Predicting Images using Convolutional Networks: Visual Scene Understanding with Pixel Maps

by

David Eigen

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

May, 2015

Rob Fergus

Acknowledgements

I would like to thank my advisor Rob Fergus, whose optimistic enthusiasm and sense of inquiry helped guide and push me to completing the different works in this thesis, and a couple others as well. He was always very available and I had many valuable discussions with him, sometimes at odd hours. I learned a lot about conducting research through these, and many fruitful ideas grew from them, too.

I would also like to thank all of my collaborators and coauthors, and the many students and postdocs in the lab, especially: Ross Goroshin, Dilip Krishnan, Pierre Sermanet, Christian Puhrsch, Li Wan, Nathan Silberman, Jason Rolfe, Matthew Zeiler, Jonathan Tompson and Arthur Szlam. I learned much from our work together and numerous inspiring conversations.

Thanks also to Yann LeCun and David Sontag for cultivating the lab on the 12th floor, and the stimulating discussions and group meetings. I also thank Marc'Aurelio Ranzato for hosting me at Google during my summer internship, and Ronan Collobert for being on my dissertation committee.

Finally and most essentially, I am grateful for all the support from my parents, family and friends in both New York and California.

Abstract

In the greater part of this thesis, we develop a set of convolutional networks that infer predictions at *each pixel* of an input image. This is a common problem that arises in many computer vision applications: For example, predicting a semantic label at each pixel describes not only the image content, but also fine-grained locations and segmentations; at the same time, finding depth or surface normals provide 3D geometric relations between points. The second part of this thesis investigates convolutional models also in the contexts of classification and unsupervised learning.

To address our main objective, we develop a versatile Multi-Scale Convolutional Network that can be applied to diverse vision problems using simple adaptations, and apply it to predict depth at each pixel, surface normals and semantic labels. Our model uses a series of convolutional network stacks applied at progressively finer scales. The first uses the entire image field of view to predict a spatially coarse set of feature maps based on global relations; subsequent scales correct and refine the output, yielding a high resolution prediction. We look exclusively at depth prediction first, then generalize our method to multiple tasks. Our system achieves state-of-the-art results on all tasks we investigate, and can match many image details without the need for superpixelation.

Leading to our multi-scale network, we also design a purely local convolutional network to remove dirt and raindrops present on a window surface, which learns to identify and inpaint compact corruptions. We also we investigate a weighted nearest-neighbors labeling system applied to superpixels, in which we learn weights for each example, and use local context to find rare class instances.

In addition, we investigate the relative importance of sizing parameters using a recursive convolutional network, finding that network depth is most critical. We also develop a Convolutional LISTA Autoencoder, which learns features similar to stacked sparse coding at a fraction of the cost, combine it with a local entropy objective, and describe a convolutional adaptation of ZCA whitening.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer vision systems may infer numerous types of estimates in order to relate an input image to the scene it captures, to the world in which the scene is a part, and to our own understanding of the world. For example, object recognition systems infer objects present in a scene by predicting class labels (*e.g.* "bed", "picture"), often along with the objects' locations, *e.g.* with a bitmask or bounding box. Such a detection system is depicted in Fig. 1.1. These systems, however, scratch only the surface of possible representations: beyond labels and bounding boxes, many other useful types of understandings can be inferred as well. For example, geometric estimates such as world-space locations, depth maps or surface normals; per-pixel object labels that provide more detailed location information; object attributes that provide more fine-grained semantic descriptions; affordances relating objects to their potential human interactions; even decompositions of the image into those portions depicting the underlying scene and those introduced by artifacts of the image capture and storage.

While each of these problems might be tackled in different ways and have numerous choices for output representation, this thesis focuses on a prevalent theme of inferring 2D *pixel maps* from a single input image. Pixel map prediction arises naturally for many problems, several of which are depicted in Fig. 1.2. Finding semantic class labels for each

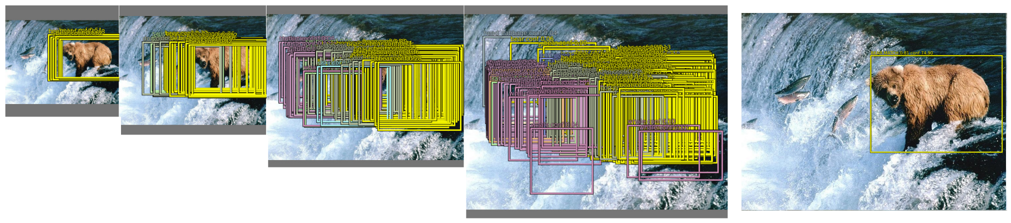


Figure 1.1: Object detection using bounding boxes. A ConvNet makes predictions at multiple locations and scales, which are then merged. Figure reproduced from [113], to which the author contributed.

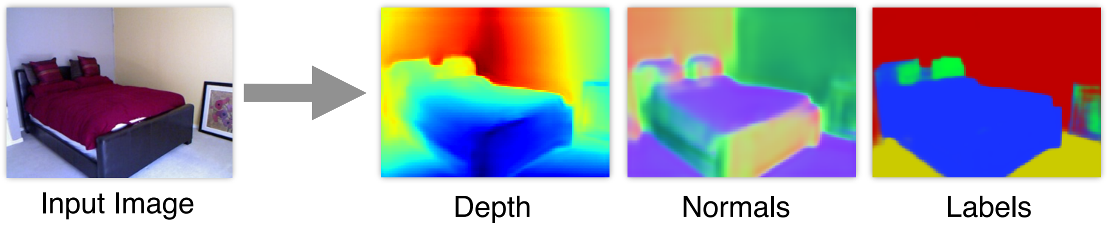


Figure 1.2: Inferring pixel maps for depth, surface normals and semantic labels using a convolutional network. Predictions made by the system described in Chapter 6.

pixel provides information both on which objects are in the scene ("what" is present) and their image location and extent ("where" they are). Estimating the depth from camera at each pixel provides a more 3D geometric understanding of the scene, as does per-pixel estimation of surface normals; these may be useful for physical applications, including robotics and 3D modeling. Likewise, denoising an image extracts relevant low-level structures away from corrupted input, producing a clean version of the image as another type of pixel map.

The following chapters explore systems for each of these tasks in turn, and focus in particular on convolutional networks that are able to learn their own internal feature representations at successive layers. Such systems require little or no hand-tuning of low- or mid-level feature descriptors, allowing intermediate representations to be learned from the data directly. These systems have recently achieved large performance gains, most notably for current object classification and detection tasks, and this thesis further extends their application to produce 2D outputs that align each pixel in the input with a set of values describing the scene content.

We start out by investigating a system to perform pixelwise semantic labeling in Chapter 3, using superpixels and hand-crafted features. By learning weights for each feature descriptor in a k-nearest-neighbor classification scheme, we see the benefit of automatically tuning the relative importance for each feature and database point (although not yet the features themselves). This anticipates our later work in multilayer networks by extending training further upstream in the dataflow. In addition, we create descriptors for mid-level context information that we use to augment the number of rare class instances available at classification time. We also evaluate a rough use of global context by varying the number of best-matching database images used for k-NN queries for each image; the latter has a loose relationship to our later use of global context in multi-scale convolutional networks.

We then apply Convolutional Networks to predict pixel maps. In Chapter 4, we employ a ConvNet to remove dirt or raindrops present on a window surface in front of the camera, thus restoring the underlying scene. This network is trained end-to-end to produce clean output images, and is able to estimate low-level natural image structures using purely local fields of view. Here we also find a benefit to *training convolutionally*, *i.e.* through the final combination of strided local predictions using an averaging step, and analyze the effect of this with an illustrative example.

Chapter 5 then combines ConvNets at both global and local levels to infer depth maps. In contrast to the denoising application, we find the global view is essential for depth prediction, and show how we integrate this view along with the original image at a local scale to produce detailed pixel-map outputs. The resulting system is simple, using a sequence of two networks that first produces a coarse estimate of the depth, then refines it to details in the original image; we obtain state-of-the-art performance in this task.

In Chapter 6 we extend the Multi-Scale ConvNet of the previous chapter to infer not only depth, but also surface normals and semantic labels at each point, and implement a third refinement scale to obtain higher resolution outputs. These provide both richer geometric descriptions and detailed class descriptions, and we achieve state-of-the-art performance for these tasks as well. We compare our semantic labeling system to the one we looked at originally in Chapter 3, and find that our newer method achieves greatly better accuracy, without the need for either hand-crafted feature descriptors or superpixel preprocessing.

Following this, in Chapter 7 we look at some of the factors involved in tuning convolutional network sizes, in the context of a single-object classification task. By constructing a network whose weights are tied between layers, we find that adding layers alone can lead to better performance. Moreover, through a series of control experiments, we also measure the relative effects of the number of feature maps and number of parameters; our results suggest that layers and parameters are the most important factors.

We also look at three ideas using related convolutional models for unsupervised feature learning in Chapter 8: We introduce Convolutional LISTA autoencoders, which emulate Deconvolutional sparse-coding networks using feed-forward autoencoders, and are able to learn similar features at a small fraction of the cost. We then explore an entropy objective that encourages feature map units to factorize into a few prototype templates with high activation, plus many deformation units concerned that edit reconstruction details. We lastly describe a convolutional ZCA whitening method that can be applied locally to arbitrarily large images using a single convolution kernel.

Finally, we discuss some future research directions and conclude in Chapter 9.

Chapter 2

Background

2.1 Convolutional Networks

Convolutional networks have been used for many computer vision applications, starting from their roots in digit classification [33, 77], to more recent systems for image classification and object detection [73, 113, 116, 125], as well as many others such as image denoising [65], pose estimation [131, 94] and stereo depth [145, 87]. They have also been applied with great success to speech recognition [95, 57] and natural language processing [14, 15]. The success of these systems stems from their ability to learn both image features, as well as the tiers of rules needed to combine them, all from the data itself. As a consequence, they can leverage large amounts of data to maximize their effectiveness, while still maintaining a compact size that can be deployed to future test cases.

A basic convolutional network, such as the one in Fig. 2.1, consists of applying multiple layers of learned *convolution kernels* together with elementwise nonlinearities, often intermixed with spatial pooling (subsampling). Each convolutional hidden layer is fed

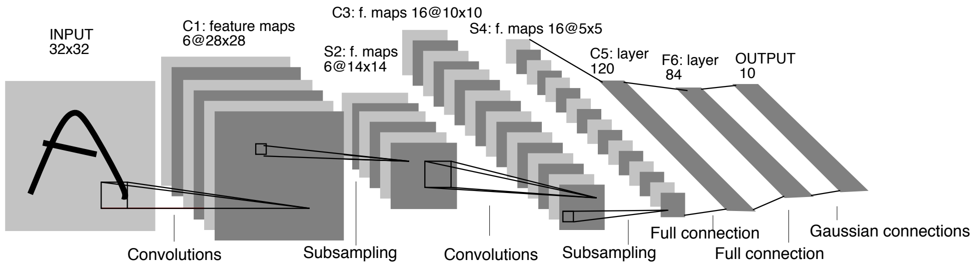


Figure 2.1: A convolutional network for classification, from LeCun *et al.* [77].

forward from the layer below:

$$\begin{aligned} h_0 &= x \\ h_{l+1} &= pool_l(f(W_l * h_l + b_l)), \quad l = 0, \dots, L \end{aligned}$$

where x is the input image, h_l is the hidden layer activations at layer l, W_l and b_l are the learned convolution kernel and feature bias, f is an activation function applied to each unit (*e.g.* rectification/thresholding; sigmoid), and $pool_l$ is the (optional) pooling at layer l. Common pooling schemes include max-, l_2 or average-pooling [7, 67], or strided convolutions, which correspond to weighted-sum pooling with trainable weights. These can serve to enforce local invariance to small feature movements, and/or increase the model's field of view more aggressively.

For a classification model, one or more fully-connected layers are placed on top, followed by a softmax output normalization, which produces a sum-normalized distribution over the classes:

$$y = \text{softmax}(W_C h_L + b_C), \quad \text{where } \text{softmax}(z) = \frac{e^z}{\sum_i e^{z_i}}$$

These models are commonly trained using backpropagation and stochastic gradient descent [77]. After defining the error to be optimized using a loss function, *e.g.* cross-entropy classification loss, the gradient of the error with respect to all model parameters is found using the chain rule through the hidden layers.

In the OverFeat detection system [113] (joint work between Pierre Sermanet, myself, Xiang Zhang, Michael Mathieu, Rob Fergus and Yann LeCun), we apply the fully-connected classification layers themselves convolutionally, using 1×1 convolution filters. This is equivalent to applying an entire classification network to large overlapping windows of the input image, but it applies all networks at once from the bottom up, thus sharing computation for the overlap regions. We simultaneously learn an object bounding box regressor, also applied at each input window, and then merge all the predictions together. In so doing, we use the multiple class/box predictions to vote on the final result, greatly improving accuracy. Furthermore, by applying the network at several different scales, we can both better align different sized objects to the ConvNet windows and boost the number of prediction samples, improving accuracy even more. This system is depicted in Fig. 1.1.

In any network, each convolution layer produces a set of output values at each spatial location (a "feature map"), using a weighted combination of the inputs from a local area. Convolutions have several intuitive interpretations:

- **Pattern response filtering**

  A unit responds when a template hits a corresponding pattern in the input.

- **Local weighted voting**

  Different feature activations vote on whether to make a next-layer unit active.

- **Spatially-constrained linear operations**

  The input is linearly transformed.

- **Weighted average pooling/unpooling**

  The input is subsampled (or upsampled) according to a weighted sum.

- **Mixtures of output patch templates**

  An output patch is splatted into a canvas.

By using local connections, as opposed to full all-to-all connections, ConvNets take advantage of local correlation and long-range decorrelation present in images, particularly at lower levels. If a fully connected network were trained for the same task, with infinite diverse data, the vast majority of connections would end up being local, and with the same weights replicated among most spatial locations. Thus convolutions provide a double-win: They regularize by enforcing zeros for long-range connections that might not otherwise be learned from a smaller finite dataset, while at the same time greatly reduce computation by not considering these zeros during propagation.

Many kinds of regularization can also be used to improve generalization and make better use of limited training data. One of the most effective is data augmentation: In the best case, adding random perturbations of the provided data adds a large diverse set of new samples; it can also encourage the model to be invariant to transformations not easily encoded in the architecture, and derails its ability to memorize exact samples. Other common techniques include several that average over injected training-time stochasticity, such as Dropout [122] or DropConnect [139], l_2 weight decay, and ensemble voting.

Features learned by convolutional networks at lower layers have analogies in both biological visual behaviors, as well as many hand-crafted computer vision techniques. In particular, the first convolutional layer in most networks tends to learn oriented gabors and color contrasts, which are then pooled. These features are then comprised chiefly of aggregated quantized edge orientations. Such representations can explain many behaviors observed in the primary visual cortex [64].

In addition, computer vision features like HoG [29] or SIFT [84] perform many of the same edge-summary operations: Each also identifies prevalent edge orientations, and combines them over spatial areas using histograms. Systems such as spatial pyramid matching [76] or deformable parts models [29] combine these further over larger areas; DPMs in particular are in essence ConvNets with few layers and large offsets [138]. However, hand-created descriptors ignore much relevant information [137], fundamentally lost in their design. Moreover, while low-level features can be possible to intuit, it is often far from clear how to combine them. The need for choices that can limit a model's effectiveness only worsens at higher layers, since these are where the system must make many of its eventual decisions.

Convolutional networks overcome this by automatically learning the parameters that define both the low-level features themselves, as well as their combination. This thesis focuses on applications of convolutional networks beyond classification and detection that require the prediction of pixel maps, *i.e.* 2D arrays containing inferred values at each pixel location.

2.2 Autoencoders

The use of neural networks to produce pixel maps has a close relation to autoencoders, which also output an image using a neural network. However, instead of predicting a pixel map of a different mode, the aim of an autoencoder is to reconstruct the original input under constraints. These have been used in particular for learning initial hidden layer representations in pretraining supervised networks [59, 37]. They also have close connections to both Sparse Coding and Restricted Boltzmann Machines.

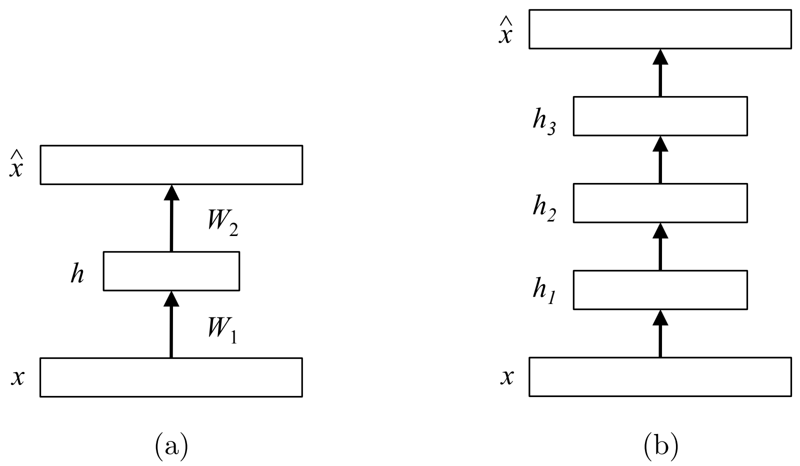


Figure 2.2: An autoencoder (a) and stacked deep autoencoder (b).

At its most basic, an autoencoder reconstructs the input from a single hidden layer; multiple hidden layers may be stacked as well to form a deep autoencoder (see Fig. 2.2).

In the case of a single layer, the output is formed using a linear combination of hidden activations, which are themselves produced according to a linear combination of the input and nonlinear activation function:

$$h_1 = f(W_1 x + b_1), \quad \hat{x} = W_2 h_1 + b_2$$

Here, x and $\hat{x}$ are the input and its reconstruction, h_1 is the hidden layer activation units, W_1 and W_2 are weight matrices, b_1 and b_2 the biases, and f is the activation function (*e.g.* sigmoid, rectified linear).

The weights W_1 and W_2 are optimized according to an objective function that defines the desired relationship between the input x and its reconstruction $\hat{x}$. For an autoencoder, we want these to be equal; a common objective in this case is the l_2 error,

$$L = \frac{1}{2} \sum_{i=1}^{N} ||x_i - \hat{x}_i||^2$$

where i ranges over all images in the training dataset.

Note if f is the identity, $\hat{x}$ is a linear projection of x, and the optimal solution for $W_2 W_1$ with respect to l_2 error is a PCA projection to the first k principal components of the data, where k is the size of h_1 [3]. That is, $W_2 W_1 = V V^T$ where V are the first k PCA directions of the training data, and the biases b_1 and b_2 are used to subtract and add back the mean of the training data. If we also enforce $W_1 = W_2^T = W$, then $W = V^T$ up to isometry.

With a nonlinear choice of f, the solution will start to move away from PCA. Additional constraints or regularizations make the learned behavior more different yet. Common variations include dimension bottlenecks [59], sparsity-inducing constraints [100, 103, 37] or contractive costs [102]. A somewhat recent technique is the denoising autoencoder [135, 136], which introduces random noise to the input and forces the network to learn to reconstruct the full original image from the incomplete data. To do so, it must learn correlations between parts of the input so that the missing regions can be restored.

Combining the l_2 reconstruction with an additive l_1 cost encourages sparser hidden representations, linking autoencoders with sparse coding. Sparse coding aims to find a hidden code that reconstructs the input, but with few active units; a prevalent form [93] is to minimize the cost

$$E_{SC}(z;W) = ||x - W^T z||_2^2 + \lambda |z|_1$$

where z is the code vector (hidden layer), and W^T is a decoding dictionary. Iterative algorithms are often used to infer a code z given the input x [5], however sparse coding may also be combined with feed-forward networks to quickly generate approximations of the code, *e.g.* PSD and LISTA [69, 45, 105]. Deconvolutional networks [147, 148] use successive layers of convolutional sparse coding to learn progressively higher-level features from unlabeled images.

Autoencoders have a particularly close connection to Restricted Boltzmann Machines (RBMs). A RBM describes a probability distribution, parameterized using a weight matrix connecting visible units with hidden units. A simple instance is where visible and hidden units are Bernoulli random variables (*i.e.* can take on the discrete states of either zero or one). However, to model continuous-valued images x, a common method is to use Gaussian units for the visibles, in which case the energy is

$$\begin{aligned} E_{gbRBM}(x,h) &= h^T W x - b_h^T h + \frac{1}{2} x^T x - b_x^T x \\ P_{gbRBM}(x,h) &= \frac{1}{Z} e^{-E_{gbRBM}(x,h)} \end{aligned}$$

Here, x is a vector of visible units (an image), h a vector of hidden units, b_x and b_h are the visible and hidden biases, and W is the weight matrix connecting visibles and hiddens.

The probability of an image x is obtained by marginalizing over h; this has the following associated energy (called "free energy"):

$$\begin{aligned} E_{gbRBM}(x) &= -\log \sum_{h \in \{0,1\}^k} \exp(-E_{gbRBM}(x,h)) \\ &= \frac{1}{2} x^T x - b_x^T x - \sum_{i=1}^{k} \log(1 + \exp(W_i \cdot x + b_{h,i})) \end{aligned}$$

Taking the derivative with respect to x and setting it to 0, we find that the energy has critical points where

$$\begin{aligned} x &= \sum_{i=1}^{k} W_i^T \frac{\exp(W_i \cdot x + b_{h,i}))}{1 + \exp(W_i \cdot x + b_{h,i}))} + b_x \\ &= W^T \sigma(Wx + b_h) + b_x \end{aligned}$$

The right hand side is a feed-forward autoencoder with sigmoid hidden activation σ and weights identical to the RBM; a fixed point of this autoencoder is a critical point of the RBM energy. In addition, the derivative of the RBM energy is equal to the difference between the input x and its autoencoder reconstruction (*i.e.* the reconstruction error).

Highly related in a more general setting, Alain *et al.* [1] show that any autoencoder trained with a contractive or denoising method models the derivative of the log data density, in the sense that the difference between the input and its reconstruction approaches the log density derivative. Although it is tempting to think of the autoencoder fixed points as energy minima, [1] note that some of these must be maxima or saddles, considering paths that lie between two minima.

Beyond applications to feature learning and density estimation, neural networks with similar architectures are starting to be used for image prediction tasks, such as image denoising [9, 10, 65, 143, 150], object detection [126] and semantic segmentation [16, 98]. This thesis explores this line of research further, applying image-generating convolutional networks to multiple tasks.

Chapter 3

Nonparametric Image Parsing using Adaptive Neighbor Sets

The work presented in this chapter appeared in CVPR 2012 [22], and was a collaboration with Rob Fergus.

3.1 Introduction

This chapter examines a k-nearest-neighbors (kNN) voting approach to dense semantic labeling that uses multiple hand-designed descriptors to classify super-pixels. This is an effective brute-force mechanism for using a database of labeled example regions, and serves as an essential point of comparison for our later convolutional network system in Chapter 6.

While simple, such kNN systems make only limited use of the data available. Features must be hand-tuned, and feature sets and data both must be carefully calibrated so that the different sources contribute relatively similar amounts and no single source dominates. Furthermore, in the semantic labeling task there may be a potentially large number of different label classes, stemming from the high diversity of the visual world, and the distribution of classes is often highly uneven (see Fig. 3.8). kNN classifiers present a trade-off here: They naturally handle large label sets, since one need only consider the labels of those points retrieved during test queries; on the other hand, rare class examples are hard to find and underrepresented in query results. Consequently, many classes will have a small number of example instances even using a large dataset, making it hard to train good classifiers.

Starting from the kNN "superparsing" method of Tighe and Lazebnik [128] as a baseline, we add two novel mechanisms that help address these issues:

1. In an off-line training phase, we learn a set of weights for each descriptor type of every segment in the training set. The weights are trained to minimize classification error in a weighted nearest-neighbor scheme. Individually weighting each descriptor has the effect of introducing a distance metric that varies throughout the descriptor space. This enables it to overcome the limitations of a global metric, as outlined above. It also allows us to discard outlier descriptors that would otherwise hurt performance (e.g. from segmentation errors).

2. At query-time, we adapt the set of points used by the weighted-NN classification based on context from the query image. We first remove segments based on a global context match. Crucially, we then add back previously discarded segments from rare classes. Here we use the local context of segments to look up rare class examples from the training set. This boosts the representation of rare classes within the kNN sets, giving a more even class distribution that improves classification accuracy.

The overall theme of these methods is the customization of the dataset for a particular query to improve performance.

In addition to Tighe and Lazebnik [128, 129], other related non-parametric approaches to recognition include: the SIFT-Flow scene parsing method of Liu *et al.* [80, 81]; scene classification using Tiny Images by Torralba *et al.* [132] and the Naive-Bayes NN approach from Boiman *et al.* [6]. However, none of these involve re-weighting of the data, and context is limited to a CRF at most.

Our classification and training procedure is much related to Neighborhood Component Analysis [38]. NCA also learns a distance metric for kNN classification using leave-one-out training. However, the metric is parameterized by a single quadratic distance matrix applied to all feature descriptors. By contrast, we find neighbors using unmodified descriptors, then tune the weights of each to influence the class predictions, effectively learning a metric that varies according to the local region. It is possible that the two approaches may be combined, however we did not explore that in this work.

Our re-weighting approach has interesting similarities to Frome *et al.* [32] (and related work from Malisiewicz & Efros [85, 86]). Motivated by the inadequacies of a single global distance metric, they use a different metric for each exemplar in their training set, which is integrated into an SVM framework. The main drawback to this is that the evaluation of a query is slow (∼minutes/image). The weights learned by our scheme are equivalent to a local modulation of the distance metric, with a large weight moving the point closer to a query, and vice-versa. Furthermore, the context-based training set adaptation in our method also effects a query-dependent metric on the data.

The re-weighting scheme we propose also has connections to a traditional machine learning approach called editing [20, 71]. In edting, individual points in the dataset may be modified, however these are usually binary in that they either keep or completely remove each training point. Of this family, the most similar to ours is Paredes and Vidal [96], who also use real-valued weights on the points. However, their approach does not handle multiple descriptor types and is demonstrated on a range of small text classification datasets.

There is extensive work on using context to help recognition [61, 92, 133, 134]; the most relevant approaches being those of Gould *et al.* [41, 42] and in particular Heitz & Koller [55] who use "stuff" to help find "things." Heitz *et al.* [54] use similar ideas in a sophisticated graphical model that reasons about objects, regions and geometry. These works have similar goals regarding the use of context but quite different methods. Our approach is simpler, relying on NN lookups and standard gradient descent for learning the weights.

Our work also has similar goals to multiple kernel learning approaches (e.g. [35]) which combine weighted feature kernels, but the underlying mechanisms are quite different: we do not use SVMs, and our weights are per-descriptor. By contrast, the weights used in these methods are constant across all descriptors of a given type. Finally, Boosting [109] is an approach that weights each datapoint individually, as we do, but it is based on parametric models rather than non-parametric ones.

3.2 Approach

Our approach builds on the nearest-neighbor voting framework of Tighe and Lazebnik [128] and uses three distinct stages to classify image segments: (i) global context selection; (ii) learning descriptor weights; (iii) adding local context segments. Stages (i) and (ii) are used in off-line training, while (i) and (iii) are used during evaluation. While stage (i) is adopted from [128], the other two stages are novel and the main focus of our paper.

A query image Q consists of a set of super-pixel segments q, each of which we need to classify into one of C classes. The training dataset T consists of super-pixel segments s, taken from images I_1 to I_M. The true class c_s^* for each segment in T is known. Each segment is represented by D different types of descriptors (the same set of 19 used in [128]. These include quantized SIFT, color, position, shape and area features. Additionally, each image I_m has a set global context descriptors, $\{g_m\}$ that capture the content of the entire image; these are computed in advance and stored in kd-trees for efficient retrieval.

3.2.1 Global Context Selection

In this stage, we use overall scene appearance to remove descriptors from scenes bearing little resemblance to the query. For example, the segments taken from a street scene are likely to be distractors when trying to parse a mountain scene. Thus their removal is expected to improve performance. A secondary benefit is that the subsequent two stages need only consider a small subset of the training dataset T, which gives a considerable speed-up for big datasets.

For each query Q we compute global context descriptors $\{g_q\}$, which consists of 4 types: (i) a spatial pyramid of vector quantized SIFT [76]; (ii) a color histogram spatial pyramid and (iii) Gist computed with two different parameter settings [92]. For each of the types, we find the nearest neighbors amongst the training set $\{g_m\}$. The ranks across the four types of context descriptor are averaged to give an overall ranking. We then form a subset G of the segment-level training database T that consists of segments belonging to the top v images from our image-level ranking. We denote the global match set $G = \textsc{GlobalMatches}(Q, v)$. v is an important parameter whose setting we explore in Section 5.4.

3.2.2 Learning Descriptor Weights

To learn the weights, we adopt a leave-one-out strategy, using each segment s (from image I_m) in the training dataset T as probe segment (a pretend query). The weights of the neighbors of s are then adjusted to increase the probability of correctly predicting the class of s.

For a query segment s, we first compute the global match set $G_s = \textsc{GlobalMatches}(I_m, v)$. Let the set of descriptors of s be D_s. Following [128], the predicted class $\hat{c}$ for each segment is the one that maximizes the ratio of posterior probabilities $P(c|D_s)/P(\bar{c}|D_s)$. After the application of Bayes rule using a uniform class prior[1] and making a naive-

[1]Using the true, highly-skewed, class distribution $P(c)/P(\bar{c})$ dramatically impairs performance for

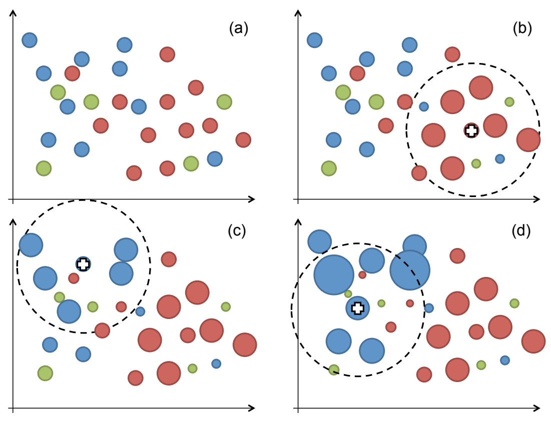


Figure 3.1: Toy example of our re-weighting scheme. (a): Initially all descriptors have uniform weight. (b), (c) & (d): a probe point is chosen (cross) and points in the neighborhood (black circle) of the same class as the probe have their weights increased. Points of a different class have their weights decreased, so rejecting outlier points. In practice, (i) there are multiple descriptor spaces, one for each descriptor type and (ii) the GlobalMatch operation removes some of the descriptors.

Bayes assumption for combining descriptor types, this is equivalent to maximizing the product of likelihood ratios for each descriptor type:

$$\hat{c} = \arg\max_c L(s, c) = \arg\max_c \prod_{d \in D_s} \frac{P(d|c)}{P(d|\bar{c})} \tag{3.1}$$

The probabilities $P(d|c)$ and $P(d|\bar{c})$ are computed using nearest-neighbor lookups in the space of the descriptor type of d, over all segments in the global match set G. In the un-weighted case (i.e. no datapoint weights), this is:

$$P(d|c) \propto p_d(c) = \frac{n_d^N(c)}{n_d(c)}, \quad P(d|\bar{c}) \propto \bar{p}_d(c) = \frac{\bar{n}_d^N(c)}{\bar{n}_d(c)}$$

where $n_d^N(c)$ is the number of points of class c in the nearest neighbor set N of d, determined by taking the closest k neighbors of d. [2] $n_d(c)$ is the total number of points in class c. $\bar{n}_d^N(c)$ is the number of points *not* of class c in the nearest neighbor set N of d

rare classes.

[2]We also include all points at zero distance from d, so $n_d^N(c)$ is occasionally larger than k.

(i.e. $\sum_{c' \neq c} n_d^N(c')$), and similarly for $\bar{n}_d$. Conceptually, both $n_d^N(c)$ and $n_d(c)$ should be computed over the match set G; in practice, this sample may be small enough that using G just for $n_d^N(c)$ and estimating $n_d(c)$ over the entire training database T can reduce noise.

To eliminate zeros in $P(d|\bar{c})$, we smooth the above probabilities using a smoothing factor t:

$$q_d(c) = (n_d^N(c) + \bar{n}_d^N(c))^2 \cdot p_d(c) + t \qquad (3.2)$$

$$\bar{q}_d(c) = (n_d^N(c) + \bar{n}_d^N(c))^2 \cdot \bar{p}_d(c) + t \qquad (3.3)$$

and define the smoothed likelihood ratio $L_d(c)$:

$$L_d(c) = \frac{q_d(c)}{\bar{q}_d(c)}$$

We now introduce weights w_{di} for each descriptor d of each segment i. This changes the definitions of n_d and n_d^N:

$$n_d(c) = \sum_{i \in T} w_{di} \delta(c_i^*, c) = W^T \Delta$$

$$n_d^N(c) = \sum_{i \in N} w_{di} \delta(c_i^*, c) = W^T \Delta^N$$

where c_i^* is the true class of point i and T is the training set. Note that when using only the match set G to estimate $n_d(c)$, the sum over T need only be performed over G. In matrix form, W is the vector of weights w_{di}, and Δ is the $|T| \times |C|$ class indicator matrix whose ci-th entry is $\delta(c_i, c)$. For neighbor counts, Δ^N is the restriction of Δ to the neighbor set N — that is, its entries in rows $i \notin N$ are zero.

Similarly, for $\bar{n}_d(c)$ and $\bar{n}_d^N(c)$ we use the complement $\bar{\Delta} = 1 - \Delta$:

$$\bar{n}_d(c) = \sum_{i \in T} w_{di} \delta(c_i^*, \bar{c}) = W^T \bar{\Delta}$$

$$\bar{n}_d^N(c) = \sum_{i \in N} w_{di} \delta(c_i^*, \bar{c}) = W^T \bar{\Delta}^N$$

To train the weights, we choose a negative log-likelihood loss:

$$\begin{aligned} J(W) &= \sum_{s \in T} J_s(W) = \sum_{s \in T} - \log L(s, c^*) + \log \sum_{c \in C} L(s, c) \\ &= \sum_{s \in T} \left(- \sum_{d \in D_s} \log L_d(c^*) + \log \sum_{c \in C} \prod_{d \in D_s} L_d(c) \right) \end{aligned}$$

The derivatives with respect to W are back-propagated through the nearest neighbor probability calculations using 5 chain rule steps. The vector of weights W_d (the weights for all segments on descriptor type d) is updated as follows:

Step 1:

$$\frac{\partial n_d}{\partial W_d} = \Delta, \quad \frac{\partial n_d^N}{\partial W_d} = \Delta^N, \quad \frac{\partial \bar{n}_d}{\partial W_d} = \bar{\Delta}, \quad \frac{\partial \bar{n}_d^N}{\partial W_d} = \bar{\Delta}^N$$

Step 2:

$$\frac{\partial p_d}{\partial W_d} = (\Delta^N - p_d \cdot \Delta)/n_d, \quad \frac{\partial \bar{p}_d}{\partial W_d} = (\bar{\Delta}^N - \bar{p}_d \cdot \bar{\Delta})/\bar{n}_d$$

Step 3:

$$\begin{aligned} \frac{\partial q_d}{\partial W_d} &= 2(n_d^N + \bar{n}_d^N) \cdot p \cdot 1^N + (n_d^N + \bar{n}_d^N)^2 \cdot \frac{\partial p_d}{\partial W_d} \\ \frac{\partial \bar{q}_d}{\partial W_d} &= 2(n_d^N + \bar{n}_d^N) \cdot \bar{p} \cdot 1^N + (n_d^N + \bar{n}_d^N)^2 \cdot \frac{\partial \bar{p}_d}{\partial W_d} \end{aligned}$$

Step 4:

$$\frac{\partial \log L_d}{\partial W_d} = \frac{1}{q_d} \frac{\partial q_d}{\partial W_d} - \frac{1}{\bar{q}_d} \frac{\partial \bar{q}_d}{\partial W_d}$$

Step 5:

$$\frac{\partial J_s}{\partial W_d} = -\frac{\partial \log L_d}{\partial W_d}(c^*) + \frac{1}{\sum_c L(c)} \sum_c L(c) \cdot \frac{\partial \log L_d(c)}{\partial W_d}$$

where $1^N = \Delta^N + \bar{\Delta}^N$, and products and divisions are performed element-wise. The weight matrix is updated using gradient descent:

$$W \leftarrow W - \eta \frac{\partial J_s}{\partial W}$$

where η is the learning rate parameter. In addition, we enforce positivity and upper bound constraints on each weight, so that $0 \leq w_{di} \leq 1$ for all d, i. We initialize the learning with all weights set to 0.5 and η set to 0.1.

The above procedure provides a principled approach to maximizing the classification performance, using the same naive-Bayes framework of [128]. It is also practical to deploy on large datasets: although the the time to compute a single gradient step is $O(|T||C|)$, we found that fixing n_d and $\bar{n}_d$ to their values with the initial weights yields good performance, and limits the time for each step to $O(|G||C|)$.

Effect of the Smoothing Parameter

Aside from smoothing the NN probabilities, the smoothing parameter t also modulates $L_d(c)$ as a function of $n_d(c)$, the number of descriptors of each class. As such, it gives a natural way to bias the algorithm toward common classes or toward rare ones.

To see this, let us assume $n_d^N(c) + \bar{n}_d^N(c) = k$ (which is usually the case; see footnote 2). This lets us rearrange $L_d(c)$ to obtain (omitting d for brevity and defining $u = t/k^2$):

$$L(c) = \frac{n^N(c)\bar{n}(c) + u \cdot n(c)\bar{n}(c)}{\bar{n}^N(c)n(c) + u \cdot n(c)\bar{n}(c)}$$

Note that $n(c)\bar{n}(c)$ depends only on the frequency of class c in the dataset, not on the NN lookup. The influence of t therefore becomes larger for progressively more common classes. So by increasing t we bias the algorithm toward rare classes, an effect we systematically explore in Section 5.4.

3.2.3 Adding Segments

The global context selection procedure discards a large fraction of segments from the training set T, leaving a significantly smaller match set G. This restriction means that rare classes may have very few examples in G — and sometimes none at all. Consequently, (i) the sample resolution of rare classes is too small to accurately represent their density, and (ii) for NN classifiers that use only a single lookup among points of all classes (as ours does), common points may fill a search window before any rare ones are reached. We seek to remedy this by explicitly adding more segments of rare classes back into G.

To decide which points to add, we index rare classes using a descriptor based on semantic context. Since the classifier is already fairly accurate at common background classes, we can use its existing output to find probable background labels around a given segment. The context descriptor of a segment is the normalized histogram of class labels in the 50 pixel dilated region around it (excluding the segment region itself). See Fig. 3.2(a) & (b) for an illustration of this operation, which we call MakeContextDescriptor.

To generate the index, we perform leave-one-out classification on each image in the training set, and index each super-pixel whose class occurs below a threshold of r times in its image's match set G. In this way, the definition of a rare class adapts naturally according to the query image. This the BuildContextIndex operation.

When classifying a test image, we first classify the image without any extra segments. These labels are used to generate the context descriptors as described above. For each super-pixel, we look up the nearest r points in the rare segments index, and add these to the set of points G used to classify that super-pixel. See Algorithm 2 for more details.

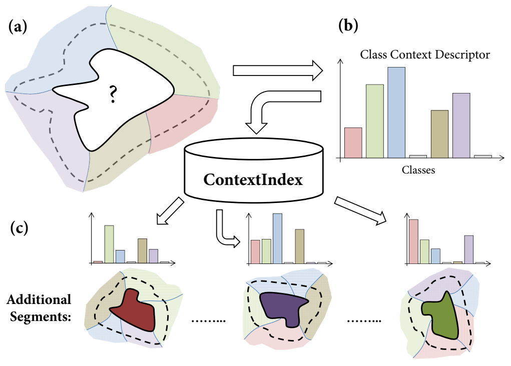


Figure 3.2: Context-based addition of segments to the global match set G. (a): Segment in the query image, surrounded by an initial label map. (b): Histogram of class labels, built by dilating the segment over the label map, which captures the semantic context of the region. This is matched with histograms built in the same manner from the training set T. (c): Segments in T with a similar surrounding class distribution are added to G.

3.3 Algorithm Overview

The overall training procedure is summarized in Algorithm 1. We first learn the weights for each segment/descriptor, before building the context index that will be used to add segments at test time. Note that we do not rely on ground truth labels for constructing this index, since not all segments in T are necessarily labeled. Instead, we use the predictions from our weighted NN classifier. NN algorithms work better with more data, so to boost performance we make a horizontally flipped copy of each training image and add it to the training set.

The evaluation procedure, shown in Algorithm 2, involves two distinct classifications. The first uses the weighted NN scheme to give an initial label set for the query image. Then we lookup each segment in the ContextIndex structure to augment G with more segments from rare classes. We then run a second weighted classification using this extended match set to give the final label map.

Algorithm 1 Training Procedure

```
1: procedure LearnWeights($T$)
2:     Parameters: $v, k$
3:     $W_{di} = 0.5$
4:     for all segments $s \in T$ do
5:         $G$ =GlobalMatches($I_m, v$)
6:         NN-lookup to obtain $\Delta^N, \bar{\Delta}^N$
7:         Compute $\frac{\partial J_s}{\partial W_d}$
8:         $W_d \leftarrow W_d - \eta \frac{\partial J_s}{\partial W_d}$
9:     end for
10: end procedure

11: procedure BuildContextIndex($T, W$)
12:     Parameters: $v, k$
13:     ContextIndex = $\varnothing$
14:     for all $I \in T$ do
15:         $G$ =GlobalMatches($I, v$)
16:         label_map = Classify($I, G, W, k$)
17:         for all Segments $s$ in $I$ with rare $\hat{c}_s$ in $G$ do
18:           desc = MakeContextDescriptor($s$, label_map)
19:           Add (desc $\rightarrow I, s$) to ContextIndex
20:         end for
21:     end for
22: end procedure

23: function Classify($I, G, W, k$)
24:     for all segments $s \in$ image $I$ do
25:         $k$NN-lookup in $G$ to obtain $\Delta^N, \bar{\Delta}^N$
26:         Use weights $W$ to compute $n_d^N(c), \bar{n}_d^N(c)$ and $L_d(c)$
27:         $\hat{c}_s = \underset{c}{\text{argmax}} \prod_d L_d(c)$
28:     end for
29:     return label_map $\hat{c}$
30: end function
```

Algorithm 2 Evaluation Procedure

```
1: procedure EvaluateTestImage($Q$)
2:     Parameters: $v, k, r$
3:     $G$ =GlobalMatches($Q, v$)
4:     init_label_map = Classify($Q, G, W, k$)
5:     for all segments $s \in Q$ do
6:      desc = MakeContextDescriptor($s$, init_label_map)
7:      $H_s$ = ContextMatches(desc,ContextIndex,r)
8:     end for
9:     final_label_map = Classify($Q, G \cup H, W, k$)
10: end procedure
```

3.4 Experiments

We evaluate our approach on two datasets: (i) Stanford background [41] (572/143 training/test images, 8 classes) and (ii) the larger SIFT-Flow [80] dataset (2488/200 training/test images, densely labeled with 33 object classes).

In evaluating sense parsing algorithms there are two metrics that are commonly used: per-*pixel* classification rate and per-*class* classification rate. If the class distribution were uniform then the two would be the same, but this is not the case for real-world scenes. A problem with optimizing pixel error alone is that rare classes are ignored since they occupy only a few percent of image pixels. Consequently, the mean class error is a more useful metric for applications that require performance on all classes, not just the common ones. Our algorithm is able to smoothly trade off between the two performance measures by varying the smoothing parameter t at evaluation time. Using a 2D plot for the pair of metrics, the curve produced by varying t gives the full performance picture for our algorithm.

Our baseline is the system described in Section 2, but with no image flips, no learned weights (i.e. they are uniform) and no added segments. It is essentially the same as the Tighe and Lazebnik [128], but with a slightly different smoothing of the NN counts. Our method relies on the same set of 19 super-pixel descriptors used by [128]. As other authors do, we compare the performance without an additional CRF layer so that any differences in local classification performance can be seen clearly. Our algorithm uses the following parameters for all experiments (unless otherwise stated): $v = 200$, $k = 10$, $r = 200$.

3.4.1 Stanford Background Dataset

Fig. 3.3 shows the performance curve of our algorithm on the Stanford Background dataset, along with the baseline system. Also shown is the result from Gould *et al.* [41], but since they do not measure per-class performance, we show an estimated range on the x-axis. While we convincingly beat the baseline and do better than Gould *et al.* [3], our best per-pixel performance of 75.3% fall short of the current state-of-the-art on the dataset, 78.1% by Socher *et al.* [121]. The small size of the training set is problematic for our algorithm, since it relies on good density estimates from the NN lookup. Indeed, the limited size of the dataset means that the global match set is most of the dataset (i.e. $|G|$ is close to $|T|$), so the global context stage is not effective. Furthermore, since there are only 8 classes, adding segments using contextual cues gave no performance gain either. We therefore focus on the SIFT-Flow dataset which is larger and better suited to our algorithm.

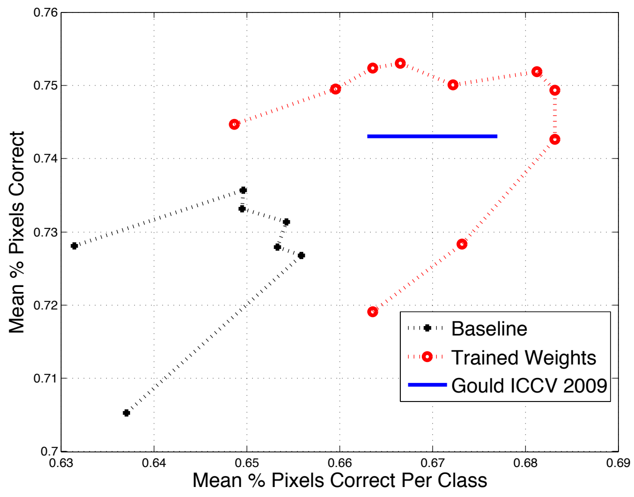


Figure 3.3: Evaluation of our algorithm on the Stanford background dataset, using local labeling only. x-axis is mean per-class classification rate, y-axis is mean per-pixel classification rate. Better performance corresponds to the top right corner. Black = Our version of [128]; Red = Our algorithm (without added segments step); Blue = Gould *et al.* [41] (estimated range).

3.4.2 SIFT-Flow Dataset

The results of our algorithm on the SIFT-Flow dataset are shown in Fig. 3.4, where we compare to other approaches using local labeling only. Both the trained weights and adding segments procedures give a significant jump in performance. The latter procedure

[3]Assuming some a per-class performance consistent with their per-pixel performance.

only gives a per-class improvement, consistent with its goal of helping the rare classes (see Fig. 3.8 for the class distribution).

To the best of our knowledge, Tighe and Lazebnik [128] is the current state-of-the-art method on this dataset (Fig. 3.4, black square). For local labeling, our overall system outperforms their approach by **10.1%** (29.1% vs 39.2%) in per-class accuracy, for the same per-pixel performance, a 35% relative improvement. The gain in per-pixel accuracy is **3.6%** (73.2% vs 76.8%).

Adding an MRF to our approach (Fig. 3.4, cyan curve) gives 77.1% per-pixel and 32.5% per-class accuracy, outperforming the best published result of Tighe and Lazebnik [128] (76.9% per-pixel and 29.4% per-class). Note that their result uses geometric features not used by our approach. Adding an MRF to our implementation of their system gives a small improvement over the baseline which is significantly outperformed by our approach + an MRF.

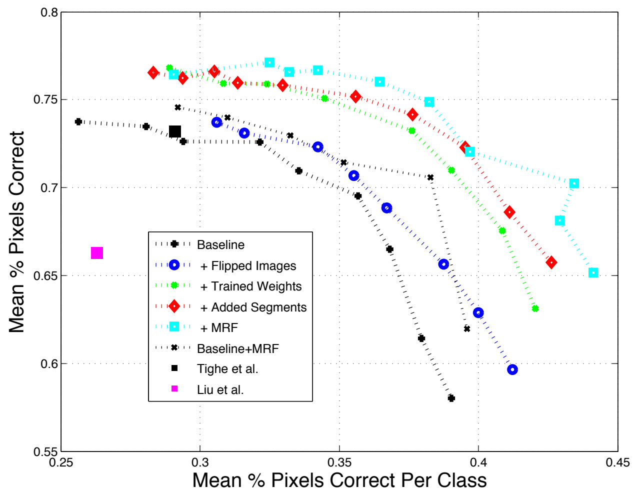


Figure 3.4: Evaluation of our algorithm on the SIFT-Flow dataset. Better performance is in the top right corner. Our implementation of [128] (black + curve) closely matches their published result (black square). Adding flipped versions of the images to the training set improves the baseline a small amount (blue). A more significant gain is seen when after training the NN weights (green). Refining our classification after adding segments (red) gives a further gain in per-class performance. Adding an MRF (cyan) also gives further gain. Also shown is Liu *et al.* [80] (magenta). Not shown is Shotton *et al.* [114]: 0.13 class, 0.52 pixel.

Sample images classified by our algorithm are shown in Fig. 5.4. We also demonstrate the significance of our results by re-running our methods on a different train/test split of the SIFT-Flow dataset. The results obtained are very similar to the original split and are shown in Fig. 3.5.

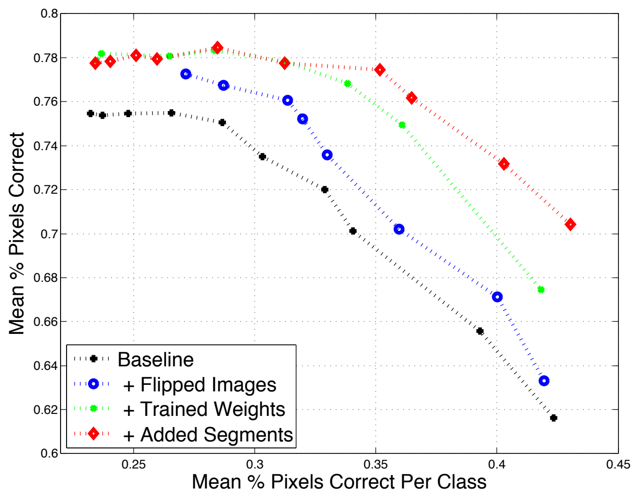


Figure 3.5: Results for a different train/test split of the SIFT-Flow dataset to one standard one used in Fig. 3.4. Similar results are obtained on both test sets.

In Fig. 3.6, we explore the role of the global context selection by varying the number of image-level matches, controlled by the v parameter which dictates $|G|$. For small values performance is poor. Intermediate v gives improved performance under both metrics. But if v is too large, G contains many unrelated descriptors and the per-class performance is decreased. This demonstrates the value of the global context selection procedure, since without it $G = T$, and the per-class performance would be poor.

In Fig. 3.7 we visualize the descriptor weights, showing how they vary across class and descriptor type (by averaging them over all instances of each class, since they differ for each segment). Note how the weights jointly vary across both class and descriptor. For example, the `min_height` descriptor usually has high weight, except for some spatially diffuse classes (e.g. desert,field) where its weight is low.

Fig. 3.8 shows the expected class distribution of super-pixels in G for the SIFT-Flow dataset before and after the adding segments procedure, demonstrating its efficacy. The

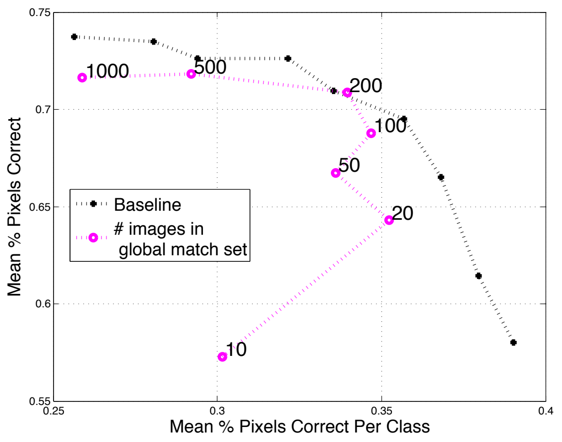


Figure 3.6: The global context selection procedure. Changing the parameter v (value at each magenta dot) affects both types of error. See text for details. For comparison, the baseline approach using a fixed $v = 200$ (and varying the smoothing t) is shown.

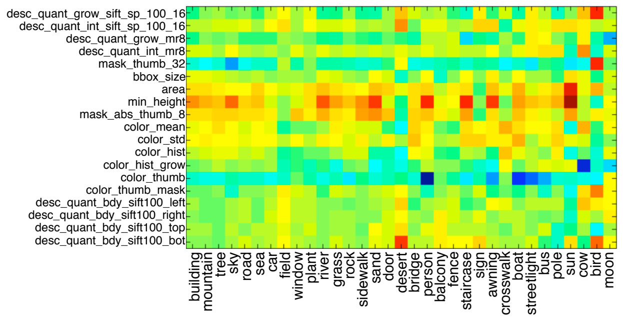


Figure 3.7: A visualization of the mean weight for different classes by descriptor type. Red/Blue corresponds to high/low weights. See text for details.

increase in rare segments is important in improving per-class accuracy (see Fig. 3.4).

	Learned Weights	Full
Global Descriptors	2.8	2.8
Segment Descriptors	3.0	3.0
GlobalMatch	0.9	0.9
Classify	3.5	3.5
ContextMatches	-	0.4
Classify	-	6.1
Total	10.3	16.6

Table 3.1: Timing breakdown (seconds) for the evaluation of a single query image using the full system and our system without adding segments (just global context match + learning weights). Note the descriptor computation makes up around half of the time.

In Table 3.1, we list the timings for each stage of our algorithm running on the SIFT-

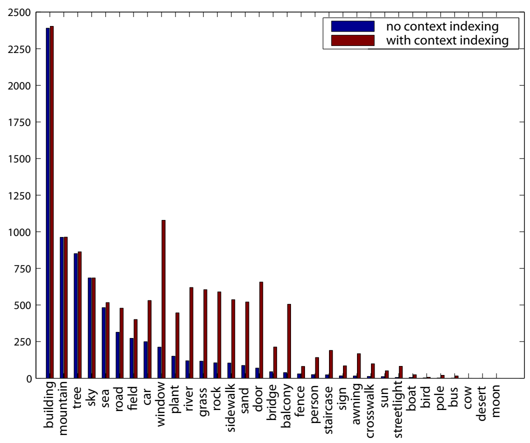


Figure 3.8: Expected number of super-pixels in G with the same true class c_s^* of a query segment, ordered by frequency (blue). Note the power-law distribution of frequencies, with many classes having fewer than 50 counts. Following the Adding Segments procedure, counts of rare classes are significantly boosted while those for common classes are unaltered (red). Queries were performed using the SIFT-Flow dataset.

Flow dataset, implemented in Matlab. Note that a substantial fraction of the time is just taken up with descriptor computations. The search parts of our algorithm run in a comparable time to other non-parametric approaches [128], being considerably faster than methods that use per-exemplar distance measures (e.g. Frome *et al.* [32] which takes 300s per image).

3.5 Discussion

In this chapter we have described two mechanisms for enhancing the performance of non-parametric scene parsing based on kNN methods. Both share the underlying idea of customizing the dataset for each kNN query. Rather than assuming that the full training set is optimally discriminative, adapting the dataset allows for better use of imperfectly generated descriptors with limited power. Learning weights focuses the classifier on more discriminative features and removes outlier points. Likewise, context-based adaptation uses information beyond local descriptors to remove distractor super-pixels whose appearances are indistinguishable from those of relevant classes. Reintroducing rare class examples improves density lost in the initial global pruning. On sufficiently

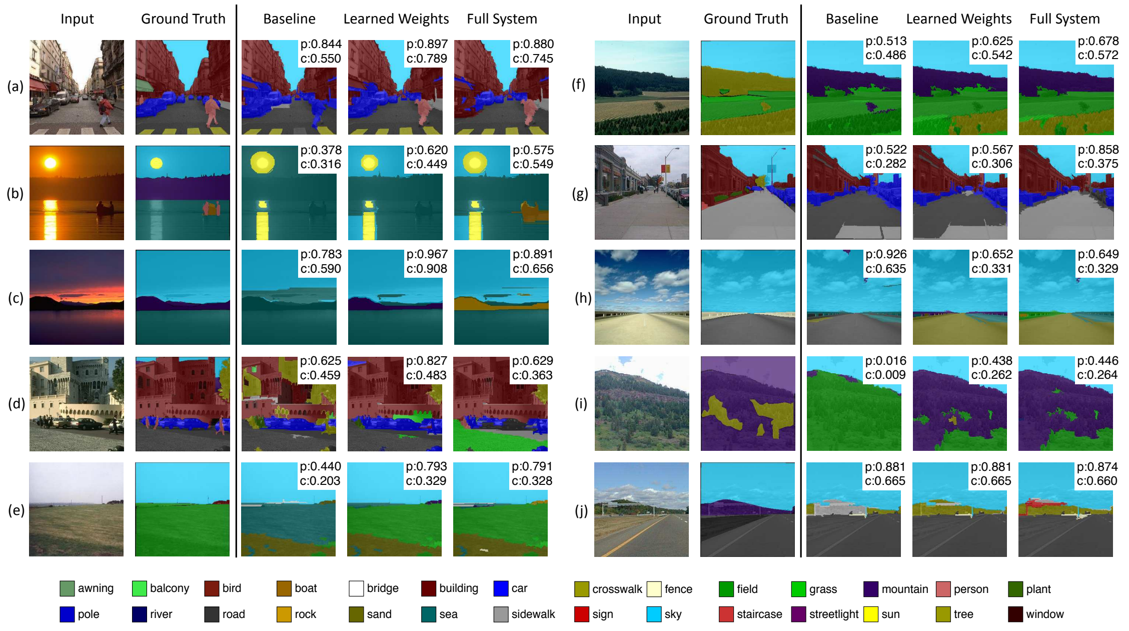

Input
Ground Truth
Baseline
Learned Weights
Full System
(a) p:0.844 c:0.550 p:0.897 c:0.789 p:0.880 c:0.745
(b) p:0.378 c:0.316 p:0.620 c:0.449 p:0.575 c:0.549
(c) p:0.783 c:0.590 p:0.967 c:0.908 p:0.891 c:0.656
(d) p:0.625 c:0.459 p:0.827 c:0.483 p:0.629 c:0.363
(e) p:0.440 c:0.203 p:0.793 c:0.329 p:0.791 c:0.328
(f) p:0.513 c:0.486 p:0.625 c:0.542 p:0.678 c:0.572
(g) p:0.522 c:0.282 p:0.567 c:0.306 p:0.858 c:0.375
(h) p:0.926 c:0.635 p:0.652 c:0.331 p:0.649 c:0.329
(i) p:0.016 c:0.009 p:0.438 c:0.262 p:0.446 c:0.264
(j) p:0.881 c:0.665 p:0.881 c:0.665 p:0.874 c:0.660
awning
balcony
bird
boat
bridge
building
car
crosswalk
fence
field
grass
mountain
person
plant
pole
river
road
rock
sand
sea
sidewalk
sign
sky
staircase
streetlight
sun
tree
window

Figure 3.9: Example images from the SIFT-Flow dataset, annotated with classification rates using per-pixel ("p") and per-class ("c") metrics. Learning weights improves overall performance. Adding rare class examples improves classification of less common classes, like the boat in (b) and sidewalk in (g). Failures include labeling the road as sand in (h) and the mountain as rock (a rarer class) in (c).

large datasets, both contributions give a significant performance gain.

While this kNN system can learn weights for each feature descriptor indicating its relative importance, it does not yet learn the features themselves, nor the steps that combine them into a classification prediction. We now begin to investigate a system that has these capabilities: By their use of trainable weights in multiple layers of computation, convolutional networks take the principle of model customization much further, learning entire stacks of features automatically tuned for the task and the data.

Chapter 4

Restoring An Image Taken Through a Window Covered with Dirt or Rain

The work presented in this chapter appeared in ICCV 2013 [24], and was a collaboration with Dilip Krishnan and Rob Fergus.

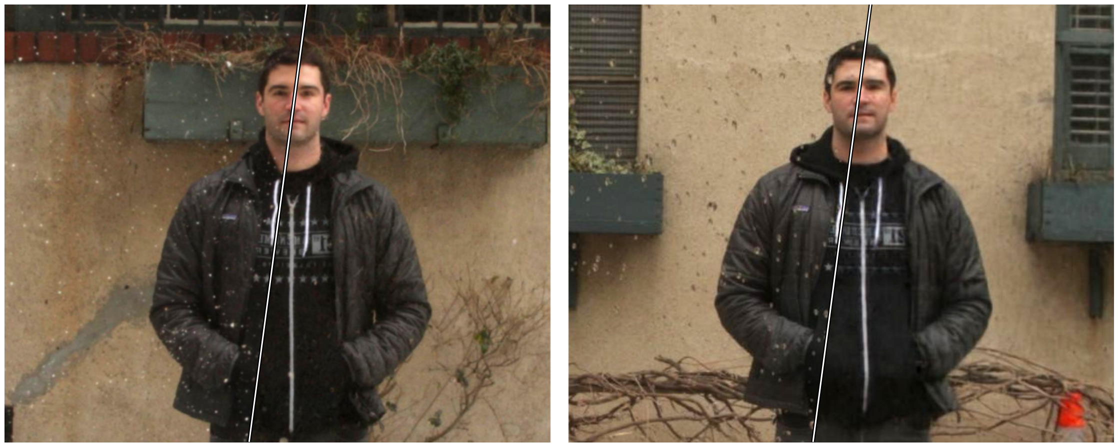


Figure 4.1: A photograph taken through a glass pane covered in dirt (left) and rain (right), along with the output of our neural network model trained to remove this type of corruption.

4.1 Introduction

In the previous chapter, we described a system for predicting per-pixel semantic labels using superpixels and hand-designed features. We now begin to apply convolutional networks to infer per-pixel outputs. As we will see in Chapter 6, the ability of these networks to learn multiple layers of weighted combinations will allow them to leverage large datasets and achieve greatly better performance on the same task, using a combination of global and local fields of view. First, however, we investigate an effective application of convolutional networks using purely *local* fields of view: restoring images that contain compact structured noise in the form of dirt or rain droplets. In this chapter we also find a benefit to *training convolutionally*, *i.e.* evaluating the loss on the final predicted image obtained after averaging individual patch predictions, and examine the effects of this.

Photographs taken through a window are often compromised by dirt or rain present on the window surface. Common cases of this include when a person is inside a car, train or building and wishes to photograph the scene outside, or exhibits in museums displayed behind protective glass. Such scenarios have become increasingly common with the widespread use of smartphone cameras. Beyond consumer photography, many cameras are mounted outside, e.g. on buildings for surveillance or on vehicles to prevent collisions. These cameras are protected from the elements by an enclosure with a transparent window.

Such images are affected by many factors including reflections and attenuation. However, in this paper we address the particular situation where the window is covered with dirt or water drops, resulting from rain. As shown in Fig. 4.1, these artifacts significantly degrade the quality of the captured image.

The classic approach to removing occluders from an image is to defocus them to the point of invisibility at the time of capture. This requires placing the camera right up against the glass and using a large aperture to produce small depth-of-field. However, in practice it can be hard to move the camera sufficiently close, and aperture control may not be available on smartphone cameras or webcams. Correspondingly, many shots with smartphone cameras through dirty or rainy glass still have significant artifacts, as shown in Fig. 4.9.

In this paper we instead restore the image post-capture, treating the dirt or rain as a structured form of image noise. Our method only relies on the artifacts being spatially compact, thus is aided by the rain/dirt being in focus — hence the shots need not be taken close to the window.

Our approach is to use a convolutional neural network to predict clean patches, given dirty *or clean* ones as input. By asking the network to produce a clean output, regardless of the corruption level of the input, it implicitly must both detect the corruption and, if present, in-paint over it. Integrating both tasks simplifies and speeds test-time operation, since separate detection and in-painting stages are avoided.

Training the models requires a large set of patch pairs to adequately cover the space inputs and corruption, the gathering of which was non-trivial and required the development of new techniques. However, although training is somewhat complex, test-time operation is simple: a new image is presented to the neural network and it directly outputs a restored image.

4.1.1 Related Work

Image denoising is a very well studied problem, with current approaches such as BM3D [17] approaching theoretical performance limits [79]. However, the vast majority of this literature is concerned with additive white Gaussian noise, quite different to the image artifacts resulting from dirt or water drops. Our problem is closer to shot-noise removal, but differs in that the artifacts are not constrained to single pixels and have characteristic structure. Classic approaches such as median or bilateral filtering have no way of leveraging this structure, thus cannot effectively remove the artifacts (see

Section 5.4).

Learning-based methods have found widespread use in image denoising, e.g. [152, 93, 99, 153]. These approaches remove additive white Gaussian noise (AWGN) by building a generative model of clean image patches. In this paper, however, we focus on more complex structured corruption, and address it using a neural network that directly maps corrupt images to clean ones; this obviates the slow inference procedures used by most generative models.

Neural networks have previously been explored for denoising natural images, mostly in the context of AWGN, e.g. Jain and Seung [65], and Zhang and Salari [150]. Algorithmically, the closest work to ours is that of Burger *et al.* [9], which applies a large neural network to a range of non-AWGN denoising tasks, such as salt-and-pepper noise and JPEG quantization artifacts. Although more challenging than AWGN, the corruption is still significantly easier than the highly variable dirt and rain drops that we address. Furthermore, our network has important architectural differences that are crucial for obtaining good performance on these tasks.

Removing localized corruption can be considered a form of blind inpainting, where the position of the corrupted regions is not given (unlike traditional inpainting [27]). Dong *et al.* [21] show how salt-and-pepper noise can be removed, but the approach does not extend to multi-pixel corruption. Recently, Xie *et al.* [143] showed how a neural network can perform blind inpainting, demonstrating the removal of text synthetically placed in an image. This work is close to ours, but the solid-color text has quite different statistics to natural images, thus is easier to remove than rain or dirt which vary greatly in appearance and can resemble legitimate image structures. Jancsary *et al.* [66] denoise images with a Gaussian conditional random field, constructed using decision trees on local regions of the input; however, they too consider only synthetic corruptions.

Several papers explore the removal of rain from images. Garg and Nayar [34] and Barnum *et al.* [4] address airborne rain. The former uses defocus, while the latter uses frequency-domain filtering. Both require video sequences rather than a single image, however. Roser and Geiger [104] detect raindrops in single images; although they do not demonstrate removal, their approach could be paired with a standard inpainting algorithm. As discussed above, our approach combines detection and inpainting.

Closely related to our application is Gu *et al.* [47], who show how lens dust and nearby occluders can be removed, but their method requires extensive calibration or a video sequence, as opposed to a single frame. Wilson *et al.* [142] and Zhou and Lin [151] demonstrate dirt and dust removal. The former removes defocused dust for a Mars Rover camera, while the latter removes sensor dust using multiple images and a physics model.

4.2 Approach

To restore an image from a corrupt input, we predict a clean output using a specialized form of convolutional neural network [77]. The same network architecture is used for all forms of corruption; however, a different network is trained for dirt and for rain. This allows the network to tailor its detection capabilities for each task.

4.2.1 Network Architecture

Given a noisy image x, our goal is to predict a clean image y that is close to the true clean image y^*. We accomplish this using a multilayer convolutional network, $y = F(x)$. The network F is composed of a series of layers F_l, each of which applies a linear convolution to its input, followed by an element-wise sigmoid (implemented using hyperbolic tangent).

Concretely, if the number of layers in the network is L, then

$$\begin{aligned} F_0(x) &= x \\ F_l(x) &= \tanh(W_l * F_{l-1}(x) + b_l), \quad l = 1, ..., L-1 \\ F(x) &= \frac{1}{m}(W_L * F_{L-1}(x) + b_L) \end{aligned}$$

Here, x is the RGB input image, of size $N \times M \times 3$. If n_l is the output dimension at layer l, then W_l applies n_l convolutions with kernels of size $p_l \times p_l \times n_{l-1}$, where p_l is the spatial support. b_l is a vector of size n_l containing the output bias (the same bias is used at each spatial location).

While the first and last layer kernels have a nontrivial spatial component, we restrict the middle layers ($2 \leq l \leq L-1$) to use $p_l = 1$, i.e. they apply a linear map at each spatial location. We also element-wise divide the final output by the overlap mask[1] m to account for different amounts of kernel overlap near the image boundary. The first layer uses a "valid" convolution, while the last layer uses a "full" (these are the same for the middle layers since their kernels have 1×1 support).

In our system, the input kernels' support is $p_1 = 16$, and the output support is $p_L = 8$. We use two hidden layers (i.e. $L = 3$), each with 512 units. As stated earlier, the middle layer kernel has support $p_2 = 1$. Thus, W_1 applies 512 kernels of size $16 \times 16 \times 3$, W_2 applies 512 kernels of size $1 \times 1 \times 512$, and W_3 applies 3 kernels of size $8 \times 8 \times 512$. Fig. 4.2 shows examples of weights learned for the rain data.

[1] $m = 1_K * 1_I$, where 1_K is a kernel of size $p_L \times p_L$ filled with ones, and 1_I is a 2D array of ones with as many pixels as the last layer input.

4.2.2 Training

We train the weights W_l and biases b_l by minimizing the mean squared error over a dataset $D = (x_i, y_i^*)$ of corresponding noisy and clean image pairs. The loss is

$$J(\theta) = \frac{1}{2|D|} \sum_{i \in D} ||F(x_i) - y_i^*||^2$$

where $\theta = (W_1, ..., W_L, b_1, ..., b_L)$ are the model parameters. The pairs in the dataset D are random 64×64 pixel subregions of training images with and without corruption (see Fig. 4.4 for samples). Because the input and output kernel sizes of our network differ, the network F produces a 56×56 pixel prediction y_i, which is compared against the middle 56×56 pixels of the true clean subimage y_i^*.

We minimize the loss using Stochastic Gradient Descent (SGD). The update for a single step at time t is

$$\theta^{t+1} \leftarrow \theta^t - \eta_t (F(x_i) - y_i^*)^T \frac{\partial}{\partial \theta} F(x_i)$$

where η_t is the learning rate hyper-parameter and i is a randomly drawn index from the training set. The gradient is further backpropagated through the network F.

We initialize the weights at all layers by randomly drawing from a normal distribution with mean 0 and standard deviation 0.001. The biases are initialized to 0. The learning rate is 0.001 with decay, so that $\eta_t = 0.001/(1 + 5t \cdot 10^{-7})$. We use no momentum or weight regularization.

4.2.3 Effect of Convolutional Architecture

A key improvement of our method over [9] is that we minimize the error of the final image prediction, whereas [9] minimizes the error only of individual patches. We found this difference to be crucial to obtain good performance on the corruption we address.

Since the middle layer convolution in our network has 1×1 spatial support, the network

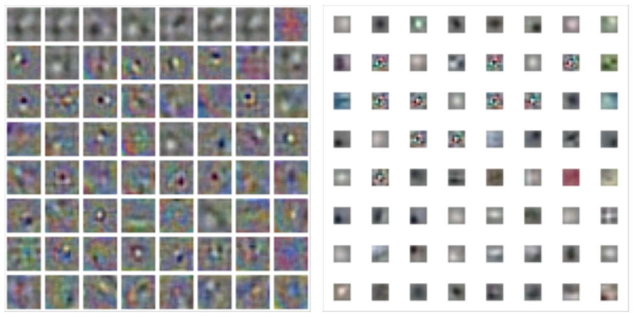


Figure 4.2: A subset of rain model network weights, sorted by l_2-norm. Left: first layer filters which act as detectors for the rain drops. Right: top layer filters used to reconstruct the clean patch.

can be viewed as first patchifying the input, applying a fully-connected neural network to each patch, and averaging the resulting output patches. More explicitly, we can split the input image x into stride-1 overlapping patches $\{x_p\} = \texttt{patchify}(x)$, and predict a corresponding clean patch $y_p = f(x_p)$ for each x_p using a fully-connected multilayer network f. We then form the predicted image $y = \texttt{depatchify}(\{y_p\})$ by taking the average of the patch predictions at pixels where they overlap. In this context, the convolutional network F can be expressed in terms of the patch-level network f as $F(x) = \texttt{depatchify}(\{f(x_p) : x_p \in \texttt{patchify}(x)\})$.

In contrast to [9], our method trains the full network F, *including patchification and depatchification*. This drives a decorrelation of the individual predictions, which helps both to remove occluders as well as reduce blur in the final output. To see this, consider two adjacent patches y_1 and y_2 with overlap regions y_{o1} and y_{o2}, and desired output y_o^*. If we were to train according to the individual predictions, the loss would minimize $(y_{o1} - y_o^*)^2 + (y_{o2} - y_o^*)^2$, the sum of their error. However, if we minimize the error of their average, the loss becomes $\left(\frac{y_{o1}+y_{o2}}{2} - y_o^*\right)^2 = \frac{1}{4}[(y_{o1} - y_o^*)^2 + (y_{o2} - y_o^*)^2 + 2(y_{o1} - y_o^*)(y_{o2} - y_o^*)]$. The new mixed term pushes the individual patch errors in opposing directions, encouraging them to decorrelate.

Fig. 4.3 depicts this for a real example. When trained at the patch level, as in the system described by [9], each prediction leaves the same residual trace of the noise, which their

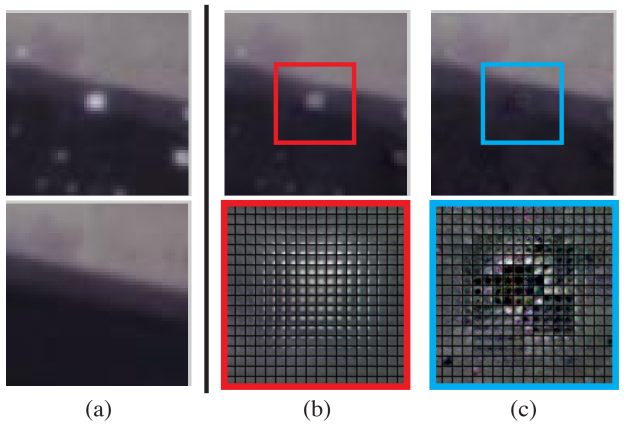


Figure 4.3: Denoising near a piece of noise. (a) shows a 64×64 image region with dirt occluders (top), and target ground truth clean image (bottom). (b) and (c) show the results obtained using non-convolutional and convolutionally trained networks, respectively. The top row shows the full output after averaging. The bottom row shows the signed error of each individual patch prediction for all 8×8 patches obtained using a sliding window in the boxed area, displayed as a montage. The errors from the convolutionally-trained network (c) are less correlated with one another compared to (b), and cancel to produce a better average.

average then maintains (b). When trained with our convolutional network, however, the predictions decorrelate where not perfect, and average to a better output (c).

4.2.4 Test-Time Evaluation

By restricting the middle layer kernels to have 1×1 spatial support, our method requires no synchronization until the final summation in the last layer convolution. This makes our method natural to parallelize, and it can easily be run in sections on large input images by adding the outputs from each section into a single image output buffer. Our Matlab GPU implementation is able to restore a 3888×2592 color image in 60s using a nVidia GTX 580, and a 1280×720 color image in 7s.

4.3 Training Data Collection

The network has 753,664 weights and 1,216 biases which need to be set during training. This requires a large number of training patches to avoid over-fitting. We now describe the procedures used to gather the corrupted/clean patch pairs[2] used to train each of the dirt and rain models.

4.3.1 Dirt

To train our network to remove dirt noise, we generated clean/noisy image pairs by synthesizing dirt on images. Similarly to [47], we also found that dirt noise was well-modeled by an opacity mask and additive component, which we extract from real dirt-on-glass panes in a lab setup. Once we have the masks, we generate noisy images according to

$$I' = p\alpha D + (1 - \alpha)I$$

Here, I and I' are the original clean and generated noisy image, respectively. α is a transparency mask the same size as the image, and D is the additive component of the dirt, also the same size as the image. p is a random perturbation vector in RGB space, and the factors $p\alpha D$ are multiplied together element-wise. p is drawn from a uniform distribution over (0.9, 1.1) for each of red, green and blue, then multiplied by another random number between 0 and 1 to vary brightness. These random perturbations are necessary to capture natural variation in the corruption and make the network robust to these changes.

To find α and αD, we took pictures of several slide-projected backgrounds, both with and without a dirt-on-glass pane placed in front of the camera. We then solved a linear least-squares system for α and αD at each pixel; further details are included in the supplementary material.

[2]The corrupt patches still have many unaffected pixels, thus even without clean/clean patch pairs in the training set, the network will still learn to preserve clean input regions.

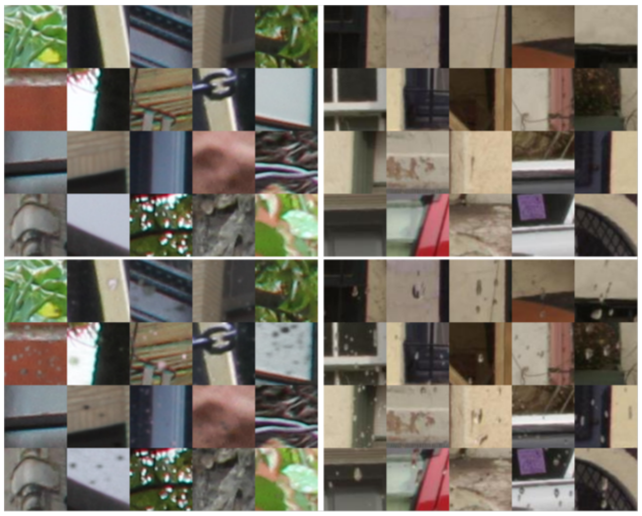


Figure 4.4: Examples of clean (top row) and corrupted (bottom row) patches used for training. The dirt (left column) was added synthetically, while the rain (right column) was obtained from real image pairs.

4.3.2 Water Droplets

Unlike the dirt, water droplets refract light around them and are not well described by a simple additive model. We considered using the more sophisticated rendering model of [46], but accurately simulating outdoor illumination made this inviable. Thus, instead of synthesizing the effects of water, we built a training set by taking photographs of multiple scenes with and without the corruption present. For corrupt images, we simulated the effect of rain on a window by spraying water on a pane of anti-reflective MgF_2-coated glass, taking care to produce drops that closely resemble real rain. To limit motion differences between clean and rainy shots, all scenes contained only static objects. Further details are provided in the supplementary material.

4.4 Baseline Methods

We compare our convolutional network against a nonconvolutional patch-level network similar to [9], as well as three baseline approaches: median filtering, bilateral filtering [130, 97], and BM3D [17]. In each case, we tuned the algorithm parameters to yield the

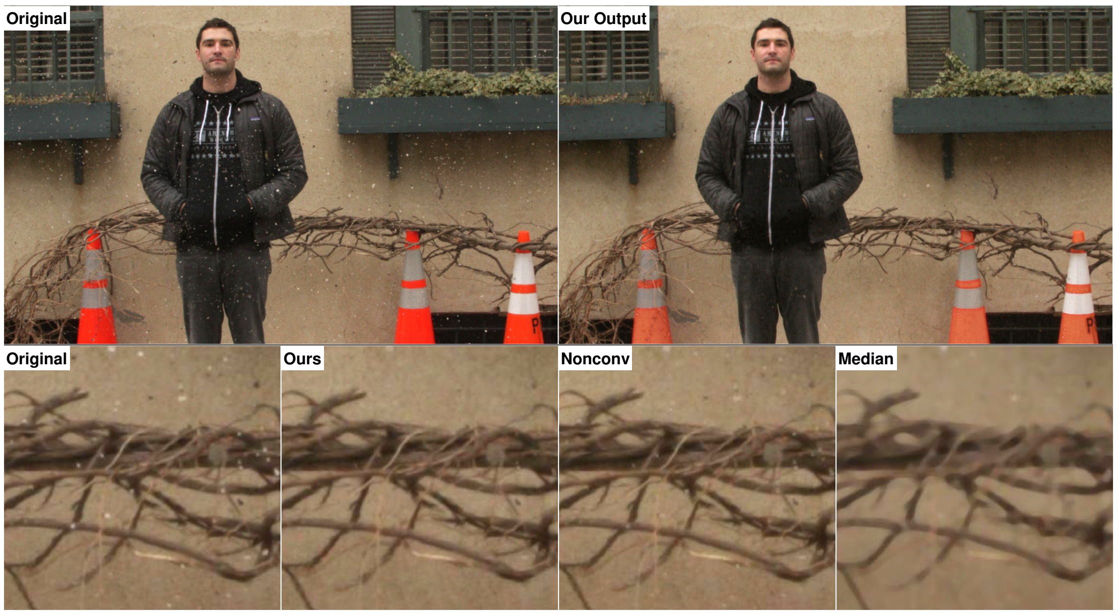


Figure 4.5: Example image containing dirt, and the restoration produced by our network. Note the detail preserved in high-frequency areas like the branches. The nonconvolutional network leaves behind much of the noise, while the median filter causes substantial blurring.

best qualitative performance in terms of visibly reducing noise while keeping clean parts of the image intact. On the dirt images, we used an 8×8 window for the median filter, parameters $\sigma_s = 3$ and $\sigma_r = 0.3$ for the bilateral filter, and $\sigma = 0.15$ for BM3D. For the rain images, we used similar parameters, but adjusted for the fact that the images were downsampled by half: 5×5 for the median filter, $\sigma_s = 2$ and $\sigma_r = 0.3$ for the bilateral filter, and $\sigma = 0.15$ for BM3D.

4.5 Experiments

4.5.1 Dirt

We tested dirt removal by running our network on pictures of various scenes taken behind dirt-on-glass panes. Both the scenes and glass panes were not present in the training set, ensuring that the network did not simply memorize and match exact patterns. We tested restoration of both real and synthetic corruption. Although the training set was composed entirely of synthetic dirt, it was representative enough for the network to perform well in both cases.

The network was trained using 5.8 million examples of 64×64 image patches with synthetic dirt, paired with ground truth clean patches. We trained only on examples where the variance of the clean 64×64 patch was at least 0.001, and also required that at least 1 pixel in the patch had a dirt α-mask value of at least 0.03. To compare to [9], we trained a non-convolutional patch-based network with patch sizes corresponding to our convolution kernel sizes, using 20 million 16×16 patches.

Synthetic Dirt Results

We first measure quantitative performance using synthetic dirt. The results are shown in Table 4.1. Here, we generated test examples using images and dirt masks held out from the training set, using the process described in Section 4.3.1. Our convolutional network substantially outperforms its patch-based counterpart. Both neural networks are much better than the three baselines, which do not make use of the structure in the corruption that the networks learn.

We also applied our network to two types of artificial noise absent from the training set: synthetic "snow" made from small white line segments, and "scratches" of random cubic splines. An example region is shown in Fig. 4.6. In contrast to the gain of +6.50 dB for dirt, the network leaves these corruptions largely intact, producing near-zero PSNR gains of -0.10 and +0.30 dB, respectively, over the same set of images. This demonstrates that the network learns to remove dirt specifically.

Dirt Results

Fig. 4.5 shows a real test image along with our output and the output of the patch-based network and median filter. Because of illumination changes and movement in the scenes,

PSNR	Input	Ours	Nonconv	Median	Bilateral	BM3D
Mean	28.93	**35.43**	34.52	31.47	29.97	29.99
Std.Dev.	0.93	**1.24**	1.04	1.45	1.18	0.96
Gain	-	**6.50**	5.59	2.53	1.04	1.06

Table 4.1: PSNR for our convolutional neural network, nonconvolutional patch-based network, and baselines on a synthetically generated test set of 16 images (8 scenes with 2 different dirt masks). Our approach significantly outperforms the other methods.

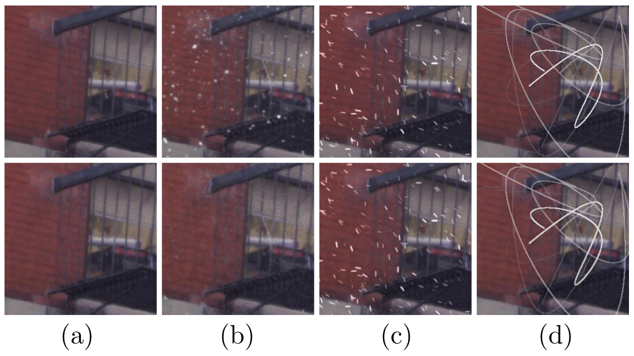


Figure 4.6: Our dirt-removal network applied to an image with (a) no corruption, (b) synthetic dirt, (c) artificial “snow” and (d) random “scratches.” Because the network was trained to remove dirt, it successfully restores (b) while leaving the corruptions in (c,d) largely untouched. Top: Original images. Bottom: Output.

we were not able to capture ground truth images for quantitative evaluation. Our method is able to remove most of the corruption while retaining details in the image, particularly around the branches and shutters. The non-convolutional network leaves many pieces of dirt behind, while the median filter loses much detail present in the original. Note also that the neural networks leave already-clean parts of the image mostly untouched.

Two common causes of failure of our model are large corruption, and very oddly-shaped or unusually colored corruption. Our 16×16 input kernel support limits the size of corruption recognizable by the system, leading to the former. The latter is caused by a lack of generalization: although we trained the network to be robust to shape and color by supplying it a range of variations, it will not recognize cases too far from those seen in training. Another interesting failure of our method appears in the bright orange cones in Fig. 4.5, which our method reduces in intensity — this is due to the fact that the training dataset did not contain any examples of such fluorescent objects. More examples are provided in the supplementary material.

4.5.2 Rain

We ran the rain removal network on two sets of test data: (*i*) pictures of scenes taken through a pane of glass on which we sprayed water to simulate rain, and (*ii*) pictures of scenes taken while it was actually raining, from behind an initially clean glass pane. Both sets were composed of real-world outdoor scenes not in the training set.

We trained the network using 6.5 million examples of 64×64 image patch pairs, captured as described in Section 4.3.2. Similarly to the dirt case, we used a variance threshold of 0.001 on the clean images and required each training pair to have at least 1 pixel difference over 0.1.

Water Droplets Results

Examples of our network removing sprayed-on water is shown in Fig. 4.7. As was the case for the dirt images, we were not able to capture accurate ground truth due to illumination changes and subject motion. Since we also do not have synthetic water examples, we analyze our method in this mode only qualitatively.

As before, our network is able to remove most of the water droplets, while preserving finer details and edges reasonably well. The non-convolutional network leaves behind additional droplets, e.g. by the subject's face in the top image; it performs somewhat better in the bottom image, but blurs the subject's hand. The median filter must blur the image substantially before visibly reducing the corruption. However, the neural networks mistake the boltheads on the bench for raindrops, and remove them.

Despite the fact that our network was trained on static scenes to limit object motion between clean/noisy pairs, it still preserves animate parts of the images well: The face and body of the subject are reproduced with few visible artifacts, as are grass, leaves

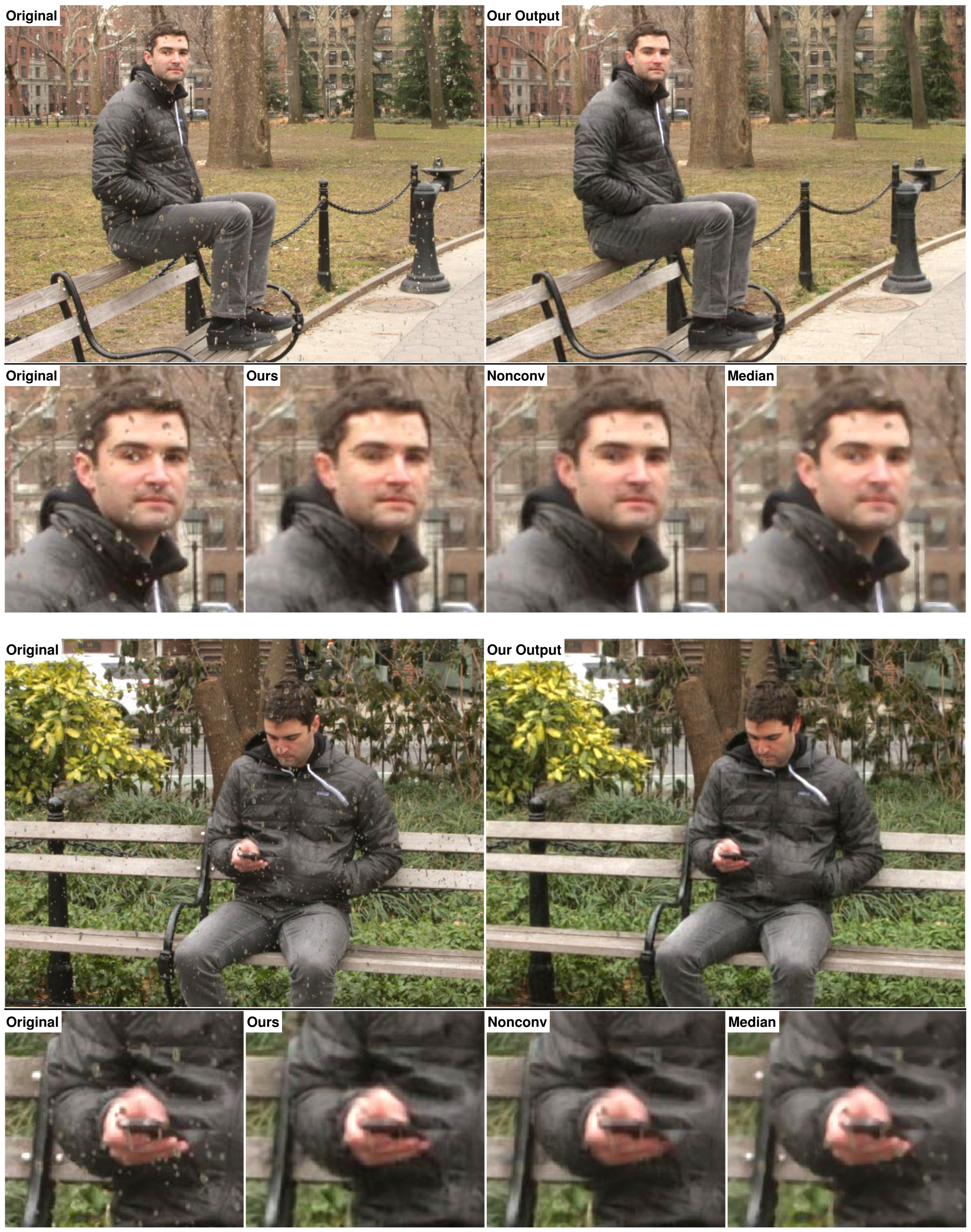


Figure 4.7: Our network removes most of the water while retaining image details; the non-convolutional network leaves more droplets behind, particularly in the top image, and blurs the subject's fingers in the bottom image. The median filter blurs many details, but still cannot remove much of the noise.

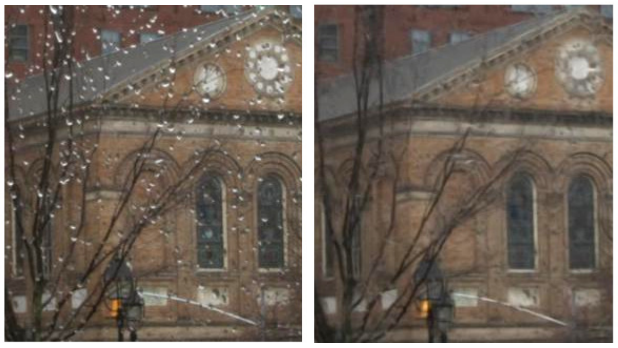


Figure 4.8: Shot from the rain video sequence (see supplementary video), along with the output of our network. Note each frame is processed independently, without using any temporal information or background subtraction.

and branches (which move from wind). Thus the network can be applied to many scenes substantially different from those seen in training.

Real Rain Results

A picture taken using actual rain is shown in Fig. 4.8. We include more pictures of this time series as well as a video in the supplementary material. Each frame of the video was presented to our algorithm independently; no temporal filtering was used. To capture the sequence, we set a clean glass pane on a tripod and allowed rain to fall onto it, taking pictures at 20s intervals. The camera was placed 0.5m behind the glass, and was focused on the scene behind.

Even though our network was trained using sprayed-on water, it was still able to remove much of the actual rain. The largest failures appear towards the end of the sequence, when the rain on the glass is very heavy and starts to agglomerate, forming droplets larger than our network can handle. Although this is a limitation of the current approach, we hope to address such cases in future work.

Lastly, in addition to pictures captured with a DSLR, in Fig. 4.9 we apply our network to a picture taken using a smartphone on a train. While the scene and reflections are preserved, raindrops on the window are removed, though a few small artifacts do remain.

This demonstrates that our model is able to restore images taken by a variety of camera types.

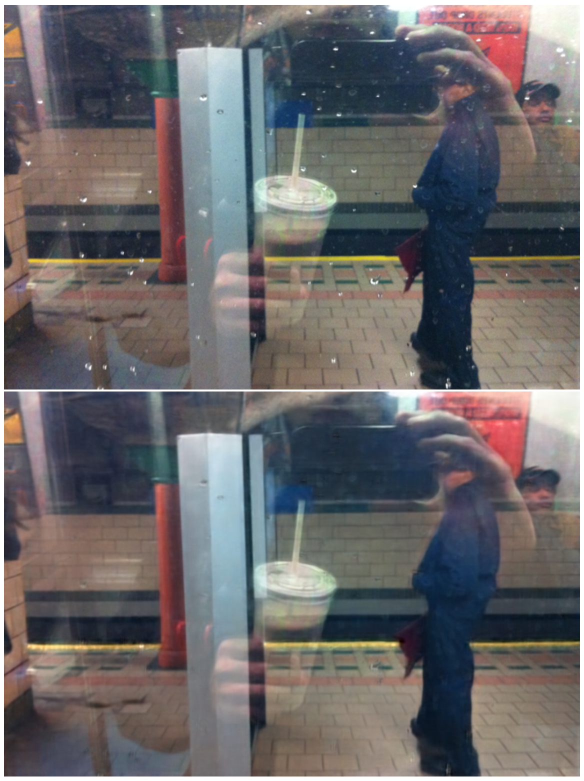


Figure 4.9: Top: Smartphone shot through a rainy window on a train. Bottom: Output of our algorithm.

4.6 Discussion

In this chapter we introduced a method for removing rain or dirt artifacts from a single image. Although the problem appears underconstrained, the artifacts have a distinctive appearance which we are able to learn with a specialized convolutional network and a carefully constructed training set. Results on real test examples show most artifacts being removed without undue loss of detail, unlike previous approaches such as median or bilateral filtering. Using a convolutional network accounts for the error in the final image prediction, providing a significant performance gain over the corresponding patch-based network.

The quality of the results does however depend on the statistics of test cases being similar to those of the training set. In cases where this does not hold, we see significant artifacts in the output. This can be alleviated by expanding the diversity and size of the training set. A second issue is that the corruption cannot be much larger than the training patches. This means the input image may need to be downsampled, e.g. as in the rain application, leading to a loss of resolution relative to the original.

Although we have only considered day-time outdoor shots, the approach could be extended to other settings such as indoor or night-time, given suitable training data. It could also be extended to other problem domains such as scratch removal or color shift correction. Our algorithm also provides the underlying technology for a number of potential applications such as a digital car windshield to aid driving in adverse weather conditions, or enhancement of footage from security or automotive cameras in exposed locations.

While a local field of view is sufficient to detect and remove compact noise structures, a more global view is needed for many other tasks in order to incorporate context and cues from the larger image area. We now turn to a more challenging task that requires both scales, predicting depth from a single image.

Chapter 5

Depth Map Prediction from a Single Image using a Multi-Scale Deep Network

The work presented in this chapter appeared in NIPS 2014 [25], and was a collaboration with Christian Puhrsch and Rob Fergus.

5.1 Introduction

Predicting depth is an essential component in understanding the 3D geometry of a scene. In this chapter we develop a convolutional network that integrates both global and local views of an input image together to generate a depth map; this map contains the depth from the camera for each pixel of the input. While for stereo images local correspondence suffices for estimation, finding depth relations from a *single image* is less straightforward, and requires integrating information from both global and local scales.

Depth relations help provide richer representations of objects and their environment compared to using RGB exclusively; including them often leads to improvements in existing recognition tasks [115], and can enable many further applications such as 3D modeling [108, 62], physics and support models [115], robotics [50, 88], and potentially reasoning about occlusions.

While there is much prior work on estimating depth based on stereo images or motion [110], there has been relatively little on estimating depth from a *single* image. Yet the monocular case often arises in practice: Potential applications include better understandings of the many images distributed on the web and social media outlets, real estate listings, and shopping sites. These include many examples of both indoor and outdoor scenes.

There are likely several reasons why the monocular case has not yet been tackled to the same degree as the stereo one. Provided accurate image correspondences, depth can be recovered deterministically in the stereo case [53]. Thus, stereo depth estimation can be reduced to developing robust image point correspondences — which can often be found using local appearance features. By contrast, estimating depth from a single image requires the use of monocular depth cues such as line angles and perspective, object sizes, image position, and atmospheric effects. Furthermore, a global view of the scene may be needed to relate these effectively, whereas local disparity is sufficient for stereo.

Moreover, the task is inherently ambiguous, and a technically ill-posed problem: Given an image, an infinite number of possible world scenes may have produced it. Of course, most of these are physically implausible for real-world spaces, and thus the depth may still be predicted with considerable accuracy. At least one major ambiguity remains, though: the global scale. Although extreme cases (such as a normal room versus a dollhouse) do not exist in the data, moderate variations in room and furniture sizes are present. We address this using a *scale-invariant error* in addition to more common scale-dependent errors. This focuses attention on the spatial relations within a scene rather than general scale, and is particularly apt for applications such as 3D modeling, where the model is often rescaled during postprocessing.

In this chapter we present a new approach for estimating depth from a single image. We *directly regress on the depth* using a neural network with two components: one that first estimates the global structure of the scene, then a second that refines it using local information. The network is trained using a loss that explicitly accounts for depth relations between pixel locations, in addition to pointwise error. Our system achieves state-of-the art estimation rates on NYU Depth and KITTI, as well as improved qualitative outputs.

5.2 Related Work

Directly related to our work are several approaches that estimate depth from a single image. Saxena *et al.* [107] predict depth from a set of image features using linear regression and a MRF, and later extend their work into the Make3D [108] system for 3D model generation. However, the system relies on horizontal alignment of images, and suffers in less controlled settings. Hoiem *et al.* [62] do not predict depth explicitly, but instead categorize image regions into geometric structures (ground, sky, vertical), which they use to compose a simple 3D model of the scene.

More recently, Ladicky *et al.* [74] show how to integrate semantic object labels with monocular depth features to improve performance; however, they rely on handcrafted features and use superpixels to segment the image. Karsch *et al.* [68] use a kNN transfer mechanism based on SIFT Flow [81] to estimate depths of static backgrounds from single images, which they augment with motion information to better estimate moving foreground subjects in videos. This can achieve better alignment, but requires the entire dataset to be available at runtime and performs expensive alignment procedures. By contrast, our method learns an easier-to-store set of network parameters, and can be applied to images in real-time.

More broadly, stereo depth estimation has been extensively investigated. Scharstein *et al.* [110] provide a survey and evaluation of many methods for 2-frame stereo correspondence, organized by matching, aggregation and optimization techniques. In a creative application of multiview stereo, Snavely *et al.* [118] match across views of many uncalibrated consumer photographs of the same scene to create accurate 3D reconstructions of common landmarks.

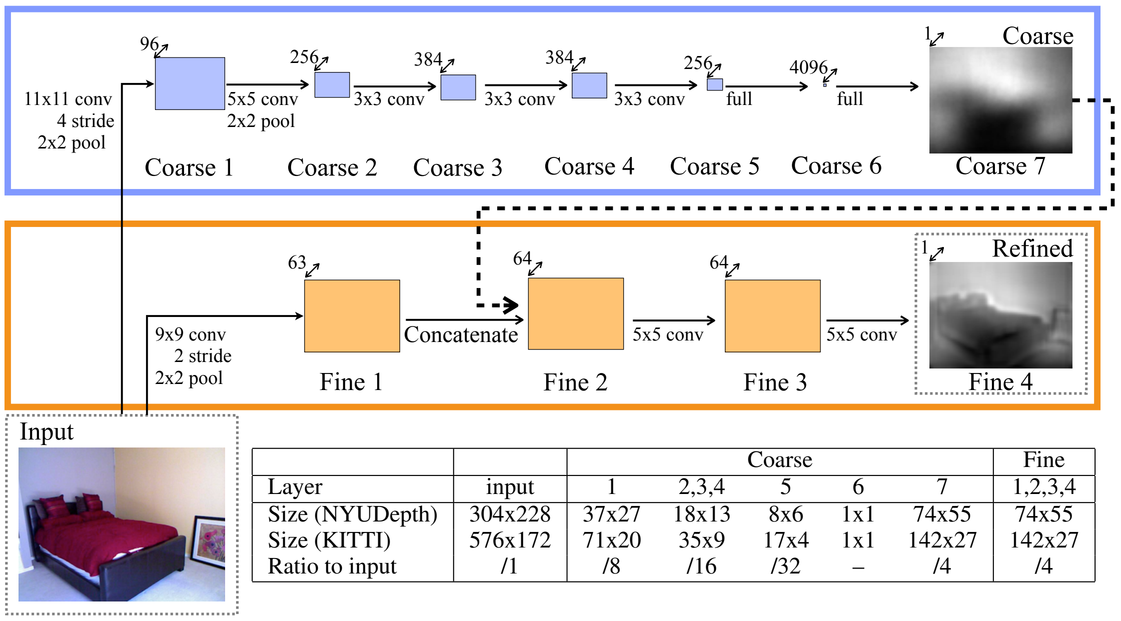
Machine learning techniques have also been applied in the stereo case, often obtaining better results while relaxing the need for careful camera alignment [70, 87, 144, 117]. Most relevant to this work is Konda *et al.* [70], who train a factored autoencoder on image patches to predict depth from stereo sequences; however, this relies on the local displacements provided by stereo.

There are also several hardware-based solutions for single-image depth estimation. Levin *et al.* [78] perform depth from defocus using a modified camera aperture, while the Kinect and Kinect v2 use active stereo and time-of-flight to capture depth. Our method makes indirect use of such sensors to provide ground truth depth targets during training; however, at test time our system is purely software-based, predicting depth from RGB images.

5.3 Approach

5.3.1 Model Architecture

Our network is made of two component stacks, shown in Fig. 5.1. A coarse-scale network first predicts the depth of the scene at a global level. This is then refined within local regions by a fine-scale network. Both stacks are applied to the original input, but in addition, the coarse network's output is passed to the fine network as additional first-layer image features. In this way, the local network can edit the global prediction to incorporate finer-scale details.



		Coarse					Fine
Layer	input	1	2,3,4	5	6	7	1,2,3,4
Size (NYUDepth)	304x228	37x27	18x13	8x6	1x1	74x55	74x55
Size (KITTI)	576x172	71x20	35x9	17x4	1x1	142x27	142x27
Ratio to input	/1	/8	/16	/32	–	/4	/4

Figure 5.1: Model architecture.

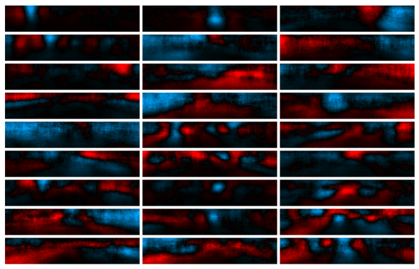
Global Coarse-Scale Network

The task of the coarse-scale network is to predict the overall depth map structure using a global view of the scene. The upper layers of this network are fully connected, and thus contain the entire image in their field of view. Similarly, the lower and middle layers are designed to combine information from different parts of the image through max-pooling operations to a small spatial dimension. In so doing, the network is able to integrate a global understanding of the full scene to predict the depth. Such an understanding is needed in the single-image case to make effective use of cues such as vanishing points, object locations, and room alignment. A local view (as is commonly used for stereo matching) is insufficient to notice important features such as these.

As illustrated in Fig. 5.1, the global, coarse-scale network contains five feature extraction layers of convolution and max-pooling, followed by two fully connected layers. The input, feature map and output sizes are also given in Fig. 5.1. The final output is at 1/4-resolution compared to the input (which is itself downsampled from the original dataset by a factor of 2), and corresponds to a center crop containing most of the input (as we describe later, we lose a small border area due to the first layer of the fine-scale network and image transformations).

Note that the spatial dimension of the output is larger than that of the topmost convolutional feature map. Rather than limiting the output to the feature map size and relying on hardcoded upsampling before passing the prediction to the fine network, we allow the top full layer to learn templates over the larger area (74x55 for NYU Depth). These are expected to be blurry, but will be better than the upsampled output of a 8x6 prediction (the top feature map size); essentially, we allow the network to learn its own upsampling based on the features. Sample output weights are shown in Fig. 5.2

All hidden layers use rectified linear units for activations, with the exception of the coarse output layer 7, which is linear. Dropout is applied to the fully-connected hidden layer 6. The convolutional layers (1-5) of the coarse-scale network are pretrained on the ImageNet classification task [19] — while developing the model, we found pretraining on ImageNet worked better than initializing randomly, although the difference was not very large[1].


(a)

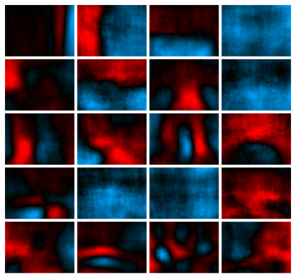

(b)

Figure 5.2: Weight vectors from layer Coarse 7 (coarse output), for (*a*) KITTI and (*b*) NYUDepth. Red is positive (farther) and blue is negative (closer); black is zero. Weights are selected uniformly and shown in descending order by l_2 norm. KITTI weights often show changes in depth on either side of the road. NYUDepth weights often show wall positions and doorways.

[1]When pretraining, we stack two fully connected layers with 4096 - 4096 - 1000 output units each, with dropout applied to the two hidden layers, as in [73]. We train the network using random 224x224 crops from the center 256x256 region of each training image, rescaled so the shortest side has length 256. This model achieves a top-5 error rate of 18.1% on the ILSVRC2012 validation set, voting with 2 flips and 5 translations per image.

Local Fine-Scale Network

After taking a global perspective to predict the coarse depth map, we make local refinements using a second, fine-scale network. The task of this component is to edit the coarse prediction it receives to align with local details such as object and wall edges. The fine-scale network stack consists of convolutional layers only, along with one pooling stage for the first layer edge features.

While the coarse network sees the entire scene, the field of view of an output unit in the fine network is 45x45 pixels of input. The convolutional layers are applied across feature maps at the target output size, allowing a relatively high-resolution output at 1/4 the input scale.

More concretely, the coarse output is fed in as an additional low-level feature map. By design, the coarse prediction is the same spatial size as the output of the first fine-scale layer (after pooling), and we concatenate the two together (Fine 2 in Fig. 5.1). Subsequent layers maintain this size using zero-padded convolutions.

All hidden units use rectified linear activations. The last convolutional layer is linear, as it predicts the target depth. We train the coarse network first against the ground-truth targets, then train the fine-scale network keeping the coarse-scale output fixed (*i.e.* when training the fine network, we do not backpropagate through the coarse one).

5.3.2 Scale-Invariant Error

The global scale of a scene is a fundamental ambiguity in depth prediction. Indeed, much of the error accrued using current elementwise metrics may be explained simply by how well the mean depth is predicted. For example, Make3D trained on NYUDepth obtains 0.41 error using RMSE in log space (see Table 5.1). However, using an oracle to substitute the mean log depth of each prediction with the mean from the corresponding ground truth reduces the error to 0.33, a 20% relative improvement. Likewise, for our system, these error rates are 0.28 and 0.22, respectively. Thus, just finding the average scale of the scene accounts for a large fraction of the total error.

Motivated by this, we use a scale-invariant error to measure the relationships between points in the scene, irrespective of the absolute global scale. For a predicted depth map y and ground truth y^*, each with n pixels indexed by i, we define the *scale-invariant mean squared error* (in log space) as

$$D(y, y^*) = \frac{1}{n} \sum_{i=1}^{n} (\log y_i - \log y_i^* + \alpha(y, y^*))^2, \tag{5.1}$$

where $\alpha(y, y^*) = \frac{1}{n} \sum_i (\log y_i^* - \log y_i)$ is the value of α that minimizes the error for a given (y, y^*). For any prediction y, e^α is the scale that best aligns it to the ground truth. All scalar multiples of y have the same error, hence the scale invariance.

Two additional ways to view this metric are provided by the following equivalent forms. Setting $d_i = \log y_i - \log y_i^*$ to be the difference between the prediction and ground truth at pixel i, we have

$$D(y, y^*) = \frac{1}{n^2} \sum_{i,j} \big((\log y_i - \log y_j) - (\log y_i^* - \log y_j^*)\big)^2 \tag{5.2}$$

$$= \frac{1}{n} \sum_i d_i^2 - \frac{1}{n^2} \sum_{i,j} d_i d_j = \frac{1}{n} \sum_i d_i^2 - \frac{1}{n^2} \left(\sum_i d_i \right)^2 \tag{5.3}$$

Eqn. 5.2 expresses the error by comparing relationships *between pairs* of pixels i, j in the output: to have low error, each pair of pixels in the prediction must differ in depth by an amount similar to that of the corresponding pair in the ground truth. Eqn. 5.3 relates the metric to the original l_2 error, but with an additional term, $-\frac{1}{n^2} \sum_{ij} d_i d_j$, that credits mistakes if they are in the same direction and penalizes them if they oppose.

Thus, an imperfect prediction will have lower error when its mistakes are consistent with one another. The last part of Eqn. 5.3 rewrites this as a linear-time computation.

In addition to the scale-invariant error, we also measure the performance of our method according to several error metrics have been proposed in prior works, as described in Section 5.4.

5.3.3 Training Loss

In addition to performance evaluation, we also tried using the scale-invariant error as a training loss. Inspired by Eqn. 5.3, we set the per-sample training loss to

$$L(y, y^*) \quad = \quad \frac{1}{n}\sum_i d_i^2 - \frac{\lambda}{n^2}\left(\sum_i d_i\right)^2 \tag{5.4}$$

where $d_i = \log y_i - \log y_i^*$ and $\lambda \in [0, 1]$. Note the output of the network is $\log y$; that is, the final linear layer predicts the log depth. Setting $\lambda = 0$ reduces to elementwise l_2, while $\lambda = 1$ is the scale-invariant error exactly. We use the average of these, *i.e.* $\lambda = 0.5$, finding that this produces good absolute-scale predictions while slightly improving qualitative output.

During training, most of the target depth maps will have some missing values, particularly near object boundaries, windows and specular surfaces. We deal with these simply by masking them out and evaluating the loss only on valid points, *i.e.* we replace n in Eqn. 5.4 with the number of pixels that have a target depth, and perform the sums excluding pixels i that have no depth value.

5.3.4 Data Augmentation

We augment the training data with random online transformations (values shown for NYUDepth; for KITTI, $s \in [1, 1.2]$, and rotations are not performed since images are horizontal from the camera mount):

- *Scale*: Input and target images are scaled by $s \in [1, 1.5]$, and the depths are divided by s.
- *Rotation*: Input and target are rotated by $r \in [-5, 5]$ degrees.
- *Translation*: Input and target are randomly cropped to the sizes indicated in Fig. 5.1.
- *Color*: Input values are multiplied globally by a random RGB value $c \in [0.8, 1.2]^3$.
- *Flips*: Input and target are horizontally flipped with 0.5 probability.

Note that image scaling and translation do not preserve the world-space geometry of the scene. This is easily corrected in the case of scaling by dividing the depth values by the scale s (making the image s times larger effectively moves the camera s times closer). Although translations are not easily fixed (they effectively change the camera to be incompatible with the depth values), we found that the extra data they provided benefited the network even though the scenes they represent were slightly warped. The other transforms, flips and in-plane rotation, are geometry-preserving. At test time, we use a single center crop at scale 1.0 with no rotation or color transforms.

5.4 Experiments

We train our model on the raw versions both NYU Depth v2 [115] and KITTI [36]. The raw distributions contain many additional images collected from the same scenes as in the more commonly used small distributions, but with no preprocessing; in particular, points for which there is no depth value are left unfilled. However, our model's natural ability to handle such gaps as well as its demand for large training sets make these fitting sources of data.

5.4.1 NYU Depth

The NYU Depth dataset [115] is composed of 464 indoor scenes, taken as video sequences using a Microsoft Kinect camera. We use the official train/test split, using 249 scenes for training and 215 for testing, and construct our training set using the raw data for these scenes. RGB inputs are downsampled by half, from 640x480 to 320x240. Because the depth and RGB cameras operate at different variable frame rates, we associate each depth image with its closest RGB image in time, and throw away frames where one RGB image is associated with more than one depth (such a one-to-many mapping is not predictable). We use the camera projections provided with the dataset to align RGB and depth pairs; pixels with no depth value are left missing and are masked out. To remove many invalid regions caused by windows, open doorways and specular surfaces we also mask out depths equal to the minimum or maximum recorded for each image.

The training set has 120K unique images, which we shuffle into a list of 220K after evening the scene distribution (1200 per scene). We test on the 694-image NYU Depth v2 test set (with filled-in depth values). We train coarse network for 2M samples using SGD with batches of size 32. We then hold it fixed and train the fine network for 1.5M samples (given outputs from the already-trained coarse one). Learning rates are: 0.001 for coarse convolutional layers 1-5, 0.1 for coarse full layers 6 and 7, 0.001 for fine layers 1 and 3, and 0.01 for fine layer 2. These ratios were found by trial-and-error on a validation set (folded back into the training set for our final evaluations), and the global scale of all the rates was tuned to a factor of 5. Momentum was 0.9.

5.4.2 KITTI

The KITTI dataset [36] is composed of several outdoor scenes captured while driving with car-mounted cameras and depth sensor. We use 56 scenes from the "city," "residential," and "road" categories of the raw data. These are split into 28 for training and 28 for testing. The RGB images are originally 1224x368, and downsampled by half to form the network inputs.

The depth for this dataset is sampled at irregularly spaced points, captured at different times using a rotating LIDAR scanner. When constructing the ground truth depths for training, there may be conflicting values; since the RGB cameras shoot when the scanner points forward, we resolve conflicts at each pixel by choosing the depth recorded closest to the RGB capture time. Depth is only provided within the bottom part of the RGB image, however we feed the entire image into our model to provide additional context to the global coarse-scale network (the fine network sees the bottom crop corresponding to the target area).

The training set has 800 images per scene. We exclude shots where the car is stationary (acceleration below a threshold) to avoid duplicates. Both left and right RGB cameras are used, but are treated as unassociated shots. The training set has 20K unique images, which we shuffle into a list of 40K (including duplicates) after evening the scene distribution. We train the coarse model first for 1.5M samples, then the fine model for 1M. Learning rates are the same as for NYU Depth.

5.4.3 Baselines and Comparisons

We compare our method against Make3D trained on the same datasets, as well as the published results of other current methods [74, 68]. As an additional reference, we also compare to the mean depth image computed across the training set. We trained Make3D on KITTI using a subset of 700 images (25 per scene), as the system was unable to scale beyond this size. Depth targets were filled in using the colorization routine in the NYUDepth development kit. For NYUDepth, we used the common distribution training set of 795 images. We evaluate each method using several errors from prior works, as well as our scale-invariant metric:

- Threshold: % of y_i s.t. $\max(\frac{y_i}{y_i^*}, \frac{y_i^*}{y_i}) = \delta < thr$
- Abs Relative difference: $\frac{1}{|T|}\sum_{y\in T} |y - y^*|/y^*$
- Squared Relative difference: $\frac{1}{|T|}\sum_{y\in T} ||y - y^*||^2/y^*$
- RMSE (linear): $\sqrt{\frac{1}{|T|}\sum_{y\in T} ||y_i - y_i^*||^2}$

- RMSE (log): $\sqrt{\frac{1}{|T|}\sum_{y \in T} || \log y_i - \log y_i^* ||^2}$
- RMSE (log, scale-invariant): The error Eqn. 5.1

Note that the predictions from Make3D and our network correspond to slightly different center crops of the input. We compare them on the intersection of their regions, and upsample predictions to the full original input resolution using nearest-neighbor. Upsampling negligibly affects performance compared to downsampling the ground truth and evaluating at the output resolution. [2]

5.5 Results

5.5.1 NYU Depth

Results for NYU Depth dataset are provided in Table 5.1. As explained in Section 5.4.3, we compare against the data mean and Make3D as baselines, as well as Karsch *et al.* [68] and Ladicky *et al.* [74]. (Ladicky *et al.* uses a joint model which is trained using both depth and semantic labels). Our system achieves the best performance on all metrics, obtaining an average 35% relative gain compared to the runner-up. Note that our system is trained using the raw dataset, which contains many more example instances than the data used by other approaches, and is able to effectively leverage it to learn relevant features and their associations.

This dataset breaks many assumptions made by Make3D, particularly horizontal alignment of the ground plane; as a result, Make3D has relatively poor performance in this task. Importantly, our method improves over it on both scale-dependent and scale-invariant metrics, showing that our system is able to predict better relations as well as better means.

[2]On NYUDepth, log RMSE is 0.285 vs 0.286 for upsampling and downsampling, respectively, and scale-invariant RMSE is 0.219 vs 0.221. The intersection is 86% of the network region and 100% of Make3D for NYUDepth, and 100% of the network and 82% of Make3D for KITTI.

Qualitative results are shown on the left side of Fig. 5.4, sorted top-to-bottom by scale-invariant MSE. Although the fine-scale network does not improve in the error measurements, its effect is clearly visible in the depth maps — surface boundaries have sharper transitions, aligning to local details. However, some texture edges are sometimes also included. Fig. 5.3 compares Make3D as well as outputs from our network trained with losses using $\lambda = 0$ and $\lambda = 0.5$. While we did not observe numeric gains using $\lambda = 0.5$ over $\lambda = 0$, it did produce slight qualitative improvements in the more detailed outputs.

	Mean	Make3D	Ladicky*&al*	Karsch*&al*	Coarse	Coarse+Fine	
threshold $\delta < 1.25$	0.418	0.447	0.542	–	**0.618**	0.611	higher
threshold $\delta < 1.25^2$	0.711	0.745	0.829	–	**0.891**	0.887	is
threshold $\delta < 1.25^3$	0.874	0.897	0.940	–	0.969	**0.971**	better
abs relative diff.	0.408	0.349	–	0.350	0.228	**0.215**	
sqr relative diff.	0.581	0.492	–	–	0.223	**0.212**	lower
RMSE (linear)	1.244	1.214	–	1.2	**0.871**	0.907	is
RMSE (log)	0.430	0.409	–	–	**0.283**	0.285	better
RMSE (log,sc.inv.)	0.304	0.325	–	–	0.221	**0.219**	

Table 5.1: Comparison on the NYUDepth dataset

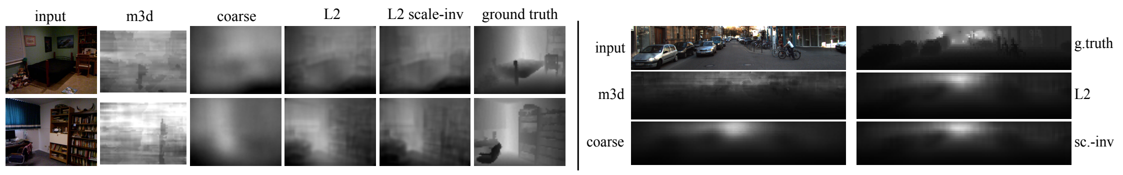


Figure 5.3: Qualitative comparison of Make3D, our method trained with l_2 loss ($\lambda = 0$), and our method trained with both l_2 and scale-invariant loss ($\lambda = 0.5$).

5.5.2 KITTI

We next examine results on the KITTI driving dataset. Here, the Make3D baseline is well-suited to the dataset, being composed of horizontally aligned images, and achieves relatively good results. Still, our method improves over it on all metrics, by an average 31% relative gain. Just as importantly, there is a 25% gain in both the scale-dependent and scale-invariant RMSE errors, showing there is substantial improvement in the predicted structure. Again, the fine-scale network does not improve much over the coarse one in the error metrics, but differences between the two can be seen in the qualitative outputs.

	Mean	Make3D	Coarse	Coarse + Fine	
threshold $\delta < 1.25$	0.556	0.601	0.679	**0.692**	higher
threshold $\delta < 1.25^2$	0.752	0.820	0.897	**0.899**	is
threshold $\delta < 1.25^3$	0.870	0.926	**0.967**	**0.967**	better
abs relative difference	0.412	0.280	0.194	**0.190**	
sqr relative difference	5.712	3.012	1.531	**1.515**	lower
RMSE (linear)	9.635	8.734	7.216	**7.156**	is
RMSE (log)	0.444	0.361	0.273	**0.270**	better
RMSE (log, scale inv.)	0.359	0.327	0.248	**0.246**	

Table 5.2: Comparison on the KITTI dataset.

The right side of Fig. 5.4 shows examples of predictions, again sorted by error. The fine-scale network produces sharper transitions here as well, particularly near the road edge. However, the changes are somewhat limited. This is likely caused by uncorrected alignment issues between the depth map and input in the training data, due to the rotating scanner setup. This dissociates edges from their true position, causing the network to average over their more random placements. Fig. 5.3 shows Make3D performing much better on this data, as expected, while using the scale-invariant error as a loss seems to have little effect in this case.

5.6 Discussion

Predicting depth estimates from a single image is a challenging task. Yet by combining information from both global and local views, it can be performed reasonably well. Our system accomplishes this through the use of two deep networks, one that estimates the global depth structure, and another that refines it locally at finer resolution. We achieve a new state-of-the-art on this task for NYU Depth and KITTI datasets, having effectively leveraged the full raw data distributions.

In the next chapter, we extend our method to also predict surface normals and semantic labels, thus providing even richer geometric outputs and object class information. We also apply successively finer-scaled networks to increase the output map resolution, providing sharper alignments and more detailed detections.

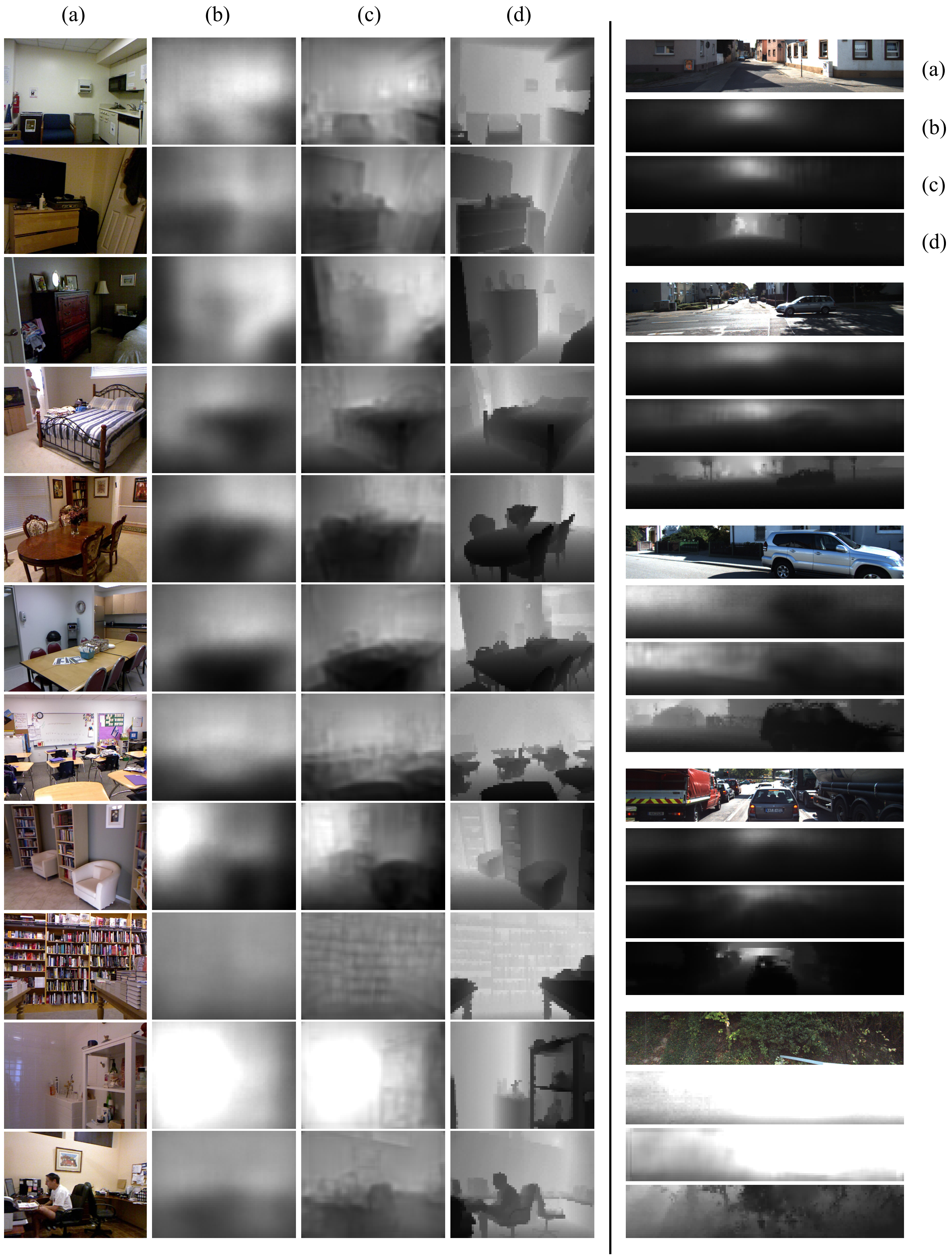


Figure 5.4: Example predictions from our algorithm. NYUDepth on left, KITTI on right. For each image, we show (a) input, (b) output of coarse network, (c) refined output of fine network, (d) ground truth. Examples are sorted from best (top) to worst (bottom).

Chapter 6

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

The work presented in this chapter was a collaboration with Rob Fergus, and is currently available on arXiv [23].

6.1 Introduction

In this chapter, we develop a multiscale convolutional network to address three different computer vision tasks: depth prediction, surface normal estimation, and semantic labeling. Our new model builds upon the approach we took in the previous chapter on depth map prediction, and contains enhancements that both enable generalization to new tasks, as well as help performance. All three tasks use the same core architecture, which requires only minor task-specific modifications to perform each effectively.

Our new method generates pixel-maps directly from an input image, without the need for low-level superpixels or contours, and is able to align to many image details by using a series of convolutional network stacks applied at increasing resolution. At test time, all three outputs can be generated in real time (∼30Hz). We achieve state-of-the art results on all three tasks we investigate, demonstrating the versatility of our approach.

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

6.2 Related Work

Single-image surface normal estimation has been addressed by Fouhey *et al.* [30, 31], Ladicky *et al.* [75], and most recently by Wang *et al.* [140], the latter in work concurrent with ours. Fouhey *et al.* match to discriminative local templates [30] followed by a global optimization on a grid drawn from vanishing point rays [31], while Ladicky *et al.* learn a regression from over-segmented regions to a discrete set of normals and mixture coefficients. Wang *et al.* [140] use convolutional networks to combine normals estimates from local and global scales, while also employing cues from room layout, edge labels and vanishing points. Importantly, we achieve as good or superior results with a more general multiscale architecture that can naturally be used to perform many different tasks.

Prior work on semantic segmentation includes many different approaches, both using RGB-only data [129, 11, 28] as well as RGB-D [115, 101, 89, 16, 51, 56, 48]. Most of these use local features to classify over-segmented regions, followed by a global consistency optimization such as a CRF. By comparison, our method takes an essentially inverted approach: We make a consistent global prediction first, then follow it with iterative local refinements. In so doing, the local networks are made aware of their place within the global scene, and can can use this information in their refined predictions.

Gupta *et al.* [48, 49] create semantic segmentations first by generating contours, then classifying regions using either hand-generated features and SVM [48], or a convolutional network for object detection [49]. Notably, [48] also performs amodal completion, which transfers labels between disparate regions of the image by comparing planes from the depth.

Most related to our method in semantic segmentation are other approaches using convolutional networks. Farabet *et al.* [28] and Couprie *et al.* [16] each use a convolutional network applied at multiple scales to find local predictions, then aggregate the predictions using superpixels. Our method differs in several important ways. First, our model has a large, full-image field of view at the coarsest scale; as we demonstrate, this is of critical importance, particularly for depth and normals tasks. Second, we do not use superpixels or any post-process smoothing — instead, our network produces fairly smooth outputs on its own, allowing us to take a simple pixel-wise maximum. Moreover, our model can naturally be applied both to piecewise-constant targets (*e.g.* labels) as well as spatially-varying outputs (*e.g.* depth/normals).

Pinheiro *et al.* [98] use a recurrent convolutional network in which each application predicts labels at the center location of an input region, given predicted labels from the previous scale and a rescaled input patch. In contrast to our model, the scales progress from local to global, incorporating progressively more context — precisely the reverse of our approach. In addition, they apply the same network parameters at all scales, while we learn distinct networks that can specialize in the edits appropriate to their stage, and communicate between the first two scales with more flexible feature maps rather than constraining to the classes; these choices are also consistent with our findings in Chapter 7, in which we find that not tying weights between layers generally helps performance. However, a drawback of this is that our model is constrained to operate on fixed-size images, whereas [98] can in theory be repeatedly applied to cover arbitrarily large images.

In concurrent work, Long *et al.* [83] adapt the recent VGG ImageNet model [116] to semantic segmentation by applying 1x1 convolutional label classifiers at feature maps from different layers, corresponding to different scales, and averaging the outputs. By contrast, we apply networks for different scales in series, which allows them to make more complex edits and refinements, starting from a full image field of view. Thus our architecture easily adapts to many tasks, whereas by using fields of view always centered on the output and summing predictions, theirs is specific to semantic labeling.

Some recent works have applied related architectures to object segmentation. Wang *et al.* [141] perform salient object segmentation using a single-scale ConvNet applied jointly with bounding box detection, but segment only one object per image and are limited to 50x50 single-channel bitmaps. Huang and Jain [63] segment neurons by recursively applying an affinity graph generator, but apply their model exclusively to neuron segmentation, use a VQ/SVM pipeline, and make different use of scale. By contrast, we apply a series of convolutional networks at successive scales, applied to a several different tasks in describing photographic scenes.

6.3 Model Architecture

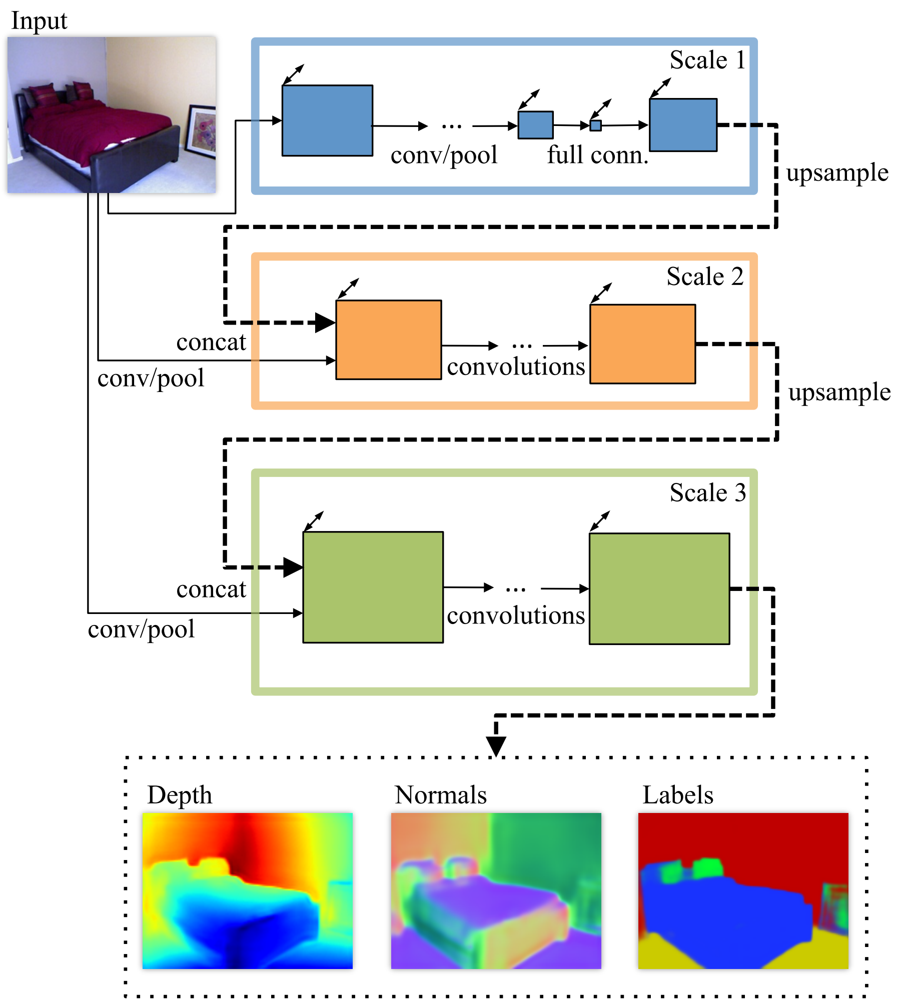
Building upon the we model introduced in Chapter 5, we employ a multi-scale deep network that first predicts a coarse global output based on the entire image area, then refines it using finer-scale local networks. This scheme is illustrated in Fig. 6.1. Our new model has several architectural improvements: First, we make the model deeper (more convolutional layers). Second, we add a third scale at higher resolution, bringing the final output resolution up to half the input, or 109×147 for NYUDepth. Third, instead of passing output *predictions* from scale 1 to scale 2, we pass multichannel *feature maps*; in so doing, we found we could also train the first two scales of the network jointly from the start, somewhat simplifying the training procedure and yielding performance gains.

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

Since the top layers are fully connected, each spatial location in the output connects to the all the image features, incorporating a very large field of view. This stands in contrast to the multiscale approach of [16, 28], who apply convolutions and pooling alone to downsampled versions of the image, producing maps whose output locations' fields of view are always centered on the output pixel. This full-view connection is especially important for depth and normals tasks, as we investigate in Section 6.7.1.

As shown in Fig. 6.1, we trained two different sizes of our model: One where this scale is based on an ImageNet-trained AlexNet [73], and one where it is initialized using the Oxford VGG network [116]. We report differences in performance between the models on all tasks, to measure the impact of model size in each.

Scale 2: Predictions The job of the second scale is to produce predictions at a



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

Figure 6.1: Model architecture. C is the number of output channels in the final prediction, which depends on the task. The input to the network is 320x240.

mid-level resolution, by incorporating a more detailed but narrower view of the image along with the full-image information supplied by the coarse network. We accomplish this by concatenating the feature maps of the coarse network with those from a single layer of convolution and pooling, performed at finer stride (see Fig. 6.1). The output of the second scale is a 74×55 prediction (for NYUDepth), with the number of channels depending on the task. We train Scales 1 and 2 of the model together jointly, using SGD on the losses described in Section 6.4.

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

6.4 Tasks

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

6.4.1 Depth

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

$$L_{depth}(D, D^*) = \frac{1}{n}\sum_i d_i^2 - \frac{1}{2n^2}\left(\sum_i d_i\right)^2 + \frac{1}{n}\sum_i [(\nabla_x d_i)^2 + (\nabla_y d_i)^2] \qquad (6.1)$$

where the sums are over valid[1] pixels i, and n is the number of valid pixels. Here, $\nabla_x d_i$ and $\nabla_y d_i$ are the horizontal and vertical image gradients of the difference.

This loss combines the l_2 and scale-invariant terms we used in Chapter 5 with a first-order matching term $(\nabla_x d_i)^2 + (\nabla_y d_i)^2$, which compares image gradients of the prediction with the ground truth. This encourages predictions to have not only close-by values, but also similar local structure. We found it indeed produces outputs that better follow depth gradients, with no degradation in measured l_2 performance.

6.4.2 Surface Normals

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

$$L_{normals}(N, N^*) = -\frac{1}{n}\sum_i N_i \cdot N_i^* = -\frac{1}{n} N \cdot N^* \tag{6.2}$$

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

For ground truth targets, we compute the normal map using the same method as in Silberman *et al.* [115], which estimates normals from depth by fitting least-squares planes to neighboring sets of points in the point cloud.

6.4.3 Semantic Labels

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

[1]We mask out pixels where the ground truth is missing.

simple pixelwise cross-entropy loss,

$$L_{semantic}(C, C^*) = -\frac{1}{n}\sum_i C_i^* \log(C_i) \tag{6.3}$$

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

When labeling the NYUDepth RGB-D dataset, we use the ground truth depth and normals as additional input channels. We convolve each of the three input types (RGB, depth and normals) with a different set of $32 \times 9 \times 9$ filters, then concatenate the resulting three feature sets along with the network output from the previous scale to form the input to the next. [2] Note the first scale is initialized using ImageNet, and we keep it RGB-only. Applying convolutions to each input type separately, rather than concatenating all the channels together in pixel space and filtering the joint input, enforces independence between the features at the lowest filter level, which we found helped performance.

6.5 Training

6.5.1 Training Procedure

We train our model in two phases using SGD: First, we jointly train both Scales 1 and 2. Second, we fix the parameters of these scales and train Scale 3. Since Scale 3 contains four times as many pixels as Scale 2, it is expensive to train using the entire image area for each gradient step. To speed up training, we instead use random crops of size 74x55: We first forward-propagate the entire image through scales 1 and 2, upsample, and crop the resulting Scale 3 input, as well as the original RGB input at the corresponding location. The cropped image and Scale 2 prediction are forward- and back-propagated through the Scale 3 network, and the weights updated. We find this speeds up training

[2] We also tried the "HHA" encoding proposed by [49], but did not see a benefit in our case, thus we opt for the simpler approach of using the depth and xyz-normals directly.

by about a factor of 3, including the overhead for inference of the first two scales, and results in about the same if not slightly better error from the increased stochasticity.

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

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

6.5.2 Data Augmentation

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

6.5.3 Combining Depth and Normals

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

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

6.6 Performance Experiments

6.6.1 Depth

We first apply our method to depth prediction on NYUDepth v2. We train using the entire NYUDepth v2 raw data distribution, using the scene split specified in the official train/test distribution. We then test on the common distribution depth maps, including filled-in areas, but constrained to the axis-aligned rectangle where there there is a valid depth map projection. Since the network output is a lower resolution than the original NYUDepth images, and excludes a small border, we bilinearly upsample our network outputs to the original 640x480 image scale, and extrapolate the missing border using a cross-bilateral filter. We compare our method to prior works Ladicky *et al.* [74], Karsh *et al.* [68], Baig *et al.* [2], Liu *et al.* [82], and our previous system from Chapter 5 (Eigen *et al.* [25]).

Results are shown in Table 6.1. Our model obtains best performance in every metric, due to our larger architecture. Qualitative results in Fig. 6.2 show considerable improvement in detail sharpness over the results from Chapter 5.

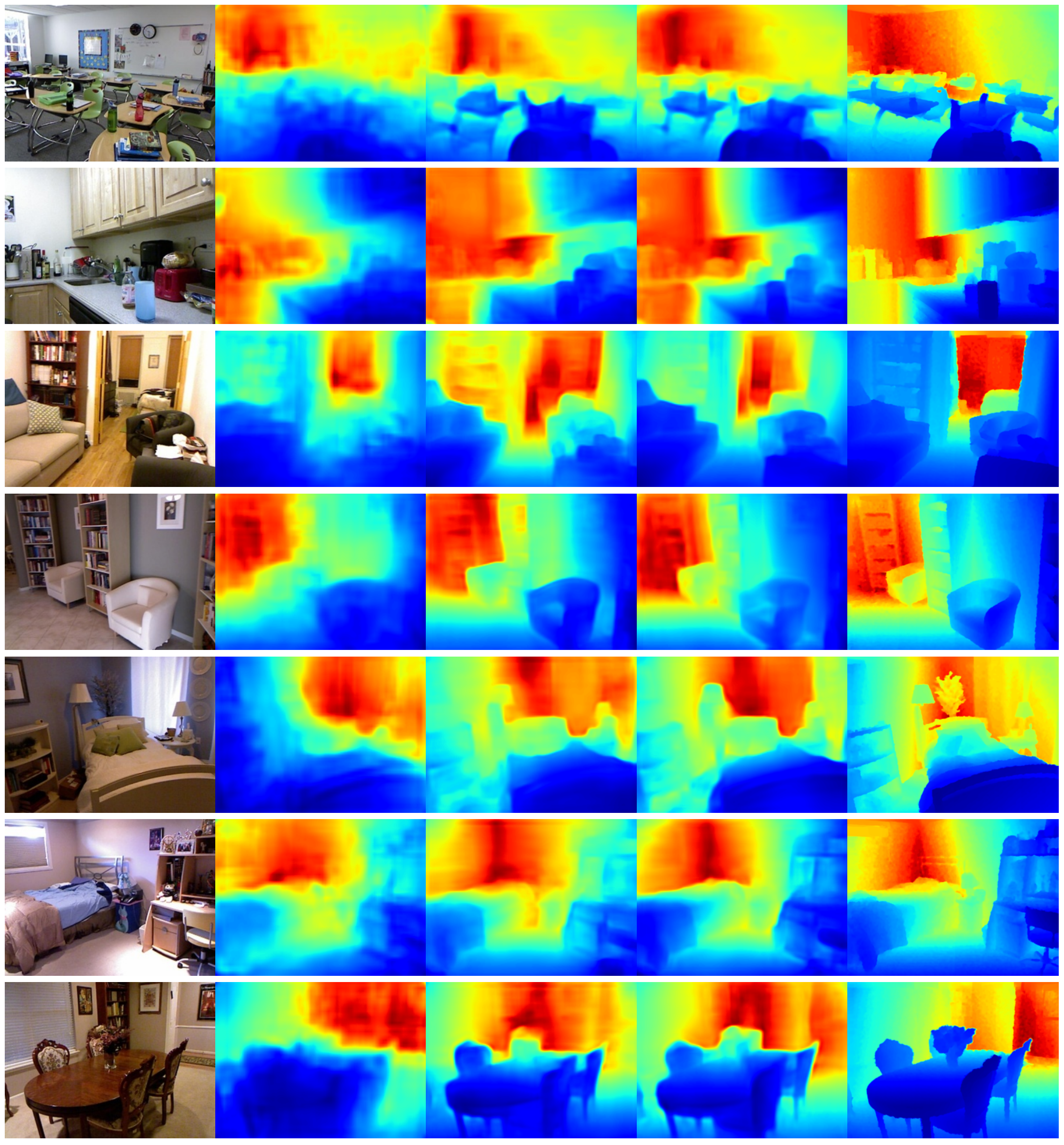


Figure 6.2: Example depth results. (a) RGB input; (b) Result from Chapter 5 [25]; (c) Our result (Scale 1: AlexNet); (d) Our result (Scale 1: VGG); (e) Ground Truth. Note the color range of each image is individually scaled.

Depth Prediction								
	Ladicky[74]	Karsch[68]	Baig [2]	Liu [82]	Eigen[25]	Ours(A)	Ours(VGG)	
$\delta < 1.25$	0.542	–	0.597	0.614	0.614	0.697	**0.769**	higher
$\delta < 1.25^2$	0.829	–	–	0.883	0.888	0.912	**0.950**	is
$\delta < 1.25^3$	0.940	–	–	0.971	0.972	0.977	**0.988**	better
abs rel	–	0.350	0.259	0.230	0.214	0.198	**0.158**	
sqr rel	–	–	–	–	0.204	0.180	**0.121**	lower
RMS(lin)	–	1.2	0.839	0.824	0.877	0.753	**0.641**	is
RMS(log)	–	–	–	–	0.283	0.255	**0.214**	better
sc-inv.	–	–	0.242	–	0.219	0.202	**0.171**	

Table 6.1: Depth estimation measurements.

6.6.2 Surface Normals

Next we apply our method to surface normals prediction. We compare against the 3D Primitives (3DP) and "Indoor Origami" works of Fouhey *et al.* [30, 31]; Ladicky *et al.* [75]; and Wang *et al.* [140]. As with the depth network, we used the full raw dataset for training, since ground-truth normal maps can be generated for all images. Since different systems have different ways of calculating ground truth normal maps, we compare using both the ground truth as constructed in [30, 31] as well as the method used in [115], using precomputed predictions supplied by the authors of method. Note that Wang *et al.* use a method similar to [30] to construct training targets, while we use the method of [115] for this purpose. We measure performance with the same metrics as in [30]: The mean and median angle from the ground truth across all unmasked pixels, as well as the percent of vectors whose angle falls within a series of three thresholds.

Results are shown in Table 6.2. The smaller version of our model performs similarly or slightly better than Wang *et al.* , while the larger version substantially outperforms all comparison methods. Note that of the ground truths, [30] is somewhat more pre-processed compared to [115], and thus [30] tends to present flatter areas, while [115] is noisier but keeps more details present.

Figure 6.3 and 6.4 show example predictions. Note the details captured by our method, such as the curvature of the blanket on the bed in the first row, sofas in the second row, and objects in the last row.

Surface Normal Estimation (GT [30])					
	Angle Distance		**Within t° Deg.**		
	Mean	**Median**	11.25°	22.5°	30°
3DP [30]	34.2	30.0	18.5	38.6	50.0
Ladicky &*al* [75]	32.5	22.3	27.4	50.2	60.1
Fouhey &*al* [31]	35.1	19.2	37.6	53.3	58.9
Wang &*al* [140]	26.6	15.3	40.1	61.4	69.0
Ours (AlexNet)	23.1	15.1	39.4	63.6	72.7
Ours (VGG)	**20.5**	**13.2**	**44.0**	**68.5**	**77.2**
Surface Normal Estimation (GT [115])					
	Angle Distance		**Within t° Deg.**		
	Mean	**Median**	11.25°	22.5°	30°
3DP [30]	37.7	34.1	14.0	32.7	44.1
Ladicky &*al* [75]	35.5	25.5	24.0	45.6	55.9
Wang &*al* [140]	28.8	17.9	35.2	57.1	65.5
Ours (AlexNet)	25.9	18.2	33.2	57.5	67.7
Ours (VGG)	**22.2**	**15.3**	**38.6**	**64.0**	**73.9**

Table 6.2: Surface normals prediction measured against different types of ground truth acquisition. Each column shows results for a different ground truth.

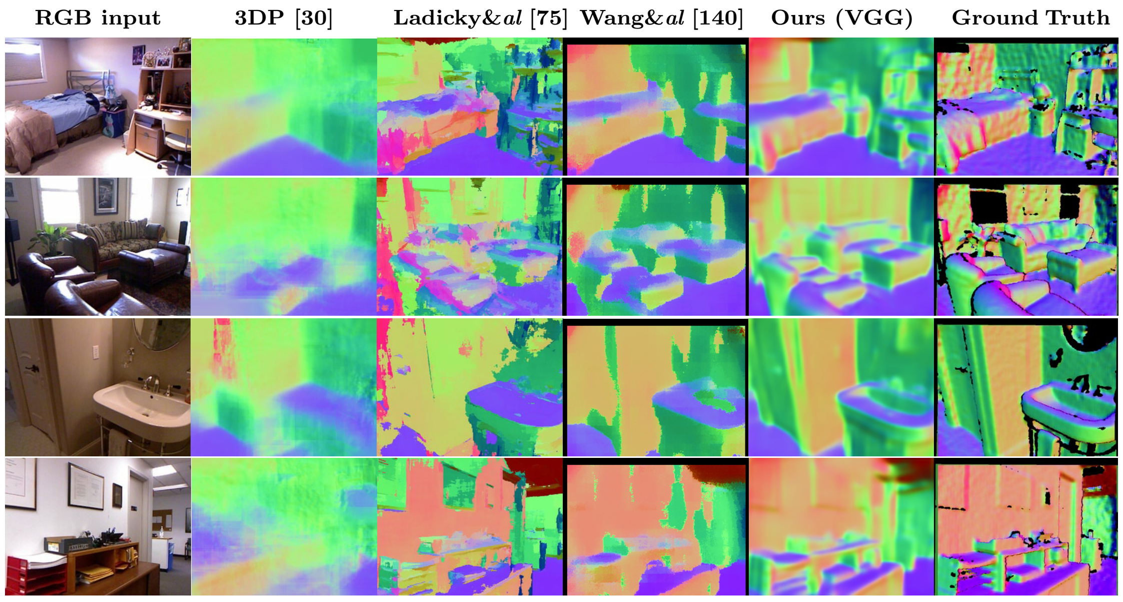


Figure 6.3: Comparison of surface normal maps.

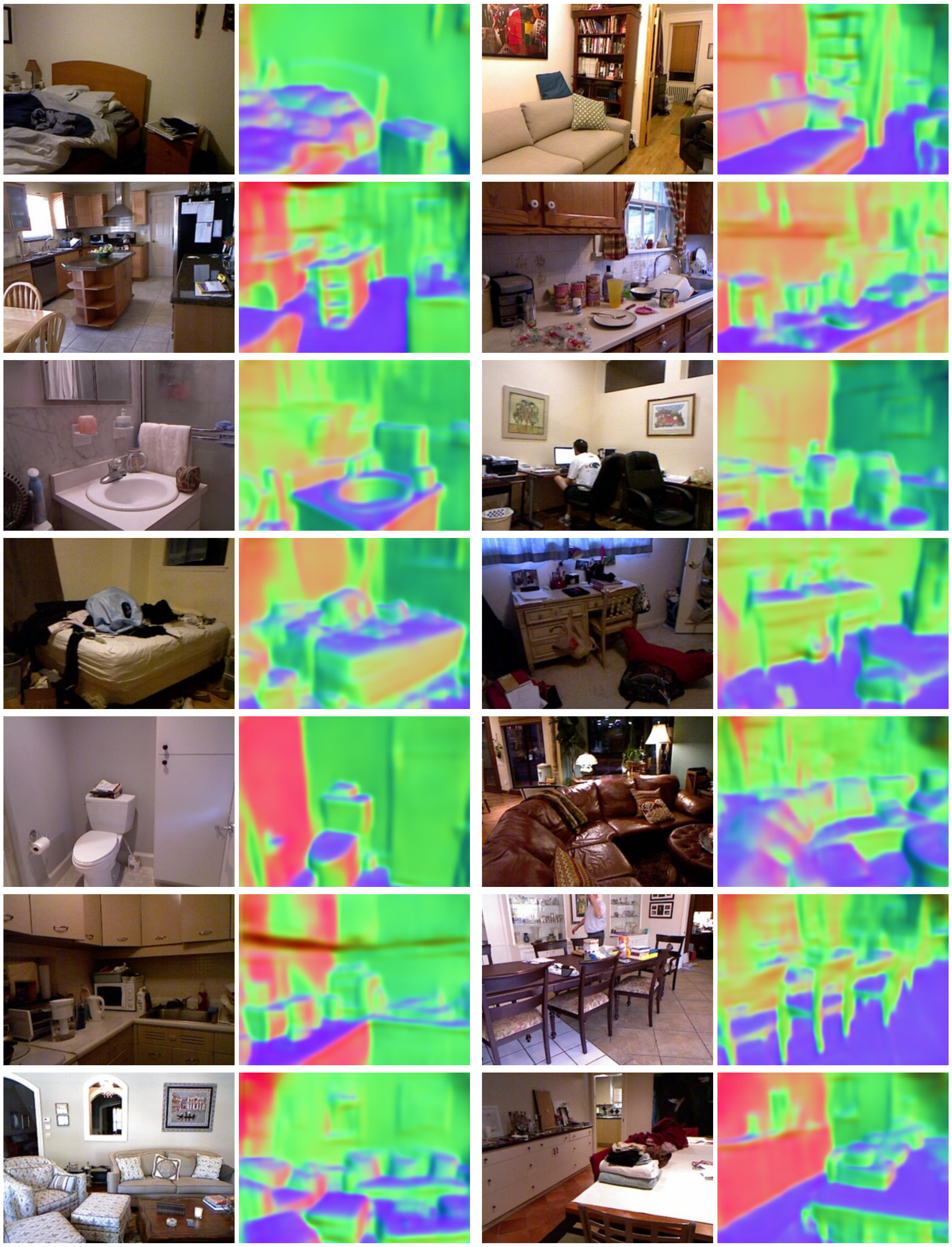


Figure 6.4: Example surface normals results.

6.6.3 Semantic Labels

NYU Depth

We finally apply our method to semantic segmentation, first also on NYUDepth. Because this data provides a depth channel, we use the ground-truth depth and normals as input into the semantic network, as described in Section 6.4.3. We evaluate our method on semantic class sets with 4, 13 and 40 labels, described in [115], [16] and [48], respectively. The 4-class segmentation task uses high-level category labels "floor", "structure", "furniture" and "props", while the 13- and 40-class tasks use different sets of more fine-grained categories. We compare with several recent methods, using the metrics commonly used to evaluate each task: For the 4- and 13-class tasks we use pixelwise and per-class accuracy; for the 40-class task, we also compare using the mean pixel-frequency weighted Jaccard index of each class, and the flat mean Jaccard index.

4-Class Semantic Segmentation		
	Pixel	Class
Couprie &*al* [16]	64.5	63.5
Khan &*al* [51]	69.2	65.6
Stuckler &*al* [123]	70.9	67.0
Mueller &*al* [89]	72.3	71.9
Gupta &*al* '13 [48]	78	–
Ours (AlexNet)	80.6	79.1
Ours (VGG)	**83.2**	**82.0**

13-Class Semantic		
	Pixel	Class
Couprie &*al* [16]	52.4	36.2
Wang &*al* [140]	–	42.2
Hermans &*al* [56]	54.2	48.0
Khan &*al* [51] *	58.3	45.1
Ours (AlexNet)	70.5	59.4
Ours (VGG)	**75.4**	**66.9**

40-Class Semantic Segmentation				
	Pix. Acc.	Per-Cls Acc.	Freq. Jaccard	Av. Jaccard
Gupta&*al*'13 [48]	59.1	28.4	45.6	27.4
Gupta&*al*'14 [49]	60.3	35.1	47.0	28.6
Long&*al* [83]	65.4	**46.1**	49.5	34.0
Ours (AlexNet)	62.9	41.3	47.6	30.8
Ours (VGG)	**65.6**	45.1	**51.4**	**34.1**

Table 6.3: Semantic labeling on NYUDepth v2

*Khan&*al* use a different overlapping label set.

Results are shown in Table 6.3. We decisively outperform the comparison methods on the 4- and 14-class tasks. In the 40-class task, our model outperforms Gupta *et al.* '14 for both model sizes, and Long *et al.* with the larger size. Qualitative results are shown in Fig. 6.7. Even though our method does not use superpixels or any piecewise constant assumptions, it nevertheless tends to produce large constant regions most of the time. Individual per-class performance is shown in Table 6.6.

Sift Flow

We confirm our method can be applied to additional scene types by evaluating on the Sift Flow dataset [81], which contains images of outdoor cityscapes and landscapes segmented into 33 categories. All images are 256x256, rather than 320x240 for NYUDepth, and so our model outputs images of a different size. Note that we do not adjust any of the convolutional kernel sizes or learning rates for this dataset — we simply transfer the values used for NYUDepth directly; however, we adjust the random crop augmentations by a few pixels so that feature maps can be combined evenly between scales.

Sift Flow Semantic Segmentation		
	Pix. Acc.	Per-Class Acc.
Chapter 3 weighted kNN	77.1	32.5
Farabet &*al* (1) [28]	78.5	29.6
Farabet &*al* (2) [28]	74.2	46.0
Tighe &*al* [129]	78.6	39.2
Pinheiro &*al* [98]	77.7	29.8
Long &*al* [83]	85.1	51.7
Ours (1)	84.0	42.0
Ours (2)	81.6	48.2
Ours VGG feats (1)	**86.8**	46.4
Ours VGG feats (2)	83.8	**55.7**

Table 6.4: Semantic labeling on the Sift Flow dataset.

We compare against Tighe *et al.* [129], Farabet *et al.* [28], Pinheiro [98] and Long *et al.* [83], as well as the weighted kNN system we presented in Chapter 3. Note that Farabet *et al.* train two models, using either empirical or rebalanced class distributions by resampling superpixels during training. We train a more class-balanced version of our model by reweighting each class in the cross-entropy loss; we weight each pixel using weight $\alpha_c = median\_freq/freq(c)$ where $freq(c)$ is the number of pixels of class c divided by the total number of pixels in images where c is present, and $median\_freq$ is the median of these frequencies.

Results are shown in Table 6.4; we compare regular (1) and reweighted (2) versions of our model against comparison methods. Our model outperforms all but Long *et al.* by substantial margins using our smaller ImageNet model, and performs similarly or better to Long *et al.* with our larger model. We also greatly improve upon the weighted kNN system we first developed in Chapter 3. Examples are shown in Fig. 6.5. This demonstrates our model's adaptability not just to different tasks but also different data.

Pascal VOC

In addition, we also verify our method using the Pascal VOC 2011 validation set. Similarly to Long *et al.* [83], we train using the 2011 training set augmented with 8498 training images collected by Hariharan *et al.* [52], and evaluate using the 736 images from the 2011 validation set not also in the Hariharan extra set. We perform online data augmentations as in our NYUDepth and Sift Flow models, and use the same learning rates. Because these images have arbitrary aspect ratio, we train our model on square inputs, and scale the smaller side of each image to 256; at test time we apply the model with a stride of 128 to cover the image (two applications are usually sufficient).

Pascal VOC Semantic Segmentation				
	Pix. Acc.	Per-Cls Acc.	Freq. Jaccard	Av. Jaccard
Long*&al* [83]	**90.3**	**75.9**	**83.2**	**62.7**
Ours (VGG)	**90.3**	72.4	82.9	62.2

Table 6.5: Semantic labeling on Pascal VOC 2011.

Results are shown in table Table 6.5 and Fig. 6.6. Our model performs comparably to Long *et al.* , even as it generalizes to multiple tasks, demonstrated by its adeptness at depth and normals prediction.

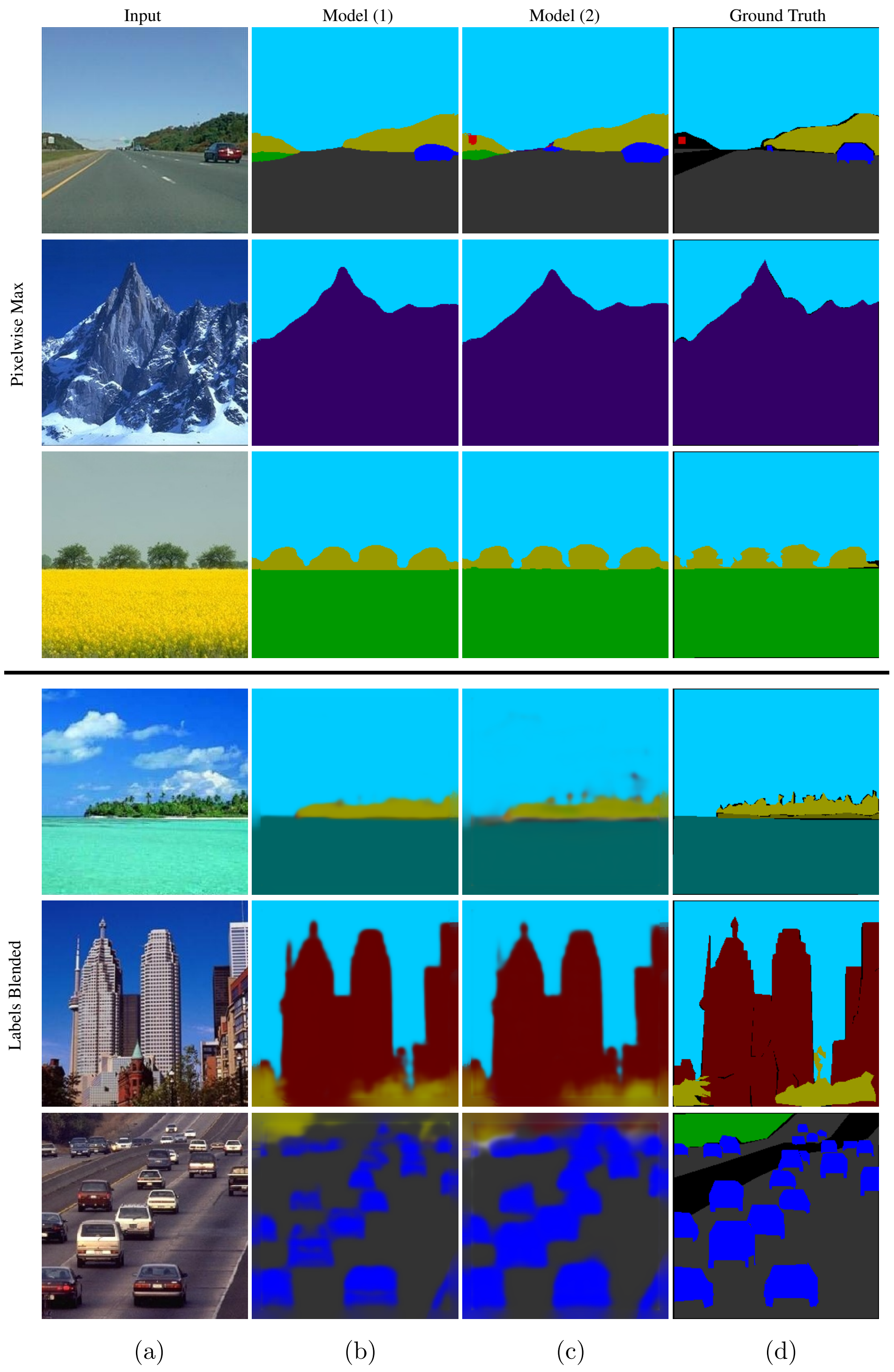


Figure 6.5: Example semantic labeling results for Sift Flow. Top: Maximum predicted label shown for each pixel; Bottom: Label colors blended according to softmax outputs. For each row, we show: (a) input image; (b) without class rebalancing; (c) with class rebalancing; (d) ground truth.

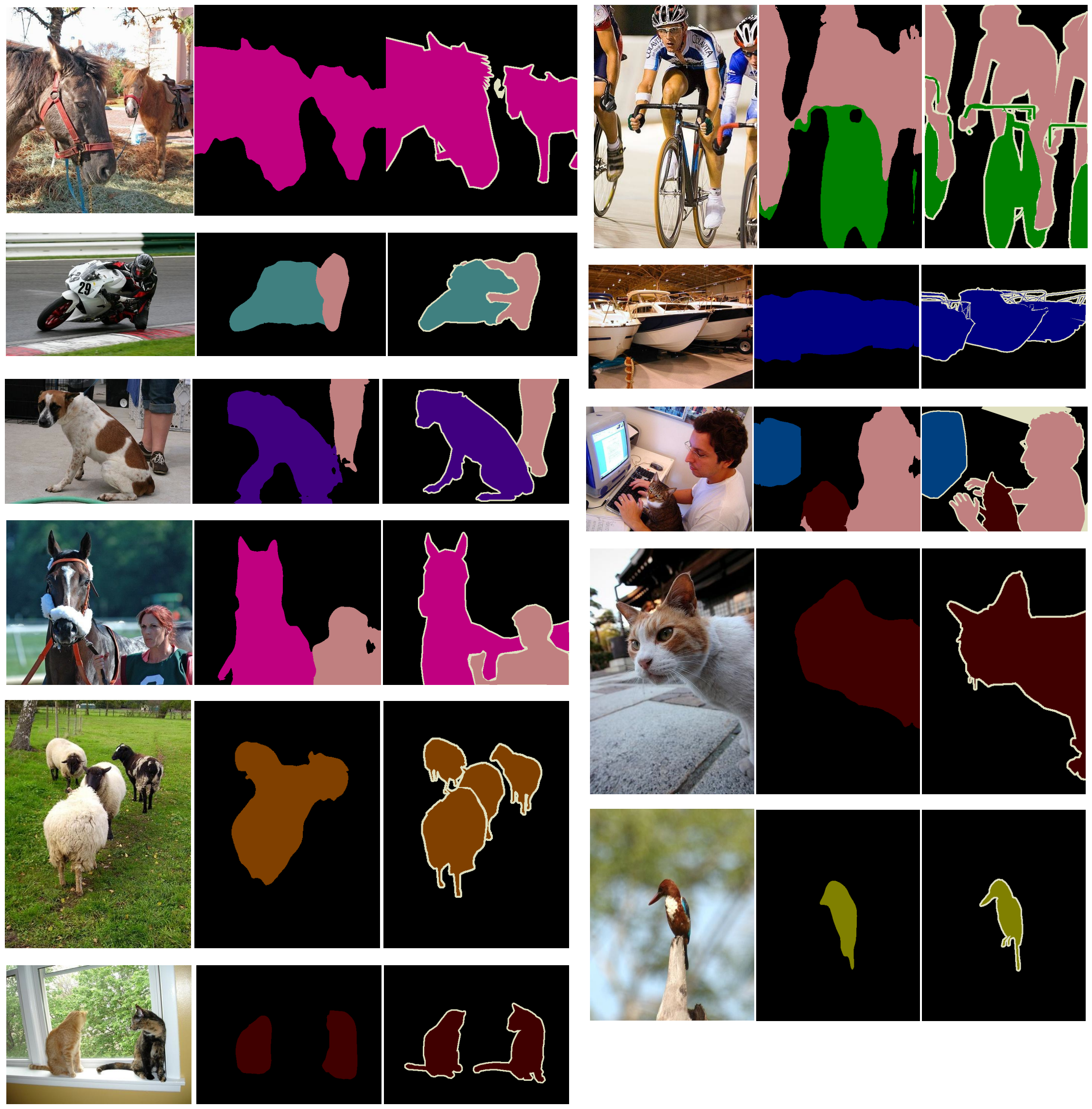


Figure 6.6: Example semantic labeling results for Pascal VOC 2011. For each image, we show RGB input, our prediction, and ground truth.

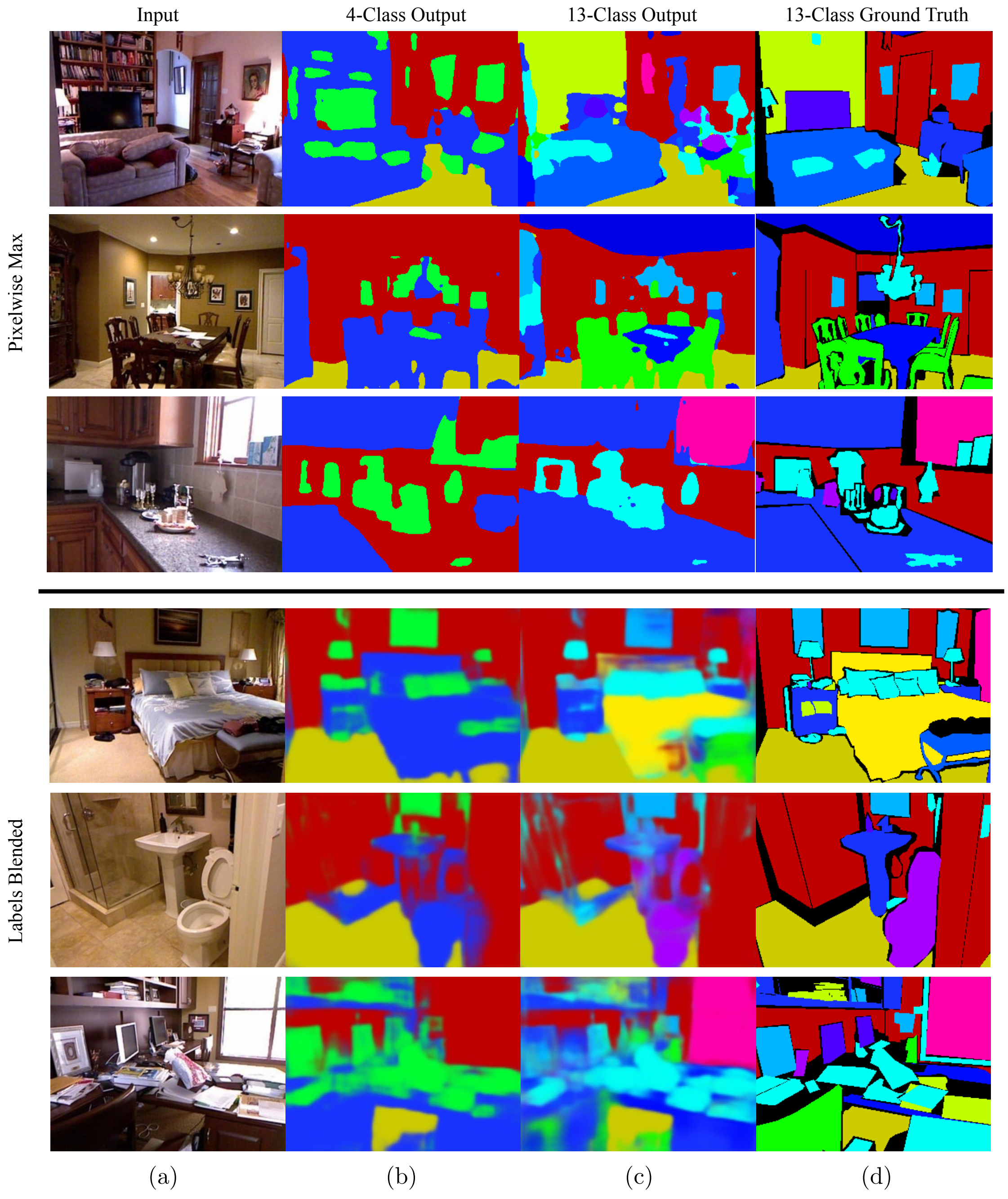


Figure 6.7: Example semantic labeling results. Top: Maximum predicted label shown for each pixel; Bottom: Label colors blended according to softmax outputs. For each row, we show: (a) input image; (b) 4-class labeling result; (c) 13-class result; (d) 13-class ground truth. Note we feed the ground-truth depth and normals along with the RGB image as input to our labeling network.

4-Class (pixel acc.)				
	floor	struct	furntr	prop
Couprie *et al.*	87.3	86.1	45.3	35.5
Khan *et al.*	87.1	88.2	54.7	32.6
Stuckler *et al.*	90.7	81.4	68.1	19.8
Mueller *et al.*	**94.9**	78.9	71.1	42.7
Ours (AlexNet)	93.9	**87.9**	**79.7**	**55.1**

13-Class (pixel acc.)													
	bed	books	ceiling	chair	floor	furniture	objects	picture	sofa	table	tv	wall	window
Couprie *et al.*	30.3	31.7	33.2	44.4	68.0	28.5	10.9	38.5	25.8	18.0	18.8	**89.4**	37.8
Wang *et al.*	47.6	45.0	68.1	23.5	84.1	16.7	12.4	26.4	39.1	35.4	32.4	65.9	52.2
Hermans *et al.*	**68.4**	**45.4**	**83.4**	41.9	91.5	37.1	8.6	35.8	28.5	27.7	**38.4**	71.8	46.1
Khan *et al.*	38.1	13.7	62.6	-	87.3	-	-	-	29.8	10.2	6.0	86.1	15.9
Ours (AlexNet)	57.7	39.9	77.6	**71.1**	**95.9**	**64.1**	**54.9**	**49.4**	**45.8**	**45.0**	25.2	87.9	**57.6**

40-Class (Jaccard index)										
	bag	bathtub	bed	blinds	book	bookshelf	box	cabinet	ceiling	chair
Gupta*&al*'13	0.65	33	55	44	4.4	20	4.8	48	59	40
Gupta*&al*'14	0.2	**38.2**	**65.0**	42.0	18.1	6.4	2.1	44.9	60.5	**47.9**
Ours (AlexNet)	1.5	29.1	51.9	43.9	16.6	31.5	4.5	45.0	**75.2**	46.0
Ours (VGG)	**2.0**	37.3	58.6	**50.7**	**20.0**	**36.4**	**6.6**	**51.6**	66.0	47.5
	clothes	counter	curtain	desk	door	dresser	floor	floor-mat	fridge	lamp
Gupta*&al*'13	6.9	47	34	10	8.3	22	81	22	15	6.8
Gupta*&al*'14	4.7	51.3	29.1	11.3	20.3	**34.8**	81.3	28.0	14.5	**34.8**
Ours (AlexNet)	12.9	53.4	35.4	11.3	17.6	26.3	83.3	27.4	15.7	33.6
Ours (VGG)	**14.1**	**57.1**	**37.3**	**12.6**	**28.7**	24.3	**84.1**	**29.8**	**26.0**	31.2
	mirror	night-stand	paper	person	picture	pillow	shelves	showerctn	sink	sofa
Gupta*&al*'13	19	20	1.9	16	40	28	5.1	18	26	44
Gupta*&al*'14	16.4	**27.2**	14.3	0.2	40.3	**34.4**	3.5	4.2	37.5	47.9
Ours (AlexNet)	**32.6**	24.2	20.3	27.5	45.0	31.0	8.9	**21.7**	39.4	40.4
Ours (VGG)	23.3	24.7	**21.2**	**37.8**	**48.5**	33.4	**9.1**	20.2	**41.3**	**49.1**
	table	toilet	towel	tv	wall	whitebd	window	other-furntr	other-prop	other-struct
Gupta*&al*'13	30	50	14	9.3	68	37	33	2	22	6.9
Gupta*&al*'14	29.9	**55.1**	16.3	31.0	68.0	14.2	32.6	6.1	23.1	7.1
Ours (AlexNet)	32.0	44.8	14.8	32.3	68.2	6.6	33.0	6.8	**29.4**	11.7
Ours (VGG)	**35.7**	44.0	**20.0**	**35.9**	**71.1**	**39.7**	**37.4**	**8.8**	28.7	**13.1**

Table 6.6: Individual class performance comparisons.

6.7 Probe Experiments

6.7.1 Contributions of Scales

We compare performance broken down according to the different scales in our model in Table 6.7. For depth, normals and 4- and 13-class semantic labeling tasks, we train and evaluate the model using just scale 1, just scale 2, both, or all three scales 1, 2 and 3. For the coarse scale-1-only prediction, we replace the last fully connected layer of the coarse stack with a fully connected layer that outputs directly to target size, *i.e.* a pixel map of either 1, 3, 4 or 13 channels depending on the task. The spatial resolution is the same as is used for the coarse features in our model, and is upsampled in the same way.

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

First, we note there is progressive improvement in all tasks as scales are added (rows

Contributions of Scales

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

Table 6.7: Comparison of networks for different scales for depth, normals and semantic labeling tasks with 4 and 13 categories. Largest single contributing scale is underlined.

Effect of Depth/Normals Inputs

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

Table 6.8: Comparison of RGB-only, predicted depth/normals, and ground-truth depth/normals as input to the 13-class semantic labeling task.

1, 3, and 4). In addition, we find the largest single contribution to performance is the coarse Scale 1 for depth and normals, but the more local Scale 2 for the semantic tasks — however, this is only due to the fact that the depth and normals channels are introduced at Scale 2 for the semantic labeling task. Looking at the labeling network with RGB-only inputs, we find that the coarse scale is again the larger contributer, indicating the importance of the global view. (Of course, this scale was also initialized with ImageNet convolution weights that are much related to the semantic task; however, even initializing randomly achieves 54.5% for 13-class scale 1 only, still the largest contribution, albeit by a smaller amount).

6.7.2 Effect of Depth and Normals Inputs

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

To study this, we trained and tested our network using either Scale 2 alone or both Scales 1 and 2 for the 13-class semantic labeling task under three input conditions: (*a*) the RGB image only, (*b*) the RGB image along with predicted depth and normals, or (*c*) RGB plus true depth and normals. Results are in Table 6.8. Using ground truth depth/normals shows substantial improvements over RGB alone. Predicted depth/normals appear to have little effect when using both scales, but a tangible improvement when using only Scale 2. We believe this is because any relevant information provided by predicted depths/normals for labeling can also be extracted from the input; thus the labeling network can learn this same information itself, just from the label targets. However, this supposes that the network structure is capable of learning these relations: If this is not the case, *e.g.* when using only Scale 2, we do see improvement. This is also consistent with Section 6.7.1, where we found the coarse network was important for prediction in all tasks — indeed, supplying the predicted depth/normals to scale 2 is able to recover much of the performance obtained by the RGB-only scales 1+2 model.

6.8 Discussion

Together, depth, surface normals and semantic labels provide a rich account of a scene. We have proposed a simple and fast multiscale architecture using convolutional networks that gives excellent performance on all three modalities. The models beat existing methods on the vast majority of benchmarks we explored. This is impressive given that many of these methods are specific to a single modality and often slower and more complex algorithms than ours. As such, our model also provides a convenient new baseline for the three tasks.

One drawback of this approach is that it currently requires a large number of relatively dense pixel maps for training. Possible future extensions of this approach include adapting it to be able to use more sparsely labeled targets, as well as its application to further tasks, such as instance segmentation.

In the past four chapters, we have applied convolutional networks to multiple different pixel-map prediction tasks: denoising, depth prediction, surface normals, and semantic labeling. We also saw the ConvNet approach soundly improve over the kNN scene parsing method first described in Chapter 3. The next two chapters look into convolutional architecture sizing patterns in some more detail, then explore some ideas that use ConvNets in unsupervised learning problems.

Chapter 7

Understanding Deep Architectures using a Recursive Convolutional Network

The work presented in this chapter appeared at the ICLR Workshops 2014 [26], and was a collaboration with Jason Rolfe, Rob Fergus and Yann LeCun.

7.1 Introduction

The previous chapters in this thesis developed convolutional network models for use in four pixel map prediction tasks: denoising, depth-from-camera, surface normals, and semantic labels. Each used multiple layers of convolution to make these predictions based on the input. However, many sizing factors needed to be set in order to define the models, including the numbers of layers, feature maps, kernel pixel width, pooling, etc. Moreover, each of these adjusted both the size of the activation units as well as the total number of parameters. Are there any intuitions for how different configurations affect performance and the system's capabilities? This chapter aims to characterize some of these in the context of a classification task, by evaluating the independent contributions of three interlinked linked variables: The numbers of layers, feature maps, and parameters.

We accomplish this via a series of three experiments using a recursive convolutional network model. This model is equivalent to a deep convolutional network where all layers have the same number of feature maps and the filters (weights) are tied across layers. By aligning the architecture of our model to existing convolutional approaches, we are able to tease apart these three factors that determine performance. For example, adding another layer increases the number of parameters, but it also puts an additional non-linearity into the system. But would the extra parameters be better used expanding the size of the existing layers? To provide a general answer to this type of issue is difficult since multiple factors are conflated: the capacity of the model (and of each layer) and its degree of non-linearity. However, we can design a recursive model to have the same number of layers and parameters as the standard convolutional model, and thereby see whether the number of feature maps (which differs) is important. Or we can match the number of feature maps and parameters to see if the number of layers (and number of non-linearities) matters.

Several recent works have found that stacks of multiple unpooled convolution layers are essential to obtain high performance on image classification tasks, including all ImageNet challenge winners for the past three years [73, 149, 113, 125, 116]. Hence the use of multiple convolution layers is vital and the development of superior models relies on understanding their properties. Our investigation in this chapter has particular bearing in characterizing these layers, and our results are very much corroborative with the circumstantial evidence provided by these recent systems.

We find that while increasing the numbers of layers and parameters each have clear benefit, the number of feature maps (and hence dimensionality of the representation) appears ancillary, and finds most of its benefit through the introduction of more weights. Our results (*i*) empirically confirm the notion that adding layers alone increases computational power, within the context of convolutional layers, and (*ii*) suggest that precise sizing of convolutional feature map dimensions is itself of little concern — more attention should be paid to the numbers of parameters in these layers instead.

7.2 Related Work

In addition to unpooled stacks of convolutional maps, the model we employ also has relations to recurrent neural networks. These are are well-studied models [60, 111, 124], naturally suited to temporal and sequential data. For example, they have recently been shown to deliver excellent performance for phoneme recognition [44] and cursive handwriting recognition [43]. However, they have seen limited use on image data. Socher *et al.* [121] showed how image segments could be recursively merged to perform scene parsing. More recently [120], they used a convolutional network in a separate stage to first learn features on RGB-Depth data, prior to hierarchical merging. In these models the input dimension is twice that of the output. This contrasts with our model which has the same input and output dimension.

Our network also has links to several auto-encoder models. Sparse coding [93] uses iterative algorithms, such as ISTA [5], to perform inference. Rozell *et al.* [105] showed how the ISTA scheme can be unwrapped into a repeated series of network layers, which can be viewed as a recursive net. Gregor & LeCun [45] showed how to backpropagate through such a network to give fast approximations to sparse coding known as LISTA. Rolfe & LeCun [103] then showed in their DrSAE model how a discriminative term can be added. Our network can be considered a purely discriminative, convolutional version of LISTA or DrSAE.

7.3 Approach

Our investigation is based on a multilayer Convolutional Network [77], for which all layers beyond the first have the same size and connection topology. All layers use rectified linear units (ReLU) [13, 37, 90]. We perform max-pooling with non-overlapping windows after the first layer convolutions and rectification; however, layers after the first use no explicit pooling. We refer to the number of feature maps per layer as M, and the number of layers after the first as L. To emphasize the difference between the pooled first layer and the unpooled higher layers, we denote the first convolution kernel by $\mathbf{V}$ and the kernels of the higher layers by $\mathbf{W}^l$. A per-map bias $\mathbf{b}^l$ is applied in conjunction with the convolutions. A final classification matrix $\mathbf{C}$ maps the last hidden layer to softmax inputs.

Since all hidden layers have the same size, the transformations at all layers beyond the first have the same number of parameters (and the same connection topology). In addition to the case where all layers are independently parameterized, we consider networks for which the parameters of the higher layers are tied between layers, so that $\mathbf{W}^i = \mathbf{W}^{\mathbf{j}}$ and $\mathbf{b}^i = \mathbf{b}^j$ for all i, j. As shown in Fig. 7.1, tying the parameters across layers renders the deep network dynamics equivalent to recurrence: rather than projecting through a stack of distinct layers, the hidden representation is repeatedly processed by a consistent nonlinear transformation. The convolutional nature of the transformation performed at each layer implies another set of ties, enforcing translation-invariance among the parameters. This novel recursive, convolutional architecture is reminiscent of LISTA [45], but without a direct projection from the input to each hidden layer.

7.3.1 Instantiation on CIFAR-10 and SVHN

We describe our models for the CIFAR-10 [72] and SVHN [91] datasets used in our experiments. In both cases, each image $\mathbf{X}^n$ is of size $32 \times 32 \times 3$. In the equations below, we drop the superscript n indicating the index in the dataset for notational simplicity. The first layer applies a set of M kernels $\mathbf{V}_m$ of size $8 \times 8 \times 3$ via spatial convolution with stride one (denoted as $*$), and per-map bias $\mathbf{b}_m^0$, followed by the element-wise rectification nonlinearity. We use a "same" convolution (i.e. zero-padding the edges), yielding a same-size representation $\mathbf{P}$ of $32 \times 32 \times M$. This representation is then max-

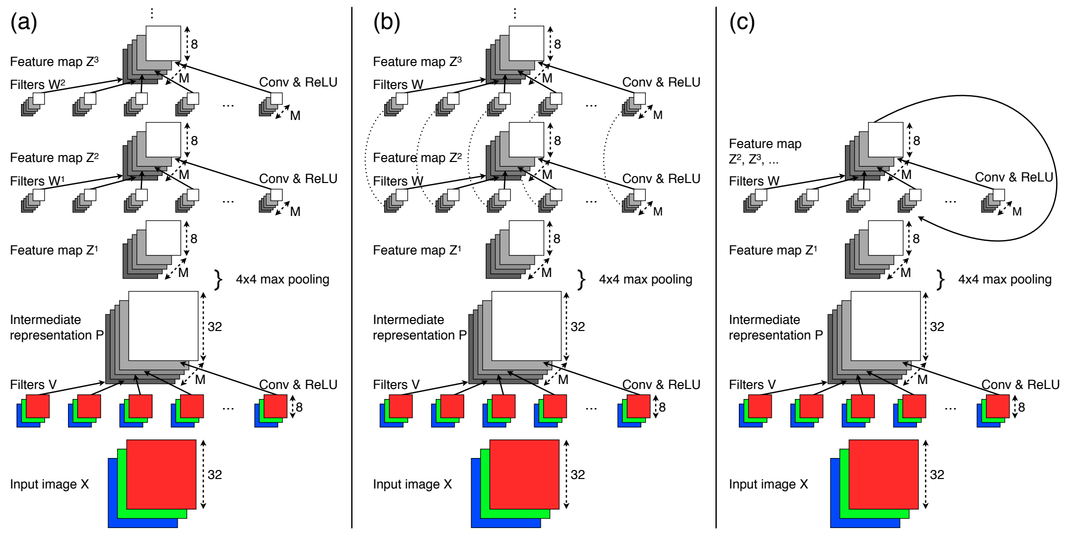


Figure 7.1: Our model architecture prior to the classification layer, as applied to CIFAR and SVHN datasets. (a): Version with un-tied weights in the upper layers. (b): Version with tied weights. Kernels connected by dotted lines are constrained to be identical. (c): The network with tied weights from (b) can be represented as a recursive network.

pooled within each feature map with non-overlapping 4×4 windows, producing a hidden layer $\mathbf{Z^1}$ of size $8 \times 8 \times M$.

$$\mathbf{P}_m = \max\left(0, \mathbf{b}_m^0 + \mathbf{V}_m * \mathbf{X}\right) \quad , \quad \mathbf{Z}^1_{i,j,m} = \max_{i',j' \in \{0,\cdots,3\}} \left(\mathbf{P}_{4\cdot i+i', 4\cdot j+j', m}\right)$$

All L succeeding hidden layers maintain this size, applying a set of M kernels $\mathbf{W}^l_m$ of size $3 \times 3 \times M$, also via "same" spatial convolution with stride one, and per-map bias $\mathbf{b}^l_m$, followed by the rectification nonlinearity:

$$\mathbf{Z}^l_m = \max\left(0, \mathbf{b}^{l-1}_m + \mathbf{W}^{l-1}_m * \mathbf{Z}^{l-1}\right)$$

In the case of the tied model (see Fig. 7.1(b)), the kernels W^l (and biases b^l) are constrained to be the same. The final hidden layer is subject to pixel-wise L2 normalization and passed into a logistic classifier to produce a prediction $\mathbf{Y}$:

$$\mathbf{Y}_k = \frac{\exp(Y'_k)}{\sum_k \exp(Y'_k)} \qquad \text{where} \qquad \mathbf{Y}'_k = \sum_{i,j,m} \mathbf{C}^k_{i,j,m} \cdot \mathbf{Z}^{L+1}_{i,j,m} / ||\mathbf{Z}^{L+1}_{i,j}||$$

The first-layer kernels $\mathbf{V}_m$ are initialized from a zero-mean Gaussian distribution with standard deviation 0.1 for CIFAR-10 and 0.001 for SVHN. The kernels of the higher layers $\mathbf{W}^l_m$ are initialized to the identity transformation $\mathbf{W}_{i',j',m',m} = \delta_{i',0} \cdot \delta_{j',0} \cdot \delta_{m',m}$, where δ is the Kronecker delta. The network is trained to minimize the logistic loss function $\mathcal{L} = \sum_n \log(\mathbf{Y}^n_{k(n)})$ and $k(n)$ is the true class of the nth element of the dataset. The parameters are not subject to explicit regularization. Training is performed via stochastic gradient descent with minibatches of size 128, learning rate 10^{-3}, and momentum 0.9:

$$\mathbf{g} = 0.9 \cdot \mathbf{g} + \sum_{n \in \text{minibatch}} \frac{\partial \mathcal{L}^n}{\partial \{\mathbf{V}, \mathbf{W}, \mathbf{b}\}} \quad ; \quad \{\mathbf{V}, \mathbf{W}, \mathbf{b}\} = \{\mathbf{V}, \mathbf{W}, \mathbf{b}\} - 10^{-3} \cdot \mathbf{g}$$

7.4 Experiments

7.4.1 Performance Evaluation

We first provide an overview of the model's performance at different sizes, with both untied and tied weights, in order to examine basic trends and compare with other current systems. For CIFAR-10, we tested the models using $M = 32, 64, 128, \text{or } 256$ feature maps per layer, and $L = 1, 2, 4, 8, \text{or } 16$ layers beyond the first. For SVHN, we used $M = 32, 64, 128, \text{or } 256$ feature maps and $L = 1, 2, 4, \text{or } 8$ layers beyond the first. That we were able to train networks at these large depths is due to the initialization of all W^l_m to the identity: this initially copies activations at the first layer up to the last layer, and gradients from the last layer to the first. Both untied and tied models had trouble learning with zero-centered Gaussian initializations at some of the larger depths.

Results are shown in Figs. 7.2 and 7.3. Here, we plot each condition on a grid according to numbers of feature maps and layers. To the right of each point, we show the test error (top) and training error (bottom). Contours show curves with a constant number of parameters: in the untied case, the number of parameters is determined by the number of feature maps and layers, while in the tied case it is determined solely by the number of feature maps; Section 7.4.2 examines the behavior along these lines in more detail.

First, we note that despite the simple architecture of our model, it still achieves competitive performance on both datasets, relative to other models that, like ours, do not use any image transformations or other regularizations such as dropout [58, 139], stochastic pooling [146] or maxout [40] (see Table 7.1). Thus our simplifications do not entail a departure from current methods in terms of performance.

We also see roughly how the numbers of layers, feature maps and parameters affect performance of these models at this range. In particular, increasing any of them tends to improve performance, particularly on the training set (a notable exception to CIFAR-10 at 16 layers in the tied case, which goes up slightly). We now examine the independent effects of each of these variables in detail.

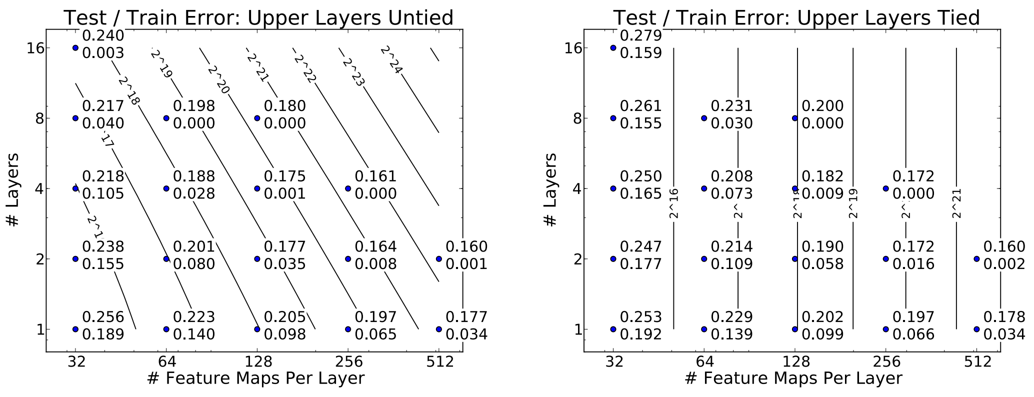


Figure 7.2: Classification performance on CIFAR-10 as a function of network size, for untied (left) and tied (right) models. Contours indicate lines along which the total number of parameters remains constant. The top number by each point is test error, the bottom training error.

7.4.2 Effects of the Numbers of Feature maps, Parameters and Layers

In a traditional untied convolutional network, the number of feature maps M, layers L and parameters P are interrelated: Increasing the number of feature maps or layers increases the total number of parameters in addition to the representational power gained by higher dimensionality (more feature maps) or greater nonlinearity (more layers). But by using the tied version of our model, we can investigate the effects of each of these

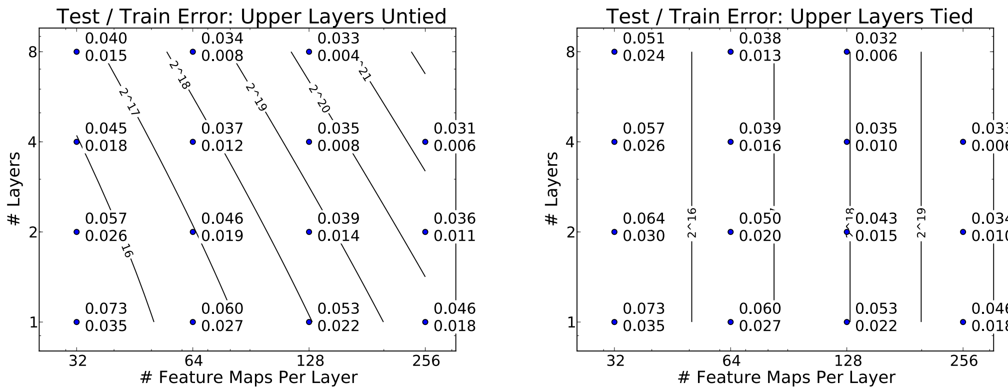


Figure 7.3: Classification performance on Street View House Numbers as a function of network size, for untied (left) and tied (right) models.

CIFAR-10	**Test error (%)**
Ours	16.0
Snoek *et al.* [119]	15.0
Ciresan *et al.* [12]	15.9
Hinton *et al.* [58]	16.6
Coates & Ng [13]	18.5

SVHN	**Test error (%)**
Ours	3.1
Zeiler & Fergus (max pool) [146]	3.8
Sermanet *et al.* [112]	4.9

Table 7.1: Comparison of our largest model architecture (measured by number of parameters) against other approaches that do not use data transformations or stochastic regularization methods.

three variables independently.

To accomplish this, we consider the following three cases, each of which we investigate with the described setup:

1. *Control for M and P, vary L*: Using the tied model (constant M and P), we evaluate performance for different numbers of layers L.

2. *Control for M and L, vary P*: Compare pairs of tied and untied models with the same numbers of feature maps M and layers L. The number of parameters P increases when going from tied to untied model for each pair.

3. *Control for P and L, vary M*: Compare pairs of untied and tied models with

the same number of parameters P and layers L. The number of feature maps M increases when going from the untied to tied model for each pair.

Note the number of parameters P is equal to the total number of independent weights and biases over all layers, including initial feature extraction and classification. This is given by the formula below for the untied model (for the tied case, substitute $L = 1$ regardless of the number of layers):

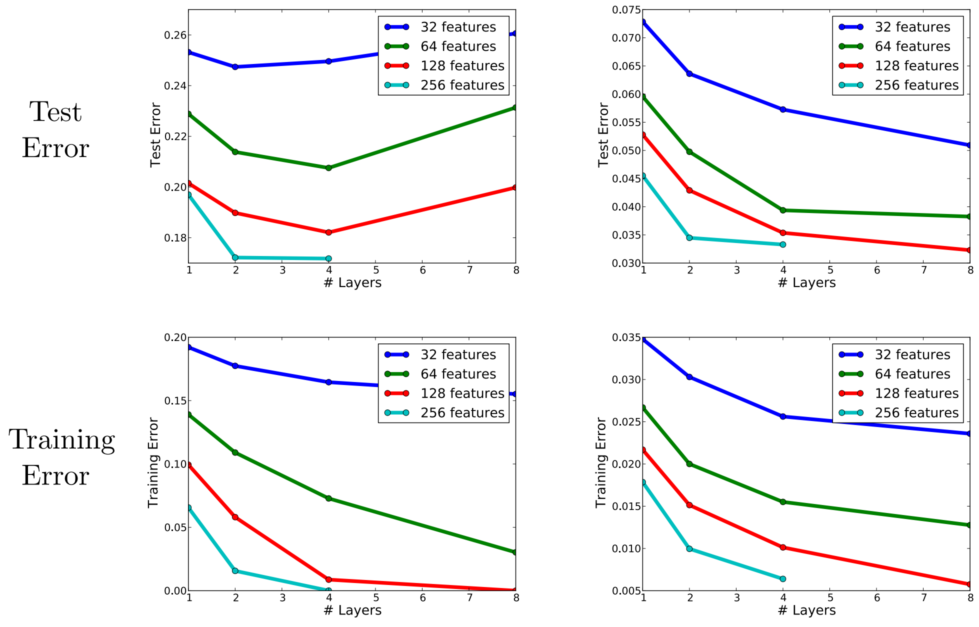
$$P = \underbrace{8 \cdot 8 \cdot 3 \cdot M}_{\text{(first layer)}} + \underbrace{3 \cdot 3 \cdot M^2 \cdot L}_{\text{(higher layers)}} + \underbrace{M \cdot (L+1)}_{\text{(biases)}} + \underbrace{64 \cdot M \cdot 10 + 10}_{\text{(classifier)}}$$

Case 1: Number of Layers

We examine the first of these cases in Fig. 7.4. Here, we plot classification performance at different numbers of layers using the tied model only, which controls for the number of parameters. A different curve is shown for different numbers of feature maps. For both CIFAR-10 and SVHN, performance gets better as the number of layers increases, although there is an upward tick at 8 layers for CIFAR-10 test error. The predominant cause of this appears to be overfitting, since the training error still goes down. At these depths, therefore, adding more layers alone tends to increase performance, even though no additional parameters are introduced. This is because additional layers allow the network to learn more complex functions by using more nonlinearities.

This conclusion is further supported by Fig. 7.5, which shows performance of the untied model according to numbers of parameters and layers. Note that vertical cross-sections of this figure correspond to the constant-parameter contours of Fig. 7.2. Here, we can also see that for any given number of parameters, the best performance is obtained with a deeper model. The exception to this is again the 8-layer models on CIFAR-10 test error, which suffer from overfitting.

Experiment 1a: Error by Layers and Features (tied model)



(a) CIFAR-10 (b) SVHN

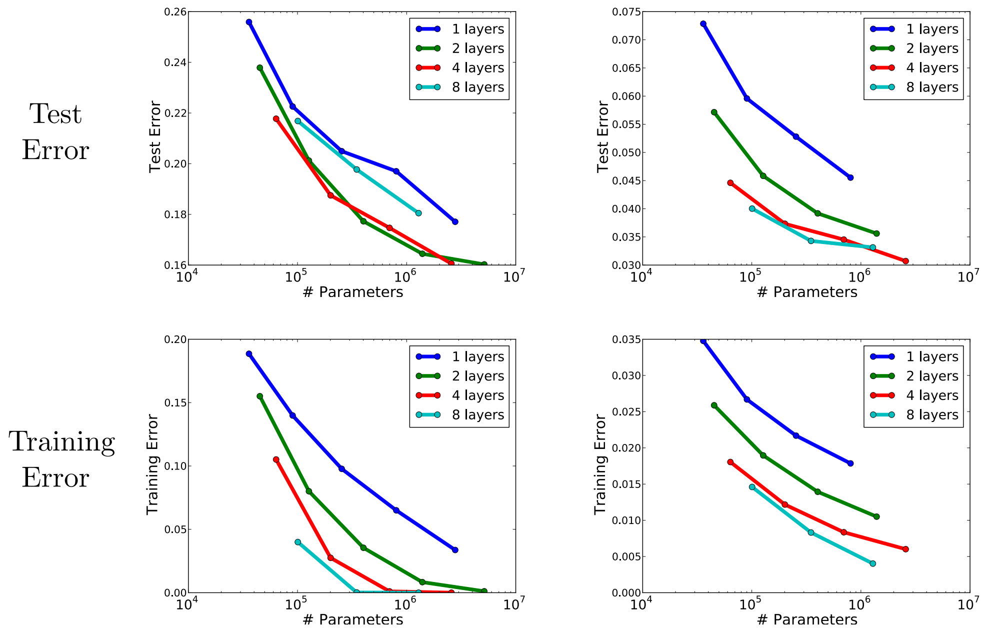
Figure 7.4: Comparison of classification error for different numbers of layers in the tied model. This controls for the number of parameters and features. We show results for both (a) CIFAR-10 and (b) SVHN datasets.

Case 2: Number of Parameters

To vary the number of parameters P while holding fixed the number of feature maps M and layers L, we consider pairs of tied and untied models where M and L remain the same within each pair. The number of parameters P is then greater for the untied model.

The result of this comparison is shown in Fig. 7.6. Each point corresponds to a model pair; we show classification performance of the tied model on the x axis, and performance of the untied model on the y axis. Since the points fall below the $y = x$ line, classification performance is better for the untied model than it is for the tied. This is not surprising, since the untied model has more total parameters and thus more flexibility. Note also that the two models converge to the same test performance as classification gets better — this is because for the largest numbers of L and M, both models have enough flexibility

Experiment 1b: Error by Parameters and Layers (untied model)



(a) CIFAR-10 (b) SVHN

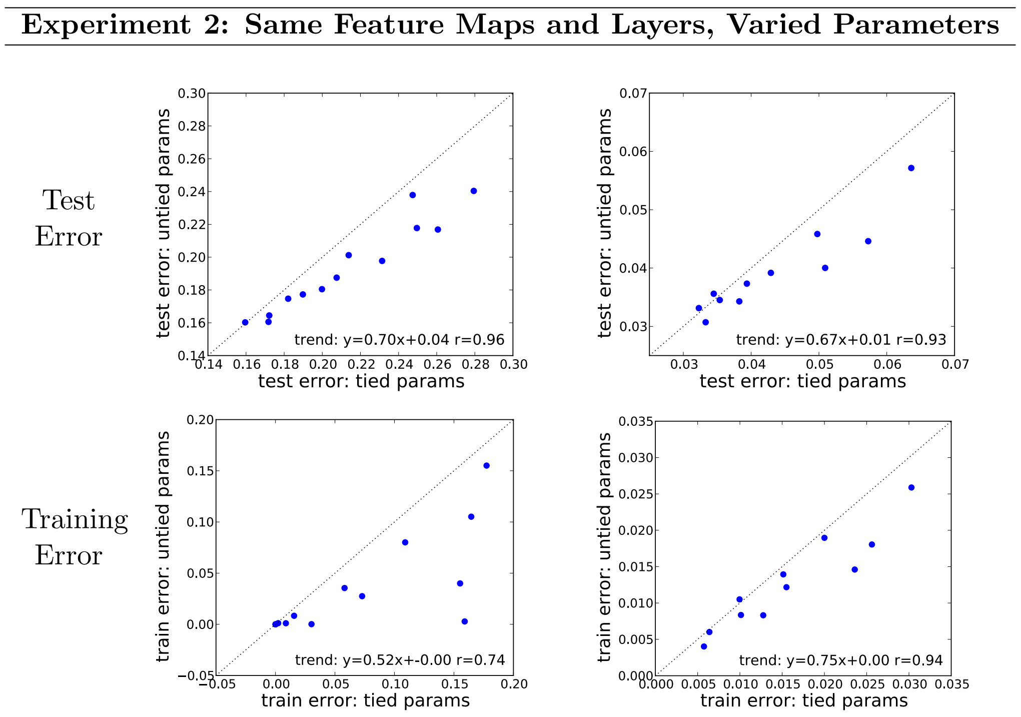
Figure 7.5: Comparison of classification error for different numbers of parameters in the untied model, for (a) CIFAR-10 and (b) SVHN datasets. Larger numbers of both parameters and layers help performance. In addition, for a fixed budget of parameters, allocating them in more layers is generally better (CIFAR-10 test error increases above 4 layers due to overfitting).

to achieve maximum test performance and begin to overfit.

Case 3: Number of Feature Maps

We now consider the third condition from above, the effect of varying the number of feature maps M while holding fixed the numbers of layers L and parameters P.

For a given L, we find model pairs whose numbers of parameters P are very close by varying the number of feature maps. For example, an untied model with $L = 3$ layers and $M = 71$ feature maps has $P = 195473$ parameters, while a tied model with $L = 3$ layers and $M = 108$ feature maps has $P = 195058$ parameters — a difference of only 0.2%. In this experiment, we randomly sampled model pairs having the same number of layers, and where the numbers of parameters were within 1.0% of each other. We



(a) CIFAR-10 (b) SVHN

Figure 7.6: Comparison of classification error between tied and untied models, controlling for the number of feature maps and layers. Linear regression coefficients are in the bottom-right corners.

considered models where the number of layers beyond the first was between 2 and 8, and the number of feature maps was between 16 and 256 (for CIFAR-10) or between 16 and 150 (for SVHN).

Fig. 7.7 shows the results. As before, we plot a point for each model pair, showing classification performance of the tied model on the x axis, and of the untied model on the y axis. This time, however, each pair has fixed P and L, and tied and untied models differ in their number of feature maps M. We find that despite the different numbers of feature maps, the tied and untied models perform about the same in each case. Thus, performance is determined by the number of parameters and layers, and is insensitive to the number of feature maps.

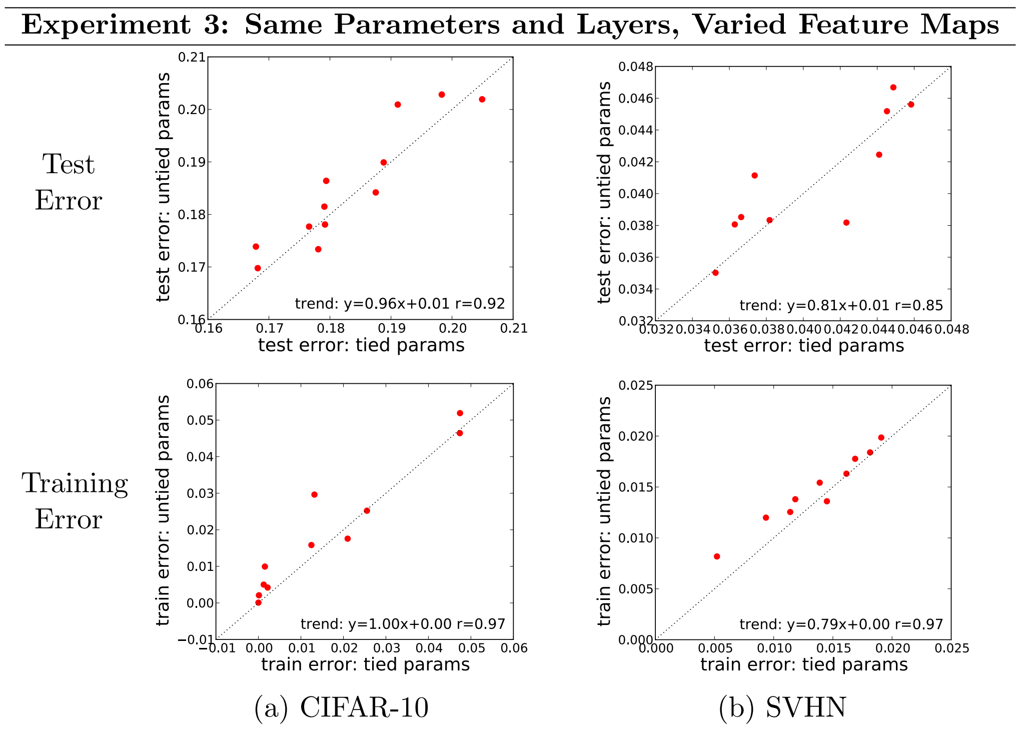


Figure 7.7: Comparison of classification error between tied and untied models, controlling for the number of parameters and layers. Linear regression coefficients in the bottom-right corners.

7.5 Discussion

Above we have demonstrated that while the numbers of layers and parameters each have clear effects on performance, the number of feature maps has little effect, once the number of parameters is taken into account. This is perhaps somewhat counterintuitive, as we might have expected the use of higher-dimensional representations to increase performance; instead we find that convolutional layers are insensitive to this size.

This observation is also consistent with Fig. 7.5: Allocating a fixed number of parameters across multiple layers tends to increase performance compared to putting them in few layers, even though this comes at the cost of decreasing the feature map dimension. This is precisely what one might expect if the number of feature maps had little effect compared to the number of layers.

Our analysis employed a special tied architecture and comes with some important caveats, however. First, while the tied architecture serves as a useful point of comparison leading to several interesting conclusions, it is new and thus its behaviors are still relatively unknown compared to the common untied models. This may particularly apply to models with a large number of layers ($L > 8$), or very small numbers of feature maps ($M < 16$), which have been left mostly unexamined. Second, our experiments all used a simplified architecture, with just one layer of pooling. While we believe the principles found in our experiments are likely to apply in more complex cases as well, this is unclear and requires further investigation to confirm. Nevertheless, many recent systems make heavy use of unpooled convolution stacks, and our investigation is particularly applicable in this case.

The results we have presented provide empirical confirmation within the context of convolutional layers that increasing layers alone can yield performance benefits (Experiment 1a). They also indicate that filter parameters may be best allocated in multilayer stacks (Experiments 1b and 3), even at the expense of having fewer feature maps. In conjunction with this, we find the feature map dimension itself has little effect on convolutional layers' performance, with most sizing effects coming from the numbers of layers and parameters (Experiments 2 and 3). Thus, focus is best placed on these variables when searching for model architectures.

Chapter 8

Convolutional Unsupervised Methods

The work presented in this chapter is currently unpublished. Section 8.2 is a joint work with Rob Fergus; Sections 8.3 and 8.4 are collaborations with Jason Rolfe, Rob Fergus and Yann LeCun.

8.1 Introduction

In this chapter we explore three convolutional unsupervised methods. First, we describe a Convolutional LISTA Autoencoder that approximates convolutional sparse coding using a fast feed-forward network. Second, we apply an entropy cost that causes convolutional features to organize into two different types, prototype templates and deformations, that factors many higher-level edge features away from the lowest pixel-level information necessary for reconstruction. Third, we outline how ZCA whitening can be adapted for convolutional local whitening.

8.2 Convolutional LISTA Autoencoder

The Convolutional LISTA Autoencoder combines ideas from Gregor *et al.*, "Learned ISTA" [45] and Zeiler *et al.*, "Deconvolutional Networks" [147, 148]. We first adapt the LISTA network described in [45] to be convolutional, and use it as the encoder half of an autoencoder: Rather than training the encoder to predict true sparse codes found by an additional and expensive sparse-coding training phase, we instead train using reconstruction error directly. We then stack autoencoder modules to learn successively higher-level features, similar to a Deconvolutional network, but using fast, feed-forward models at both training and test time.

8.2.1 Background

Before describing our LISTA autoencoder network, we first review the basics of ISTA and LISTA. Given an input x and (convolutional) filter dictionary W, the Iterative Shrinkage and Thresholding Algorithm (ISTA) [18, 105, 5] finds codes z that minimize the sparse error $\frac{1}{2}||W^T * z - x||_2^2 + \lambda|z|_1$, producing codes that are both sparse (have few nonzero elements) and reconstruct the input well. The procedure is based on a gradient descent on z:

$$\begin{aligned} z_0 &= 0 \\ z_t &= sh_\alpha(z_{t-1} + \eta(W * x - S * z_{t-1})) \end{aligned} \tag{8.1}$$

where $S = W * W^T$ is the lateral inhibition matrix of kernel similarities, and sh_α is a shrink and threshold operation, *i.e.* $sh_\alpha(z) = \max(0, z - \alpha)$ if the codes z are constrained to be nonnegative. The scalars α and η come from the strength λ of the sparsity term and the descent rate, and can be determined analytically.

Each step of ISTA adds to the code z a small amount of the input filtered by W, thus increasing the activations of matching filters, while at the same time subtracting out activations of features similar to each another via the lateral inhibitions S (equal to the all-to-all similarities). This effects an explaining-away mechanism to produce sparse activations, however often takes a fairly large (*e.g.* 100 or more) number of iterations.

Learned ISTA (LISTA) [45] modifies this by decoupling the inhibition weights S from the dictionary W, allowing each to be trained separately. The inference procedure Eqn. 8.1 is unrolled for fixed small number of steps (*e.g.* five), and trained to predict the true sparse codes z of a training dataset via backpropagation. The network can use the weight relaxations to learn to "shortcut" many of the steps needed for inference, so that it quickly produces codes with few iterations. This comes at a cost of restricted generalizability — the inference model is trained on a training set and may perform arbitrarily on unrepresented inputs — however, we generally are only interested in a small subset of the full unrestricted space, *e.g.* the set of natural images versus the set of all real-valued images, so this is often a negligible tradeoff.

8.2.2 Single Layer

Our convolutional LISTA autoencoder, depicted in Fig. 8.1(a), uses a LISTA network as the encoder half of an autoencoder. Rather than train a LISTA model to predict true sparse codes, we further extend the network with separate encoder and decoder weights W_e and W_d, in addition to inhibition weights S, and train it end-to-end using a reconstruction objective. As in LISTA, our model produces a code z_n from an input x by applying an encoding matrix W_e, followed by successive applications of an inhibition matrix S. The decoder then performs a linear convolution back to the input space using the decoding weights W_d:

$$\begin{aligned} z_0 &= 0 \\ z_t &= \max(0, z_{t-1} + S * z_{t-1} + W_e * x - b) \qquad (8.2) \\ x' &= W_d^T * z_n \end{aligned}$$

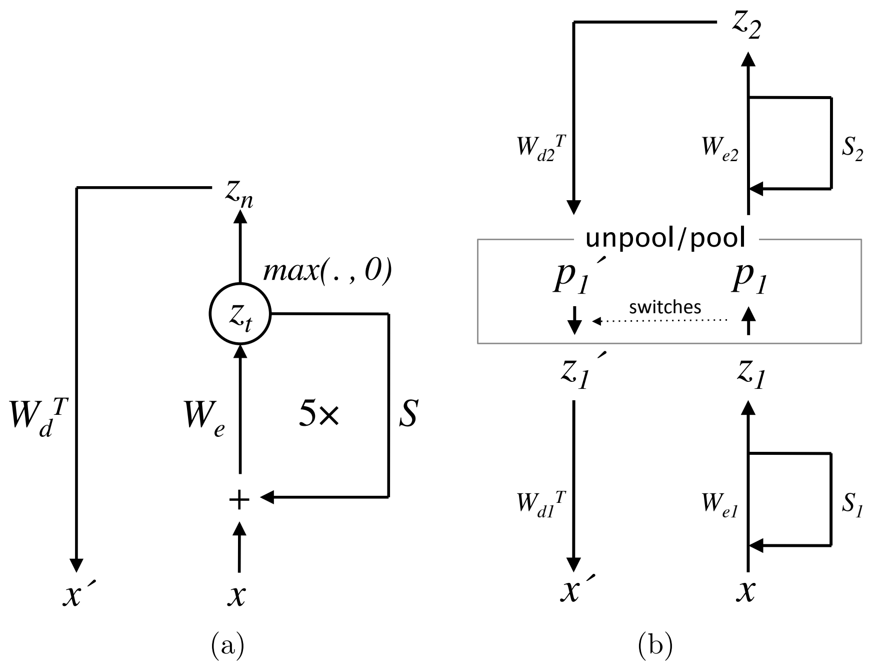


Figure 8.1: Convolutional LISTA Autoencoder architecture. (*a*) Single layer. (*b*) A stack of two layers, with max-pooling switches transfered between encoder and decoder.

The operations perfomormed by this network are thus similar to ISTA sparse coding inference, but with separately trainable encoder, decoder and lateral inhibition kernels.

For training, we use a l_2 reconstruction and l_1 sparsity-inducing loss, $L(x) = \frac{1}{2}||x' - x||_2^2 + \lambda|z_n|_1$. We also constrain the decoder weights so that each filter is unit norm.

Another variation that we also tried instead uses an inhibition kernel S_p to project back to the input (pixel) space, rather than working between features in the code space. This generally has fewer connections, but is faster, and the inhibition kernels can be visualized directly. In this case, the network is defined by

$$\begin{aligned} z_0 &= 0 \\ z_t &= \max(0, z_{t-1} + W_e * (S_p * z_{t-1} + x) - b) \end{aligned}$$

Filters learned using $n = 5$ iterations on MNIST and CIFAR-10 are shown in Fig. 8.2.

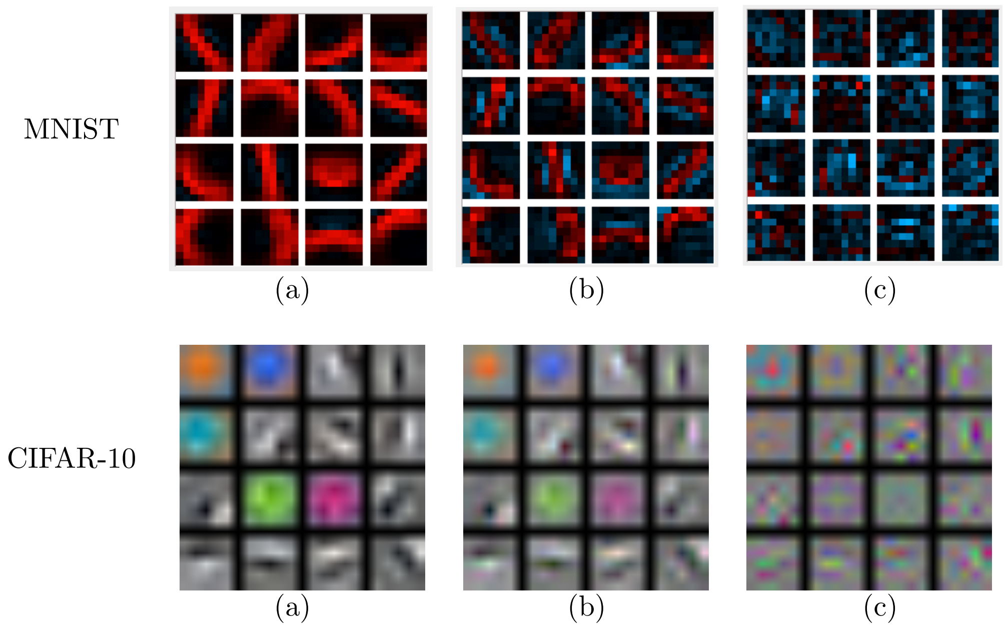


Figure 8.2: Filters learned using MNIST and CIFAR-10 (for MNIST, red is positive and blue negative). (*a*) Decoder W_d, (*b*) Encoder W_e, (*c*) Inhibition S_p. The encoder has negative "shadows" around positive stroke centers that help turn off the activation soon as the filter becomes unaligned; the inhibition kernels subtract out the stroke as it is explained.

8.2.3 Stacking Multiple Layers

We can stack multiple LISTA autoencoders together, combined with spatial max-pooling layers, to form a deep autoencoder network (Fig. 8.1(b)). This stacking is similar to that in [147, 148], but uses LISTA autoencoders instead of iterative sparse coding inference. Since the feed-forward encoder network performs many fewer iterations (about one one-hundredth the number), it is much faster. Yet since it has a trained inference network, it is still able to produce reasonable representations and filters.

Specifically, we first train a convolutional LISTA autoencoder as described in the above section on the input images, using five z-iterations. This produces first-layer codes z^1, on which we perform spatial max-pooling to form pooled activations p^1. We then sequentially train a stack of two additional autoencoders and pooling, by minimizing the reconstruction error of the pooled maps immediately below. This forms a stack of three autoencoder applications.

	Classification Error
Layer 1	0.42
Layer 2	0.37
Layer 3	0.34

Table 8.1: Classification error on CIFAR-10 using features from each layer of a Convolutional LISTA Autoencoder. We train a single-layer softmax on top of fixed features from each layer to evaluate their relation with semantic class labels. By comparison, a similar fully-supervised network achieves an error of around 0.15.

We show reconstructions from the top 10 activations of each feature map in Figures 8.3 and 8.4, when trained on CIFAR-10 with ZCA whitening. We see that the network is able to capture extended edges, as well as corners and edge or color combinations, at a fraction of the cost of a similarly stacked Deconvolutional network.

Fixing the features at each layer, we also trained a 10-unit softmax classifier on top of each layer's features to check their relationship to predicting semantic labels. Results are in Table 8.1: Classification performance improves between the first and second layers, and the third layer is slightly better yet, though not by much.

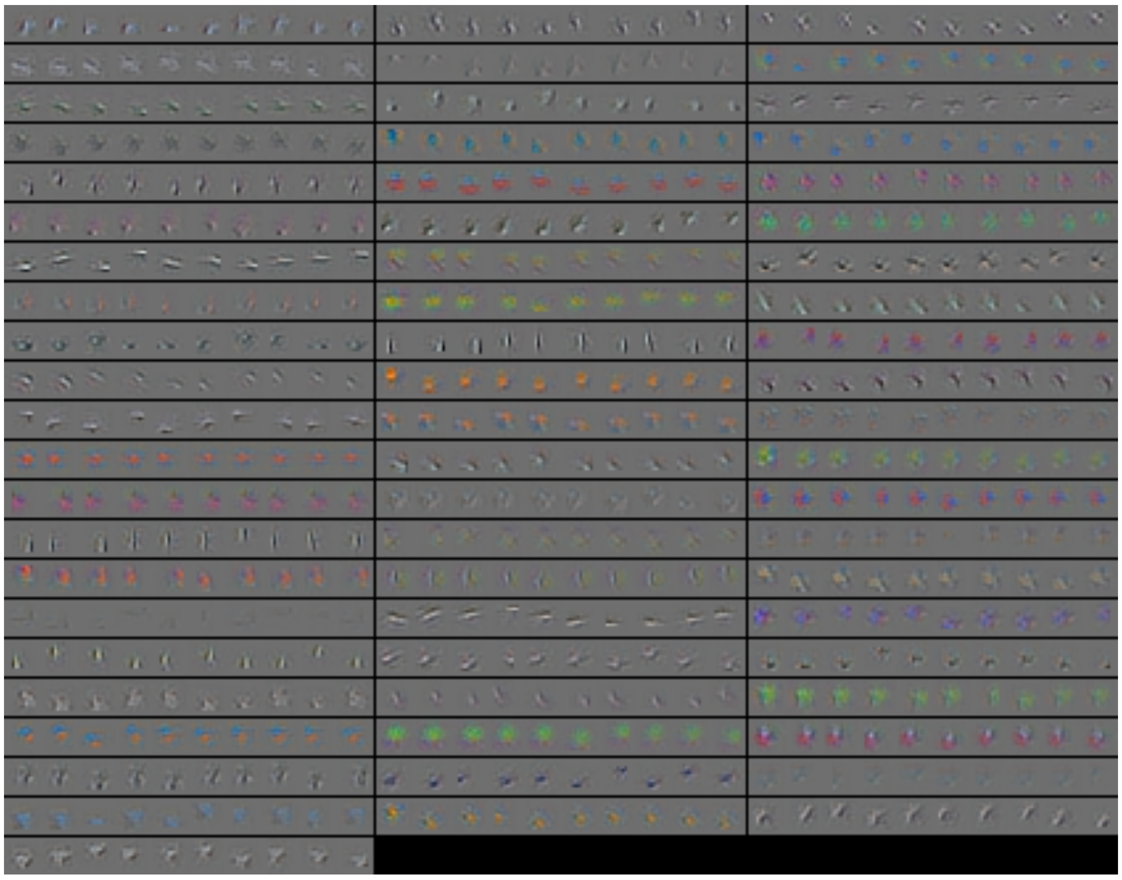


Figure 8.3: Layer 2 reconstructions. We show reconstructions for the top 10 activations for each second layer unit across the test set. The decoder unpools using the pooling locations determined by the encoder.

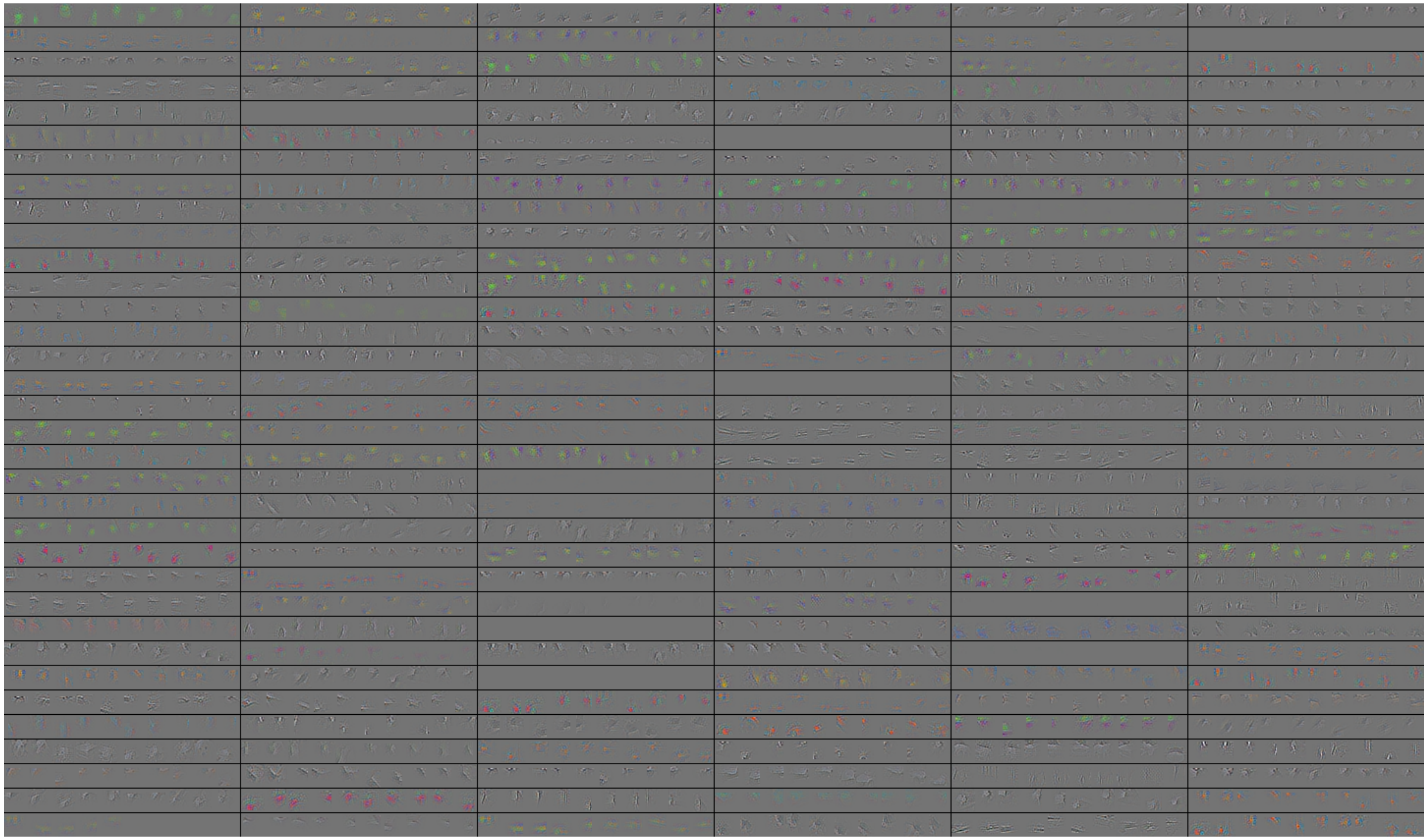


Figure 8.4: Layer 3 reconstructions. We show reconstructions for the top 10 activations for each third layer unit across the test set. The decoder unpools through two pooling layers, using the locations determined in the encoder pooling layers.

8.3 Entropy Prototypes

Above, we described a convolutional LISTA autoencoder that learns approximate sparse codes with an objective combining l_2 reconstruction and l_1 activation norm. A related model, although not convolutional, is that of Rolfe *et al.* [103], which adds a classification objective in addition to these two. They find that with all three objective terms, weights are learned so that the activation units organize themselves into "prototypes" and "deformation" units. Prototypes capture a mean template for each class, while deformations edit the template to better reconstruct the input. The prototype units display several characteristics that differentiate them from the deformations: They are fewer in number, activate higher in magnitude, and turn on later in the network stack, indicating that they form a higher-level semantic understanding.

In this section, we describe a system that learns units with similar characteristics convolutionally and in a purely unsupervised fashion, by substituting an entropy term for the classification cost in [103]. Using this entropy cost, we can find convolutional "prototypes" and "deformations"; applied with strided convolutions, these can factor out larger low-level structures from small pixel details at each input window.

Given an input x, we use the convolutional LISTA autoencoder described in Section 8.2.2 to find a code z and reconstruction x'; however, the loss including entropy term is now

$$L(x) = \frac{1}{2}||x' - x||_2^2 + \lambda|z|_1 + \mu H\left(\frac{e^z}{\sum_k e_k^z}\right)$$

$$\text{where} \quad H(y) = -\sum_{i,j,k} y_{ijk} \log y_{ijk}$$

That is, the additional cost tries to minimize the entropy of the features (indexed by k) at each spatial location i, j. This happens when each spatial location has a strong activation (prototype) and possibly several weak ones (deformations). It parallels a classification cross-entropy loss, substituting the feature activations themselves for the ground-truth class labels.

We also bound the l_2 norm of the encoder weights W_e, and normalize z across the feature dimension at each location before passing it to the softmax in the entropy cost. These prevent the entropy term being satisfied by a degenerate case where a very large activation is always placed on a single z unit at each spatial location (the corresponding decoder for that unit is a self-canceling convolution kernel with checkers of positive and negative that has zero net contribution over the image).

We trained this model on both MNIST and NORB. In each case, we first pretrained the autoencoder using just the reconstruction and sparsity terms, then added in the entropy cost. We used 8x8 kernels with a stride of 4 for the convolutions, so that in fact they encoded overlapping tiles. NORB was preprocessed using convolutional ZCA (described in the next section).

We show the filters the system learns in Fig. 8.5, sorted descending by mean nonzero activation. For MNIST, the first few units (corresponding to prototypes) are all-positive with uniform thickness. The deformation units that follow have both positive and negative values, which often edit the stroke thickness. For NORB, the prototypes are simple edges, while deformation units are more complex. We also plot the mean nonzero activations for each unit in Fig. 8.6. For each dataset, there is a distinctive split between prototype units, which have a large activation, and the deformations, which tend to be smaller.

Reconstructions are shown in Figures 8.7 and 8.8. We show the pixel-space reconstruction using only prototype units (chosen using a threshold on the mean nonzero activation), only deformation units, and both. For MNIST, strokes tend to become more uniform in thickness for prototype-only reconstruction, and are more limited in orientation due to the more discrete set of prototypes. For NORB, prototype-only reconstructions also use a more limited set of edges, appearing similar to line drawings. Thus many pixel-level details are explained by the deformation units, allowing essential templates to be factored into the prototypes.

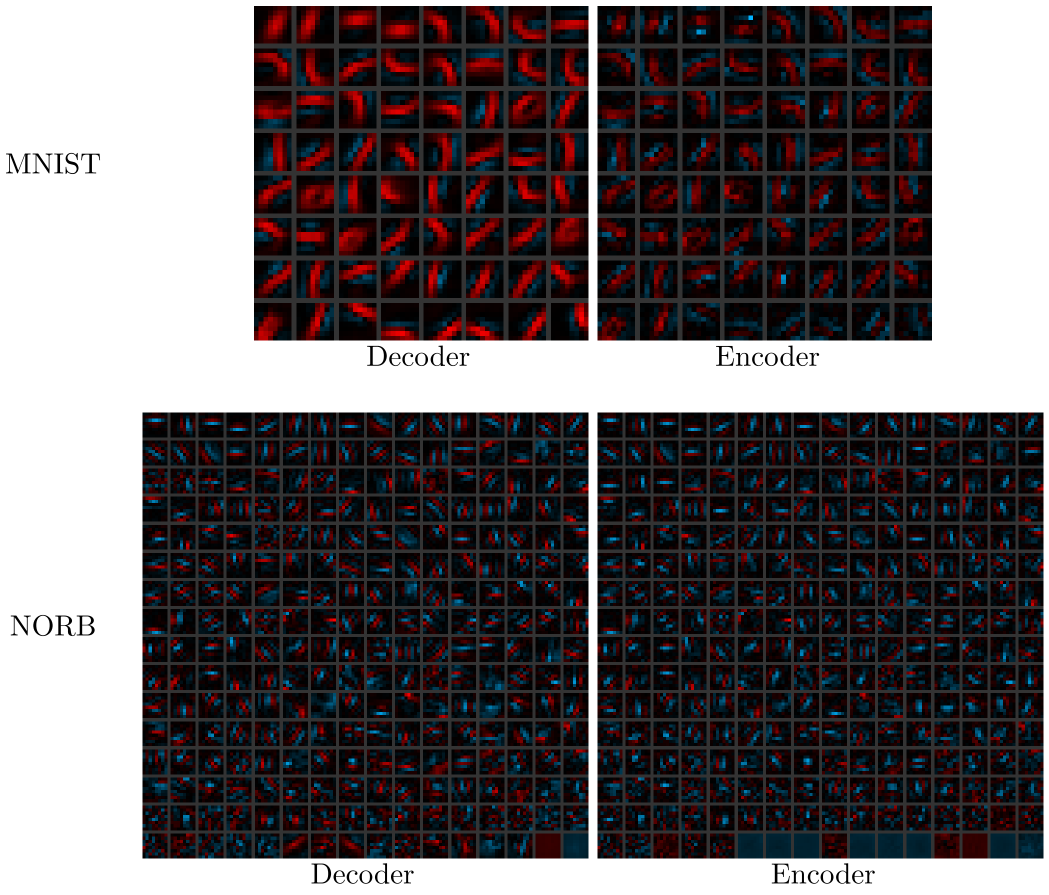


Figure 8.5: Autoencoder weights trained with per-location entropy cost, sorted descending by mean nonzero activation. Prototype units appear in the beginning. For MNIST, these are all-positive with around uniform thickness; the deformation units that follow have both positive and negative values, often editing the stroke thickness.

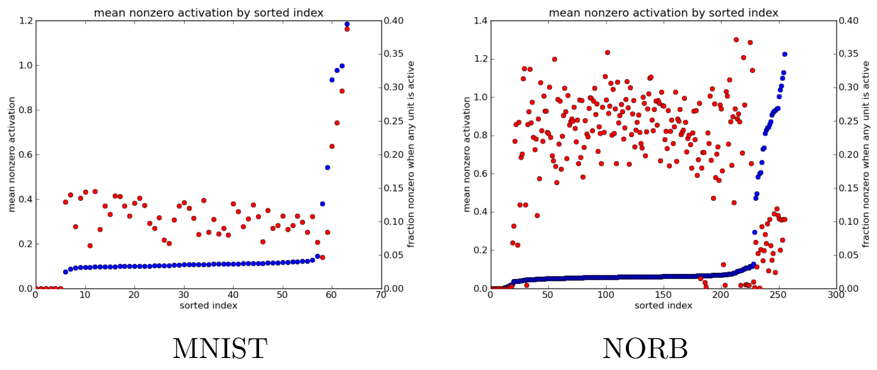


Figure 8.6: Mean nonzero activations: For each unit, we plot the mean of its activation out of times it is nonzero (blue, left axis). We also show the fraction of the time the unit is active when any unit at the same location is active (red, right axis). Units are sorted according to mean nonzero activation. We find that there are two distinct kinds of units, corresponding to prototypes and deformations.

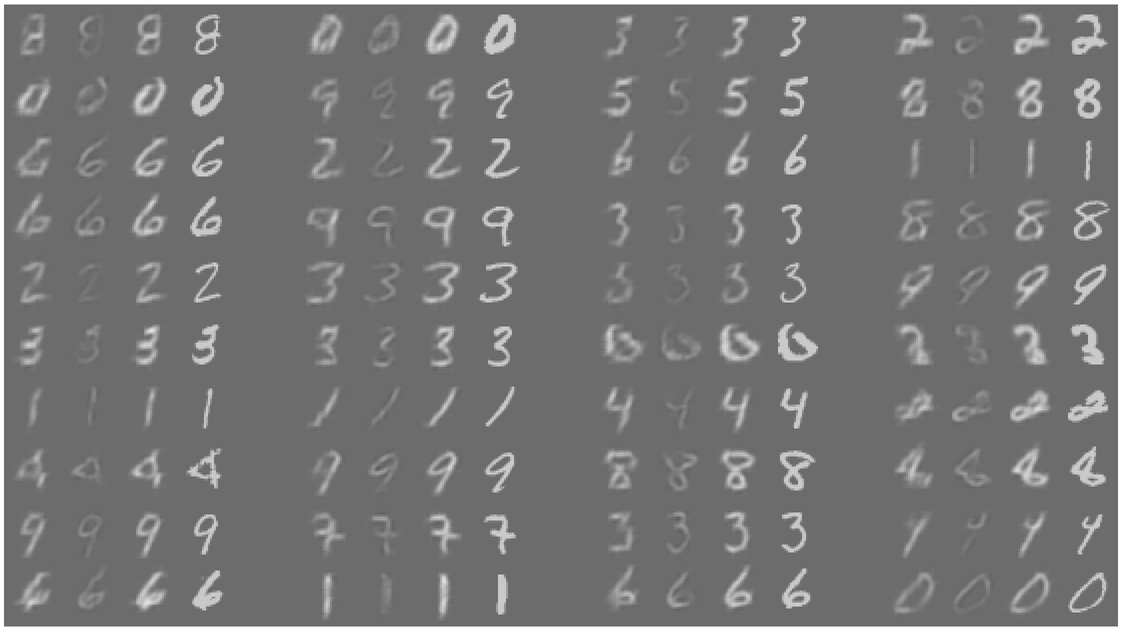


Figure 8.7: Reconstructions of MNIST with per-location entropy cost. For each image, we show (*i*) reconstruction using prototypes, (*ii*) using only deformations, (*iii*) reconstruction with all units, (*iv*) original input image.

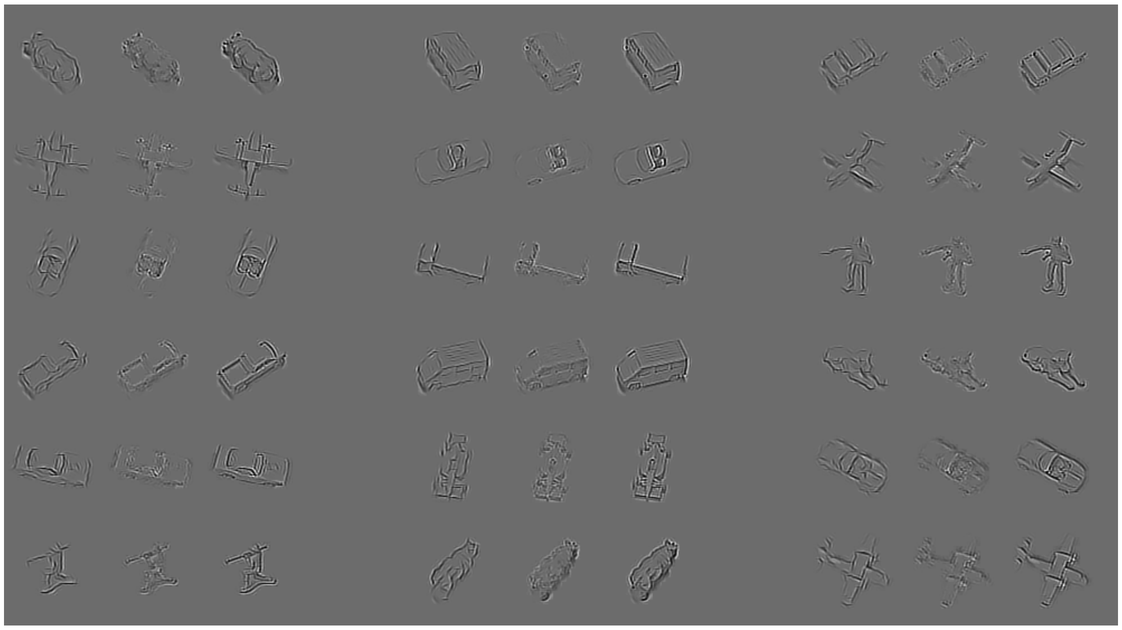


Figure 8.8: Reconstructions of NORB with per-location entropy cost. For each image, we show (*i*) reconstruction using prototypes, (*ii*) using only deformations, (*iii*) reconstruction with all units.

8.4 Convolutional ZCA Whitening

Whitening transformations aim to equalize frequencies in an image, so that low frequencies do not dominate reconstruction costs; this enables unsupervised methods based on reconstruction to learn codes that represent the full spectrum of the image, rather than just constant regions. An effective whitening transformation is ZCA [72], which constructs a linear transformation that explicitly scales each singular value to be 1. However, this transform is applied to the entire image, and requires the covariance between all pixels. While reasonable for small images such as the 32x32 CIFAR, this is expensive for larger data.

A simple adaptation of ZCA to convolutional application is to find the ZCA whitening transformation for a sample of local image patches across the dataset, and then apply this transform to every patch in a larger image. We then use the center pixel of each ZCA'd patch to create the conv-ZCA output image. The operations of applying local ZCA and selecting the center pixel can be combined into a single convolution kernel, resulting in the following algorithm (explained using RGB inputs and 9x9 kernel):

1. Sample $\sim$10M random 9x9 image patches (each with 3 colors)
2. Perform PCA on these to get eigenvectors V and eigenvalues D.
3. Optionally remove small eigenvalues, so V has shape [*npca* x 3 x 9 x 9].
4. Construct the whitening kernel k:
   for each pair of colors (c_i, c_j), set $k[c_j, c_i, :, :] = V[:, c_j, x_0, y_0]^T D^{-1/2} V[:, c_i, :, :]$
   where (x_0, y_0) is the center pixel location (*e.g.* (5,5) for a 9x9 kernel)

Note the matrix multiplies in step 4 work on the PCA dimension of V and are "broadcasted" over each spatial component of the V on the right (which maps the input to the eigenspace).

We show the top 100 singular values for a random sample of 1M patches both before and after convolutional ZCA processing in Fig. 8.9, using the RGB part of the NYU Depth v2 dataset, rescaled to 320x240. We display plots for both 9x9 and 15x15 patches, but used the same 9x9 ZCA kernel in each case. The kernel was computed using 10M 9x9 patch samples and keeping 50 PCA components.

Whitening kernels found using this method on NORB (grayscale, 96x96) and NYU Depth v2 (using RGB only, rescaled to 320x240) are shown in Fig. 8.10. In both cases we use 9x9 kernels and keep 50 PCA components. We show transformed images in Fig. 8.11.

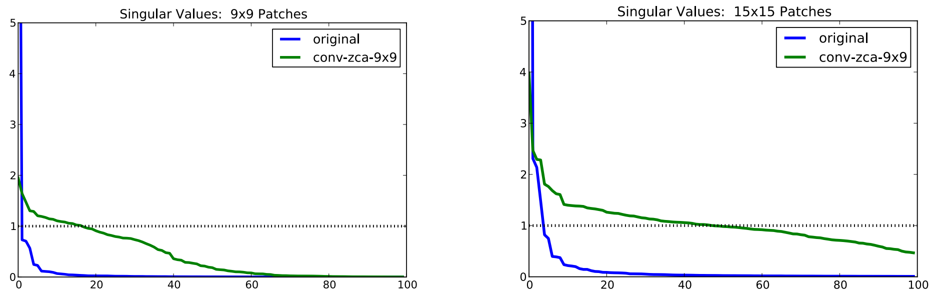


Figure 8.9: Top 100 singular values for 9x9 and 15x15 patches, before and after applying convolutional ZCA with 9x9 filters and 50 PCA components, using the RGB part of NYU Depth v2 rescaled to 320x240.

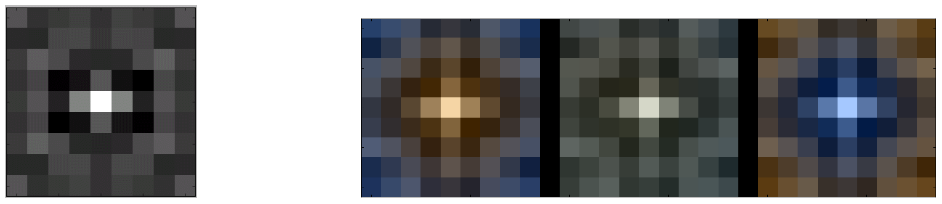


Figure 8.10: Convolutional ZCA whitening kernels, trained on NORB (left) and the RGB part of NYU Depth v2 (right).

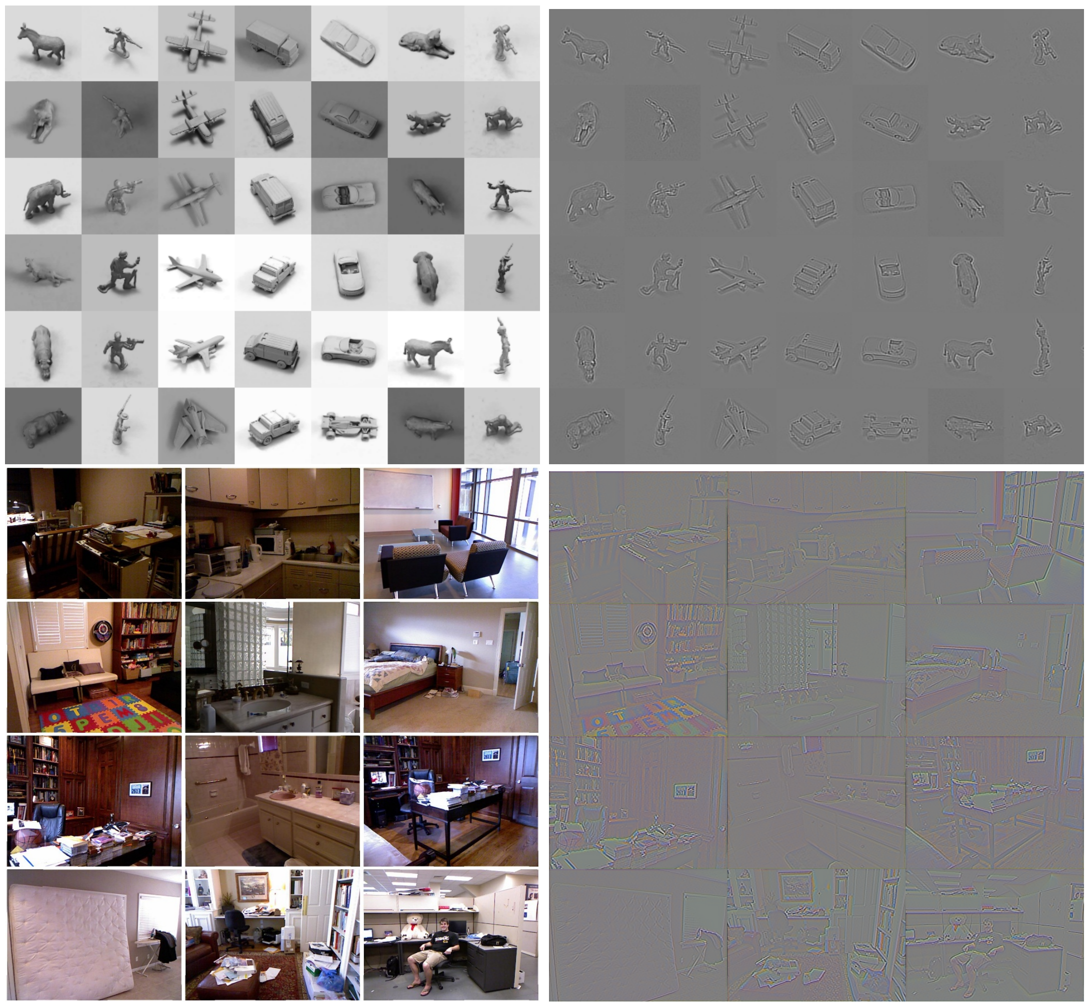


Figure 8.11: Example images before and after convolutional ZCA processing, for NORB (top) and NYU Depth RGB (bottom).

Chapter 9

Conclusion

In this thesis we have developed convolutional network models that infer 2D pixel maps for a variety of tasks, as well as exploring several related systems. In particular, we have contributed the following.

1. We introduce methods to learn datapoint weights within kNN classification, and use context to augment rare class instances (Chapter 3).

2. We develop a new method to restore images corrupted by dirt and rain on an intervening glass pane using a local convolutional network, and show that training convolutionally has the effect of decorrelating output patches. (Chapter 4).

3. We introduce a new system for predicting depth from a single image that integrates global and local views using a series of convolutional networks applied at different scales; we also introduce a scale-invariant error to handle much of the ambiguity fundamentally present in this task (Chapter 5).

4. We develop a more general multiscale convolutional network model that can be easily adapted to predict many types of 2D outputs effectively from a single image, and apply it to predict depth, surface normals and semantic labels. Our model uses a global-scale network whose field of view includes the entire image area to find low-

resolution feature maps, then refines predictions through a series of progressively more local scales. (Chapter 6).

5. We investigate the independent effects of the numbers of layers, parameters and feature maps by employing a recursive classification network, and find that higher depth alone can result in higher performance while the number of feature maps is less critical (Chapter 7).

6. We introduce Convolutional LISTA Autoencoders, which are computationally inexpensive feed-forward emulations of deconvolutional networks, as well as an low-entropy objective that factors prototype template features away from reconstruction details. We also construct a convolutional ZCA whitening operation that can be applied using a single convolution kernel. (Chapter 8).

While the systems we presented are able to tackle a variety of problems, there are several limitations and possible directions for future work.

Firstly, our systems require a relatively large amount of densely labeled data for training. While this is fairly inexpensive to acquire for depth and surface normals, it is more difficult to obtain for semantic tasks, often requiring detailed human annotations. Some of this might be relieved by handling sparser target maps, where many pixels are unlabeled. While we can handle relatively small unlabeled regions by excluding them from the training loss, it is unclear how well our methods would perform if most of the data is unlabeled. In particular, there may be a problem in recognizing object boundaries, since the precise boundary may not be available in the label map. Furthermore, not all unlabeled regions can used directly as negative examples, since the model will eventually learn to predict correct positives at these locations.

In addition, our models have only a limited interchange between between top-down and bottom-up information. Our multiscale network includes top-down information influencing bottom-up flow: The network looks back to the original image when refining the output of coarser layers, thus the coarse prediction is recombined with a bottom-up signal from the image to produce each finer-scale prediction. However, the new finer-scale prediction cannot then be used to influence the coarser scale — there is no continual circulation between the scales and layers. Such signals might help disambiguate cases with multiple interpretations, and enable the system to predict each individually, rather than an average that splits the difference in error. For example, Deep Boltzmann Machines can achieve this through iterative inference [106], settling on single interpretations rather than averaging between them. Such methods may help in similar ways for our case.

Another possible extension is to use losses different from pixelwise accuracy. Pixelwise loss leads to predictions that essentially average nearby plausible outputs in pixel space. Other losses might push these averages into a more complex space, leading to qualitatively different errors. For instance, adversarial networks [39] have very recently been used to generate small images. One might imagine applying this as a loss, feeding the output of our network concatenated with the RGB input to a second discriminator network, which attempts to distinguish it from the true data. Our network might then average over different outputs not in pixel-space, but in "convolutional-network-space".

Additionally, it may be possible to use data from device sensors as a means for "unsupervised" feature learning. For instance, depth maps can be captured automatically, at less expense than hand-annotated labels, yet predicting them still requires extracting many relevant features. These representations may be able to generalize to other tasks better than features found by reconstruction (although worse than labels created with the new task in mind). Learning from sensor data may fall between "supervised" learning with human annotations, and "unsupervised" learning in which only the inputs are available.

Lastly, it is also possible to extend convolutional networks to domains beyond even images. There has already been much work using ConvNets for speech [57], text [14, 15] and video [127], and the idea of a convolution also can even be generalized to non-grid topologies [8], showing promise for a range of other graphs such as 3D meshes, meteorological data, or network systems. Generalizing multi-scale networks in a similar way may be effective for these areas as well.

Bibliography

[1] G. Alain and Y. Bengio. What regularized auto-encoders learn from the data generating distribution. *CoRR*, abs/1211.4246, 2014.

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

[3] P. Baldi and K. Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, 2(1):53–58, 1989.

[4] P. Barnum, S. Narasimhan, and K. Takeo. Analysis of rain and snow in frequency space. *IJCV*, 86(2):256–274, 2010.

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

[6] O. Boiman, E. Shechtman, and M. Irani. In defense of nearest-neighbor based image classification. In *CVPR*, 2008.

[7] Y.-L. Boureau, J. Ponce, and Y. LeCun. A theoretical analysis of feature pooling in visual recognition. In *ICML*, 2010.

[8] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *ICLR*, 2014.

[9] H. Burger, C. Schuler, and S. Harmeling. Image denoising: Can plain neural networks compete with BM3D? In *CVPR*, 2012.

[10] H. Burger, C. Schuler, and S. Harmeling. Image denoising with multi-layer perceptrons, part 2: training trade-offs and analysis of their mechanisms. In *arXiv preprint arXiv:1211.1552*, 2012.

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

[12] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *IJCAI*, 2011.

[13] A. Coates and A. Y. Ng. The importance of encoding versus training with sparse coding and vector quantization. In *ICML*, volume 8, pages 921–928, 2011.

[14] R. Collobert. Deep learning for efficient discriminative parsing. In *AISTATS*, 2011.

[15] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537, 2011.

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

[17] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Image denoising with block-matching and 3D filtering. In *Proc. SPIE Electronic Imaging*, 2006.

[18] I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on pure and applied mathematics*, 2004.

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

[20] P. Devijver and J. Kittler. *Pattern Recognition. A Statistical Approach.* Prentice Hall, 1992.

[21] B. Dong, H. Ji, J. Li, Z. Shen, and Y. Xu. Wavelet frame based blind image inpainting. *Applied and Comp'l Harmonic Analysis*, 32(2):268–279, 2011.

[22] D. Eigen and R. Fergus. Nonparametric image parsing using adaptive neighbor sets. In *CVPR*, 2012.

[23] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. *arXiv preprint arXiv:1411.4734*, 2014.

[24] D. Eigen, D. Krishnan, and R. Fergus. Restoring an image taken through a window covered with dirt or rain. In *ICCV*, 2013.

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

[26] D. Eigen, J. Rolfe, R. Fergus, and Y. LeCun. Understanding deep architectures using a recursive convolutional network. *ICLR Workshop*, 2014.

[27] M. Elad and M. Aharon. Image denoising via learned dictionaries and sparse representation. In *CVPR*, 2006.

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

[29] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1627–1645, 2010.

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

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

[32] A. Frome, Y. Singer, and J. Malik. Image retrieval and classification using local distance functions. In *NIPS*, 2006.

[33] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 1980.

[34] K. Garg and S. Nayar. Detection and removal of rain from videos. In *CVPR*, pages 528–535, 2004.

[35] P. V. Gehler and S. Nowozin. On feature combination for multiclass object classification. In *ICCV*, 2009.

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

[37] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier networks. In *AISTATS*, volume 15, pages 315–323, 2011.

[38] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood components analysis. 2004.

[39] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, 2014.

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

[41] S. Gould, R. Fulton, and D. Koller. Decomposing a scene into geometric and semnatically consistent regions. In *CVPR*, 2009.

[42] S. Gould, J. Rodgers, D. Cohen, G. Elidan, and D. Koller. Multi-class segmentation with relative location prior. *IJCV*, 2008.

[43] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, and J. Schmidhuber. A novel connectionist system for improved unconstrained handwriting recognition. *PAMI*, 31(5), 2009.

[44] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *ICASSP*, 2013.

[45] K. Gregor and Y. LeCun. Learning fast approximations of sparse coding. In *ICML*, 2010.

[46] J. Gu, R. Ramamoorthi, P. Belhumeur, and S. Nayar. Dirty Glass: Rendering Contamination on Transparent Surfaces. In *Eurographics*, Jun 2007.

[47] J. Gu, R. Ramamoorthi, P. Belhumeur, and S. Nayar. Removing Image Artifacts Due to Dirty Camera Lenses and Thin Occluders. *SIGGRAPH Asia*, Dec 2009.

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

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

[50] R. Hadsell, P. Sermanet, J. Ben, A. Erkan, M. Scoffier, K. Kavukcuoglu, U. Muller, and Y. LeCun. Learning long-range vision for autonomous off-road driving. *Journal of Field Robotics*, 26(2):120–144, 2009.

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

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

[53] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[54] G. Heitz, S. Gould, A. Saxena, and D. Koller. Cascaded classifcation models: Combining models for holistic scene understanding. In *NIPS*, 2008.

[55] G. Heitz and D. Koller. Learning spatial context: using stuff to find things. In *CVPR*, 2008.

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

[57] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[58] G. Hinton, N. Srivastave, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580*, 2012.

[59] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

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

[61] D. Hoiem, A. Efros, and M. Hebert. Closing the loop on scene interpretation. In *CVPR*, 2008.

[62] D. Hoiem, A. A. Efros, and M. Hebert. Automatic photo pop-up. In *ACM SIGGRAPH*, pages 577–584, 2005.

[63] G. B. Huang and V. Jain. Deep and wide multiscale recursive networks for robust image labeling. *arXiv:1310.0354*, 2013.

[64] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1), 1962.

[65] V. Jain and S. Seung. Natural image denoising with convolutional networks. In *NIPS*, 2008.

[66] J. Jancsary, S. Nowozin, and C. Rother. Loss-specific training of non-parametric image restoration models: A new state of the art. In *ECCV*, 2012.

[67] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *Proceedings of the 12th International Conference on Computer Vision*, pages 2146–2153. IEEE, 2009.

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

[69] K. Kavukcuoglu, M. Ranzato, R. Fergus, and Y. LeCun. Learning invariant features through topographic filter maps. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1605–1612. IEEE, 2009.

[70] K. Konda and R. Memisevic. Unsupervised learning of depth and motion. In *arXiv:1312.3429v2*, 2013.

[71] J. Koplowitz and T. Brown. On the relation of the performance to editing in nearest neighbor rules. *Pattern Recognition*, 13(3):251–255, 1981.

[72] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical Report TR-2009, University of Toronto, 2009.

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

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

[75] L. Ladickỳ, B. Zeisl, and P. Marc. Discriminatively trained dense surface normal estimation. *ECCV*, 2014.

[76] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, pages 2169–2178, 2006.

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

[78] A. Levin, R. Fergus, F. Durand, and W. T. Freeman. Image and depth from a conventional camera with a coded aperture. In *SIGGRAPH*, 2007.

[79] A. Levin and B. Nadler. Natural image denoising: Optimality and inherent bounds. In *CVPR*, 2011.

[80] C. Liu, J. Yuen, and A. Torralba. Nonparametric scene parsing: label transfer via dense scene alignment. In *CVPR*, 2009.

[81] C. Liu, J. Yuen, A. Torralba, J. Sivic, and W. Freeman. Sift flow: dense correspondence across difference scenes. In *ECCV*, 2008.

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

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

[84] D. G. Lowe. Object recognition from local scale-invariant features. In *Proc. of the International Conference on Computer Vision ICCV, Corfu*, pages 1150–1157, 1999.

[85] T. Malisiewicz and A. Efros. Recognition by association via learning per-exemplar distances. In *CVPR*, 2008.

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

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

[88] J. Michels, A. Saxena, and A. Y. Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In *ICML*, pages 593–600, 2005.

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

[90] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010.

[91] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop*, 2011.

[92] A. Oliva and A. Torralba. Modeling the shape of the scene: a holistic representation of the spatial envelope. *IJCV*, 42:145–175, 2001.

[93] B. A. Olshausen and D. J. Field. Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision Research*, 37(23):3311–3325, 1997.

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

[95] D. Palaz, R. Collobert, and M. Magimai-Doss. Estimating phoneme class conditional probabilities from raw speech signal using convolutional neural networks. In *Interspeech*, 2013.

[96] R. Paredes and E. Vidal. Learning weighted metrics to minimize nearest-neighbor classification error. *IEEE PAMI*, 28(7):1100–1100, 2006.

[97] S. Paris and F. Durand. A fast approximation of the bilateral filter using a signal processing approach. In *ECCV*, pages IV: 568–580, 2006.

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

[99] J. Portilla, V. Strela, M. J. Wainwright, and E. P. Simoncelli. Image denoising using scale mixtures of Gaussians in the wavelet domain. *IEEE Trans Image Processing*, 12(11):1338–1351, November 2003.

[100] M. Ranzato, C. Poultney, S. Chopra, and Y. L. Cun. Efficient learning of sparse representations with an energy-based model. In *NIPS*, 2006.

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

[102] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011.

[103] J. Rolfe and Y. LeCun. Discriminative recurrent sparse auto-encoders. In *ICLR*, 2013.

[104] M. Roser and A. Geiger. Video-based raindrop detection for improved image registration. In *ICCV Workshop on Video-Oriented Object and Event Classification*, Kyoto, Japan, September 2009.

[105] C. J. Rozell, D. H. Johnson, R. G. Baraniuk, and B. A. Olshausen. Sparse coding via thresholding and local competition in neural circuits. *Neural Computation*, 20(10):2526–2563, October 2008.

[106] R. Salakhutdinov and G. E. Hinton. Deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, 2009.

[107] A. Saxena, S. H. Chung, and A. Y. Ng. Learning depth from single monocular images. In *NIPS*, 2005.

[108] A. Saxena, M. Sun, and A. Y. Ng. Make3d: Learning 3-d scene structure from a single still image. *TPAMI*, 2008.

[109] R. Schapire. The boosting approach to machine learning: An overview. In D. D. Denison, M. H. Hansen, C. Holmes, B. Mallick, and B. Yu, editors, *Nonlinear Estimation and Classification*. Springer, 2003.

[110] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *IJCV*, 47:7–42, 2002.

[111] J. Schmidhuber, D. Wierstra, M. Gagliolo, and F. Gomez. Training recurrent networks by evolino. *Neural Computation*, 19(3):757–779, 2007.

[112] P. Sermanet, S. Chintala, and Y. LeCun. Convolutional neural networks applied to house numbers digit classification. In *ICPR*, 2012.

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

[114] J. Shotton, M. Johnson, and R. Cipolla. Semantic texton forests for image categorization and segmentation. In *CVPR*, 2008.

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

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

[117] F. H. Sinz, J. Q. Candela, G. H. Bakır, C. E. Rasmussen, and M. O. Franz. Learning depth from stereo. In *Pattern Recognition*, pages 245–252. Springer, 2004.

[118] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: Exploring photo collections in 3d. 2006.

[119] J. Snoek, H. Larochelle, and R. Adams. Practical bayesian optimzation of machine learning algorithms. In *NIPS*, 2012.

[120] R. Socher, B. Huval, B. Bhat, C. D. Manning, and A. Y. Ng. Convolutional-Recursive Deep Learning for 3D Object Classification. In *NIPS*, 2012.

[121] R. Socher, C. C. Lin, A. Y. Ng, and C. D. Manning. Parsing natural scenes and natural language with recursive neural networks. In *ICML*, 2011.

[122] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

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

[124] I. Sutskever and G. Hinton. Temporal kernel recurrent neural networks. *Neural Networks*, 23:239–243, 2010.

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

[126] C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. In *Advances in Neural Information Processing Systems*, pages 2553–2561, 2013.

[127] G. W. Taylor, R. Fergus, Y. LeCun, and C. Bregler. Convolutional learning of spatio-temporal features. In *Computer Vision–ECCV 2010*, pages 140–153. Springer, 2010.

[128] J. Tighe and S. Lazebnik. Superparsing: Scalable nonparametric image parsing with superpixels. In *ECCV*, 2010.

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

[130] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *CVPR*, 1998.

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

[132] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: a large database for non-parametric object and scene recognition. *IEEE PAMI*, 30(11):1958–1970, November 2008.

[133] A. Torralba, K. P. Murphy, and W. T. Freeman. Contextual models for object detection using brfs. In *NIPS*, 2005.

[134] Z. Tu. Auto-context and its application to high-level vision tasks. In *CVPR*, 2008.

[135] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[136] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, 11:3371–3408, 2010.

[137] C. Vondrick, A. Khosla, T. Malisiewicz, and A. Torralba. Hoggles: Visualizing object detection features. In *ICCV*, 2013.

[138] L. Wan, D. Eigen, and R. Fergus. End-to-end integration of a convolutional network, deformable parts model and non-maximum suppression. *CoRR*, abs/1411.5309, 2014.

[139] L. Wan, M. Zeiler, Z. Sixin, Y. LeCun, and R. Fergus. Regularization of neural networks using dropconnect. In *ICML*, 2013.

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

[141] X. Wang, L. Zhang, L. Lin, Z. Liang, and W. Zuo. Deep joint task learning for generic object extraction. *NIPS*, 2014.

[142] R. G. Willson, M. W. Maimone, A. E. Johnson, and L. M. Scherr. An optical model for image artifacts produced by dust particles on lenses. In *i-SAIRAS*, volume 1, 2005.

[143] J. Xie, L. Xu, and E. Chen. Image denoising and inpainting with deep neural networks. In *NIPS*, 2012.

[144] K. Yamaguchi, T. Hazan, D. Mcallester, and R. Urtasun. Continuous markov random fields for robust stereo estimation. In *arXiv:1204.1393v1*, 2012.

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

[146] M. Zeiler and R. Fergus. Stochastic pooling. In *ICLR*, 2013.

[147] M. Zeiler, D. Krishnan, G. Taylor, and R. Fergus. Deconvolutional networks. In *CVPR*, 2010.

[148] M. Zeiler, G. Taylor, and R. Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *ICCV*, 2011.

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

[150] S. Zhang and E. Salari. Image denosing using a neural network based non-linear filter in the wavelet domain. In *ICASSP*, 2005.

[151] C. Zhou and S. Lin. Removal of image artifacts due to sensor dust. In *CVPR*, 2007.

[152] S. C. Zhu and D. Mumford. Prior learning and gibbs reaction-diffusion. *PAMI*, 19(11):1236–1250, 1997.

[153] D. Zoran and Y. Weiss. From learning models of natural image patches to whole image restoration. In *ICCV*, 2011.