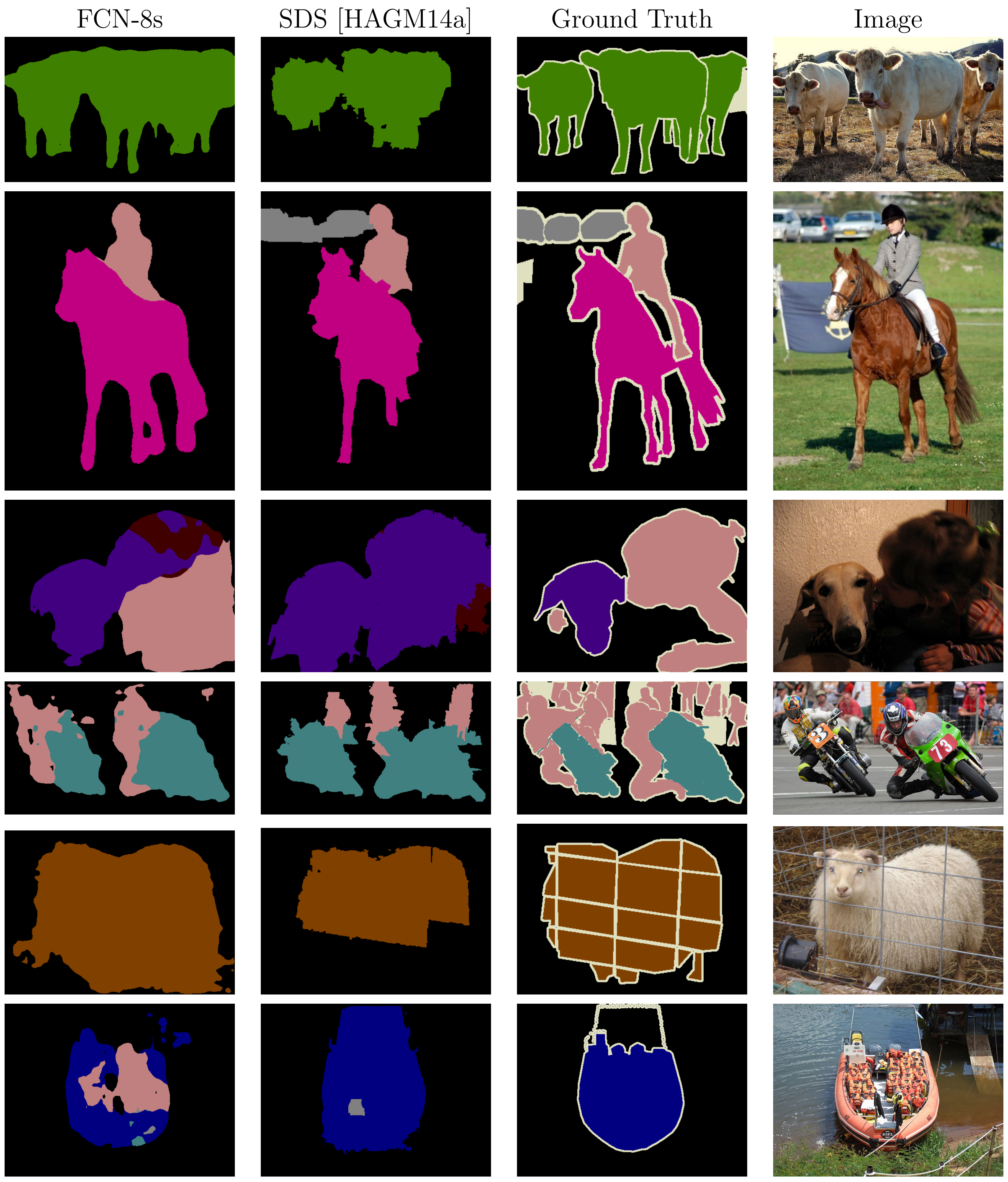


Figure 3.6: Fully convolutional segmentation nets produce state-of-the-art performance on PASCAL. The left column shows the output of our highest performing net, FCN-8s. The second shows the segmentations produced by the previous state-of-the-art system by Hariharan *et al.* [HAGM14a]. Notice the fine structures recovered (first row), ability to separate closely interacting objects (second row), and robustness to occluders (third row). The fourth row shows a failure case: the net sees lifejackets in a boat as people.

Table 3.6: Results on SIFT Flow[9] with class segmentation (center) and geometric segmentation (right). Tighe [TL10] is a non-parametric transfer method. Tighe 1 is an exemplar SVM while 2 is SVM + MRF. Farabet is a multi-scale convnet trained on class-balanced samples (1) or natural frequency samples (2). Pinheiro is a multi-scale, recurrent convnet, denoted RCNN$_3$ ($\circ^3$). The metric for geometry is pixel accuracy.

	pixel acc.	mean acc.	mean IU	f.w. IU	geom. acc.
Liu *et al.* [LYT11]	76.7	-	-	-	-
Tighe *et al.* [TL10]	-	-	-	-	90.8
Tighe *et al.* [TL13] 1	75.6	41.1	-	-	-
Tighe *et al.* [TL13] 2	78.6	39.2	-	-	-
Farabet *et al.* [FCNL13] 1	72.3	50.8	-	-	-
Farabet *et al.* [FCNL13] 2	78.5	29.6	-	-	-
Pinheiro *et al.* [PC14]	77.7	29.8	-	-	-
FCN-16s	**85.2**	**51.7**	39.5	76.1	**94.3**

factor	mean IU
128	50.9
64	73.3
32	86.1
16	92.8
8	96.4
4	98.5

Pixel-perfect prediction is clearly not necessary to achieve mean IU well above state-of-the-art, and, conversely, mean IU is a not a good measure of fine-scale accuracy.

3.6 More Results

We further evaluate our FCN for semantic segmentation.

PASCAL-Context [MCL$^+$14] provides whole scene annotations of PASCAL VOC 2010. While there are over 400 distinct classes, we follow the 59 class task defined by [MCL$^+$14] that picks the most frequent classes. We train and evaluate on the training and val sets respectively. In Table 3.7, we compare to the joint object + stuff variation of Convolutional Feature Masking [DHS14] which is the previous state-of-the-art on this task. FCN-8s scores 37.8 mean IU for a 20% relative improvement.

Table 3.7: Results on PASCAL-Context. *CFM* is the best result of [DHS14] by convolutional feature masking and segment pursuit with the VGG net. *O_2P* is the second order pooling method [CCBS12] as reported in the *errata* of [MCL+14]. The 59 class task selects the 59 most frequent classes for evaluation.

59 class	pixel acc.	mean acc.	mean IU	f.w. IU
O_2P	-	-	18.1	-
CFM	-	-	31.5	-
FCN-32s	65.4	47.2	35.1	50.3
FCN-16s	66.8	49.6	37.6	52.3
FCN-8s	**67.0**	**50.7**	**37.8**	**52.5**

3.7 Conclusion

Fully convolutional networks are a rich class of models, of which modern classification convnets are a special case. Recognizing this, extending these classification nets to segmentation, and improving the architecture with multi-resolution layer combinations dramatically improves the state-of-the-art, while simultaneously simplifying and speeding up learning and inference.

3.8 Further Progress in Segmentation

In this historical addendum, we examine progress on the PASCAL VOC segmentation benchmark before and after the public release of the system described in this chapter. Figure 3.7 plots publicly-posted scores [com] on the PASCAL VOC "comp6" test set against the time of their submission. Note that the points on this plot reflect both technological and social forces; each point appears only after it becomes feasible to obtain the corresponding score, and effort is invested to actually do so. Keeping this conflation in mind, a number of interesting trends are revealed by this plot. Observe:

- Little progress was made in the first two years of competition results. This is despite the AlexNet-precipitated advances in classification, which began near the left side of the plot, and subsequent advances in detection.
- Significant progress was made simultaneously by a number of works, corresponding to the CVPR 2015 submission cycle. Specifically, the results presented here (originally in [LSD15]), Hypercolumn SDS [HAGM15b], Zoomout [MYS15], and CFM [DHS14] all achieved relative improvements of 13-22% over the previous state-of-the-art. The VGG networks [SZ14b] (and, in particular, the 16-layer "VGG-D" network, as used here) are a key unifying element in these works. Note that these networks were made

publicly available on September 24, 2014. (The first results were posted on November 12, 2014—even though these results can be obtained in hours from the VGG weights.)

- With the exception of further Zoomout results, all subsequent segmentation numbers on this benchmark have built on the ideas and networks developed herein, and the rate of submission has increased dramatically. This is likely due to a combination of excitement resulting from the original jump in performance, along with the simplification of experimental methods (and corresponding drop in iteration time) afforded by addressing the problem end-to-end with a small set of very efficient parts. The improvements over the original FCN number result from a menagerie of sources, including additional CRF processing, much larger training sets (especially COCO [LMB+14]), better training methodology and hyperparameters, normalization techniques, and network architectural improvements [CPK+15, ZJRP+15, LRB15, NHH15, RFB15, YK16].

- At the time of this writing, there appears to be a performance plateau below 80% I/U. It's likely that new ideas are necessary to make further significant progress. (Although those may not be new segmentation ideas, but rather ideas resulting in a more powerful initial network.)

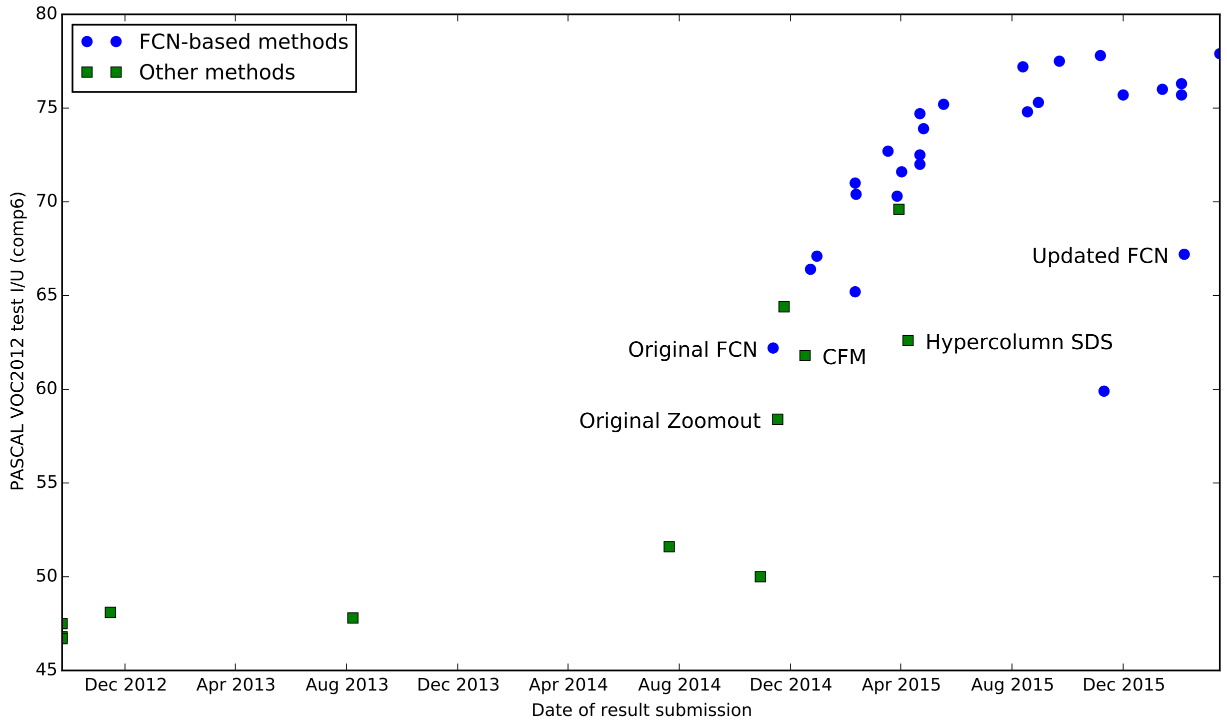


Figure 3.7: Plotting performance on the PASCAL "comp6" segmentation benchmark reveals a number of interesting trends; see text for analysis. In this plot, submission time is on the x-axis, while performance on the "comp6" test set, in terms of mean intersection-over-union, is on the y-axis. (Note that scores may be made public after they have been submitted for evaluation, so it is possible, though unlikely, for additional points to retroactively appear on this plot in the future.) A few points are annotated with method names, including the four works which originally and independently achieved a signficant boost in performance by exploiting the VGG architecture and pretraining, as well as an updated result obtained by the "vanilla" method developed in this chapter, but with improved hyperparameters [FCN]. The blue circles are methods that are or include fully convolutional networks, generally trained end-to-end. The green squares are either traditional methods, lacking convnet features altogether, or methods that use convnet features but don't build networks directly connecting image to output, instead relying on extra proposal and/or feature extraction steps. See [com] for the raw data, including a complete list of method names, descriptions, and citations.

Chapter 4

Producing Object Segments: Pairwise Networks

4.1 Introduction

As we've learned in preceeding chapters, convolutional networks are natural tools for *local* prediction, where input and output are translation-covariant. In particular, as we've seen in the last chapter, convnets are state-of-the-art models for semantic segmentation, or labeling each pixel with the category of its containing object.

Although semantic segmentation identifies category boundaries, a pixel-labeling network lacks a notion of objects, and cannot distinguish adjacent objects of the same class. In fact, translation-covariance means that *no* pure convnet output can assign a label to every pixel so that adjacent but otherwise identical objects have distinct labels.

In this chapter, we view the task of finding object regions as a classification problem on *pairs* of pixels. Rather than labeling every pixel to indicate the object it belongs to, we'd like to label every pair of pixels as either connected or disconnected. Note that there is no conflict with translation-covariance here—the connectedness of each pixel to other pixels is a function of the local content of the image. Furthermore, despite the apparent quadratic increase in output size, we can achieve remarkably efficient learning and inference by sharing most of the computation and making coarse predictions where possible.

With this conception of computation on pairs comes a new palette of linear operations on higher-dimensional arrays. We'll explore this in detail and relate the operations we describe to existing and contemporary approaches. Then, we'll conduct a detailed empirical look at existing region-generating networks [PCD15], which will lead to the finding that one of the simplest such operations—a type of transposition—improves region quality at almost no cost.

In addition, we compress the time needed for inference and learning by introducing the *natural feature pyramid* into region generation. Rather than obtaining multiscale regions by running each plane of an image pyramid through an entire net, we build a single net

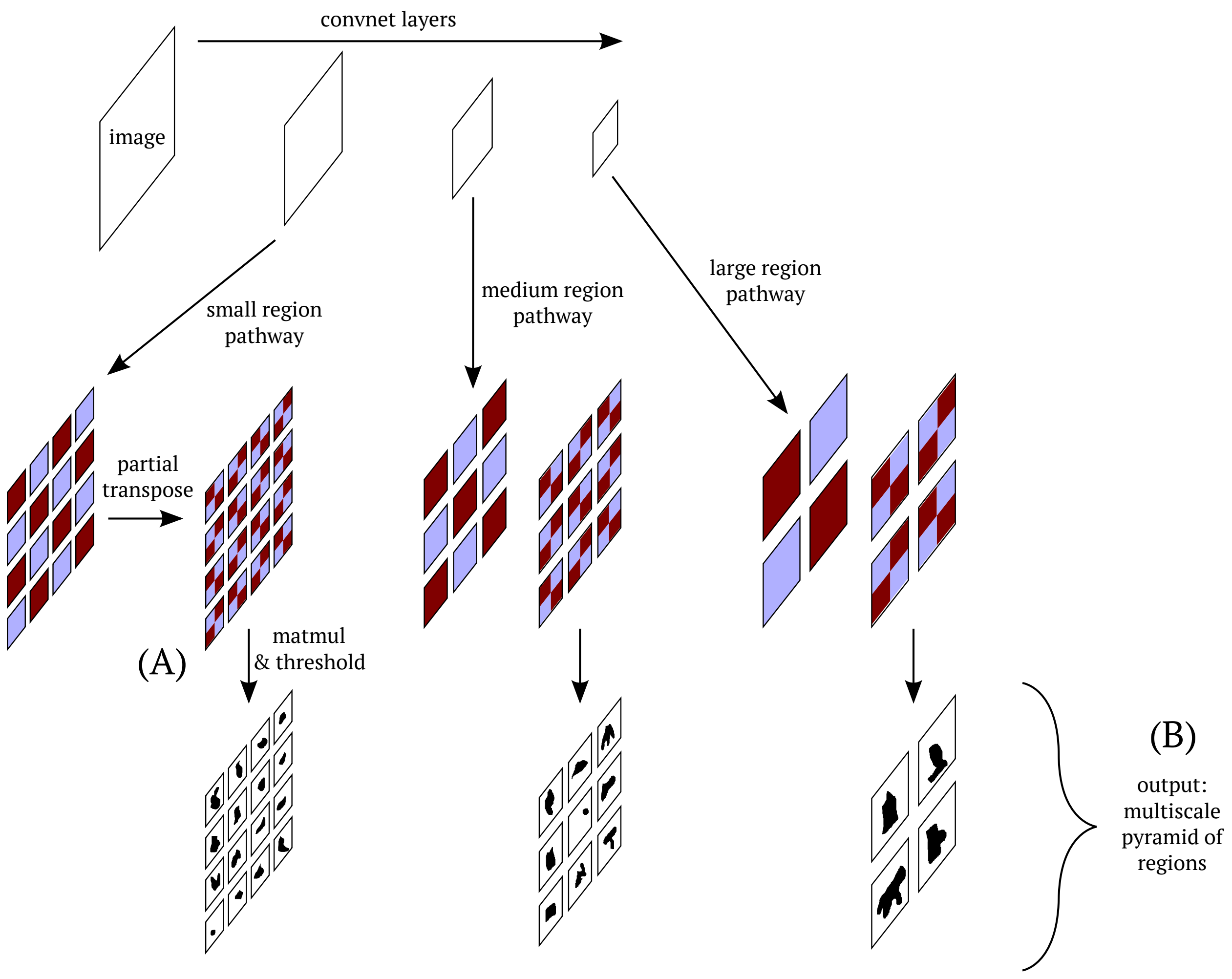


Figure 4.1: Efficient and effective region generating networks result from the right simple but subtle design choices. This schematic illustrates the two main elements we develop in this chapter. First, (A) inserting a special transposition and an extra matrix multiplication boosts the effectiveness of convolutional region prediction, as detailed in Section 4.3.3. Second, (B) building multiple region prediction pathways that branch off different layers is an efficient way to produce regions at multiple scales, especially when compared to the standard practice of sending an entire image pyramid through the network (Section 4.3.4).

with multiple output pathways that generate regions of different scales. In our model, both learning and inference are performed on whole images, without any resizing or cropping, and all pathways are trained jointly and end-to-end.

4.2 Related Work

4.2.1 DeepMask

Our design is most closely allied to that of Pinheiro et al. [PCD15], who also generate regions directly from convolutional networks, treating the problem as binary classification. We maintain and corroborate many of the choices made by DeepMask. In particular, we find that when training region generating networks, it's useful to ignore non-object regions, to group positive region examples by scale, and to avoid downsampling ground truth. However, we take a broader view of region prediction in terms of pairwise networks, and in doing so we find a simple way to improve any DeepMask-like system. In addition, we demonstrate the advantages of the natural feature pyramid over DeepMask's (and others') image pyramids. We directly compare to DeepMask in Section 4.4.5.

4.2.2 Semantic Segmentation

Our single-network, jointly trained, end-to-end design draws inspiration from similar directions in semantic segmentation [LSD15]. Although our goal is also segmentation, our need to distinguish between distinct instances rather than merely labeling pixels leads us to a network design that departs significantly from those used for pixel labeling.

4.2.3 Proposal Generation

A recently popular strategy for object segmentation and related problems divides the task into two stages: first, generating a large but managable set of candidate regions which attempt to cover all objects of interest; and second, processing those regions with more powerful machinery (e.g., a classification network) to finish the task. Recent examples of region proposal systems include MCG[APTB$^+$14], GOP [KK14], LPO [KK15], and the RPN used in Faster R-CNN [RHGS15], as well as DeepMask [PCD15] (discussed in more detail above). The region-generating networks we build in this chapter may be considered proposal systems, and we do evaluate our regions using proposal metrics. However, we are also interested in understanding the accuracy of region shapes independently from how they may be scored. In fact, systems like Faster R-CNN [RHGS15] and MNC [DHS15] erode the boundary between proposals and detection; since detection-quality scoring is no longer significantly more expensive than proposal-quality scoring, it makes sense to consider ways to evaluate region quality independently from scoring mechanisms, which can be adjusted separately. We'll explore this further in Section 4.4.5.

4.2.4 Simultaneous Detection and Segmentation

A number of systems have been proposed for the task of *simultaneous detection and segmentation*, which requires the production of *scored* object regions, which are evaluated according to detection metrics. Examples include SDS [HAGM14b], Hypercolumn SDS [HAGM15b], and

MNC [DHS15], as well as systems built by combining a proposal system with the scoring pipline of Fast R-CNN [Gir15]. MNC in particular has achieved remarkable results, winning the COCO 2015 detection challenge. While MNC includes a powerful region-generating pipeline, our networks take a fundamentally different and ultimately more powerful approach. This is because MNC (based on predecessors like SDS) relies on intermediate bounding boxes to find object regions. In general, bounding boxes are a degenerate representation of objects: distinct objects can have the same bounding box. (In practice, objects with sufficiently similar bounding boxes are also conflated, because bounding boxes are reduced through an NMS step.) Our networks have no such limitation.

In this chapter we focus on improvements to the quality of regions themselves rather than on orthogonal choices which improve scoring. We believe that our system could be usefully extended for SDS, a belief which is strongly encouraged by positive results for similar systems.

4.3 Building Region-Producing Networks

4.3.1 Single-Convolution Regions

A typical convnet produces layers of intermediate values as three-index arrays: two indices correspond to spatial locations in the input, and a third indexes across responses of different filters. For our purposes, it will be most convenient to think of each array of features as a vector-valued function $\mathbf{f} : G \subset \mathbb{R}^2 \to \mathbb{R}^d$ which associates the center point of each receptive field with a d-dimensional feature vector. The domain of each such function is a discrete, regular 2d grid G, so that each function is uniquely determined by a three-index array of values. We'll denote the vector of feature values at a particular location by $\mathbf{f}(\mathbf{x})$, where $\mathbf{x} = (x, y) \in G$ is a location on that grid.

We'd like a way to compute a four-index array of values that indicates how likely two points on the image are to belong to the same object. Let's denote such an array as a scalar function $p : G \times R \to \mathbb{R}$, which takes a 2d point on a regular grid of image locations (as above), as well as a 2d point on a grid of *relative* image coordinates. Thus we'd like $p(\mathbf{x}, \boldsymbol{\Delta})$ to indicate how likely it is that the points $\mathbf{x}$ and $\mathbf{x} + \boldsymbol{\Delta}$ belong to the same object.

The most straightforward way to produce such an array is by a convolution with *two* output indices, i.e., by computing

$$p(\mathbf{x}, \boldsymbol{\Delta}) = \sum_{\mathbf{x}'} \mathbf{w}(\mathbf{x}', \Delta) \cdot \mathbf{f}(\mathbf{x} + \mathbf{x}'),$$

where:

- p is our four-index output;
- $\mathbf{f}$ is a three-index convnet feature layer;
- and $\mathbf{w}$ is a five-index array of convolution weights (to be learned).

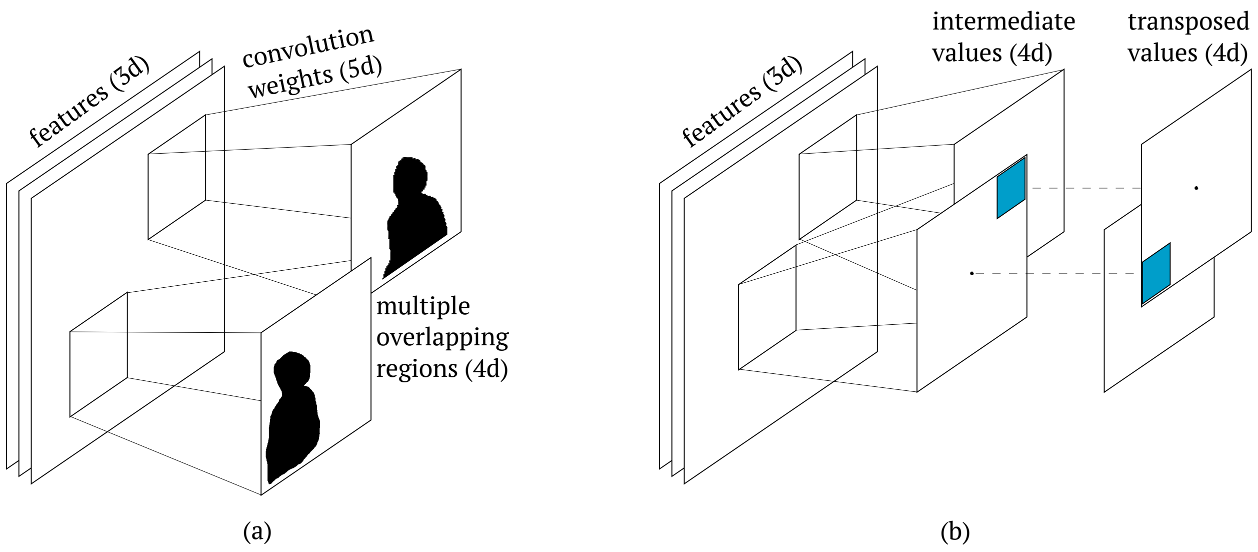


Figure 4.2: A simple architectural modification improves the performance of region generating networks at almost no cost. Subfigure (a) illustrates a direct approach to convolutional region generation, as in [PCD15]. Subfigure (b) illustrates our approach: after an initial convolution as in (a), the intermediate values undergo a type of transposition, followed by a matrix multiplication with few parameters and shared weights (the latter is not shown). Our approach increases receptive field size and improves region quality. Figure 4.3 illustrates the empirical and intuitive motivation for this structure.

This one-stage convolutional prediction, illustrated schematically in Figure 4.2(a), is the essential structure of the DeepMask region proposal network[1] [PCD15] (see the previous section for a detailed comparison to this work). With an appropriate loss, p can be interpreted (after thresholding) as a set of region masks, each centered around a point in G. (We'll call the points in G *region centers*, although they are really the centers of rectangular windows *containing* each region.)

4.3.2 Pairwise Intermediate Values

While a single convolution on powerful features can produce a high-quality set of overlapping regions, it feels underwhelming for that purpose. For one, this simple architecture forces the filter to cover all features relevant to an entire region, resulting in large filter sizes and an explosion of parameters.

Rather than viewing this single convolution as the final step, and its output as object regions, consider a more powerful perspective: view this convolution as an intermediate step which produces intermediate four-index values. This step transitions our network from

[1]Recent versions of DeepMask also include a dimensionality reduction step, which will not be relevant to the discussion here.

dealing in values which are each relevant to a *single* spatial location to dealing in values that encode information about *pairs* of spatial locations. Note that (in order to keep this simple and memory-efficient), we'll only consider *scalar* values of the latter type, while a deep net normally deals in *vector* values of the former type. (This is why we are upgrading from three-index arrays to four-index ones, even though an additional spatial coordinate adds two indices. The further generalization to vector values is straightforward.)

Pair-indexed intermediate values give rise to new ways of structuring computation. In particular, we can think of each such as array as a matrix representing a linear transformation from $\mathbb{R}^G$ to $\mathbb{R}^{G+R}$. With this view, we can perform matrix operations on these values. For example, suppose we have two four-index arrays, $p_1 : G \times R \to \mathbb{R}$ and $p_2 : G' \times R' \to \mathbb{R}$. If the "output" coordinates of p_1 line up with the "region center" coordinates of p_2, i.e., if $G + R = \{g + r \mid g \in G, r \in R\} = G'$, then it makes sense to combine p_1 and p_2 by matrix multiplication by computing

$$p_{12}(\mathbf{x}, \mathbf{\Delta}) = \sum_{\delta \in R \text{ s.t. } \mathbf{\Delta} - \delta \in R'} p_1(\mathbf{x}, \delta) \times p_2(\mathbf{x} + \delta, \mathbf{\Delta} - \delta).$$

Note that the sum and product operations on p-values, while denoted Σ and $\times$, need not be ordinary addition and multiplication, but can be other operations, affording a large space of (potentially nonlinear) inference procedures. By iterating computations of this type, we establish a sequence of intermediate predictions $p_1, \ldots, p_k$, which add a series of four-index layers to our network. Note that, while the whole network, viewed across the "region center" indices, is still translation-covariant, the breakdown into locally-connected computation is essential to continuing the inference, so that each region evolves based on the local content of the image.

Even more simply, given a single four-index array p, we can take a matrix transpose by computing

$$p^T(\mathbf{x}, \mathbf{\Delta}) = p(\mathbf{x} + \mathbf{\Delta}, -\mathbf{\Delta}).$$

(Notice that in absolute rather than relative coordinates, this amounts to switching the order of the arguments.) We'll see the usefulness of this operation in Section 4.3.3.

The form of network we've described encompasses a broad design space of inference procedures and hyperparameters. We make the following observations to outline the space:

- Each output value, defined at a point $(g, r) \in G_1 \times \left(\sum_{i=1}^{k} R_i\right)$, is a reduction over values computed on all paths from g to $g + r$ going through the connectivity structure of the network.

- If $p_i = p_j$ for all i and j (where both are defined), the computation is recurrent, and corresponds to a reduction over all paths of length k on a weighted directed graph.

- If $p_i(\mathbf{x}, 0) = 1$ (i.e., the identity for the $\times$ operation), as will typically be the case (since connectedness is reflexive), the reduction is over all paths of length at most k.

- If p_i is symmetric in the sense that $p_i(\mathbf{x}, \mathbf{\Delta}) = p_i(\mathbf{x} + \mathbf{\Delta}, -\mathbf{\Delta})$, the graph becomes undirected.
- If the p_i are distinct, the cost becomes "time-dependent" as the path progresses.
- If the operations used are $(\min, +)$, the computation finds the shortest path ($p_i \geq 0$ should hold everywhere in this case). (This is akin to GOP [KK14], where object regions are determined by geodesics.)
- If R_j is spaced more finely than R_i for $j > i$, the computation performs a coarse-to-fine region refinement.

4.3.3 Restructuring Region Prediction

The possibilties outlined in the previous section afford a great deal of flexibility in the design space of region-producing networks (or any networks that need to explicitly consider pairs of spatial locations). In this section, we'll find that a direct, visual examination of learned weights immediately suggests a particular type of restructuring involving the special transpose operation just defined.

Figure 4.3 visualizes a set of weights learned for region prediction by a single-stage region prediction network, as described in Section 4.3.1 and shown in Figure 4.2(a). Recall that the weights comprise a five-index array. For this visualization, we compute the L1 norm over the channel index. We show explicitly the resulting four-index array of norms as a 2d array of 2d arrays; the outer indices move across the dimensions of the output region, while the inner indices move across the dimensions of the input features. In other words, each small square illustrates the kernel which produces a particular output pixel (in terms of L1 norms).

This visualization strongly suggests both that this structure is inefficient (as a large portion of the weights appear to be insignificant) and that the learned weights are redundant (as the weights used by adjacent predictions appear to have similar structure). Staring at this figure gives rise to the following idea: rather than directly predicting whether two points belong to the same region, perhaps we should predict how much *more likely* a point is to belong to a given region *than neighboring points.* In other words, rather than learning the weights illustrated in Figure 4.3(a), perhaps we should learn the *differences* between adjacent blocks of weights. Computing these differences on the existing learned weights (in the y direction, for example), as shown in Figure 4.3(b), encourages this idea. Notice that the significant (i.e., bright) values in each block form a tight cluster (and, indeed, *tighter* clusters than above), and that each cluster is shifted *according to the location of the block.* In fact, the significant values lie directly over the *corresponding* part of the image.

Thus: rather than learning the weights in Figure 4.3(a), we'd prefer to rearrange the computation to learn the weights in Figure 4.3(b). In doing so, we expect to be able to use a smaller kernel size (or, alternatively, to keep the same kernel size but obtain higher-quality regions).

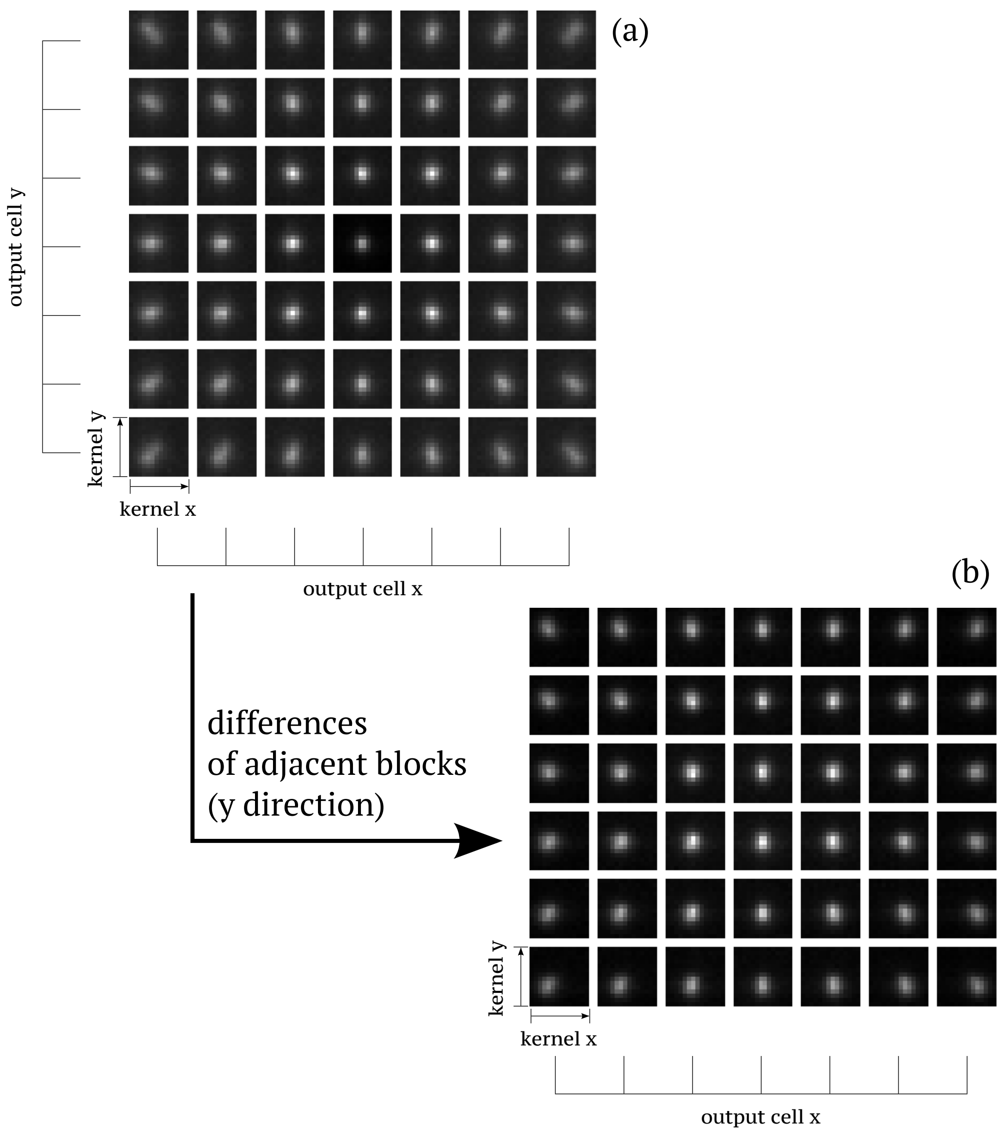


Figure 4.3: Visualizing the weights learned in a single stage of naïve convolutional region prediction reveals both inefficiency and redundancy. In the 4d illustration (a), each block corresponds to a particular output cell, and each pixel within a block corresponds to a particular input feature vector. The brightness of each point indicates the L1 norm of weights which connect a particular output cell (the "outer" dimensions) to input features (the "inner" dimensions). Notice that most of the weights are insignificant (i.e., dark) (indicating inefficiency), and that adjacent blocks appear nearly identical (indicating redundancy). In (b), examining the *differences* between adjacent blocks (in the y direction) of the weights illustrated in (a) points the way to a new computation which mitigates the redundancy of the naïve one. Notice that the weights in each block-of-differences are contained in a small region centered on the features under the *corresponding* point on the image, suggesting that these differences could be directly predicted by those underlying features.

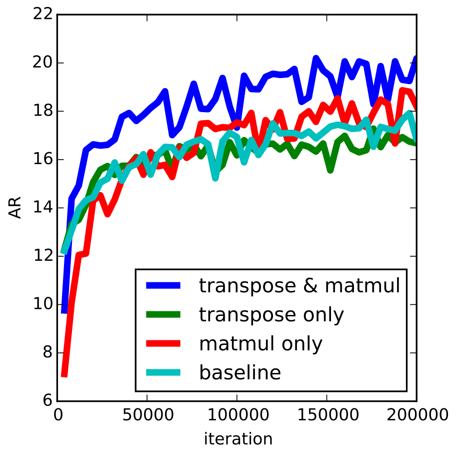


Figure 4.4: Performing region prediction under the special transposition described in the text, together with an extra matrix multiplication step, improves region quality. Either of these operations alone is less effective, in accord with the analysis in Section 4.3.3.

Now, actually computing differences would require a cumbersome step of defining exactly which pairs we want to predict differences between (which, depending on our choice, might overspecify the final output). So rather than literally computing differences, we'll use this intuition to restructure the computation in a more general way, according to the following pattern. First, we'll perform a single convolution to obtain a four-index array of intermediate values. Then, we'll perform the transposition operation defined at the end of Section 4.3.2. (Equivalently, we can think of the weights as connecting each output cell to a window of features directly under that cell, rather than, as before, connecting each output cell to a window of features under the *center* of the predicted region. This is what enables us to exploit the structure visible in Figure 4.3(b).) In addition, note that this operation increases the total receptive field size for each region. Furthermore, this increase comes nearly "for free", since the transposition above adds no additional parameters and is amenable to a highly efficient implementation (or can even be combined with previous or subsequent operations).

Finally, we'll add a matrix multiplication as a final step on top of the transposed values. We'll separately multiply each region by the same matrix. (Note that if we did interpret the transposed values as differences, we might compute the final output using a least-squares solution, i.e., by multiplying each region by a *fixed* matrix. By learning the weights of this final step, we arrive at a more general *and* easier to specify procedure which leaves the exact meaning of the intermediate values open to interpretation.) Note that this final matrix multiplication is an especially inexpensive one, both because the weights are shared between regions, and because this step is performed after the feature dimension has been eliminated.

Figure 4.4 plots the results of an initial empirical test of this idea. We trained four networks for region prediction (for this test, via a single pathway), which were otherwise identical except for the two components of this strategy: each network either includes or omits the region transpose and the final matrix multiply. (See Section 4.4.2 for more experimental details.) The results accord with our intuition. The transposition alone has at most a small effect (recall that it increases receptive field size per region). The matrix multiplication alone has a larger effect, as it adds parameters and modifies learning. However, both together boost performance more significantly than either alone.

4.3.4 Exploiting the Feature Pyramid

Object regions exist in a broad spectrum of sizes. The blue upper boundary in Figure 4.6, for example, histograms the distribution of object sizes (in terms of square root of pixel area) in (a subset of) the popular COCO dataset. Objects range from just a few pixels to occupying entire images, with smaller objects being especially prevalent.

Finding all of these objects requires multiscale reasoning; it's not reasonable to perform an expensive computation on every pixel of a 600×600 region, and it's also not possible to find a several-pixel object from aggressively downsampled features. It's therefore important to employ both dense, inexpensive computation, necessary for noticing small regions, with coarser, more expensive computation, necessary for pooling semantic information across large regions. Fortunately, the progressive, layered structure of a deep net is well fit to this challenge.

To produce regions at many scales, we build multiple region generating pathways which split off from different layers of the net. Small regions come from shallow features; they are produced cheaply and densely. Large regions come from deep features; they are produced coarsely and with the full semantic power of a deep net. Figure 4.1 gives a schematic illustration of this approach, while Figure 4.5 shows regions generated by this approach.

Note that the entire computation is performed by a single network, trained jointly and end-to-end, which takes images at "natural" resolution—no resizing or cropping.

Contrast this with the common practice of running a network on a feature pyramid. While this can be a way to produce high-quality multiscale output, it's not an efficient means of doing so. Even though common networks include substantial *downsampling*, producing output at the finest scales requires *upsampling* the image, a process which adds no additional information. Using the natural feature pyramid instead of an image pyramid bypasses this apparent circular computation. While networks built on the natural feature pyramid are not exactly equivalent to running on an image pyramid, the results herein show that it is nevertheless a more efficient way to produce multiscale output. In addition, unlike the image pyramid, the natural feature pyramid provides *reuse* of computation—most of the computation in the small region pathway is fed directly into the remaining pathways.

In fact, we can quantify this comparison for a particular region generating net. Consider a typical image of size 640×480 (this is the mode size in the COCO dataset). For the net described in Section 4.4.1, computing the coarsest pathway alone takes 161 ms. Computing the same on a $2\times$ upsampled image takes 416 ms; on a $3\times$ upsampled image, it takes 922 ms. (Proceeding further requires working around memory limitations.) Our full feature pyramid computation, which produces regions at scales of 1×, 2×, 4×, 8×, and 16×, requires only 292 ms total, nearly half the time required to add even *the first* additional equivalent level from the image pyramid.

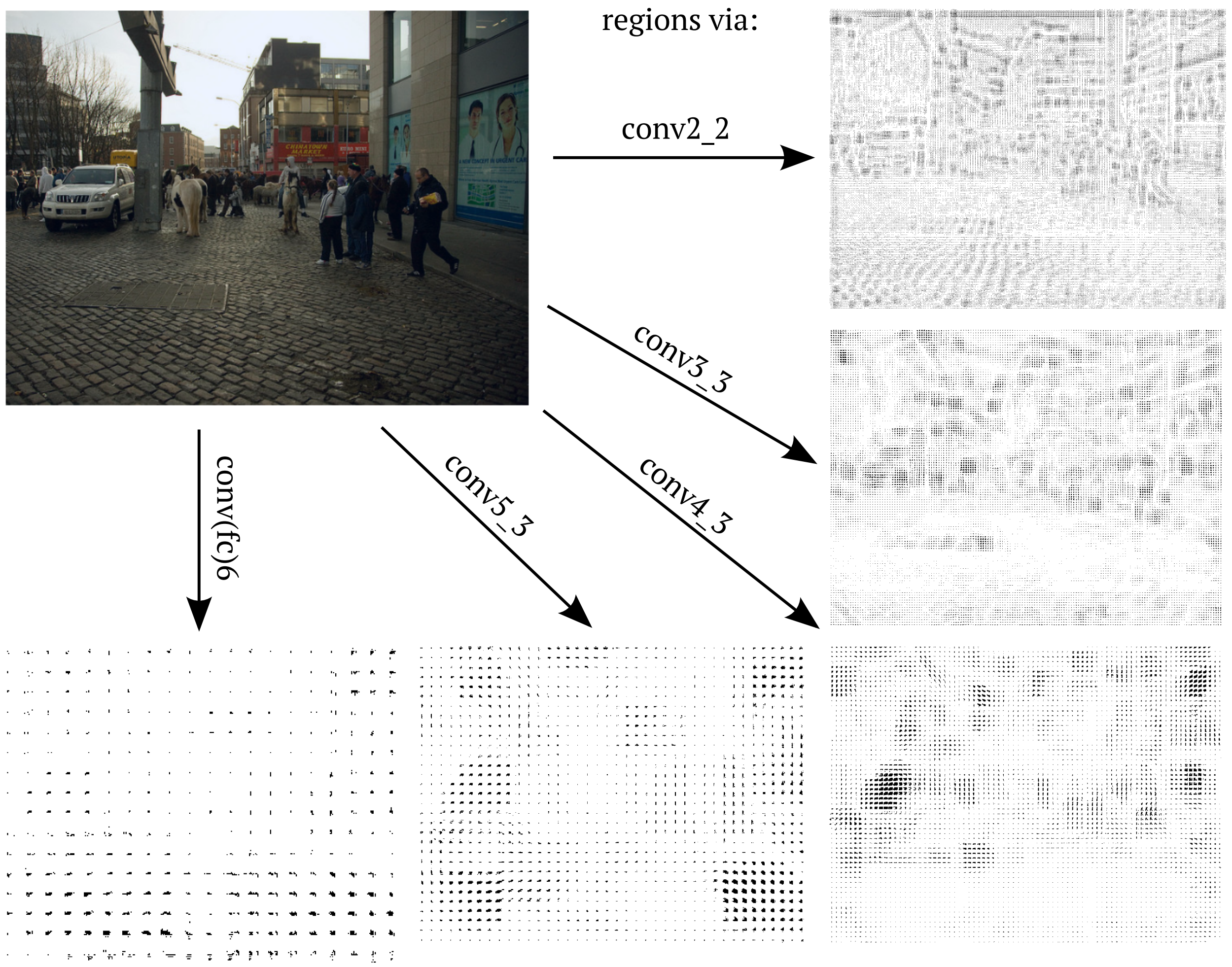


Figure 4.5: A network with several region pathways at different layers efficiently produces a pyramid of multiscale output. The image shown was run through the five-pathway network described in Section 4.4.1. Each output plane shows a set of regions covering the image at a particular scale. Each small dark blob indicates the shape of the predicted region, and regions are arranged according to their locations in the image. (Note that adjacent regions overlap.) Layer names refer to branches taken from layers in the VGG-16 network [SZ14b].

4.3.5 Non-max suppression

Our convolutional networks predict regions on a regular, highly overlapping grid of candidate windows. It's important that these windows overlap, as they must both be large enough to avoid cutting off regions, and be dense enough to avoid missing regions. However, this makes non-max suppression important to prune redundant regions. Note that our regions are especially amenable to NMS, as regions covering the same object will often be nearly identical. Furthermore, because regions are predicted on a series of regular grids, the set of regions which can overlap at all with a given region is known *a priori*. In particular, that means that NMS can be performed in *linear* rather than *quadratic* time (in the number of regions).[2] Finally, note that while our predicted regions are upsampled to the resolution of the input image, NMS can be performed before this upsampling step, making it even more efficient.

4.3.6 Loss

The loss is the final ingredient needed to train a region predictor using segment ground truth. We treat region prediction as a binary classification problem and use a binary logistic loss. The ground truth class is defined to be positive at a point (g, r) if g and r are contained by the same segment, and negative if g and r are contained in different segments. Following DeepMask, we do not include in the loss points for which g is not contained in any segment; this performed poorly in preliminary experiments. We evaluate the loss without downsampling the ground truth, instead upsampling coarsely-predicted regions (using bilinear interpolation, and backpropagating through this step).[3] Again, we found skipping this step hurt performance in preliminary experiments; upsampling predictions provides an averaging effect in the learning phase allowing more of the ground truth to be used, even when the predictions are made at a coarse scale.

4.4 Empirical Evaluation

The previous section describes a set of tools for improving region-producing networks. In this section, we put those tools together to build a single network, and study its behavior.

4.4.1 Network Assembly

The backbone of our network is the VGG-16 classification net [SZ14b], which achieves excellent performance on ILSVRC [ILS12] and has been shown to finetune well to other tasks. (Note, however, that any progressively subsampling network can be adapted into a region-producing network following the steps here.) VGG-16 downsamples by factors of two in five stages, going from features of stride one at the first convolutional layers to features of stride thirty-two at the final pooling layer. We split a region-predicting pathway off of each layer immediately before a pooling layer (except the first pooling layer), resulting in a five-level

[2]This is especially relevant for applications to *unbounded* images, such as satellite imagery.

[3]Since storing all regions at full resolution can be memory-intensive, we perform upsampling on-the-fly when computing the loss.

Table 4.1: Details of our region-generating network. Feature layers of VGG-16 [SZ14b] are split off into region-predicting pathways, each of which is described by a line below. Each pathway has its own stride, kernel size, output size (i.e., "window size" or maximum region size), and upsampling factor.

Feature Layer	Region Stride	Kernel Size	Window Size	Upsampling Factor
`conv2_2`	2px × 2px	5 × 5	28px × 28px	2×
`conv3_3`	4px × 4px	5 × 5	56px × 56px	4×
`conv4_3`	8px × 8px	9 × 9	144px × 144px	8×
`conv5_3`	16px × 16px	9 × 9	288px × 288px	16×
`conv(fc)6`	32px × 32px	9 × 9	576px × 576px	32×

pyramid of regions. Table 4.1 summarizes the detailed structure of our net. At the finest level, regions are predicted from the `conv2_2` layer. Region centers are located every two pixels, but regions are limited to 28×28 pixels in size (and are predicted at a 2× downsampling factor). At the coarsest level, regions are predicted from the `conv(fc)6`[4] layer. Region centers are spaced thirty-two pixels apart, can be up to 576×576 pixels in size, and are predicted at a 32× downsampling factor. Note that, while not compulsory, it's natural to use the same value for the stride between regions and the final upsampling factor. This simplifies the form of the transposition operation, and adapts the network structure to the region size; larger regions are searched for more coarsely, *and* their shapes are predicted more coarsely.

Following an initial convolution, each pathway includes a special transposition step, followed by a local matrix multiplication (with weights shared between all regions in a given pathway), just as dictated by the arguments and results of the previous section. Each pathway is trained using a binary logistic loss. The loss is computed on ground truth pixels which are not downsampled or modified in any way; instead, the predictions are upsampled as needed to match the resolution of the images. We include a region as ground truth for a given prediction only if the prediction center lies within the region, *and* the region fits entirely within the prediction window. This has two important effects: it inhibits prediction of regions that are artificially cut off at window boundaries, and it allows smaller pathways to focus on smaller objects, rather than learning to predict nearby boundaries of larger objects. Conversely, we do not take any special steps to prevent larger pathways from learning about smaller objects.[5]

[4]As in [LSD15], we treat fully connected layers as convolutions, including finetuning from classification weights.

[5]While larger pathways are less likely to encounter smaller objects, since their region centers are further apart, smaller pathways are very likely to encounter the borders of larger objects, which are especially long.

4.4.2 Experimental Setup

4.4.3 Data

We'll use COCO [LMB+14] as our dataset for training and evaluation. Training will use the COCO training set, including object segment labels for over 80,000 images and for 80 object categories. (Note that category labels are ignored in our training.) Validation will use a subset of the COCO val set.

4.4.4 Learning Details

All learning is performed by SGD with momentum. Each image is treated as a batch; gradients are computed one "whole" image at a time, averaging loss over all predicted regions, and parameters are updated for each image. No image resizing or patch extraction ever takes place. Rather than adjusting batch size, we adjust momentum. For all of our experiments, we find that a momentum of 0.99 works well. In contrast to common practice in SGD training, we do not pass through training examples in a preselected, shuffled order. Instead, we use a "true" stochastic selection of examples.[6] Our implementation runs in Caffe [JSD+14], but also includes custom CUDA code, as well as code written in Halide [RKBA+13][7].

4.4.5 Metrics

We'll analyze our method both in a scoring-independent way, as well as adding a simple scoring pathway which allows a comparison to existing methods using region proposal metrics.

We report the following standard measures of recall. These metrics are computed using the area of intersection divided by the area of union of each predicted region with each ground truth region, denoted I/U.

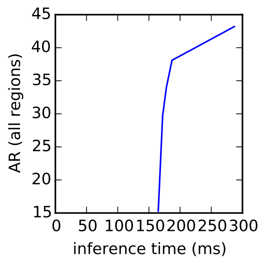
- R@x: recall at x, the portion of ground truth regions that match some predicted region at I/U $\geq x$.
- ABO: average best overlap, the average over all ground truth regions of the maximum I/U with a predicted region.
- AR: average recall, R@x averaged over the interval $[0.5, 1]$. (This metric was introduced and shown to correlate well with detection mAP by [HBDS15].)

[6]Note that our iteration time is sufficient for a simultaneous disk read, since we are performing much more processing per image than, e.g., single-label classification.

[7]which we find to be an effective tool for rapid but efficient prototyping of CUDA kernels, as well a convenient way to inline functions that would otherwise be memory-intensive, such as the upsampling necessary to evaluate loss at every output pixel

Table 4.2: Adjusting the density of region prediction trades off inference time and performance. The "feature computation" portion of the net has a fixed cost which is significant relative to the marginal cost of region generation. As a result, performance climbs dramatically as inference time is allowed to increase, until most regions have been found and diminishing returns are met.

stride at lowest layer	R@50	R@70	ABO	AR	Inf. Time
2	85.5	57.2	69.9	43.2	287ms
4	79.7	50.0	66.5	38.1	187ms
6	74.1	44.3	63.5	34.0	178ms
8	67.0	77.9	59.7	29.7	172ms
16	37.8	18.8	42.8	15.3	165ms



Region proposals are generally evaluated both in terms of region quality and scoring accuracy, typically by limiting the the number of regions per image to a fixed value. Since the methods under study here address region quality rather than scoring accuracy (which can be addressed in orthogonal ways outside the scope of this chapter), we'll begin our analysis by considering these metrics independently from scoring. The systematic nature of our method provides a natural way to analyze region quality while ignoring scoring but still taking into account the number of regions produced: we consider recall as a function of *region density*.

Table 4.2 provides a look at these measures of recall as a function of region density. We measure density in terms of the region stride at the finest (lowest) prediction layer. Each higher layer will always predict regions at twice the stride of the previous layer.[8] This analysis exhibits the tradeoff between performance and inference time for our method. While the base net is a significant fixed cost, performance rises rapidity with little marginal computation.

Although these metrics provide some interesting analysis, it's also important to situate our method with respect to existing ones. To this end, we also train a simple scoring pathway (jointly with region generation), and evaluate according to standard proposal metrics. As observed, these metrics depend on both region quality and scoring quality. However, we think that our scoring pathways are reasonably equivalent to those of DeepMask [PCD15], providing a fair comparison.

Even though our inference time is about 4× faster than DeepMask, we achieve similar AR, with a larger boost at fewer regions per image.

[8]Note that the total number of regions is therefore a geometric sum which is bounded by 4/3 of the number of regions at the smallest scale.

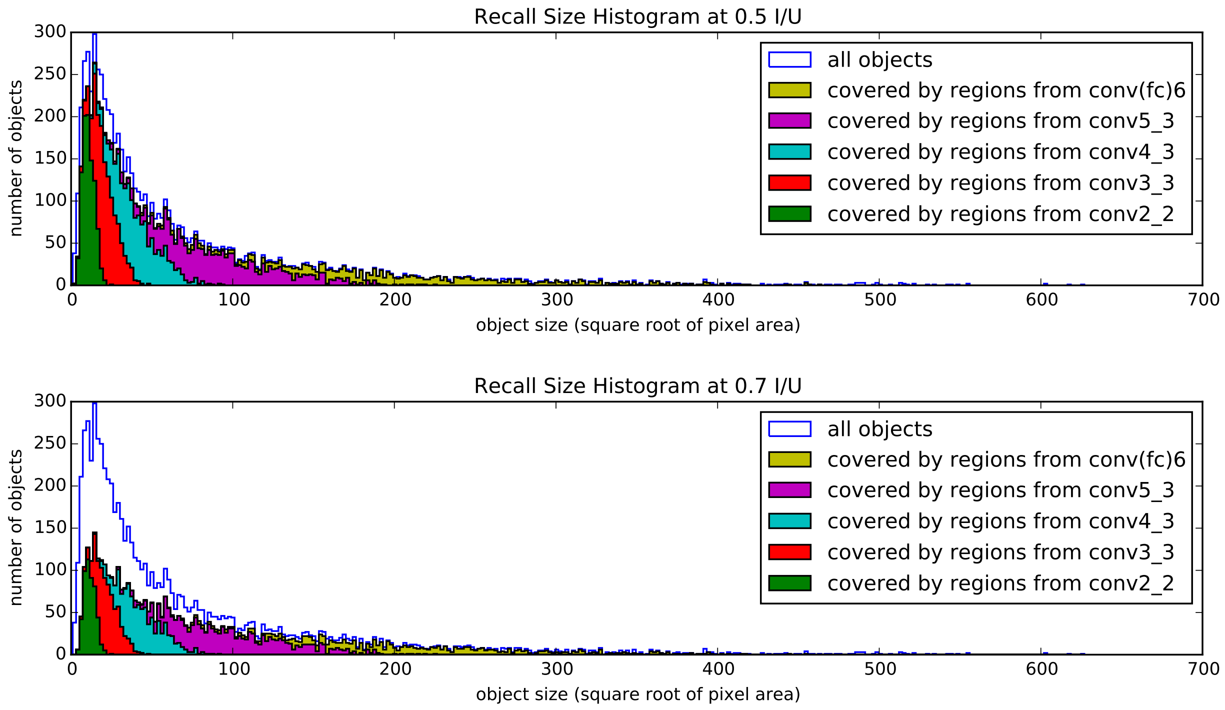


Figure 4.6: Using a series of prediction pathways from different layers is a natural, efficient way to generate regions at many scales, without needing an image pyramid. The histogram shows how a region net with five pathways (as described in the text) finds object regions at all scales. The blue outline histograms the distribution of ground truth regions in a subset of COCO val. Each ground truth region is identified with the pathway with the highest-overlapping predicted regions. The ground truth regions recalled at overlap at least 0.5 (above) or 0.7 (below) are shown as stacked histograms over ground truth region size, colored according to the associated pathway.

4.5 Output Visualization

Figures 4.7 and 4.8 visualize the regions produced by the method described in thiis Chapter.

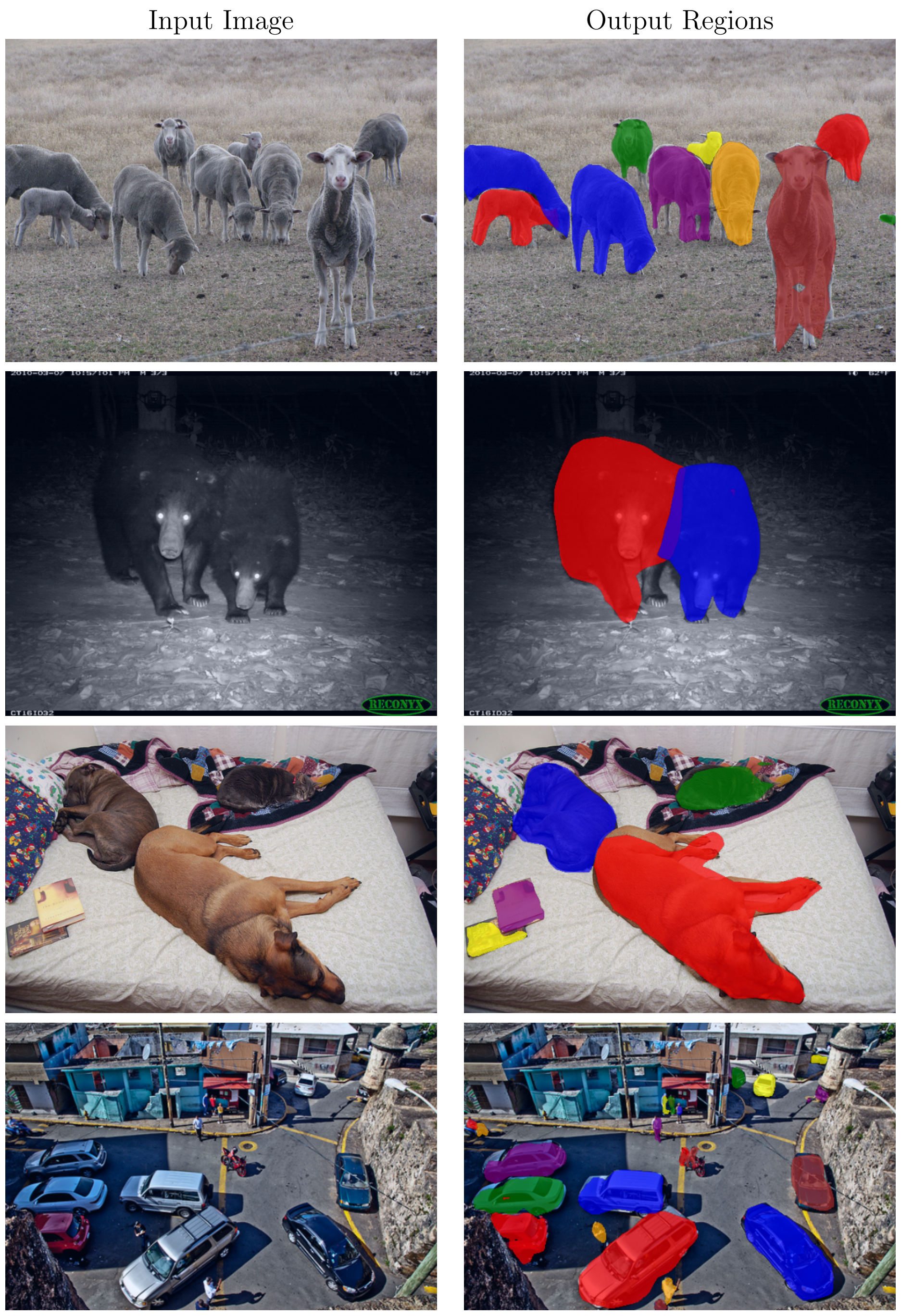


Figure 4.7: Output visualization. For each of these images from the validation set, we show the closest matching predicted region to each ground truth region (in terms of I/U). (Note that some objects are correctly predicted, but no region is shown because a label does not exist in the ground truth.) Note the ability to find occluded objects, to separate objects without sharp boundaries (second example), and to find small objects in cluttered scenes (without neglecting large objects).

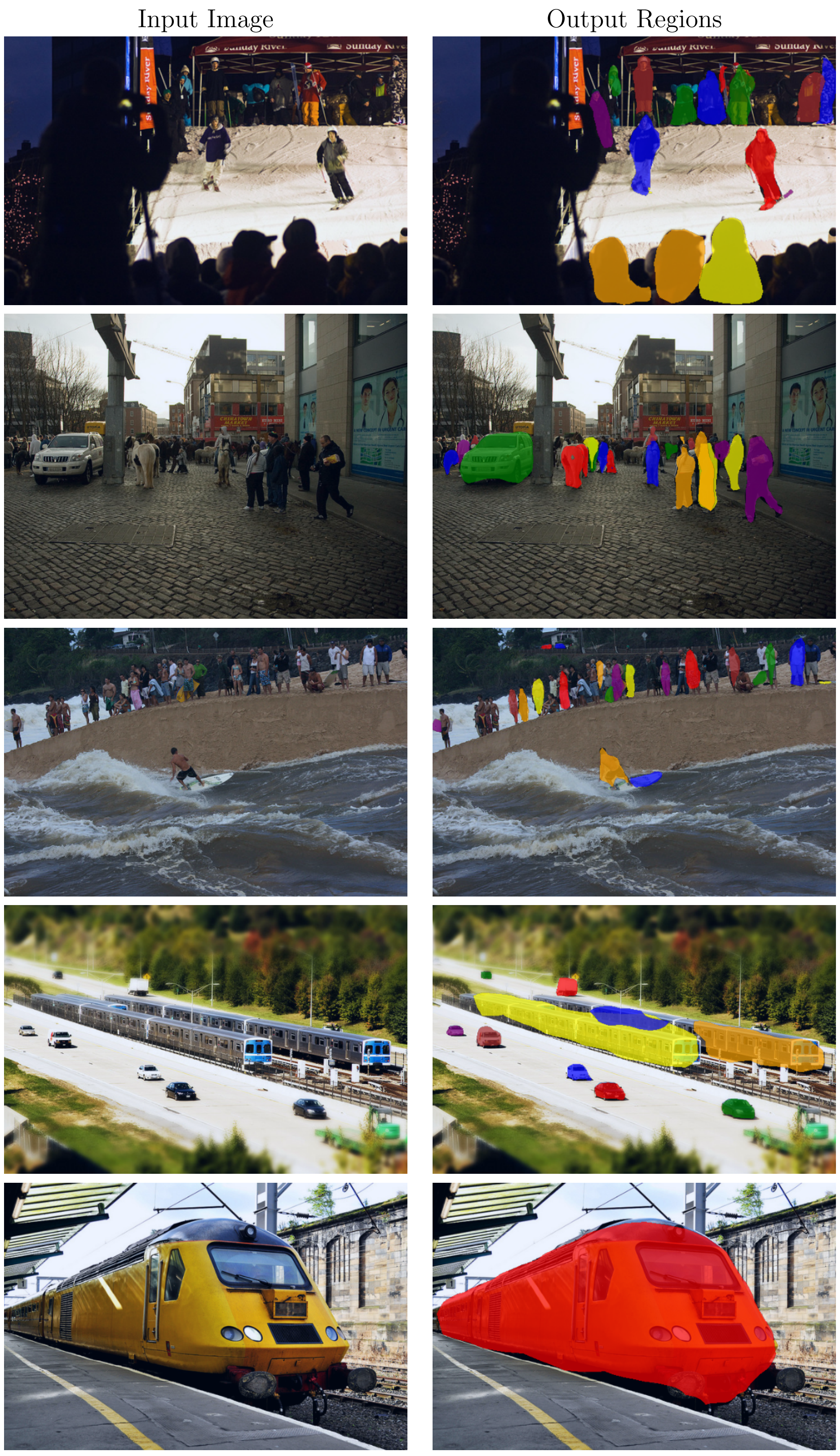


Figure 4.8: Additional output visualization. See Figure 4.7 for explanation.

Table 4.3: Training a simple scoring pathway jointly with region prediction produces competitive results on proposal metrics, with a state-of-the-art inference time. We compare methods according to average recall at a fixed number of regions per image, as described in the text. MCG [APTB$^+$14] is the best performing method pre-DeepMask. DeepMaskZoom gives the highest numbers reported in [PCD15]; the inference time is not reported, although it is noted to be larger than DeepMask.

Method	AR@10	AR@100	AR@1000	Inf. Time
Ours	16.0	25.5	32.8	281ms (Titan X) / 409ms (K40)
MCG [APTB$^+$14]	7.7	18.6	29.9	$\sim$30s (CPU)
DeepMask [PCD15]	12.6	24.5	33.1	1.6s (K40)
DeepMaskZoom [PCD15]	12.7	26.1	36.6	*

Chapter 5

Conclusion

We've seen that state-of-the art convolutional networks, trained only for whole-image classification, naturally contain features that carry semantic information locally, about just a few surrounding pixels. Guided by this positive result, we've built networks that *predict* labels for individiual pixels, learning end-to-end from per-pixel supervision. Finally, we've examined one way to produce richer information than just per-pixel labels, by upgrading one spatial index to two, gaining the ability to understand object regions—the sense in which pixels belong to other each, rather than just to semantic classes.

Along the way, we've built and analyzed systems for a number of tasks, including intraclass object alignment, keypoint detection, semantic segmentation, and region generation. While system-building work in computer vision has commonly relied on ad-hoc combinations of hand-engineered features and machine learning techniques (and, more recently, modern convnet features), it's been our goal to build *exclusively* with a small set of well-studied and highly optimized parts, or, in other words, to build "pure" convnet systems. Exactly which parts are considered to be in that set is a matter of interpretation, empiricism, and taste, but the general principles are fixed, namely:

- Learning should be *joint* and *end-to-end* wherever that makes sense; otherwise, performance is left on the table.

- Components should be *simple* to compute, with *simple derivatives*. Simple operations lead to efficient implementations, which leads to larger training data sets, and shorter iteration time. Simple derivatives are likewise essential to perform gradient-based learning. Using only differentiable operations factors out model building from choice of learning method (although these still interact in significant ways).

- Components should be *flexible* enough, with enough adjustable parameters, to support a wide range of architectures and tasks (but not so flexible that finding effective values of those parameters becomes an impossible task).

- Components should be *reusable*, in terms of both their archictectural parameters and their *learned weights*. This reusability enables transfer learning (in this context, also called finetuning, though note that "fine" need not reflect the magnitude of the effect!). Building a new system does not mean designing from scratch, nor learning from scratch, but rather taking a machine that already works well and molding it to the task at hand, both with new parts and new data.

By viewing convnets not simply as new features for old recognition tasks, but rather as (re)inspiring certain design principles, we make progress towards a vision of the future where building perceptual systems is as easy as playing with legos.

5.1 Epilogue: Future Directions

Of course, there is still much work to be done to achieve that vision. We'll point out three directions in particular that stand out as natural places to go from here, all of which are active research areas.

- Making convnet learning *robust*. Currently there is a great deal of manual effort involved in searching for effective hyperparameters (which have very little relevance to the task at hand). It's easy to imagine that with the right training methodology, this manual tweaking could be circumvented.
- Finding new methods (of the type described above) to produce new kinds of rich output. While this and other works have taken first steps in this direction, there is still a large gap between the types of information that can be accurately assessed in a fraction of a second by a convnet, and the types of information likewise accessible to an animal.
- Finding new methods for unsupervised, weakly supervised, or self-supervised learning. All of the systems explored here rely on ground truth labels as defined by the task, provided by human annotation. For many tasks, especially the dense-labeling tasks studied here, those annotations can be quite expensive, which limits available training data and ultimately performance. While we already benefit by transferring from the more cheaply-annotated classification task, performance is still highly dependent on the training set size for the final task. As we desire new types of output and high levels of performance even in the "long tail" where training data is necessarily sparse, it's important to find new ways to expand the power of a few labels.

Bibliography

[Ama98] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

[APTB$^+$14] P. Arbeláez, J. Pont-Tuset, J. Barron, F. Marques, and J. Malik. Multiscale combinatorial grouping. In *Computer Vision and Pattern Recognition*, 2014.

[BA83] P. Burt and E. Adelson. The laplacian pyramid as a compact image code. *Communications, IEEE Transactions on*, 31(4):532–540, 1983.

[BB13] T. Berg and P. N. Belhumeur. POOF: Part-based one-vs.-one features for fine-grained categorization, face verification, and attribute estimation. In *CVPR*, 2013.

[Bis06] Christopher M Bishop. *Pattern recognition and machine learning*, page 229. Springer-Verlag New York, 2006.

[BJKK11] P. N. Belhumeur, D. W. Jacobs, D. J. Kriegman, and N. Kumar. Localizing parts of faces using a consensus of exemplars. In *CVPR*, 2011.

[BM09] L. Bourdev and J. Malik. Poselets: Body part detectors trained using 3d human pose annotations. In *ICCV*, 2009.

[CCBS12] J. Carreira, R. Caseiro, J. Batista, and C. Sminchisescu. Semantic segmentation with second-order pooling. In *ECCV*, 2012.

[CGGS12] Dan C Ciresan, Alessandro Giusti, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In *NIPS*, pages 2852–2860, 2012.

[com] PASCAL VOC Segmentation Results: comp6 VOC2012. http://host.robots.ox.ac.uk:8080/leaderboard/displaylb.php?challengeid=11&compid=6. Accessed: 2016-5-9.

[CPK$^+$15] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected CRFs. In *ICLR*, 2015.

[DDS$^+$09] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009.

[DHS14] Jifeng Dai, Kaiming He, and Jian Sun. Convolutional feature masking for joint object and stuff segmentation. *arXiv preprint arXiv:1412.1283*, 2014.

[DHS15] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. *arXiv preprint arXiv:1512.04412*, 2015.

[DJV$^+$14] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. DeCAF: A deep convolutional activation feature for generic visual recognition. In *ICML*, 2014.

[EKF13] David Eigen, Dilip Krishnan, and Rob Fergus. Restoring an image taken through a window covered with dirt or rain. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 633–640. IEEE, 2013.

[EPF14] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. *arXiv preprint arXiv:1406.2283*, 2014.

[EVGW$^+$] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2011 (VOC2011) Results. http://www.pascal-network.org/challenges/VOC/voc2011/workshop/index.html.

[FCN]

[FCNL13] Clément Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 2013.

[FDB14a] P. Fischer, A. Dosovitskiy, and T. Brox. Descriptor Matching with Convolutional Neural Networks: a Comparison to SIFT. *ArXiv e-prints*, May 2014.

[FDB14b] Philipp Fischer, Alexey Dosovitskiy, and Thomas Brox. Descriptor matching with convolutional neural networks: a comparison to SIFT. *CoRR*, abs/1405.5769, 2014.

[FH04] Pedro Felzenszwalb and Daniel Huttenlocher. Distance transforms of sampled functions. Technical report, Cornell University, 2004.

[FH06] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient belief propagation for early vision. *International journal of computer vision*, 70(1):41–54, 2006.

[FMJZ08] Vittorio Ferrari, Manuel Marín-Jimenéz, and Andrew Zisserman. Progressive search space reduction for human pose estimation. In *CVPR*, 2008.

[Fuk80] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[GABM13] G. Gkioxari, P. Arbelaez, L. Bourdev, and J. Malik. Articulated pose estimation using discriminative armlet classifiers. In *CVPR*, 2013.

[GAM13] Saurabh Gupta, Pablo Arbelaez, and Jitendra Malik. Perceptual organization and recognition of indoor scenes from RGB-D images. In *CVPR*, 2013.

[GCM$^+$13] Alessandro Giusti, Dan C Cireşan, Jonathan Masci, Luca M Gambardella, and Jürgen Schmidhuber. Fast image scanning with deep max-pooling convolutional neural networks. In *ICIP*, 2013.

[GDDM14a] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition*, 2014.

[GDDM14b] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. 2014.

[GGAM14] Saurabh Gupta, Ross Girshick, Pablo Arbelaez, and Jitendra Malik. Learning rich features from RGB-D images for object detection and segmentation. In *ECCV*. Springer, 2014.

[GHGM14] G. Gkioxari, B. Hariharan, R. Girshick, and J. Malik. Using k-poselets for detecting people and localizing their keypoints. In *CVPR*, 2014.

[Gir15] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015.

[GL14] Yaroslav Ganin and Victor Lempitsky. N^4-fields: Neural network nearest neighbor fields for image transforms. In *ACCV*, 2014.

[HAB$^+$11] Bharath Hariharan, Pablo Arbelaez, Lubomir Bourdev, Subhransu Maji, and Jitendra Malik. Semantic contours from inverse detectors. In *International Conference on Computer Vision (ICCV)*, 2011.

[HAGM14a] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Simultaneous detection and segmentation. In *European Conference on Computer Vision (ECCV)*, 2014.

[HAGM14b] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Simultaneous detection and segmentation. In *Computer vision–ECCV 2014*, pages 297–312. Springer, 2014.

[HAGM15a] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Hypercolumns for object segmentation and fine-grained localization. In *Computer Vision and Pattern Recognition*, 2015.

[HAGM15b] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Hypercolumns for object segmentation and fine-grained localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 447–456, 2015.

[HBDS15] Jan Hosang, Rodrigo Benenson, Piotr Dollár, and Bernt Schiele. What makes for effective detection proposals? 2015.

[HJLM07] G. B. Huang, V. Jain, and E. Learned-Miller. Unsupervised joint alignment of complex images. In *ICCV*, 2007.

[HKMMT89] Matthias Holschneider, Richard Kronland-Martinet, Jean Morlet, and Ph Tchamitchian. A real-time algorithm for signal analysis with the help of the wavelet transform. pages 286–297, 1989.

[HMLLM12] G. B. Huang, M. A. Mattar, H. Lee, and E. Learned-Miller. Learning to align from scratch. In *NIPS*, 2012.

[HZRS14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*, 2014.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[ILS12] ILSVRC. ImageNet Large-scale Visual Recognition Challenge, 2010-2012.

[JOP$^+$] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

[JSD$^+$14] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[JTA$^+$14] Arjun Jain, Jonathan Tompson, Mykhaylo Andriluka, Graham W. Taylor, and Ghristoph Bregler. Learning human pose estimation features with convolutional networks. In *ICLR*, 2014.

[KK14] Philipp Krähenbühl and Vladlen Koltun. Geodesic object proposals. In *Computer Vision–ECCV 2014*, pages 725–739. Springer, 2014.

[KK15] Philipp Krahenbuhl and Vladlen Koltun. Learning to propose objects. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 1574–1582. IEEE, 2015.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[KvD87] Jan J Koenderink and Andrea J van Doorn. Representation of local geometry in the visual system. *Biological cybernetics*, 55(6):367–375, 1987.

[LB13] J. Liu and P. N. Belhumeur. Bird part localization using exemplar-based models with enforced pose and subcategory consistenty. In *ICCV*, 2013.

[LBBH98] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[LBD+89] Yang LeCun, B. Boser, J.S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to hand-written zip code recognition. In *Neural Computation*, 1989.

[LBOM98] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 1998.

[lio] Lions vs antelope - BBC wildlife. https://www.youtube.com/watch?v=Qr7v-Kf4064. Accessed: 2016-4-17.

[LMB+14] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[Low99] D.G. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150–1157 vol.2, 1999.

[LRB15] Wei Liu, Andrew Rabinovich, and Alexander C Berg. Parsenet: Looking wider to see better. *arXiv preprint arXiv:1506.04579*, 2015.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[LYT11] Ce Liu, Jenny Yuen, and Antonio Torralba. Sift flow: Dense correspondence across scenes and its applications. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(5):978–994, 2011.

[LZD14] Jonathan Long, Ning Zhang, and Trevor Darrell. Do convnets learn correspondence? In *NIPS*, 2014.

[Mal99] Stéphane Mallat. *A wavelet tour of signal processing.* Academic press, 2nd edition, 1999.

[MBLD91] Ofer Matan, Christopher JC Burges, Yann LeCun, and John S Denker. Multi-digit recognition using a space displacement neural network. In *NIPS*, pages 488–495. Citeseer, 1991.

[MCL+14] Roozbeh Mottaghi, Xianjie Chen, Xiaobai Liu, Nam-Gyu Cho, Seong-Whan Lee, Sanja Fidler, Raquel Urtasun, and Alan Yuille. The role of context for object detection and semantic segmentation in the wild. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 891–898. IEEE, 2014.

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

[MYS15] Mohammadreza Mostajabi, Payman Yadollahpour, and Gregory Shakhnarovich. Feedforward semantic segmentation with zoom-out features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3376–3385, 2015.

[NDL+05] Feng Ning, Damien Delhomme, Yann LeCun, Fabio Piano, Léon Bottou, and Paolo Emilio Barbano. Toward automatic phenotyping of developing embryos from videos. *Image Processing, IEEE Transactions on*, 14(9):1360–1371, 2005.

[NHH15] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *ICCV*, 2015.

[PB13] Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*, 2013.

[PC14] Pedro HO Pinheiro and Ronan Collobert. Recurrent convolutional neural networks for scene labeling. In *ICML*, 2014.

[PCD15] Pedro O. Pinheiro, Ronan Collobert, and Piotr Dollár. Learning to segment object candidates. *CoRR*, abs/1506.06204, 2015.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

[RHGS15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015.

[RKBA+13] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *ACM SIGPLAN Notices*, 48(6):519–530, 2013.

[Sch15] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[SEZ$^+$14] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *ICLR*, 2014.

[SHKF12] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[SKCL13] Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala, and Yann LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In *CVPR*, 2013.

[SLJ$^+$14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[SS11] M. Sun and S. Savarese. Articulated part-based model for joint object detection and pose estimation. In *ICCV*, 2011.

[SZ14a] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[SZ14b] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[SZS$^+$13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.

[TJLB14] Jonathan Tompson, Arjun Jain, Yann LeCun, and Christoph Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. *CoRR*, abs/1406.2984, 2014.

[TL10] Joseph Tighe and Svetlana Lazebnik. Superparsing: scalable nonparametric image parsing with superpixels. In *ECCV*, pages 352–365. Springer, 2010.

[TL13] Joseph Tighe and Svetlana Lazebnik. Finding things: Image parsing with regions and per-exemplar detectors. In *CVPR*, 2013.

[TS14] A. Toshev and C. Szegedy. DeepPose: Human pose estimation via deep neural networks. In *CVPR*, 2014.

[UvdSGS13] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. *IJCV*, 2013.

[Vai90] Parishwad P Vaidyanathan. Multirate digital filters, filter banks, polyphase networks, and applications: A tutorial. *Proceedings of the IEEE*, 78(1):56–93, 1990.

[VF08] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. http://www.vlfeat.org/, 2008.

[VKMT13] C. Vondrick, A. Khosla, T. Malisiewicz, and A. Torralba. HOGgles: Visualizing Object Detection Features. *ICCV*, 2013.

[WP94] Ralph Wolf and John C Platt. Postal address block location using a convolutional locator network. *Advances in Neural Information Processing Systems*, pages 745–745, 1994.

[WZZ$^+$13] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066, 2013.

[Yan] Debate on Yann LeCun's Google+ page. https://plus.google.com/+YannLeCunPhD/posts/JBBFfv2XgWM. Accessed: 2014-5-31.

[YK16] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016.

[YR11] Y. Yang and D. Ramanan. Articulated pose estimation using flexible mixtures of parts. In *CVPR*, 2011.

[ZDGD14] Ning Zhang, Jeff Donahue, Ross Girshick, and Trevor Darrell. Part-based r-cnns for fine-grained category detection. In *Computer Vision–ECCV 2014*, pages 834–849. Springer, 2014.

[ZF14] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014*, pages 818–833. Springer, 2014.

[ZJRP$^+$15] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip Torr. Conditional random fields as recurrent neural networks. In *ICCV*, 2015.

[ZR12] X. Zhu and D. Ramanan. Face detection, pose estimation, and landmark localization in the wild. In *CVPR*, 2012.