Joint Training of a Neural Network and a Structured Model for Computer Vision

by

Li Wan

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

January, 2015

Rob Fergus

Dedication

Large Dedicated to my parents.

Acknowledgements

Firstly, I would like to thank my advisor Prof Rob Fergus for his patience and insightful guidance. Rob introduce me to computer vision and deep learning. He was always eager to discuss new ideas at any time help me understand the research field quickly. Rob give me freedom and support to work on what I am really exciting. Rob is extremely supportive through my PhD time. I also thank to Prof Yann LeCun and Prof David Sontag for their guidance in the world of machine learning.

Many thanks to my colleague: David Eigen, Matt Zeiler and Sixin Zhang for sharing sharing numerous late nights of work and intense collaboration. I would also to like to thank Ross Goroshin for insightful discussion, especially on unsupervised learning.

Leo Zhu, Yuanhao Chen, Pierre Sermanet, and Clement Farabet for introducing me the subfield of detection in computer vision. Their suggestions and encouragement helped me overcome countless difficulties. Their knowledge and passion directly led me to choose this field for my PhD. I'm especially grateful to Yuanhao and Clement, who shared with me their GPU code.

Furthermore, I am grateful for having worked with great people at NYU: Nathan Silberman, Sainbayar Sukhbaatar, Wojiech Zaremba, Emily Denton, Tom Schaul, Xiang Zhang.

Finally, I thank my parents for their support and love during this long journey.

Abstract

Identifying objects and telling where they are in real world images is one of the most important problems in Artificial Intelligence. The problem is challenging due to: occluded objects, varying object viewpoints and object deformations. This makes the vision problem extremely difficult and cannot be efficiently solved without learning.

This thesis explores hybrid systems that combine a neural network as a trainable feature extractor and structured models that capture high level information such as object parts. The resulting models combine the strengths of the two approaches: a deep neural network which provides a powerful non-linear feature transformation and a high level structured model which integrates domain-specific knowledge. We develop discriminative training algorithms to jointly optimize these entire models end-to-end.

First, we proposed a unified model which combines a deep neural network with a latent topic model for image classification. The hybrid model is shown to outperform models based solely on neural networks or topic model alone. Next, we investigate techniques for training a neural network system, introducing an effective way of regularizing the network called DropConnect. DropConnect allows us to train large models while avoiding over-fitting. This yields state-of-the-art results on a variety of standard benchmarks for image classification. Third, we worked on object detection for PASCAL challenge. We improved the deformable parts model and proposed a new non-maximal suppression algorithm. This system was the joint winner of the 2011 challenge. Finally, we develop a new hybrid model which integrates a deep network, deformable parts model and non-maximal suppression. Joint training of our hybrid model shows clear advantage over train each component individually, and achieving competitive result on standard benchmarks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

People are very good at recognizing objects and understand the location of objects. Such task is trivial to humans because we learn from birth to understand the highly diverse and complex world. Furthermore, human are improving their ability to recognize object everyday. The understand of how human's vision system works have attracted the curiosity of researchers for long time. Towards the end of last century, people began to build machines to understand images and video signals. Understanding of image signals is difficult for machine due to the dimension of data is too high. This work aims to develop learning algorithms that enable machine to understand and interact with real world.

Structured models that capture high-level information are a natural way to describe data such as constellations model [17], deformable part model [15], topic model [6] and other types of probabilistic graphical models [26, 1]. Explicit variables (usually corresponding to some tangible entity) are linked in a sparse dependency structure, which is typically specified by hand using domain knowledge. They achieve great success in both image classification and object detection based on fixed and carefully crafted image features

such as HoG [9], SIFT [44] and LBP [50].

Neural networks especially convolutional neural networks have proven effective learning good feature representation from the image pixels [39] [34]. Instead of building handcraft and domain-specific features, we learn from large amounts of data to get robust and invariant features in a hierarchical manner. However, The variables in neural network are lack of explicit meaning comparing with strucutred models, such as constellations model and object part model.

In this thesis, we explore hybrid models which are designed to combine the strengths of both domain-appropriate structured models with powerful non-linear features provided by neural network We propose end-to-end training algorithms for hybrid models by transforming structured model in to neural network layers.

1.2 Contributions and organization of the thesis

The main contributions of this thesis are as follows:

1. In Chapter 3, we introduce a hybrid model that combines a neural network with a latent topic model. The neural network provides a low-dimensional embedding for the input data, whose subsequent distribution is captured by the topic model. The neural network thus acts as a trainable feature extractor while the topic model captures the group structure of the data. Following an initial pre-training phase to separately initialize each part of the model, a unified training scheme is introduced that allows for discriminative training of the entire model. The approach is evaluated on visual data in scene classification task, where the hybrid model is shown to outperform models based solely on neural networks or topic models, as well as other baseline methods.

2. In Chapter 4, We introduce DropConnect, a generalization of Dropout [25], for regularizing large fully-connected layers within neural networks. When training

with Dropout, a randomly selected subset of activations are set to zero within each layer. DropConnect instead sets a randomly selected subset of weights within the network to zero. Each unit thus receives input from a random subset of units in the previous layer. We derive a bound on the generalization performance of both Dropout and DropConnect. We then evaluate DropConnect on a range of datasets, comparing to Dropout, and show state-of-the-art results on several image recognition benchmarks by aggregating multiple DropConnect-trained models.

3. In Chapter 5, we develop a state-of-the-art detection system based deformable part model with handcrafted features [78]. We also proposed an iterative non-maximal suppression algorithm which is a huristic way of merge results from detector. Such algorithm is proven to be critical to achieve good performance in PASCAL challenge.

4. In Chapter 6, we introduce a hybrid model that integrate the whole detection pipeline: combines convolutional neural network model (ConvNets), deformable parts model (DPM) and post-processing non-maximal suppression (NMS). DPM and ConvNets each have achieved notable performance in object detection. Yet these two approaches find their strengths in complementary areas: DPMs are well-versed in object composition, modeling fine-grained spatial relationships between parts; likewise, ConvNets are adept at producing powerful image features, having been discriminatively trained directly on the pixels. We train this model using a new structured loss function that considers all bounding boxes within an image, rather than isolated object instances. This enables the non-maximal suppression (NMS) operation, previously treated as a separate post-processing stage, to be integrated into the model. This allows for discriminative training of our combined Convnet + DPM + NMS model in end-to-end fashion. We evaluate our system on PASCAL VOC 2007 and 2011 datasets, achieving competitive results on both benchmarks.

Chapter 2

Literature survey

2.1 Object Recognition with Handcrafted feature

The appearance of an image region can be effectively described by handcraft fixed features: SIFT (Scale Invariant Feature Transform) [44], HoG (Histogram of Gradient) [9] and LBP (Local Binary Pattern) [50]. These low-level hand-crafted features have been quite successful. Generally speaking, all these features essentially describe the low order statistics of the edge distribution. Each image region generate fixed size vector to describe properties of the region. Different descriptors describe different information of images patches: SIFT for appearance, HoG for edges and LBP for textures. Depending on different task, those features and their variations are either used separately [57] [38] [15] [17] or combined [78] [69]. Low-level features are extracted on a regular dense grid [38] or at sparse locations in the image [44]

The bag-of-words (BoW) model originated in the field of text processing have been successfully applied to image classification tasks. Features learnt from dictionary learning [61] can be used for classifying images. The general pipeline is as follows: 1) extract features from image regions 2) build a global dictionary (codebook) with feature of local image patches. 3) summarize index of feature in dictionary for each region into normal-

ized histograms. 4) supervised classifier such as SVM or random forests is trained based on image labels and normalized histograms. These methods have been proven successful in visual object recognition task. Further improvements are obtained by using spacial information [38]. Advanced kernel for svm also helps, such as histogram intersection or chi-squared kernel [57].

Probabilistic graphical models [6, 26, 1] build on low level handcrafted features also achieve success for various image recognition task [64][63][17][13]. Fergus et al. [17] model objects with flexible constellations of parts achieve excellent results in both geometrically constrained classes and flexible objects. Objects shape, appearance, occlusion and relative scale are modeled under probabilistic framework. Learning the parameters of model to maximize object likelihood in image with expectation-maximization (EM) algorithm. The "generic" knowledge learnt by graphical models could even be applied to models of unrelated categories as "prior" knowledge. Fei-Fei et al. [13] show such prior knowledge is useful for object classification with just a few images.

2.2 Object Detection with Handcrafted feature

Handcrafted features can be applied to object detection. One example is pedestrian detection with HoG features by Dalal et al. [9]. It achieved nearly perfect separation on the MIT pedestrian dataset. They train their model based on HoG features as follows: 1) HoG features are densely computed on each image 2) Linear SVM trained with fixed person patches as positive examples and person-free patches as negative examples. 3) The method then re-train using augmented set including "hard-exmaples" which the classifier fails. In the final stages, duplicate detector activation is removed by non-maximal suppression (NMS) operation.

General object detection is also an important task in computer vision which requires to predict both object labels and object locations at the same time. Detection systems have been build based on the concept of "part-based" models [15][78][77][7][4][58][73]

with patch appearance features such as HoG, SIFT, or LBP. Felzenszwalb et al. [15] build an object detection system call deformable part model (DPM) based on mixtures of models with root and part structure. It has been applied successfully in PASCAL object detection challenges. Such a model is learnt with latent svm on image pyramid (multi-scale image). The part locations in such system are modeled as latent-variables in the learning process. Such latent SVM formulation lead to an iterative learning algorithm that alternates between fixing part locations for positive examples and estimate model parameters for patch features. Zhu et al. [78] extends the concept of "part-based" model to a deeper structure by introducing one more level of "part-on-part", they applies an extension of Concave Convex procedure (CCCP) Algorithm [74] called iCCP to learn such one level deeper model. In addition to use HoG feature for previous two systems, further improvement introduced by Chen et al. [7], who included both HoG feature and vector quantized SIFT feature.

Different Object detection results can either enhance or inhibit each other within a scene or context. This has been an area of active research in object recognition[11][67] [46][18] [22] [27][2][36]. The general idea of objection detection with context information is as follows: different object classes may be correlated with each other either by presents or absence or by spatial layout. Desai et al. [11] and Torralba et al. [67] present models that learn such contextual statistics from data. Such information provide strong prior information enhancing the performance of individual detector.

End-to-end training of a multi-classes detector and post-processing has also been discussed in Desai et al. [11]. Their approach reformulates NMS as a contextual relationship between locations. They replace NMS, which removes duplicate detections, with a greedy search that adds detection results using an object class-pairs context model. Such a model also handles interactions between different types of objects.

2.3 Feature Learning

Deep architectures (neural networks), that learn hierarchies of features with increasing levels of invariance and complexity, are well fitted for vision task that requires highly complex features [21][76]. Multi-layer network trained end-to-end with little prior knowledge. In this framework, feature representations are jointly learnt with classifiers. Convolutional Networks (ConvNets) [39] are a special multi-layer neural network with end-to-end supervise learning on raw image pixels. Unsupervised techniques have also been successful applied at training or pre-training (initializing) multi-layer networks, such as restricted Boltzmann machines [23], stacked auto-encoders [3].

Hinton et al. [23] have introduced a layer-by-layer strategy to initialize neural network. One way of unsupervised learning objective is training a network to reconstruct its input. Thus, the deep network is constructed as a stack of restricted Boltzmann machines (RBM). Each layer trained to learn the distribution of layer input by maximizing an approximation of the data likelihood. Auto-encoders proposed by Bengio et al. [3] initialize network layers directly by minimizing reconstruction error of input from output. After initialization of each layer in the network, the whole network is jointly fine-tunned with back-propagation based on label information.

The hybrid model in this thesis also following similar strategy:

- In Chapter 3: 1) layer-wise initialization of network. 2) train a topic model based on network output. 3) convert topic model into neural network layer and 4) perform joint back-propagation based on label information.
- In Chapter 6: 1) initialize ConvNet based on ImageNet dataset. 2) train DPM with fixed part location 3) convert DPM to network layer 4) joint training of whole model with flexible part location.

Convolutional Networks [39] has been successfully applied for image classification for many years. The object can be handwritten characters [39], house numbers [59][8][70],

tiny image of objects [70][33], toys with texture background [8][41], and traffic signs [59][8]. Recently, it achieves a huge success form 1000-category ImageNet dataset [35]. Figure 2.1 illustrates a ConvNet for classifying handwritten characters created by LeCun et al. [39]. From a low-level point of view, images exhibit spatial relationship between neighboring

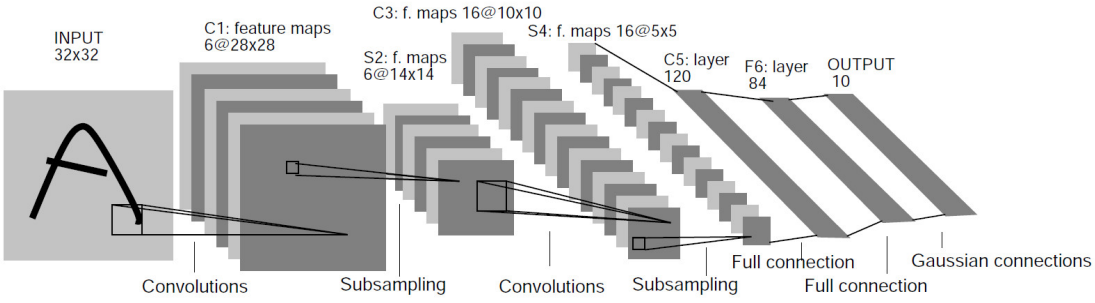


Figure 2.1: Reproduced from LeCun [39]: a ConvNet composed a set of convolutions and pooling operators as feature extractor followed by two fully connected layers. Final Gaussian layer for classify handwritten digits.

locations, i.e. 1) the correlation of neighboring pixel is high and faraway pixel is almost zero 2) each image patch is more or less independent from each other. Based on such properties of image, ConvNet effectively reduced number of network parameters in feature extractor by having shared and trainable filter bank for each input at all locations. In Chapter 6 and Chapter 4, we adapt similar ConvNet feature extractor for both image classification and image detction.

Kavukcuoglu et al. [28] proposed an unsupervised learning algorithm to initialize ConvNet layer by layer. It trained convolutional sparse auto-encoder with a linear decoder to reconstruct image from sparse features. Such technique improves the performance when the amount of label information is relative small. For example, training ConvNet for pedestrian detection on the INRIA Pedestrian dataset [9]. Such initialization step can be skipped when dataset is sufficient large [35].

2.4 Object Detection with Neural Network

The first system to combine structured learning with a ConvNet is LeCun et al. [39], who train a ConvNet to classify individual digits, then train with hand written strings of digits discriminatively. Very recently, Tompson et al. [66] trained a model for human pose estimation that combines body part location estimates into a convolutional network, in effect integrating an MRF-like model with a ConvNet. Their system, however, requires annotated body part locations and is applied to pose estimation, whereas our system does not require annotated parts and is applied to object detection.

Some recent works have applied ConvNets to object detection directly: Sermanet et al. [60] train a network to regress on object bounding box coordinates at different strides and scales, then merge predicted boxes across the image. Szegedy et al. [65] regress to a bitmap with the bounding box target, which they then apply to strided windows. Both of these approaches directly regress to the bounding box from the convolutional network features, potentially ignoring many important spatial relationships. By contrast, we use the ConvNet features as input to a DPM. In this way, we can include a model of the spatial relationships between object parts.

In the R-CNN model, Girshick et al. [19] take a different approach in the use of ConvNets. Instead of integrating a location regressor into the network, they instead produce candidate region proposals with a separate mechanism, then use the ConvNet to classify each region. However, this explicitly resizes each region to the classifier field of view (fixed size), performing significant distortions to the input, and requires the entire network stack to be recomputed for each region. Instead, our integration runs the features in a convolutional bottom-up fashion over the whole image, preserving the true aspect ratios and requiring only one computational pass.

Most closely related is the concurrent work of Girshick et al. [20], who also combine a DPM with ConvNet features in a model called DeepPyramid DPM (DP-DPM). Their work, however, is limited to integrating fixed pretrained ConvNet features with a DPM.

We independently corroborate the conclusion that using ConvNet features in place of HoG greatly boosts the performance of DPMs. Furthermore, we show how using a post-NMS online training loss improves response ordering and addresses errors from the NMS stage. We also perform joint end-to-end training of the entire system in Chapter 6.

2.5 Training Deep Neural Network

Network with millions or billions of parameters can easily over-fit to training data: network tend to memorize the training data without promising generalization performance. To overcome this problem, a wide range of techniques have been developed. Those techniques are generally divided into two categories: 1) data augmentation algorithm [35][8] and 2) network regularization algorithm [35][45][72][70]. Those two categories are not mutually exclusive, they are usually used combined to achieve good results. Further performance improvement could be introduced by voting of multiple instance of network with different initialization of weights [70][8].

Ciresan et al. [8] train a set of neural networks for image classification task, including handwritten digits, traffic signs, and Chinese characters. Building invariance into data is a critical step of getting good performance: 1) translate image data 2) rotate image 3) rescale image and 4) flip image. By having augmented dataset, the original training set became validation set to decide when to stop network.

Adding an ℓ_2 penalty on the network weights is a simple but effective approach, other forms of neural network regularization include: bayesian methods [45], weight elimination [72] and early stopping of training. However, all of these requires hand-tunning some regularization parameter.

Recently, Hinton et al. proposed a new form of parameter free regularization called Dropout [25]: For each training example, forward propagation involves randomly deleting half of the activation in each layer. Extensive experiments shown that this significantly

reduces over-fitting and improve testing performance. In this thesis, we proposed DropConnect which generalizes Dropout and shown this to be an even better regularization than Dropout from both empirically and theoretically point of view.

Chapter 3

A Hybrid Neural Network-Latent Topic Model

3.1 Introduction

Probabilistic graphical models [6, 26, 1] and neural networks [29, 52, 43] are the two prevalent types of belief network in machine learning. In the first of these, explicit variables (usually corresponding to some tangible entity) are linked in a sparse dependency structure, which is typically specified by hand using domain knowledge. This is quite different to a neural network where variables lack explicit meaning and are densely connected. In principle, the model is less dependent on the details of the problem, but in practice selections must be made regarding the model architecture and the training protocol (*eg*[40]).

Each model is appropriate for different settings. Probabilistic graphical models are a natural way to represent the high level structure of a signal. Equally, deep neural networks have proven effective at automatically learning good feature representations from the raw signal, a situation where detailing the precise form of the dependencies is problematic.

We proposes a model that combines a deep neural network with a latent topic models and presents a joint learning scheme that allows the combined model to be trained in a discriminative fashion. The resulting model combines the strengths of the two approaches: the deep belief network provides a powerful non-linear feature transformation for the domain-appropriate topic model.

Our main technical contribution is a novel way of transforming the graphical model during inference to form additional neural network layers. This transformation allows the back-propagation [40] to be performed in a straightforward manner on the unified model. We demonstrate this transformation for a latent topic model but the operation is valid for any model where (approximate) inference is possible in closed-form.

We demonstrate our model in a computer vision setting, using it to perform scene classification. We choose this domain because (a) developing good feature representations for visual data is an active area of research [43, 52] and the neural network part of our model addresses this task and (b) latent topic models, such as latent Dirichlet allocation (LDA) [6], have been shown to be effective for image classification, using a bag-of-words image representations [10, 14].

The training procedure for our model involves a preliminary unsupervised phase where the parameters of the deep-belief network are initialized. This is performed using restricted Boltzmann machines (RBMs) in conjunction with contrastive divergence learning, in the style of Hinton and Salakhutdinov [24]. The same layer-wise greedy scheme of [24, 3] is used.

Our graphical model is a variant of latent Dirichlet allocation [6] and is closely related to the author-topic model [54], introduced to the vision community by Sudderth et al. [64]. However, unlike these unsupervised models, ours is discriminative and incorporates the class label, similar to the Disc-LDA model of Lacoste-Julien et al. [37] and the Supervised-LDA model of Blei and McAuliffe [5]. Ngiam et al. [49] show how a feedforward neural-network can be used as a deterministic transformation as input to a

deep-belief network (DBN), showing results on MNIST and NORB datasets. In contrast, we use a topic model instead of the DBN and evaluate on a more complex scene dataset. Another related paper to ours is Salakhutdinov et al. [55], who combine a Gaussian process with a deep-belief network. However, this is a simpler graphical model than ours and lacks the modeling capabilities of latent topic models. The most closely related paper is the contemporaneous work of Salakhutdinov [56] who combine an Hierarchical Dirichlet Process with a Deep Boltzmann Machine.

3.2 The hybrid model

Our hybrid model is a combination of one particular probabilistic model, hierarchical topic model (HTM), and a neural network (NN). We consider a set of N images $\{I_1, \ldots, I_i, \ldots, I_N\}$ with labels y. From each we extract a set of SIFT descriptors [44] $\{v_1, \ldots, v_j, \ldots, v_{n_i}\}$. Each descriptor v_j is individually mapped by a neural network with parameters w to a feature vector x_j in R^d, d being the number of units in the top layer of the network. The transformation of $v \rightarrow x$ is denoted by $f_w(v)$. The structure of neural network is encoded by layers of hidden units h. The connections of hidden units are given in section 3.2.1.

Each image is represented by a topic model where each topic is a probability distribution over visual words in a vocabulary. In prevalent topic models such as LDA[6, 14] and its variants [64], the vocabulary is given by vector quantization, but not integrated into the topic model. In this paper, we want to learn the feature representation and the topic model jointly. We extend the topic model by adding an extra latent variable to encode visual vocabulary. Unlike other topic models, our model is directly defined over image features, and capable of learning vocabulary and topic distribution simultaneously. The new topic model is of hierarchical structure (see Figure 3.1) whose distribution is given in section 3.2.2.

The hierarchical topic model is combined with the neural network to form a hybrid model

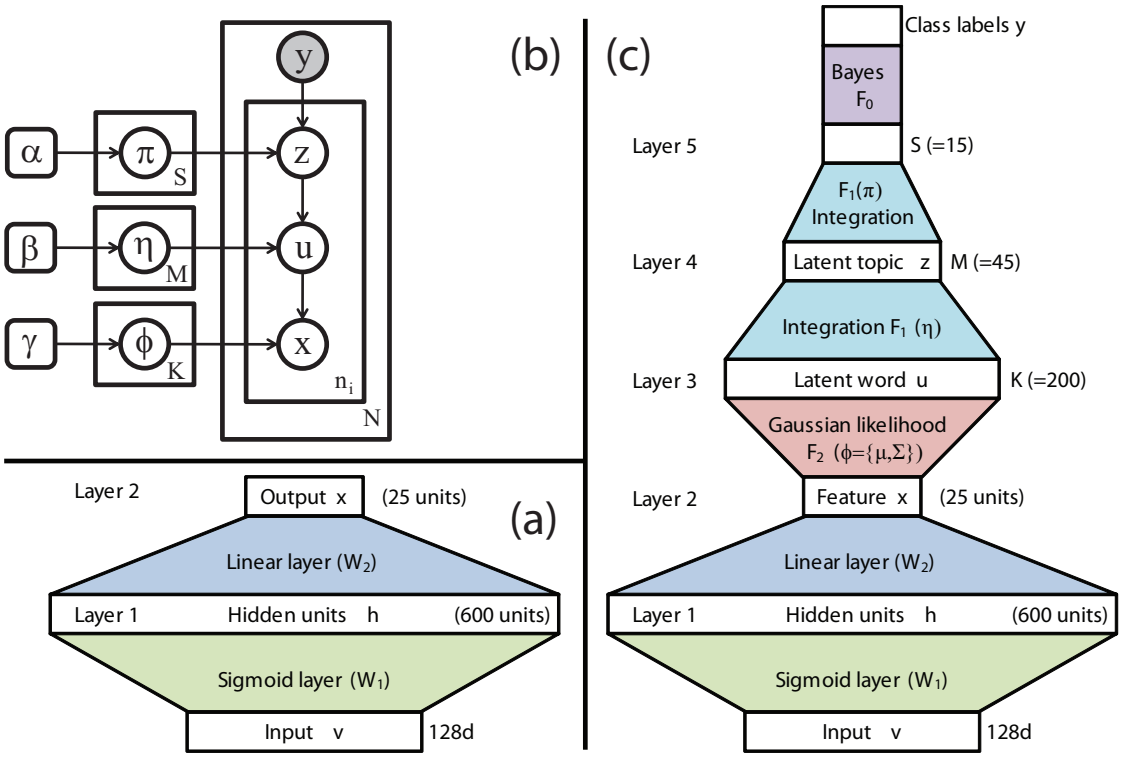


Figure 3.1: (a) Neural network (NN). (b) Hierarchical topic model (HTM). (c) Our hybrid model that combines the NN and HTM.

by treating the output x of the network as the bottom nodes of the hierarchy.

3.2.1 Neural Network

In this paper, we consider a neural network with two hidden layers, as shown in Figure 3.1(a). The first hidden one is a sigmoid layer which maps the input features v into a binary representation h via a sigmoid function, i.e. $h = \sigma(w_1 v + b_1)$ where $\sigma(t) = 1/(1 + exp(-t))$ and w_1, b_1 are the parameters of this layer. The second hidden layer performs linear dimension reduction $x = hw_2 + b_2$, with w_2, b_2 being parameters. The output of the units x correspond to the transformation $f_w(v)$ provided the whole network. An arbitrary number of extra hidden layers could be inserted between these two layers if more a complex transformation is preferred. Let $w = \{w_1, b_1, w_2, b_2\}$ denote all parameters of the network. Training the network is performed by back propagation

[40] on w. The initialization of w is obtained by learning a Restricted Boltzmann Machine (RBM) [23] with the same structure of network. We will give details of the training procedure in section 3.3.2.

3.2.2 Hierarchical topic model

Given an image I_i, the neural network will transform each raw feature v_j into a vector x_j, which is input to the hierarchical topic model. We assume that each x_j is generated by a Gaussian distribution of the corresponding word cluster u_j. The Gaussian is parametrized by $\phi_{u_j} = \{\mu_{u_j}, \Sigma_{u_j}\}$. The word u_j is generated by a multinomial word distribution with parameters of η_{z_j}. The distribution of topics z_j is a multinomial parametrized by π_{y_i} where y_i is a label of scene category for image I_i. y_i is provided in the learning stage. The overall model is shown in Figure **??**(b). The hierarchical generative process is given by:

1. Draw latent topic $z_j \sim Multi(\pi_{y_i})$
2. Draw latent word $u_j \sim Multi(\eta_{z_i})$
3. Draw feature vector $x_j \sim Gaussian(\phi_{u_j})$.

The prior distributions on π, η, ϕ are defined as follows:

$$\begin{aligned} \pi_y &\sim Dir(\alpha) \text{ for } y \in \{1, 2, \dots S\} \\ \eta_z &\sim Dir(\beta) \text{ for } z \in \{1, 2, \dots M\} \\ \phi_u &\sim \mathcal{NIW}(\gamma) \text{ for } u \in \{1, 2, \dots K\} \end{aligned}$$

where $Dir(.)$ is Dirichlet distribution, $\mathcal{NIW}(\gamma)$ is normal-inverse-Wishart distribution, parametrized by $\gamma = \{\mu_0, \kappa, \nu, \Lambda_0\}$. S is the number of scene categories (class labels), M is the number of latent topics, and K is the size of word vocabulary.

The hyper-parameters α, β, γ are set by hand, their exact value not being too important. Learning the model involves the estimation of the parameters π, η, ϕ. In section 3.3.3, we will discuss the learning of the hierarchical model.

Note that, unlike standard LDA[6], our model has no image (or document) specific prior of topic distribution. The most related representation is the scene model [64] which essentially is a author-topic model [54].

3.2.3 The hybrid model: coupling the neural network and topic model

The hybrid model is designed to combine the strengths of the two models defined above. The input x to the hierarchical topic model is the transformed output $f_w(v)$ of the neural network. Note, in the hybrid model, x is not observable. The topic model captures high-level scene structure of an image while the neural network offers approximate low-dimensional embedding of raw features. Typically, the probabilistic topic model is a sparse graph while neural network is a densely connected graph.

3.3 Learning the hybrid model

3.3.1 Brief description

The task of learning the hybrid model is to maximize the posterior distribution of class label y, given the input data v. The loss function is given by:

$$L(w, \pi, \eta, \phi) = -\log p(y|v, w, \pi, \eta, \phi) \tag{3.1}$$

Since $x = f_w(v)$, $p(y|x, w, \pi, \eta, \phi) = p(y|v, w, \pi, \eta, \phi)$, and applying Bayes rule, we thus have:

$$L = -\log p(f_w(v)|y, \pi, \eta, \phi) + \log \sum_{\tilde{y}=1}^{S} p(f_w(v)|\tilde{y}, \pi, \eta, \phi) \tag{3.2}$$

The likelihood function of the hierarchical topic model for an image I_i is defined as $p(f_w(v)|y)$:

$$\prod_{j=1}^{n_i} \sum_{z_j=1}^{M} \left(\sum_{u_j=1}^{K} p\left(f_w(v_j)|u_j, \phi\right) p\left(u_j|z_j, \eta\right) \right) p\left(z_j|y, \pi\right)$$

The optimization of the loss function is performed by gradient descent. The learning procedure consists of two steps:

1. We first initialize the parameters $\{w^0, \pi^0, \eta^0, \phi^0\}$ obtained by pre-training the neural network and the hierarchical topic model which will be introduced in sections 3.3.2 and 3.3.3.

2. The parameters are then updated according to the following rules (where c is a learning rate):

$$\begin{aligned} w^{t+1} &= w^t - c\frac{\partial L}{\partial w},\ \pi^{t+1} = \pi^t - c\frac{\partial L}{\partial \pi} \\ \eta^{t+1} &= \eta^t - c\frac{\partial L}{\partial \eta},\ \phi^{t+1} = \phi^t - c\frac{\partial L}{\partial \phi} \end{aligned} \tag{3.3}$$

The major technical issues of learning are (i) how to obtain a good initialization; (ii) how to calculate the gradients of the loss function defined over the joint model. We will address these two issues in the following sections.

3.3.2 Pre-training of neural network

We first pre-train the neural network by learning RBMs with the same structure in an unsupervised manner. A RBM is an undirected graphical model with connections between visible units v and hidden units h. The joint distribution $p(v, h) \propto \exp(-E(v, h))$ with the normalization constant $Z = \sum_{v,h} \exp(-E(v, h))$. The transformation provided by neural network is defined as $p(h|v)$ in the corresponding RBM. We apply the contrastive divergence (CD) algorithm [23] to train the network. CD algorithm is a greedy layer-wise learning method for stack of Restricted Boltzmann machine (RBMs) which

maximize the variational lower bound of the model likelihood. Hinton and Osindero [23] show that such initialization works well for discriminative training.

Each layer of NN is initialized by the corresponding RBM model. More precisely, the RBM energy for pre-training of the bottom layer is $E(v,h) = -\frac{1}{2}(v-b)^T(v-b) - h^Tc - v^TWh$. The energy of RBM for pre-training of the second (top) layer is $E(h,x) = -\frac{1}{2}(x-c)^T(x-c) - h^Tb - h^TWx$.

3.3.3 Pre-training of the hierarchical topic Model

Once we have pre-trained the neural network, the resulting $x = f_w(v)$ are fixed and used as input to the hierarchical topic model. The training of the graphical model is performed by Gibbs sampling in the style adopted by Sudderth et al. [64] in their similar scene model. The sampling procedure is given in Algorithm 1:

Algorithm 1 Gibbs sampler for the hierarchical topic model.

for image i **do**
  for feature j in image i **do**
    remove u_{ij} form cached statistics for latent variable $k = z_{ij}$ and $m = u_{ij}$
    – $\pi_{yk} \leftarrow \pi_{yk} - 1$
    – $\eta_{km} \leftarrow \eta_{km} - 1$
    – $\phi_m \leftarrow$ remove u_{ij} from ϕ_m
    Sample z_{ij} and u_{ij} jointly from multinomial distribution:
    – $p(z_{ij}, u_{ij} | x_{ij}, \pi, \eta, \phi, \alpha, \beta, \gamma) \propto p(x_{ij}|u_{ij}, \phi, \gamma)p(u_{ij}|\eta_{z_{ij}}, \beta)p(z_{ij}|\pi_{y_i}, \alpha)$
    Add back feature x_{ij} to the cached statistic for its new latent variable $k = z_{ij}$ and $m = u_{ij}$:
    – $\pi_{yk} \leftarrow \pi_{yk} + 1$,$\eta_{km} \leftarrow \eta_{km} + 1$
    – $\phi_m \leftarrow$ add u_{ij} to ϕ_m.
  end for
end for

3.3.4 Joint optimization by gradient descent

We now have the full pre-trained model with all parameters initialized. Gradient descent is applied to update the parameters. The loss function defined in Eq. 3.2 can be expanded

as follows:

$$\begin{aligned} L &= -\sum_j \log p(f_w(v_j)|y,\pi,\eta,\phi) + \\ &\quad \log \sum_{i=1}^{S} \prod_j p(f_w(v_j)|y=i,\pi,\eta,\phi) \end{aligned}$$

Let $F_0(A,y) \equiv L$ where A is a $n_i \times S$ matrix, $A_{ji} = [p(f_w(v_j)|y=i,\pi,\eta,\phi]_{ji}$ and F_0 is a function which takes an input of a matrix A and a label information y. $F_0(A,y)$ is a form of:

$$F_0(A,y) = -\sum_j \log A_{jy} + \log \sum_i \prod_j A_{ji} \tag{3.4}$$

We can decompose each element of A as follows:

$$\begin{aligned} A_{ji} &= p(f_w(v_j)|y=i,\pi,\eta,\phi) \\ &= \sum_{z_j=1}^{M} p(f_w(v_j)|z_j,\eta,\phi)p(z_j|y=i,\pi) \\ &= [F_1(B,C)]_{ji} \\ &\quad \text{where } B_{jm} = p(f_w(v_j)|z=m,\eta,\phi) \\ &\quad \text{and } C_{mi} = p(z=m|y=i,\pi) \end{aligned}$$

Here matrix B is of size $n_i \times M$ and matrix C is of size $M \times S$. Function F_1 is a multiplication of two matrices, i.e. $A = F_1(B,C) = BC$.

The decomposition of matrix B is given by:

$$\begin{aligned} B_{jm} &= p(f_w(v_j)|z=m,\eta,\phi) \\ &= \sum_{u_j=1}^{K} p(f_w(v_j)|u_j,\phi)p(u_j|z=m,\eta) \\ &= [F_1(D,E)]_{jm} \\ &\quad \text{where } D_{jk} = p(f_w(v_j)|u=k) \\ &\quad \text{and } E_{km} = p(u_j=k|z=m,\eta) \end{aligned}$$

where E is the η matrix of size $K \times M$ and D is of size $n_i \times K$. We can decompose the matrix D:

$$\begin{aligned} D_{jk} &= p(f_w(v_j)|u=k,\phi) \\ &= |\Sigma_k|^{-1} \exp\left(-\frac{1}{2}\left(f_w(v_j)-\mu_k\right)\Sigma_K^{-1}\left(f_w(v_j)-\mu_k\right)^T\right) \\ &= [F_2(x,\phi)]_{jk} \text{ where } \phi_k = \{\mu_k, \Sigma_k\} \end{aligned}$$

Function F_2 evaluates the likelihood (dropping out the normalization constant) of the input data v_i for different Gaussian centers $\phi_k = \{\mu_k, \Sigma_k\}$ for $k = 1, 2, \ldots, K$. The output of $F_2(f_w(v), \phi)$ is $|\Sigma_k|^{-1} \exp(-\frac{1}{2}(f_w(v_j) - \mu_k)\Sigma_k^{-1}(f_w(v_j) - \mu_k)^T)$.

After defining the functions F_0, F_1, F_2, the loss function can be rewritten:

$$L = \underbrace{F_0(F_1(\overbrace{F_1(\overbrace{F_2\left(f_w(v),\phi\right)}^{D}, \overbrace{\eta}^{E})}^{B}, \overbrace{\pi}^{C}), y)}_{A} \tag{3.5}$$

and

$$\begin{aligned} \frac{\partial L}{\partial \pi} &= \frac{\partial F_0(A,y)}{\partial A}\frac{\partial F_1(B,\pi)}{\partial \pi} \\ \frac{\partial L}{\partial \eta} &= \frac{\partial F_0(A,y)}{\partial A}\frac{\partial F_1(B,\pi)}{\partial B}\frac{\partial F_1(D,\eta)}{\partial \eta} \\ \frac{\partial L}{\partial \phi_k} &= \frac{\partial F_0(A,y)}{\partial A}\frac{\partial F_1(B,\pi)}{\partial B} \\ &\quad \frac{\partial F_1(D,\eta)}{\partial D}\frac{\partial F_2(f_w(v),\phi)}{\partial \phi_k} \\ \frac{\partial L}{\partial w} &= \frac{\partial F_0(A,y)}{\partial A}\frac{\partial F_1(B,\pi)}{\partial B}\frac{\partial F_1(D,\eta)}{\partial D} \\ &\quad \sum_{j=1}^{n_i}\frac{\partial F_2(f_w(v_j),\phi)}{\partial f_w(v_j)}\frac{\partial f_w(v_j)}{\partial w} \end{aligned}$$

The gradients of L w.r.t w, ϕ, η, π can be obtained by applying chain rule, provided the gradients of F_0, F_1, F_2:

1. Function $F_0(A, y)$ evaluation:

$$F_0(A, y) = -\sum_j \log A_{jy} + \log \sum_{i=1}^{S} \prod_{j=1}^{n_i} A_{ji}$$

and gradient computation:

$$\frac{\partial F_0(A, y)}{A_{ji}} = -1 + \frac{\exp(F_0(A, y))}{A_{ji}}$$

2. Function $F_1(B, \pi)$ evaluation:

$$[F_1(B, \pi)]_{ik} = \frac{\sum_j B_{ij}\pi_{jk}}{\sum_k \pi_{jk}}$$

and gradient computation:

$$\frac{\partial [F_1(B, \pi)]_{ik}}{\partial B_{ij}} = \frac{\pi_{jk}}{\sum_t \pi_{jt}}$$

and

$$\frac{\partial [F_1(B, \pi)]_{ik}}{\partial \pi_{jk}} = \frac{B_{ij}}{\sum_t \pi_{jt}} - \frac{[F_1(B, \pi)]_{ik}}{\sum_t \pi_{jt}}$$

The reason that we divide the output by $\sum_k R_{jk}$ is because the second parameter π of this function is always sufficient static of multinomial distribution, thus must always sum to one. The derivative for $F_1(D, \eta)$ takes a similar form.

3. Function $F_2(x, \phi)$,where $\phi = \{\phi_1, \phi_2, \ldots, \phi_K\}$ each of ϕ_k is a Gaussian center $\phi_k = \{\mu_k, \Sigma_k\}$ $[F_2(x, \phi)]_{jk} = |\Sigma_k|^{-1} \exp(-\frac{1}{2}(\mathbf{x}_j - \mu_k)\Sigma_k^{-1}(\mathbf{x}_j - \mu_k)^T)$ Taking gradients we obtain (omitting the Σ_k update for brevity):

$$\begin{aligned} \frac{\partial [F_2(x, \phi)]_{jk}}{\partial x_j} &= -[F_2(x, \phi)]_{jk}(x_j - \mu_k)\Sigma_k^{-1} \\ \frac{\partial [F_2(x, \phi)]_{jk}}{\partial \mu_k} &= [F_2(x, \phi)]_{jk}(x_j - \mu_k)\Sigma_k^{-1} \end{aligned}$$

3.3.5 Unifying probabilistic hierarchical model and neural network: back-propagation

The gradient descent scheme introduced in section 3.3.4 can be interpreted as a back-propagation algorithm on a new neural network which unifies the hierarchical topical model and the neural network. The decomposition of the loss function in Eq. 3.5 allows us applying chain rule to calculate the gradients w.r.t. all the parameters in the hybrid model. It suggests a strong connection with the back-propagation algorithm if we define four consecutive layers from top to bottom in the following order:

1. *Bayesian Layer:* Function F_0
2. *Integration Layer on z:* Function F_1 with π as the second parameter
3. *Integration Layer on u:* Function F_1 with η as the second parameter
4. *Gaussian Likelihood Layer:* Function F_2

The transformation of each layer is defined by the corresponding function F and its parameters. We can simply add these four invented layers on the top of the original neural network, yielding a unified neural network. By back-propagating through this unified network, we can estimate the parameters of the hybrid model.

3.3.6 Inference of the hybrid model: feed-forward

Inference is performed as a feed-forward procedure on the unified neural network. Given a testing image v, the first two layers of the neural network produce encoded features $x = f_w(v)$. According to the definition of the four additional layers, the output at the top layer is $p(y|f_w(v))$ for $y = 1, 2, \ldots, S$. The task of inference $y^* = \arg\max_y p(y|f_w(v))$ is the same as passing x through Gaussian likelihood layer(F_2), Integration layer($F_1(\cdot, \eta)$), Integration layer($F_1(\cdot, \pi)$) and Bayesian layer F_0 (see Figure 1(c)). Note that our hybrid approach can be extended to any graphical model where the (approximate) inference can be performed in closed-form.

3.4 Toy experiments

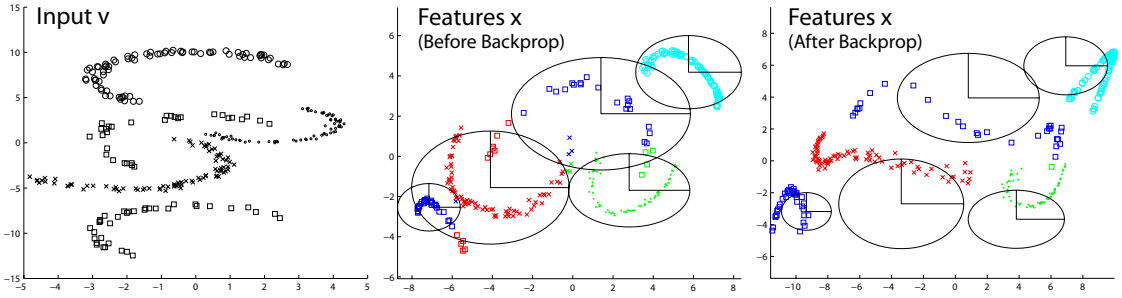


Figure 3.2: Toy experiment using 2D data (left), with 5 clusters drawn from 4 classes (cross, dot, square, circle). Note that there are multiple clusters per class. Middle: Visualization of 2D feature space x after NN pre-training and Gibbs sampling of supervised topic model with K=5 clusters and Y=4 classes. The ellipses show the mean and covariance of each Gaussian cluster in the HTM. In the HTM, the color indicates the predicted class of each data point (the labels: red=cross, green=dot, blue=square, cyan=circle). Note that several points are mislabeled. Right: The feature space after back-propagation of unified model. The NN has distorted the feature space to make classification easier for the topic model, with only a single data point now being misclassified.

We illustrate the different stages of training in our model with toy 2D data, as shown in Figure 3.4. The data is drawn from 4 classes, arranged in 5 crescent-shaped clusters (which cannot easily be separated by a Gaussian mixture model). Pre-training the NN (having a 2-50-2 architecture) for the most part preserves the structure of the input data (see Figure 2(middle)) in the feature space, thus the topic model, using Gaussian distributions makes many classification errors. But following back-propagation of the entire model (Figure 2(right)), the NN provides a significant warping of the input space, thus making it easier to separate the clusters. Note that the Gaussian clusters of the topic model do not lie directly on top of the features since the topic model optimizes a discriminative criterion, rather than a generative one.

3.5 Vision experiments

In this section, we provide a quantitative evaluation of our hybrid model on a vision dataset and compare its performance with alternative methods such as a standard neural

network, hierarchical topic model, pLSA, LDA and their variants.

3.5.1 Dataset and image features

We evaluated our hybrid model on challenging image scene recognition dataset [38] where the experimental results of standard probabilistic graphical models such as pLSA and LDA have been reported. This dataset consists of 1500 training images and 2998 test images each of which is labeled by one of 15 scene categories such as street, kitchen, coast, etc (Shown in Figure 3.3). Each image is represented by a set of SIFT descriptors which are sampled every 16 pixels (giving ~ 240 per image). Each SIFT descriptor is a 128-dimensional vector which encodes the histogram of gradients of a local image patch with size of 32×32. Since we focus on the theoretical issues involved when combining NN with graphical models, the pyramid representation [38] which leads to better performance is not used.

Figure 3.3: Images from 15-Scene DataSet

3.5.2 Scene modeling: topic model, neural network and the hybrid

Topic model. The baseline model of image scene is standard LDA [6, 14]. We first form visual vocabulary with size of 200 (following [14]) where visual words are obtained by vector quantization of SIFT descriptors. An image scene is represented by LDA which learns the latent topic distributions of visual words. When labels are provided in training, the variants of LDA such as supervised LDA [5] and discriminative LDA [37] can be applied. The performance of LDA is directly based on [14].

Hierarchical topic model. Unlike LDA, the hierarchical topic model introduced in section 3.2.2 integrates the representation of dictionary. The observation data input to HTM is SIFT features. The HTM is capable of learning the visual vocabulary and the topic distribution jointly. Our HTM method which makes use of class labels is related to the discriminative version of LDA [37]. We also follow the standard way [38, 37] to study the discriminative power of the inferred latent topics by training a SVM with the assigned topics as classification features. Note that the input to all variants of both HTM and LDA models are fixed visual vocabulary without the ability of learning transformation of low-level feature representation, i.e. SIFT descriptors.

Neural network. We use the same architecture of the neural network as described in section 3.2.1. In order to predict scene labels, we extend the neural network by imposing a softmax layer on the top which performs logistic regression of the scene labels and the output of the feature transformation. Learning the extended neural network is performed by standard back-propagation [40]. Unlike the topic models, this approach lacks a high level model of the scene.

Hybrid model: The hybrid model is a combination of the hierarchical topic model and the neural network which integrate the ability of learning low-level feature transformation and high-level scene representation. The input to the model is SIFT features, used by the other approaches. The free parameters and hyper-parameters are set to $S = 15$ (categories), $M = 45$ (topics), $K = 200$ (words), $\alpha = 1/3, \beta = 1/3$. Let d denote the

pLSA+SVM [38]	LDA [14]	Supervised LDA [5]	Neural Network	HTM
63.3	65.2	67.0	51.6 ± 1.1	64.9 ± 1.2
HTM+ SVM	Hybrid model no pre-train	Hybrid model pre-train NN only	Hybrid model pre-trained	Hybrid model fully trained
65.5 ± 1.5	47.2	52.5	65.7 ± 0.4	70.1 ± 0.6

Table 3.1: Classification rates of our model and other approaches on a scene classification dataset [38]. Our implementation of discriminative hierarchical topic model (HTM) is similar to Sudderth's scene model[64]. The performance of the HTM alone is close to the other two probabilistic models (pLSA+SVM) reported in [38] and discriminative LDA [14] which is evaluated on 13 categories. The method of "HTM+SVM" is a multi-class SVM with the input features of the latent topic assignments of HTM. Our hybrid model is a combination of neural network and HTM. We report the results of both pre-training and joint optimization, with the latter achieving a performance of 70.1%.

dimension of μ (=25). γ is set to $(\mu_0 = \mathbf{0}_d, \kappa = 0.1, \nu = d + 5, \Lambda_0 = \mathbf{I}_d)$. The NN has a $128 - 600 - 25$ architecture. Back-propagation is performed using conjugate gradients, with mini-batches of 75 images by ~ 200 features/image. Convergence occurs in about 70 iterations. The corresponding separate HTM and NN use the same parameter settings.

3.5.3 Results

We report the classification accuracy of the three types of methods in Table 3.1. Each model is trained on 5 random splits of training and test sets. We can see that the neural network, which lacks high-level representation, performs badly with classification accuracy of 51.6%. We also tried adding an extra hidden layer to better match the capacity of hybrid model, which resulted in a performance of 50.7%. The baseline HTM achieves 64.9% which is significantly better than neural network, and is comparable with other latent topic models LDA [14] (65.2%) and pLSA [38] (63.3%). The method of "HTM+SVM" which is a multi-class SVM using latent topic assignments of HTM as classification features, provides slightly improved predictions (65.5% vs 64.9%).

The hybrid model is analyzed after: i) pre-training and ii) full training with joint optimization. The pre-trained hybrid model achieves 65.7% , slightly better than HTM, which shows that simple pre-trained feature transformation offers similar predictions. The fully trained hybrid model further improves the classification accuracy to 70.1%

which is significantly better than HTM. It shows that joint optimization is capable of learning better low-level feature transformations for high-level topic modeling.

3.6 Discussion

We have introduced a unified representation that unifies two distinct classes of model that are widely used in machine learning and an end-to-end training scheme for the model. A number of improvements to our model could easily be incorporated. For example, a convolutional form of NN [40] could be used to directly learning from image pixels, or a spatial structure could be incorporated into the topic model, in the style of Sudderth et al. [64].

Our approach for joint training of the two models is a simple one that can be applied to more complex types of graphical model, provided (approximate) inference is possible in closed form. Finally, our model is not limited to image data and could easily be applied to other modalities such as text or audio.

Chapter 4

Regularization of Neural Networks using DropConnect

4.1 Introduction

Neural network (NN) models are well suited to domains where large labeled datasets are available, since their capacity can easily be increased by adding more layers or more units in each layer. However, big networks with millions or billions of parameters can easily overfit even the largest of datasets. Correspondingly, a wide range of techniques for regularizing NNs have been developed. Adding an ℓ_2 penalty on the network weights is one simple but effective approach. Other forms of regularization include: Bayesian methods [45], weight elimination [72] and early stopping of training. In practice, using these techniques when training big networks gives superior test performance to smaller networks trained without regularization.

Recently, Hinton et al. proposed a new form of regularization called Dropout [25]. For each training example, forward propagation involves randomly deleting half the activations in each layer. The error is then backpropagated only through the remaining activations. Extensive experiments show that this significantly reduces over-fitting and

improves test performance. Although a full understanding of its mechanism is elusive, the intuition is that it prevents the network weights from collaborating with one another to memorize the training examples.

In this paper, we propose DropConnect which generalizes Dropout by randomly dropping the weights rather than the activations. Like Dropout, the technique is suitable for fully connected layers only. We compare and contrast the two methods on four different image datasets.

4.2 Motivation

To demonstrate our method we consider a fully connected layer of a neural network with input $v = [v_1, v_2, \ldots, v_n]^T$ and weight parameters W (of size $d \times n$). The output of this layer, $r = [r_1, r_2, \ldots, r_d]^T$ is computed as a matrix multiply between the input vector and the weight matrix followed by a non-linear activation function, a, (biases are included in W with a corresponding fixed input of 1 for simplicity):

$$r = a(u) = a(Wv) \tag{4.1}$$

4.2.1 Dropout

Dropout was proposed by [25] as a form of regularization for fully connected neural network layers. Each element of a layer's output is kept with probability p, otherwise being set to 0 with probability $(1 - p)$. Extensive experiments show that Dropout improves the network's generalization ability, giving improved test performance.

When Dropout is applied to the outputs of a fully connected layer, we can write Eq. 4.1 as:

$$r = m \star a(Wv) \tag{4.2}$$

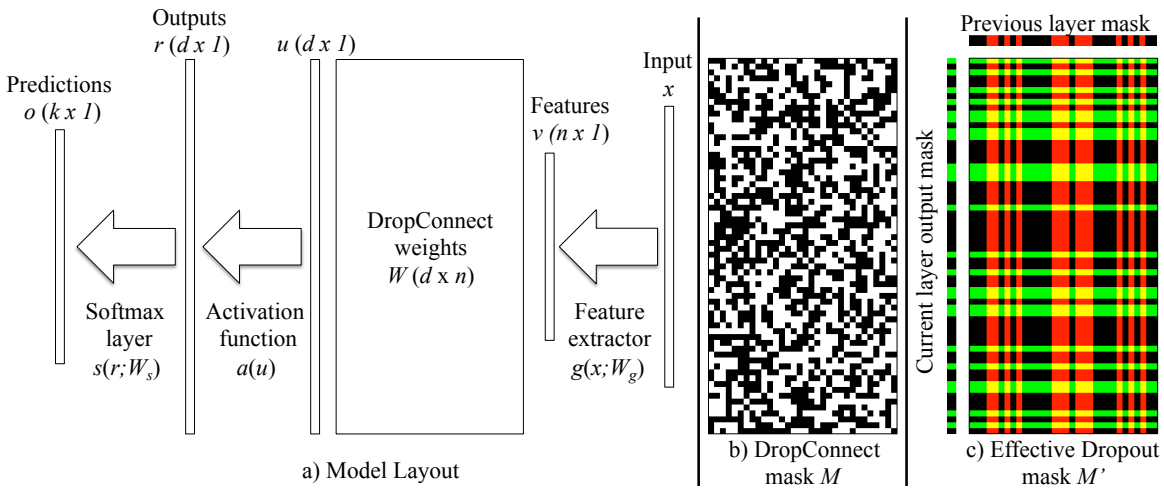


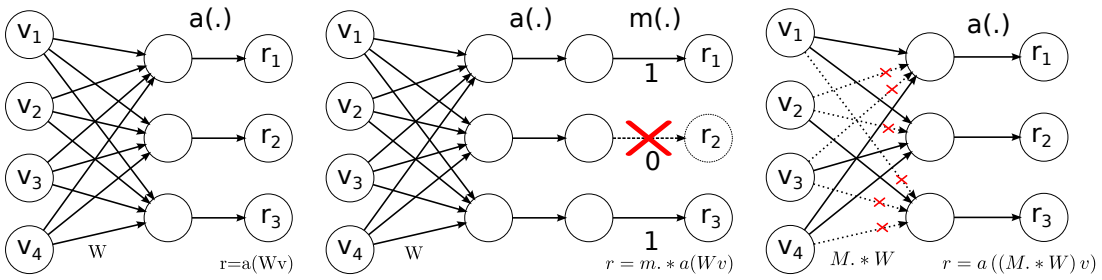
Figure 4.1: (a): An example model layout for a single DropConnect layer. After running feature extractor $g()$ on input x, a random instantiation of the mask M (e.g. (b)), masks out the weight matrix W. The masked weights are multiplied with this feature vector to produce u which is the input to an activation function a and a softmax layer s. For comparison, (c) shows an effective weight mask for elements that Dropout uses when applied to the previous layer's output (red columns) and this layer's output (green rows). Note the lack of structure in (b) compared to (c).

where $\star$ denotes element wise product and m is a binary mask vector of size d with each element, j, drawn independently from $m_j \sim Bernoulli(p)$.

Many commonly used activation functions such as *tanh*, centered *sigmoid* and *relu* [47], have the property that $a(0) = 0$. Thus, Eq. 4.2 could be re-written as, $r = a(m \star Wv)$, where Dropout is applied at the inputs to the activation function.

4.2.2 DropConnect

DropConnect is the generalization of Dropout in which each connection, rather than each output unit, can be dropped with probability $1 - p$. DropConnect is similar to Dropout as it introduces dynamic sparsity within the model, but differs in that the sparsity is on the weights W, rather than the output vectors of a layer. In other words, the fully connected layer with DropConnect becomes a sparsely connected layer in which the connections are chosen at *random* during the training stage. Note that this is not



(a) Normal Network (b) DropOut Network (c) DropConnect Network

Figure 4.2: Comparison between Normal Network, DropOut Network and DropConnect Network

equivalent to setting W to be a *fixed* sparse matrix during training.

For a DropConnect layer, the output is given as:

$$r = a\left(\left(M \star W\right) v\right) \tag{4.3}$$

where M is a binary matrix encoding the connection information and $M_{ij} \sim Bernoulli(p)$. Each element of the mask M is drawn independently for each example during training, essentially instantiating a different connectivity for each example seen. Additionally, the biases are also masked out during training. From Eq. 4.2 and Eq. 4.3, it is evident that DropConnect is the generalization of Dropout to the full connection structure of a layer[1]. Figure 4.2 illustration the difference between different kinds of networks.

The paper structure is as follows: we outline details on training and running inference in a model using DropConnect in section 3, followed by theoretical justification for DropConnect in section 4, GPU implementation specifics in section 5, and experimental results in section 6.

[1]This holds when $a(0) = 0$, as is the case for *tanh* and *relu* functions.

4.3 Model Description

We consider a standard model architecture composed of four basic components (see Figure 4.1a):

1. Feature Extractor: $v = g(x; W_g)$ where v are the output features, x is input data to the overall model, and W_g are parameters for the feature extractor. We choose $g()$ to be a multi-layered convolutional neural network (CNN) [40], with W_g being the convolutional filters (and biases) of the CNN.

2. DropConnect Layer: $r = a(u) = a((M \star W)v)$ where v is the output of the feature extractor, W is a fully connected weight matrix, a is a non-linear activation function and M is the binary mask matrix.

3. Softmax Classification Layer: $o = s(r; W_s)$ takes as input r and uses parameters W_s to map this to a k dimensional output (k being the number of classes).

4. Cross Entropy Loss: $A(y, o) = -\sum_{i=1}^{k} y_i log(o_i)$ takes probabilities o and the ground truth labels y as input.

The overall model $f(x; \theta, M)$ therefore maps input data x to an output o through a sequence of operations given the parameters $\theta = \{W_g, W, W_s\}$ and randomly-drawn mask M. The correct value of o is obtained by summing out over all possible masks M:

$$o = \mathbf{E}_M \left[f(x; \theta, M) \right] = \sum_M p(M) f(x; \theta, M) \tag{4.4}$$

This reveals the mixture model interpretation of DropConnect (and Dropout), where the output is a mixture of $2^{|M|}$ different networks, each with weight $p(M)$. If $p = 0.5$, then these weights are equal and $o = \frac{1}{|M|} \sum_M f(x; \theta, M) = \frac{1}{|M|} \sum_M s(a((M \star W)v); W_s)$

4.3.1 Training

Training the model described in Section 4.3 begins by selecting an example x from the training set and extracting features for that example, v. These features are input to the DropConnect layer where a mask matrix M is first drawn from a $Bernoulli(p)$ distribution to mask out elements of both the weight matrix and the biases in the DropConnect layer. A key component to successfully training with DropConnect is the selection of a different mask for each training example. Selecting a single mask for a subset of training examples, such as a mini-batch of 128 examples, does not regularize the model enough in practice. Since the memory requirement for the M's now grows with the size of each mini-batch, the implementation needs to be carefully designed as described in Section 4.5.

Once a mask is chosen, it is applied to the weights and biases in order to compute the input to the activation function. This results in r, the input to the softmax layer which outputs class predictions from which cross entropy between the ground truth labels is computed. The parameters throughout the model θ then can be updated via stochastic gradient descent (SGD) by backpropagating gradients of the loss function with respect to the parameters, A'_θ. To update the weight matrix W in a DropConnect layer, the mask is applied to the gradient to update only those elements that were active in the forward pass. Additionally, when passing gradients down to the feature extractor, the masked weight matrix $M \star W$ is used. A summary of these steps is provided in Algorithm 2.

4.3.2 Inference

At inference time, we need to compute $r = 1/|M| \sum_M a((M \star W)v)$, which naively requires the evaluation of $2^{|M|}$ different masks – plainly infeasible.

The Dropout work [25] made the approximation: $\sum_M a((M \star W)v) \approx a(\sum_M (M \star W)v)$, i.e. averaging before the activation rather than after. Although this seems to work in

Algorithm 2 SGD Training with DropConnect

Input: example x, parameters θ_{t-1} from step $t-1$, learning rate η
Output: updated parameters θ_t
Forward Pass:
Extract features: $v \leftarrow g(x; W_g)$
Random sample M mask: $M_{ij} \sim Bernoulli(p)$
Compute activations: $r = a((M \star W)v)$
Compute output: $o = s(r; W_s)$
Backpropagate Gradients:
Differentiate loss A'_θ with respect to parameters θ:
Update softmax layer: $W_s = W_s - \eta A'_{W_s}$
Update DropConnect layer: $W = W - \eta(M \star A'_W)$
Update feature extractor: $W_g = W_g - \eta A'_{W_g}$

Algorithm 3 Inference with DropConnect

Input: example x, parameters θ, # of samples Z.
Output: prediction u
Extract features: $v \leftarrow g(x; W_g)$
Moment matching of u:
$\mu \leftarrow E_M[u] \quad \sigma^2 \leftarrow V_M[u]$
for z $= 1 : Z$ **do** %% Draw Z samples
for i $= 1 : d$ **do** %% Loop over units in r
Sample from 1D Gaussian $u_{i,z} \sim \mathcal{N}(\mu_i, \sigma_i^2)$
$r_{i,z} \leftarrow a(u_{i,z})$
end for
end for
Pass result $\hat{r} = \sum_{z=1}^{Z} r_z/Z$ to next layer

practice, it is not justified mathematically, particularly for the *relu* activation function.[2]

We take a different approach. Consider a single unit u_i before the activation function $a()$: $u_i = \sum_j (W_{ij} v_j) M_{ij}$. This is a weighted sum of Bernoulli variables M_{ij}, which can be approximated by a Gaussian via moment matching. The mean and variance of the units u are: $E_M[u] = pWv$ and $V_M[u] = p(1-p)(W \star W)(v \star v)$. We can then draw samples from this Gaussian and pass them through the activation function $a()$ before averaging them and presenting them to the next layer. Algorithm 3 summarizes the method. Note that the sampling can be done efficiently, since the samples for each unit and example can be drawn in parallel. This scheme is only an approximation in the case of multi-layer network, it works well in practise as shown in Experiments.

[2]Consider $u \sim N(0,1)$, with $a(u) = max(u, 0)$. $a(E_M(u)) = 0$ but $E_M(a(u)) = 1/\sqrt{2\pi} \approx 0.4$.

Mask Weight Impl	Time(ms)				Speedup
	fprop	bprop acts	bprop weights	total	
CPU-float	480.2	1228.6	1692.8	3401.6	1.0 ×
CPU-bit	392.3	679.1	759.7	1831.1	1.9 ×
GPU-float global memory	21.6	6.2	7.2	35.0	97.2 ×
GPU-float tex1D memory	15.1	6.1	6.0	27.2	126.0 ×
GPU-bit tex2D memory	2.4	2.7	3.1	**8.2**	414.8 ×
GPU-Lower Bound cuBlas+read mask	0.3	0.3	0.2	0.8	

Table 4.1: Performance comparison between different implementations of our DropConnect layer on NVidia GTX580 GPU relative to a 2.67Ghz Intel Xeon (compiled with `-O3` flag). Input dimension and Output dimension are 1024 and mini-batch size is 128. As reference we provide traditional matrix multiplication using the cuBlas library.

4.4 Model Generalization Bound

We now show a novel bound for the Rademacher complexity of the model $\hat{R}_\ell(\mathcal{F})$ on the training set (see Section 4.8 for derivation):

$$\hat{R}_\ell(\mathcal{F}) \leq p\left(2\sqrt{k}dB_s n\sqrt{d}B_h\right)\hat{R}_\ell(\mathcal{G}) \tag{4.5}$$

where $max|W_s| \leq B_s$, $max|W| \leq B$, k is the number of classes, $\hat{R}_\ell(\mathcal{G})$ is the Rademacher complexity of the feature extractor, n and d are the dimensionality of the input and output of the DropConnect layer respectively. The important result from Eq. 4.5 is that the complexity is a linear function of the probability p of an element being kept in DropConnect or Dropout. When $p = 0$, the model complexity is zero, since the input has no influence on the output. When $p = 1$, it returns to the complexity of a standard model.

4.5 Implementation Details

Our system involves three components implemented on a GPU: 1) a feature extractor, 2) our DropConnect layer, and 3) a softmax classification layer. For 1 and 3 we utilize the Cuda-convnet package [32], a fast GPU based convolutional network library. We implement a custom GPU kernel for performing the operations within the DropConnect layer. Our code is available at `http:///cs.nyu.edu/~wanli/dropc`.

A typical fully connected layer is implemented as a matrix-matrix multiplication between the input vectors for a mini-batch of training examples and the weight matrix. The difficulty in our case is that each training example requires it's own random mask matrix applied to the weights and biases of the DropConnect layer. This leads to several complications:

1. For a weight matrix of size $d \times n$, the corresponding mask matrix is of size $d \times n \times b$ where b is the size of the mini-batch. For a 4096×4096 fully connected layer with mini-batch size of 128, the matrix would be too large to fit into GPU memory if each element is stored as a floating point number, requiring 8G of memory.

2. Once a random instantiation of the mask is created, it is non-trivial to access all the elements required during the matrix multiplications so as to maximize performance.

The first problem is not hard to address. Each element of the mask matrix is stored as a single bit to encode the connectivity information rather than as a float. The memory cost is thus reduced by 32 times, which becomes 256M for the example above. This not only reduces the memory footprint, but also reduces the bandwidth required as 32 elements can be accessed with each 4-byte read. We overcome the second problem using an efficient memory access pattern using 2D texture aligned memory. These two improvements are crucial for an efficient GPU implementation of DropConnect as shown in Table 4.1. Here we compare to a naive CPU implementation with floating point masks and get a 415× speedup with our efficient GPU design.

4.6 Experiments

We evaluate our DropConnect model for regularizing deep neural networks trained for image classification. All experiments use mini-batch SGD with momentum on batches of 128 images with the momentum parameter fixed at 0.9.

We use the following protocol for all experiments unless otherwise stated:

- Augment the dataset by: 1) randomly selecting cropped regions from the images, 2) flipping images horizontally, 3) introducing 15% scaling and rotation variations.
- Train 5 independent networks with random permutations of the training sequence.
- Manually decrease the learning rate if the network stops improving as in [32] according to a schedule determined on a validation set.
- Train the fully connected layer using Dropout, DropConnect, or neither (No-Drop).
- At inference time for DropConnect we draw $Z = 1000$ samples at the inputs to the activation function of the fully connected layer and average their activations.

To anneal the initial learning rate we choose a fixed multiplier for different stages of training. We report three numbers of epochs, such as 600-400-200 to define our schedule. We multiply the initial rate by 1 for the first such number of epochs. Then we use a multiplier of 0.5 for the second number of epochs followed by 0.1 again for this second number of epochs. The third number of epochs is used for multipliers of 0.05, 0.01, 0.005, and 0.001 in that order, after which point we report our results. We determine the epochs to use for our schedule using a validation set to look for plateaus in the loss function, at which point we move to the next multiplier. [3]

Once the 5 networks are trained we report two numbers: 1) the mean and standard deviation of the classification errors produced by each of the 5 independent networks,

[3] In all experiments the bias learning rate is $2\times$ the learning rate for the weights. Additionally weights are initialized with $N(0, 0.1)$ random values for fully connected layers and $N(0, 0.01)$ for convolutional layers.

and 2) the classification error that results when averaging the output probabilities from the 5 networks before making a prediction. We find in practice this voting scheme, inspired by [8], provides significant performance gains, achieving state-of-the-art results in many standard benchmarks when combined with our DropConnect layer.

4.6.1 MNIST

The MNIST handwritten digit classification task [40] consists of 28×28 black and white images, each containing a digit 0 to 9 (10-classes). Each digit in the $60,000$ training images and $10,000$ test images is normalized to fit in a 20×20 pixel box while preserving their aspect ratio. We scale the pixel values to the $[0, 1]$ range before inputting to our models.

For our first experiment on this dataset, we train models with two fully connected layers each with 800 output units using either *tanh*, *sigmoid* or *relu* activation functions to compare to Dropout in [25]. The first layer takes the image pixels as input, while the second layer's output is fed into a 10-class softmax classification layer. In Table 4.2 we show the performance of various activations functions, comparing No-Drop, Dropout and DropConnect in the fully connected layers. No data augmentation is utilized in this experiment. We use an initial learning rate of 0.1 and train for 600-400-20 epochs using our schedule.

neuron	model	error(%) 5 network	voting error(%)
relu	No-Drop	1.62 ± 0.037	1.40
	Dropout	1.28 ± 0.040	1.20
	DropConnect	1.20 ± 0.034	**1.12**
sigmoid	No-Drop	1.78 ± 0.037	1.74
	Dropout	1.38 ± 0.039	**1.36**
	DropConnect	1.55 ± 0.046	1.48
tanh	No-Drop	1.65 ± 0.026	1.49
	Dropout	1.58 ± 0.053	1.55
	DropConnect	1.36 ± 0.054	**1.35**

Table 4.2: MNIST classification error rate for models with two fully connected layers of 800 neurons each. No data augmentation is used in this experiment.

From Table 4.2 we can see that both Dropout and DropConnect perform better than not using either method. DropConnect mostly performs better than Dropout in this task, with the gap widening when utilizing the voting over the 5 models.

To further analyze the effects of DropConnect, we show three explanatory experiments in Figure 4.3 using a 2-layer fully connected model on MNIST digits. Figure 4.3a shows test performance as the number of hidden units in each layer varies. As the model size increases, No-Drop overfits while both Dropout and DropConnect improve performance. DropConnect consistently gives a lower error rate than Dropout. Figure 4.3b shows the effect of varying the drop rate p for Dropout and DropConnect for a 400-400 unit network. Both methods give optimal performance in the vicinity of 0.5, the value used in all other experiments in the paper. Our sampling approach gives a performance gain over mean inference (as used by Hinton [25]), but only for the DropConnect case. In Figure 4.3c we plot the convergence properties of the three methods throughout training on a 400-400 network. We can see that No-Drop overfits quickly, while Dropout and DropConnect converge slowly to ultimately give superior test performance. DropConnect is even slower to converge than Dropout, but yields a lower test error in the end.

In order to improve our classification result, we choose a more powerful feature extractor network described in [8] (*relu* is used rather than *tanh*). This feature extractor consists of a 2 layer CNN with 32-64 feature maps in each layer respectively. The last layer's output is treated as input to the fully connected layer which has 150 *relu* units on which No-Drop, Dropout or DropConnect are applied. We report results in Table 4.3 from training the network on a) the original MNIST digits, b) cropped 24×24 images from random locations, and c) rotated and scaled versions of these cropped images. We use an initial learning rate of 0.01 with a 700-200-100 epoch schedule, no momentum and preprocess by subtracting the image mean.

We note that our approach surpasses the state-of-the-art result of 0.23% [8], achieving a **0.21**% error rate, without the use of elastic distortions (as used by [8]).

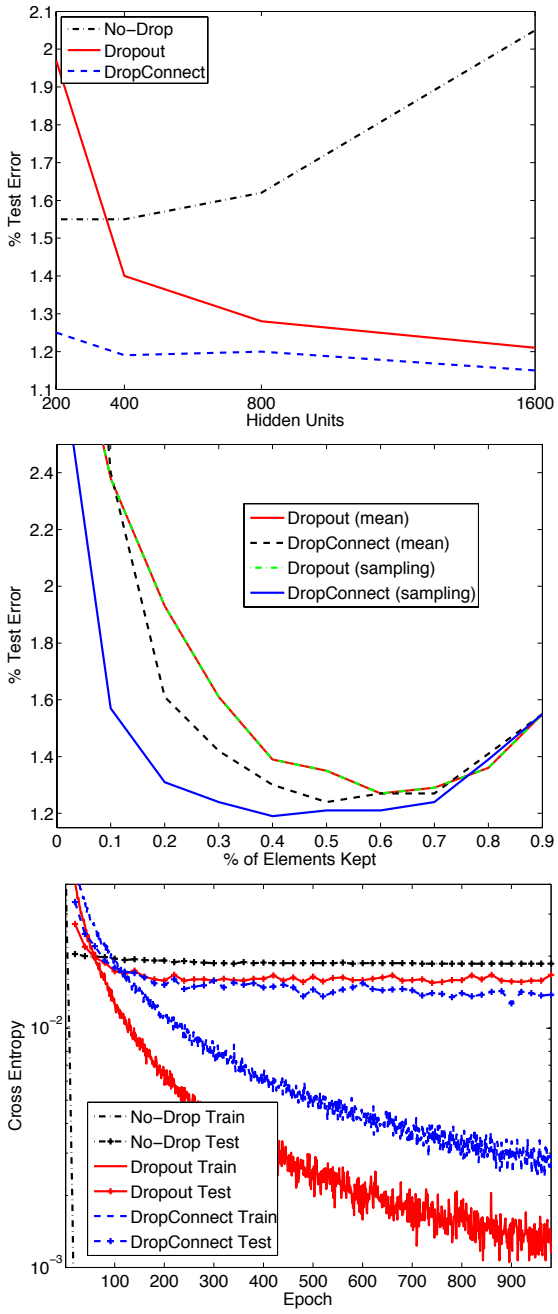


Figure 4.3: Using the MNIST dataset, in a) we analyze the ability of Dropout and DropConnect to prevent overfitting as the size of the 2 fully connected layers increase. b) Varying the drop-rate in a 400-400 network shows near optimal performance around the $p = 0.5$ proposed by [25]. c) we show the convergence properties of the train/test sets. See text for discussion.

crop	rotation scaling	model	error(%) 5 network	voting error(%)
no	no	No-Drop	0.77 ± 0.051	0.67
		Dropout	0.59 ± 0.039	**0.52**
		DropConnect	0.63 ± 0.035	0.57
yes	no	No-Drop	0.50 ± 0.098	0.38
		Dropout	0.39 ± 0.039	0.35
		DropConnect	0.39 ± 0.047	**0.32**
yes	yes	No-Drop	0.30 ± 0.035	**0.21**
		Dropout	0.28 ± 0.016	0.27
		DropConnect	0.28 ± 0.032	**0.21**

Table 4.3: MNIST classification error. Previous state of the art is *0.47%* [75] for a single model without elastic distortions and *0.23%* with elastic distortions and voting [8].

4.6.2 CIFAR-10

CIFAR-10 is a data set of natural 32x32 RGB images [31] in 10-classes with $50,000$ images for training and $10,000$ for testing. Figure 4.4 shows a few images from CIFAR-10. Before inputting these images to our network, we subtract the per-pixel mean computed over the training set from each image.

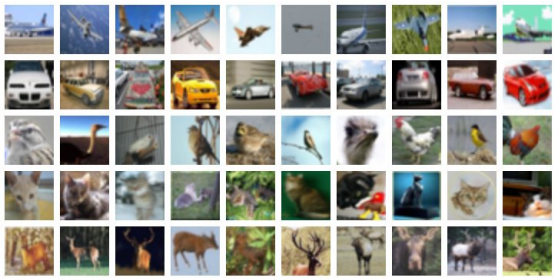


Figure 4.4: Images from CIFAR-10

The first experiment on CIFAR-10 (summarized in Table 4.4) uses the simple convolutional network feature extractor described in [32](layers-80sec.cfg) that is designed for rapid training rather than optimal performance. On top of the 3-layer feature extractor we have a 64 unit fully connected layer which uses No-Drop, Dropout, or DropConnect. No data augmentation is utilized for this experiment. Since this experiment is not aimed at optimal performance we report a single model's performance without voting. We train for 150-0-0 epochs with an initial learning rate of 0.001 and their default weight decay.

DropConnect prevents overfitting of the fully connected layer better than Dropout in this experiment.

model	error(%)
No-Drop	23.5
Dropout	19.7
DropConnect	**18.7**

Table 4.4: CIFAR-10 classification error using the simple feature extractor described in [32](layers-80sec.cfg) and with no data augmentation.

Table 4.5 shows classification results of the network using a larger feature extractor with 2 convolutional layers and 2 locally connected layers as described in [32](layers-conv-local-11pct.cfg). A 128 neuron fully connected layer with *relu* activations is added between the softmax layer and feature extractor. Following [32], images are cropped to 24x24 with horizontal flips and no rotation or scaling is performed. We use an initial learning rate of 0.001 and train for 700-300-50 epochs with their default weight decay. Model voting significantly improves performance when using Dropout or DropConnect, the latter reaching an error rate of 9.41%. Additionally, we trained a model with 12 networks with DropConnect and achieved a state-of-the-art result of **9.32**%, indicating the power of our approach.

model	error(%) 5 network	voting error(%)
No-Drop	11.18± 0.13	10.22
Dropout	11.52± 0.18	9.83
DropConnect	11.10± 0.13	**9.41**

Table 4.5: CIFAR-10 classification error using a larger feature extractor. Previous state-of-the-art is 9.5% [62]. Voting with 12 DropConnect networks produces an error rate of **9.32**%, significantly beating the state-of-the-art.

4.6.3 SVHN

The Street View House Numbers (SVHN) dataset includes $604,388$ images (both training set and extra set) and $26,032$ testing images [48]. Similar to MNIST, the goal is to

classify the digit centered in each 32x32 RGB image. Images from SVHN is illustrate in Figure 4.5.

Figure 4.5: Images from SVHN

Due to the large variety of colors and brightness variations in the images, we pre-process the images using local contrast normalization as in [75]. The feature extractor is the same as the larger CIFAR-10 experiment, but we instead use a larger 512 unit fully connected layer with *relu* activations between the softmax layer and the feature extractor. After contrast normalizing, the training data is randomly cropped to 28×28 pixels and is rotated and scaled. We do not do horizontal flips. Table 4.6 shows the classification performance for 5 models trained with an initial learning rate of 0.001 for a 100-50-10 epoch schedule.

Due to the large training set size both Dropout and DropConnect achieve nearly the same performance as No-Drop. However, using our data augmentation techniques and careful annealing, the per model scores easily surpass the previous 2.80% state-of-the-art result of [75]. Furthermore, our voting scheme reduces the relative error of the previous state-of-to-art by 30% to achieve **1.94**% error.

model	error(%) 5 network	voting error(%)
No-Drop	2.26 ± 0.072	**1.94**
Dropout	2.25 ± 0.034	1.96
DropConnect	2.23 ± 0.039	**1.94**

Table 4.6: SVHN classification error. The previous state-of-the-art is *2.8%* [75].

4.6.4 NORB

In the final experiment we evaluate our models on the 2-fold NORB (jittered-cluttered) dataset [41], a collection of stereo images of 3D models (Examples in Figure 4.6). For each image, one of 6 classes appears on a random background. We train on 2-folds of $29,160$ images each and the test on a total of $58,320$ images. The images are downsampled from 108×108 to 48×48 as in [8].

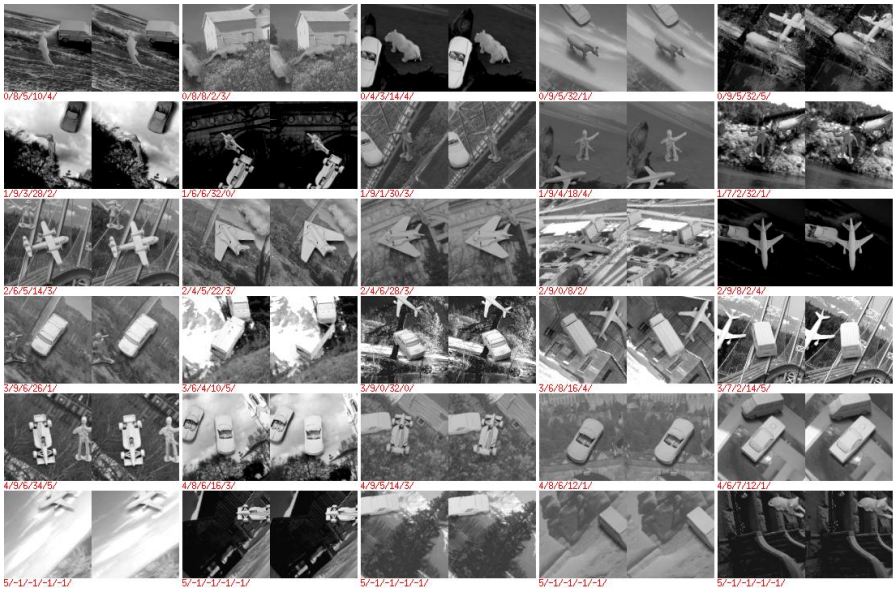


Figure 4.6: Images from NORB

model	error(%) 5 network	voting error(%)
No-Drop	4.48 ± 0.78	3.36
Dropout	3.96 ± 0.16	**3.03**
DropConnect	4.14 ± 0.06	3.23

Table 4.7: NORM classification error for the jittered-cluttered dataset, using 2 training folds. The previous state-of-art is *3.57%* [8].

We use the same feature extractor as the larger CIFAR-10 experiment. There is a 512

unit fully connected layer with *relu* activations placed between the softmax layer and feature extractor. Rotation and scaling of the training data is applied, but we do not crop or flip the images as we found that to hurt performance on this dataset. We trained with an initial learning rate of 0.001 and anneal for 100-40-10 epochs.

In this experiment we beat the previous state-of-the-art result of 3.57% using No-Drop, Dropout and DropConnect with our voting scheme. While Dropout surpasses DropConnect slightly, both methods improve over No-Drop in this benchmark as shown in Table 4.7.

4.7 Discussion

We have presented DropConnect, which generalizes Hinton et al. 's Dropout [25] to the entire connectivity structure of a fully connected neural network layer. We provide both theoretical justification and empirical results to show that DropConnect helps regularize large neural network models. Results on a range of datasets show that DropConnect often outperforms Dropout. While our current implementation of DropConnect is slightly slower than No-Drop or Dropout, in large models models the feature extractor is the bottleneck, thus there is little difference in overall training time. DropConnect allows us to train large models while avoiding overfitting. This yields state-of-the-art results on a variety of standard benchmarks using our efficient GPU implementation of DropConnect.

4.8 Theoretical Analysis of DropConnet Network

4.8.1 Preliminaries

Definition 1 (DropConnect Network)**.** *Given data set S with ℓ entries: $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_\ell\}$ with labels $\{y_1, y_2, \ldots, y_\ell\}$, we define DropConnect network as a mixture model:*

$$\mathbf{o} = \sum_m p(M) f(\mathbf{x}; \theta, M) = \mathbf{E}_m \left[f(\mathbf{x}; \theta, M) \right] \tag{4.6}$$

Each network $f(x; \theta, M)$ has weights $p(M)$ and network parameters are $\theta = \{W_s, W, W_g\}$. W_s are the softmax layer parameters, w are the DropConnect layer parameters and W_g are the feature extractor parameters. Further more, m is the DropConnect layer mask.

Remark 1. *when each element of M_i has equal probability of being on and off ($p = 0.5$), the mixture model has equal weights for all sub-models $f(\mathbf{x}; \theta, M)$, otherwise the mixture model has larger weights in some sub-models than others.*

Reformulate cross-entropy loss on top of soft-max into a single parameter function that combines soft-max output and labels. Same as logistic.

Definition 2 (Logistic Loss)**.** *The following loss function defined on k-class classification is call logistic loss function:*

$$A_y(o) = -\sum_i y_i \ln \frac{\exp o_i}{\sum_j \exp(o_j)} = -o_i + \ln \sum_j \exp(o_j)$$

where y is binary vector with i^{th} bit set on

Lemma 1. *Logistic loss function A has the following properties:*

1. $A_y(0) = \ln k$
2. $-1 \leq A'_y(o) \leq 1$
3. $A''_y(o) \geq 0$.

The first one says $A(0)$ is depend on some constant related with number of labels. The

second one says A is Lipschitz function with $L = 1$. The third one says A is a convex function w.r.t x.

Definition 3 (Rademacher complexity)**.** *For a sample $S = \{x_1, \ldots, x_\ell\}$ generated by a distribution D on set X and a real-valued function class $\mathcal{F}$ in domain X, the empirical Rademacher complexity of $\mathcal{F}$ is the random variable:*

$$\hat{R}_\ell(\mathcal{F}) = \mathbf{E}_\sigma \left[\sup_{f \in \mathcal{F}} \left| \frac{2}{\ell} \sum_{i=1}^{\ell} \sigma_i f(x_i) \right| \mid x_1, \ldots, x_\ell \right]$$

where sigma $= \{\sigma_1, \ldots, \sigma_\ell\}$ are independent uniform $\{\pm 1\}$-valued (Rademacher) random variables. The Rademacher complexity of $\mathcal{F}$ is $R_\ell(\mathcal{F}) = \mathbf{E}_S \left[\hat{R}_\ell(\mathcal{F}) \right]$.

Theorem 1 ([30])**.** *Fix $\delta \in (0, 1)$ and let $\mathcal{F}$ be a class of functions mapping from M to $[0, 1]$. Let $(M_i)_{i-1}^{\ell}$ be drawn independently according to a probability distribution D. Then with probability at least $1 - \delta$ over random draws of samples of size ℓ, every $f \in \mathcal{F}$ satisfies:*

$$\begin{aligned} \mathbf{E}[f(M)] &\leq \hat{\mathbf{E}}[f(M)] + R_\ell(\mathcal{F}) + \sqrt{\frac{ln(2/\delta)}{2\ell}} \\ &\leq \hat{\mathbf{E}}[f(M)] + \hat{R}_\ell(\mathcal{F}) + 3\sqrt{\frac{ln(2/\delta)}{2\ell}} \end{aligned}$$

4.8.2 Bound Derivation

Theorem 2 ([42])**.** *Let $\mathcal{F}$ be class of real functions. If $\mathcal{A}$: $\mathbf{R} \to \mathbf{R}$ is Lipschitz with constant L and satisfies $\mathcal{A}(0) = 0$, then $\hat{R}_\ell(\mathcal{A} \circ F) \leq 2L\hat{R}(\mathcal{F})$*

Lemma 2. *Let $\mathcal{F}$ be class of real functions and $\mathcal{H} = [\mathcal{F}_j]_{j=1}^{k}$ be a k-dimensional function class. If $\mathcal{A}$: $\mathbf{R}^k \to \mathbf{R}$ is a Lipschitz function with constant L and satisfies $\mathcal{A}(0) = 0$, then $\hat{R}_\ell(\mathcal{A} \circ \mathcal{H}) \leq 2kL\hat{R}_\ell(\mathcal{F})$*

Lemma 3 (Classifier Generalization Bound)**.** *Generalization bound of a k-class classifier*

with logistic loss function is directly related Rademacher complexity of that classifier

$$\mathbf{E}[A_y(o)] \leq \frac{1}{\ell}\sum_{i=1}^{\ell} A_{y_i}(o_i) + 2k\hat{R}_\ell(\mathcal{F}) + 3\sqrt{\frac{ln\,(2/\delta)}{2\ell}}$$

Proof. From Lemma 1, Logistic loss function $(A-c)(x) \in \mathcal{A}$ due to $(A-c)'(x) \leq 1$ and $(A-c)(0) = 0$ with some constant c. By Lemma 2: $\hat{R}_\ell((A-c)\circ\mathcal{F}) \leq 2k\hat{R}_\ell(\mathcal{F})$ □

Lemma 4. *For all neuron activations: sigmoid, tanh and relu, we have:* $\hat{R}_\ell(a\circ\mathcal{F}) \leq 2\hat{R}_\ell(\mathcal{F})$

Lemma 5 (Network Layer Bound)**.** *Let* $\mathcal{G}$ *be the class of real functions* $R^d \to R$ *with input dimension* $\mathcal{F}$*, i.e.* $\mathcal{G} = [\mathcal{F}_j]_{j=1}^d$ *and* $\mathcal{H}_B$ *is a linear transform function parameterized by* W *with* $\|W\|_2 \leq B$*, then* $\hat{R}_\ell(\mathcal{H}\circ\mathcal{G}) \leq \sqrt{d}B\hat{R}_\ell(\mathcal{F})$

Proof.

$$\begin{aligned}
& \hat{R}_\ell(\mathcal{H}\circ\mathcal{G}) \\
= \;& \mathbf{E}_\sigma\left[\sup_{h\in\mathcal{H},g\in\mathcal{G}}\left|\frac{2}{\ell}\sum_{i=1}^{\ell}\sigma_i h\circ g(x_i)\right|\right] \\
= \;& \mathbf{E}_\sigma\left[\sup_{g\in\mathcal{G},\|W\|\leq B}\left|\left\langle W, \frac{2}{\ell}\sum_{i=1}^{\ell}\sigma_i g(x_i)\right\rangle\right|\right] \\
\leq \;& B\mathbf{E}_\sigma\left[\sup_{f^j\in\mathcal{F}}\left\|\left[\frac{2}{\ell}\sum_{i=1}^{\ell}\sigma_i^j f^j(x_i)\right]_{j=1}^d\right\|\right] \\
= \;& B\sqrt{d}\mathbf{E}_\sigma\left[\sup_{f\in\mathcal{F}}\left|\frac{2}{\ell}\sum_{i=1}^{\ell}\sigma_i f(x_i)\right|\right] = \sqrt{d}B\hat{R}_\ell(\mathcal{F})
\end{aligned}$$

□

Remark 2. *Given a layer in our network, we denote the function of all layers before as* $\mathcal{G} = [\mathcal{F}_j]_{j=1}^d$*. This layer has the linear transformation function* $\mathcal{H}$ *and activation function* a*. By Lemma 4 and Lemma 5, we know the network complexity is bounded by:*

$$\hat{R}_\ell(\mathcal{H}\circ\mathcal{G}) \leq c\sqrt{d}B\hat{R}_\ell(\mathcal{F})$$

where $c = 1$ for identity neuron and $c = 2$ for others.

Lemma 6. *Let $\mathcal{F}_M$ be the class of real functions that depend on m, then $\hat{R}_\ell(\mathbf{E}_M[\mathcal{F}_M]) \leq \mathbf{E}_M\left[\hat{R}_\ell(\mathcal{F}_M)\right]$*

Proof.

$$\begin{aligned}\hat{R}_\ell(\mathbf{E}_M[\mathcal{F}_M]) &= \hat{R}_\ell\left(\sum_M p(M)\mathcal{F}_M\right) \leq \sum_M \hat{R}_\ell(p(M)\mathcal{F}_M) \\ &\leq \sum_M |p(M)|\hat{R}_\ell(\mathcal{F}_M) = \mathbf{E}_M\left[\hat{R}_\ell(\mathcal{F}_M)\right]\end{aligned}$$

because of common fact: 1) $\hat{R}_\ell(c\mathcal{F}) = |c|\hat{R}_\ell(\mathcal{F})$ and 2) $\hat{R}_\ell(\sum_i \mathcal{F}_i) \leq \sum_i \hat{R}_\ell(\mathcal{F}_i)$ □

Theorem 3 (DropConnect Network Complexity)**.** *Consider the DropConnect neural network defined in Definition 1. Let $\hat{R}_\ell(\mathcal{G})$ be the empirical Rademacher complexity of the feature extractor and $\hat{R}_\ell(\mathcal{F})$ be the empirical Rademacher complexity of the whole network. In addition, we assume:*

1. *weight parameter of DropConnect layer $|W| \leq B_h$*
2. *weight parameter of s, i.e. $|W_s| \leq B_s$ (L2-norm of it is bounded by $\sqrt{dk}B_s$).*

Then we have:

$$\hat{R}_\ell(\mathcal{F}) \leq p\left(2\sqrt{k}dB_s n\sqrt{d}B_h\right)\hat{R}_\ell(\mathcal{G})$$

Proof.

$$\begin{aligned}&\hat{R}_\ell(\mathcal{F}) = \hat{R}_\ell(\mathbf{E}_M[f(\mathbf{x};\theta,M)]) \\ \leq\ & \mathbf{E}_M\left[\hat{R}_\ell(f(\mathbf{x};\theta,M)\right] && (4.7)\\ =\ & \mathbf{E}_M\left[\hat{R}_\ell(s\circ a\circ h_m\circ g)\right] \\ \leq\ & (\sqrt{dk}B_s)\sqrt{d}\mathbf{E}_M\left[\hat{R}_\ell(a\circ h_m\circ g)\right] && (4.8)\\ =\ & 2\sqrt{k}dB_s\mathbf{E}_M\left[\hat{R}_\ell(h_m\circ g)\right] && (4.9)\end{aligned}$$

where $h_m = (M \star W)v$. Equation (4.7) is based on Lemma 6, Equation (4.8) is based

on Lemma 5 and Equation (4.9) follows from Lemma 4.

$$
\begin{aligned}
& \mathbf{E}_M\left[\hat{R}_\ell(h_m \circ g)\right] \\
= \; & \mathbf{E}_{m,\sigma}\left[\sup_{h\in\mathcal{H},g\in\mathcal{G}}\left|\frac{2}{\ell}\sum_{i=1}^{\ell}\sigma_i w^T D_M g(x_i)\right|\right] && (4.10)\\
= \; & \mathbf{E}_{m,\sigma}\left[\sup_{h\in\mathcal{H},g\in\mathcal{G}}\left|\left\langle D_M w, \frac{2}{\ell}\sum_{i=1}^{\ell}\sigma_i g(x_i)\right\rangle\right|\right] \\
\leq \; & \mathbf{E}_M\left[\max_w \|D_M w\|\right]\mathbf{E}_\sigma\left[\sup_{g^j\in\mathcal{G}}\left\|\left[\frac{2}{\ell}\sum_{i=1}^{\ell}\sigma_i g^j(x_i)\right]_{j=1}^{n}\right\|\right] && (4.11)\\
\leq \; & B_h p\sqrt{nd}\left(\sqrt{n}\hat{R}_\ell(\mathcal{G})\right) = pn\sqrt{d}B_h\hat{R}_\ell(\mathcal{G})
\end{aligned}
$$

where D_M in Equation (4.10) is an diagonal matrix with diagonal elements equal to m and inner product properties lead to Equation (4.11). Thus, we have

$$\hat{R}_\ell(\mathcal{F}) \leq p\left(2\sqrt{k}dB_s n\sqrt{d}B_h\right)\hat{R}_\ell(\mathcal{G})$$

□

Remark 3. *Theorem 3 implies that p is an additional regularizer we have added to network when we convert a normal neural network to a network with DropConnect layers. Consider the following extreme cases:*

1. *$p = 0$: the network generalization bound equals to 0, which is true because classifier does not depends on input any more*
2. *$p = 1$: reduce to normal network*

Symbol	Description	Related Formula
y	Data Label, can either be integer label for bit vector(depends on context)	
x	Network input data	
$g(.)$	Feature extractor function with parameter W_g	
v	Feature extractor network output	$v = g(x, W_g)$
M	DropConnect connection information parameter (weight mask)	
$h(.)$	DropConnect transformation function with parameter W, M	
u	DropConnect output	$u = h(v; W, M)$
$a(.)$	DropConnect activation function	
r	DropConnect after activation	$r = a(u)$
$s(.)$	Dimension reduction layer function with parameter W_s	
o	Dimension reduction layer output (network output)	$o = s(r; W_s)$
θ	All parameter of network expect weight mask	$\theta = \{W_s, W, W_g\}$
$f(.)$	Overall classifier(network) output	$o = f(x; \theta, M)$
λ	Weight penalty	
$A(.)$	Data Loss Function	$A(o - y)$
$L(.)$	Over all objective function	$L(x, y) = \sum_i A(o_i - y_i) + 1/2\lambda\|W\|_2^2$
n	Dimension of feature extractor output	
d	Dimension of DropConnect layer output	
k	number of class	$dim(y) = k$

Table 4.8: Symbol Table

Chapter 5

Detection Model for PASCAL Challenge

5.1 Introduction

An object is represented by a mixture of hierarchical tree models where the nodes represent object parts. The nodes can move spatially to allow both local and global shape deformations [78][7]. The image features are histograms of words (HOWs) [44] and oriented gradients (HOGs) [9] which enable rich appearance representation of both structured (eg, cat face) and textured (eg,cat body) image regions. Learning the hierarchical model is a latent SVM problem which can be solved by the incremental concave-convex procedure (iCCCP) [74]. Object detection is performed by scanning sub-windows using dynamic programming. The detections are rescored by a context model which encodes the correlations of 20 object classes by using both object detection and image classification.

By extending the work from Zhu et al. [78][7] (Section 5.2) and a novel non-maximum suppression algorithm (Section 5.3), we achieve 1st in PASCAL VOC 2011 detection challenge.

5.2 Model Description

An object class consists of 4-6 templates from 2-3 different views each of which is represent by a 3-layer tree-structure model. The structure of model is given in Figure 5.1. The first layer has one root node which represent the object bounding box. The root node has 9 child nodes at second layer in a 3×3 grid layout. Furthermore, each node at the second layer has 4 child nodes at the third layer. The nodes in second and third layer can move spatially with respect to its parent node with linear penalty.

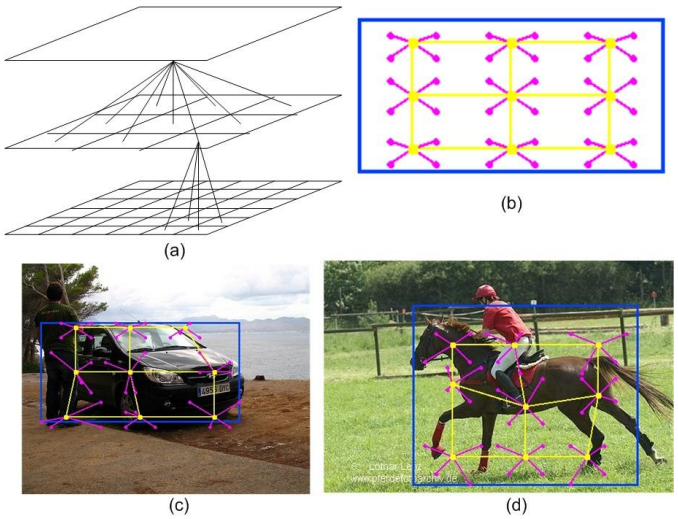


Figure 5.1: Reproduce from Zhu [78] (a) 3-layer tree model. The structure has tree layers withe node in grid layout (b) A reference template without part displacement (no deformation) Blue rectangle is the bounding box of the root node. Yellow dots indicate the center of 36 parts at bottom layer. (c,d) examples of part displacement.

Each node of tree contains two kind of image feature: HoW and HoG as show in Figure 5.2. Thus, there are three types of potential terms in overall model:

1. Spatial deformation terms (dx, dy) specifies how much child node deform from its default location with data terms: $\phi_{shape} = [dx^2, dy^2, dx, dy]$

2. Edge-like HoG feature ϕ_{HoG} with linear kernel.

3. Regional appearance HoW features ϕ_{HoW} defines by histograms of words with quasi-linear kernel. The words are K-means centroid of SIFT feature from image

patches.

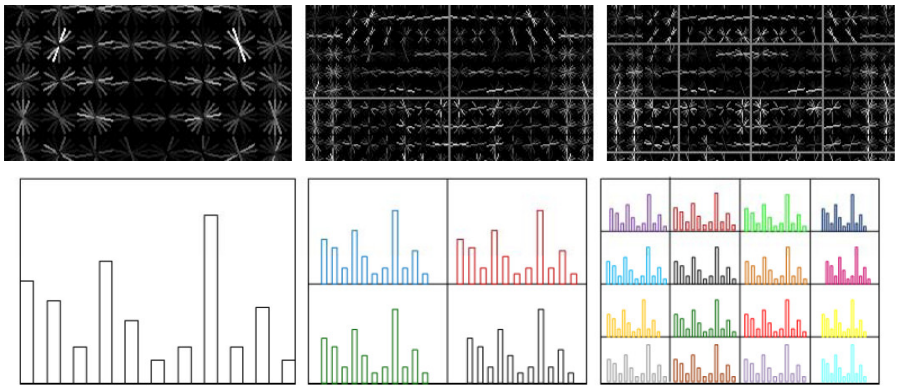


Figure 5.2: Upper row: Edge-like HoG feature Bottom row: Regional Histogram of Words(SIFT)

Each tree model is associated with a latent variable h which defines all parts node locations. Thus, the objection is detected by solving:

$$y^*, h^* = \arg\max w^T \phi(x, y, h)$$

where y^* is final detection score for current window and $\phi(x, y, h)$ counts all three kind of features ϕ_{shape}, ϕ_{HoG} and ϕ_{HoW} for each node with latent variable h.

The general idea of learning procedure is optimize the following function via CCCP algorithm with λ as regularization term.

$$\begin{aligned} & \min_w \lambda \frac{1}{2}\|w\|_2^2 + \sum_{i=1}^{N} \left[\max_{y,h} \left[w^T \phi(x_i, y, h) \right] + L(y_i, y, h) - \max_h \left[w^T \phi(x_i, y_i, h) \right] \right] \\ = & \min_w \left[\lambda \frac{1}{2}\|w\|_2^2 + \sum_{i=1}^{N} \max_{y,h} \left[w^T \phi(x_i, y, h) + L(y_i, y, h) \right] \right] - \left[\max_h \left[w^T \phi(x_i, y_i, h) \right] \right] \end{aligned}$$

where $L(y_i, y, h)$ is the loss function. For object detection problem $L(y_i, y, h) = 1$ if $y_i = y$ and $L(y_i, y, h) = 0$ if $y_i \neq y$. The model minimize when unconstrained y agrees with y_i for each term.

5.3 Non-Maximum Suppression

Non-Maximum Suppression (NMS) is an important post-processing task to removes near duplicate detection result and retain only the local maximal in a set of detection result. Removing such duplications is important because it introduce false alarm in final performance evaluation. We will begin with standard NMS algorithm [9][15] and gradually develop better versions of it.

Given a set of detection result assignment $A = \{(b_i, r_i)\}_{i=1,2,\ldots,n}$. Here r_i is the response of i^th bounding box b_i. We want to produce $A' = \{(b'_i, r'_i)\}_{i=1,2,\ldots,m} \subset A$ to remove duplication detection results and improve final mAP performance. Usually, $m \ll n$.

The first version of NMS given in Algorithm 4 which is used in many of popular systems [78][15].

Algorithm 4 Non-Maximum Suppression (NMS)

```
1: sort (r, b) ∈ A in descending order with r
2: A' = {}
3: for b_i ∈ A do
4:    match = false
5:    for b_j ∈ A' do
6:       if overlap(b_i, b_j) ≥ 0.5 then
7:          match = true
8:          break
9:       end if
10:   end for
11:   if not match then
12:      A' ← A' ∩ {(b_i, r_i)}
13:   end if
14: end for
```

The $overlap(b, b')$ in Line 6 of Algorithm 4 is defined as

$$overlap(b, b') = \frac{Area(b' \cap b)}{Area(b' \cup b)} \tag{5.1}$$

The *overlap* function in Eq. 5.1 is the same as PASCAL evaluation: A predict bounding box is correct iff it's overlap $\geq 50\%$ (define in Eq. 5.1) with ground truth bounding box.

Duplicate matchings are consider as wrong prediction.

We found an extension of previous *overlap* function that works better in practise is defined as:

$$overlap(b, b') = \max\left(\frac{Area(b')}{Area(b' \cup b)}, \frac{Area(b)}{Area(b' \cup b)}\right) \tag{5.2}$$

It introduces about 2mAP performance boost compare to standard form in Eq. 5.1. All overlap function in NMS will refer to Eq. 5.2 except explicitly stated.

Our final version of NMS shown in Algorithm 5 which extends Algorithm 4 to an iterative process by using old version as initialization step. It repeats the process of compute centroid and updating centroid process:

1. compute maximal index a_j of each bounding box b_j with respect to each center b'_i by using *overlap* function (Line 3-5).

2. update b'_i based on all bounding box with maximal index $a_j = i$. The merge of bounding box simply takes the maximal score and a larger bounding box includes all smaller ones (Line 6-9).

Iterative NMS consistently improves final detection further by $1 - 2$ mAP.

Algorithm 5 Iterative Non-Maximum Suppression

```
1: init center A' = {(b'_i, r'_i)} from Algorithm 4
2: for step = 1, 2, ..., 20 do
3:    for (r_j, b_j) ∈ A do
4:       a_j ← arg_i max overlap(b_j, b'_i)
5:    end for
6:    for (b'_i, r'_i) ∈ A' do
7:       b'_i ← merge(b_j ∈ A ∧ a_j = i)
8:       r'_i ← max(r_j ∈ A ∧ a_j = i)
9:    end for
10: end for
```

5.4 Experiment Results and Discussion

We have develop state-of-the-art detection system, mainly by improving 1) image features and its corresponding learning algorithm 2) non-maximum suppression. A summary of system performance is available at `http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2011/results/index.html` (Competition "comp3" NYUUCLA_HIERARCHY result).

The final model for detection task is divided into following stages:

1. Extract features from image patch. The most useful feature: HoG, SIFT, LBP
2. Lean hierarchical part-based model
3. Apply NMS algorithm to detection result set
4. Re-score each detection result by context, including classification and detection results from the other class.

The above 4 steps performs separately from each other and was tunned independently to achieve good performance. However, these steps are highly correlated. Furthermore, recent advances in deep learning shown that image features learnt by ConvNet generally better than hand crafted features by Krizhevsky et al. [35]. Thus, it leads us to an model integrates all of 4 steps with trainable ConvNet features. In Chapter 6, we presents such integrated model and its corresponding end-to-end training algorithm.

Chapter 6

End-to-End Integration of a ConvNet, Deformable Parts Model and Non-Maximum Suppression

6.1 Introduction

Object detection has been addressed using a variety of approaches, including sliding-window Deformable Parts Models [15, 78, 16], region proposal with classification [19, 68], and location regression with deep learning [60, 65]. Each of these methods have their own advantages, yet are by no means mutually exclusive. In particular, structured parts models capture the composition of individual objects from component parts, yet often use rudimentary features like HoG [9] that throw away much of the discriminative information in the image. By contrast, deep learning approaches [34, 76, 60], based on Convolutional Networks [39], extract strong image features, but do not explicitly model object composition. Instead, they rely on pooling and large fully connected layers to

combine information from spatially disparate regions; these operations can throw away useful fine-grained spatial relationships important for detection.

The basic building blocks of our model architecture come from the DPMs of Felzenszwalb et al. [15] and Zhu et al. [78][7], and the ConvNet of Krizhevsky et al. [34]. We make crucial modifications in their integration that enables the resulting model to achieve competitive object detection performance. In particular, we develop ways to transfer the ConvNet from classification to the detection environment, as well as changes to the learning procedure to enable joint training of all parts.

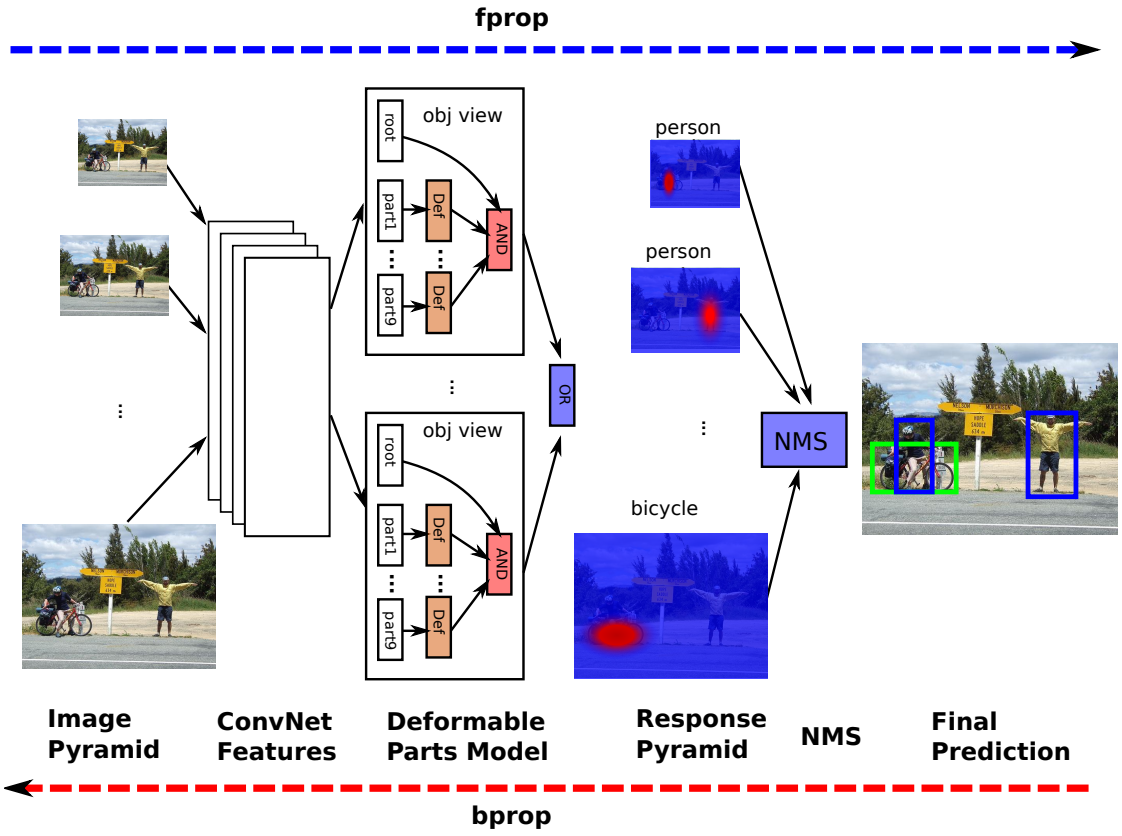


Figure 6.1: An overview of our system: (i) a convolutional network extracts features from an image pyramid; (ii) a set of deformable parts models (each capturing a different view) are applied to the convolutional feature maps; (iii) non-maximum suppression is applied to the resulting response maps, yielding bounding box predictions. Training is performed using a new loss function that enables back-propagation through all stages.

In this paper, we propose a framework (shown in Figure 6.1) that combines these two approaches, fusing together structured learning and deep learning to obtain the advan-

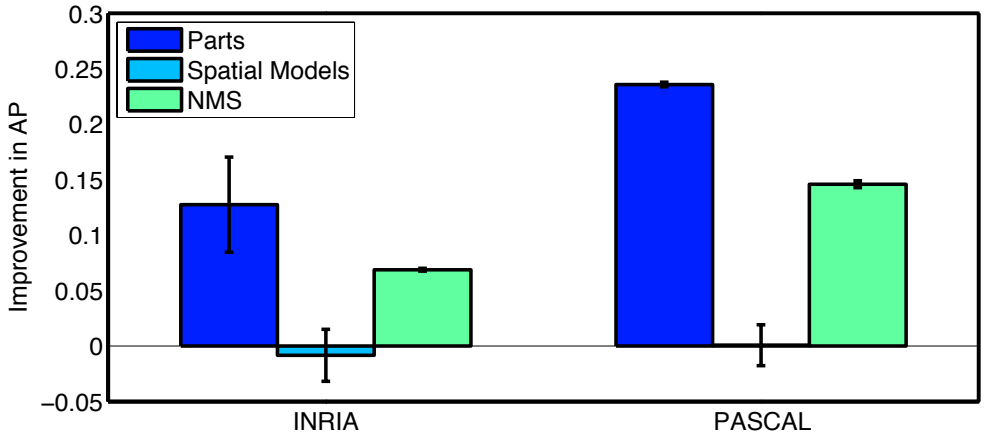


Figure 6.2: Reproduced from Parikh and Zitnick [51]: an ablation study of the stages in a DPM model [15] . Their figure shows how significant performance improvements could be obtained by replacing the parts detection and non-maximum suppresssion stages with human subjects. This suggests that these stages limit performance within the model. Our work focuses on improving each of these, replacing the part detectors with a Convnet and integrating NMS into the model.

tages of each. We use a DPM for detection, but replace the HoG features with features learned by a convolutional network. This allows the use of complex image features, but still preserves the spatial relationships between object parts during inference.

An often overlooked aspect of many detection systems is the non-maximum suppression stage, used to winnow multiple high scoring bounding boxes around an object instance down to a single detection. Typically, this is a post-processing operation applied to the set of bounding boxes produced by the object detector. As such, it is not part of the loss function used to train the model and any parameters must be tuned by hand. However, as demonstrated by Parikh and Zitnick [51], NMS can be a major performance bottleneck (see Figure 6.2). We introduce a new type of image-level loss function for training that takes into consideration of all bounding boxes within an image. This differs with the losses used in existing frameworks that consider single cropped object instances. Our new loss function enables the NMS operation trained as part of the model, jointly with the Convnet and DPM components.

6.2 Model Architecture

The architecture of our model is shown in Figure 6.3. For a given input image x, we first construct an image pyramid (a) with five intervals over one octave[1] We apply the ConvNet (b) at each scale x_s to generate feature maps $\phi_A(x_s)$. These are then passed to the DPM (c) for each class; as we describe in Section 6.2.2, the DPM may also be formulated as a series of neural network layers. At training time, the loss is computed using the final detection output obtained after NMS (d), and this is then back-propagated end-to-end through the entire system, including NMS, DPM and ConvNet.

6.2.1 Convolutional Network

We generate appearance features $\phi_A(x)$ using the first five layers of a Convolutional Network pre-trained for the ImageNet Classification task. We first train an eight layer classification model, which is composed of five convolutional feature extraction layers, plus three fully-connected classification layers[2]. After this network has been trained, we throw away the three fully-connected layers, replacing them instead with the DPM. The five convolutional layers are then used to extract appearance features.

Note that for detection, we apply the convolutional layers to images of arbitrary size (as opposed to ConvNet training, which uses fixed-size inputs). Each layer of the network is applied in a bottom-up fashion over the entire spatial extent of the image, so that the total computation performed is still proportional to the image size. This stands in contrast to [19], who apply the ConvNet with a fixed input size to different image regions, and is more similar to [60].

Applying the ImageNet classification model to PASCAL detection has two scale-related

[1] We use as many octaves as required to make the smallest dimension 48 pixels in size.

[2] The fully connected layers have 4096 - 4096 - 1000 output units each, with dropout applied to the two hidden layers. We use the basic model from [76], which trains 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.

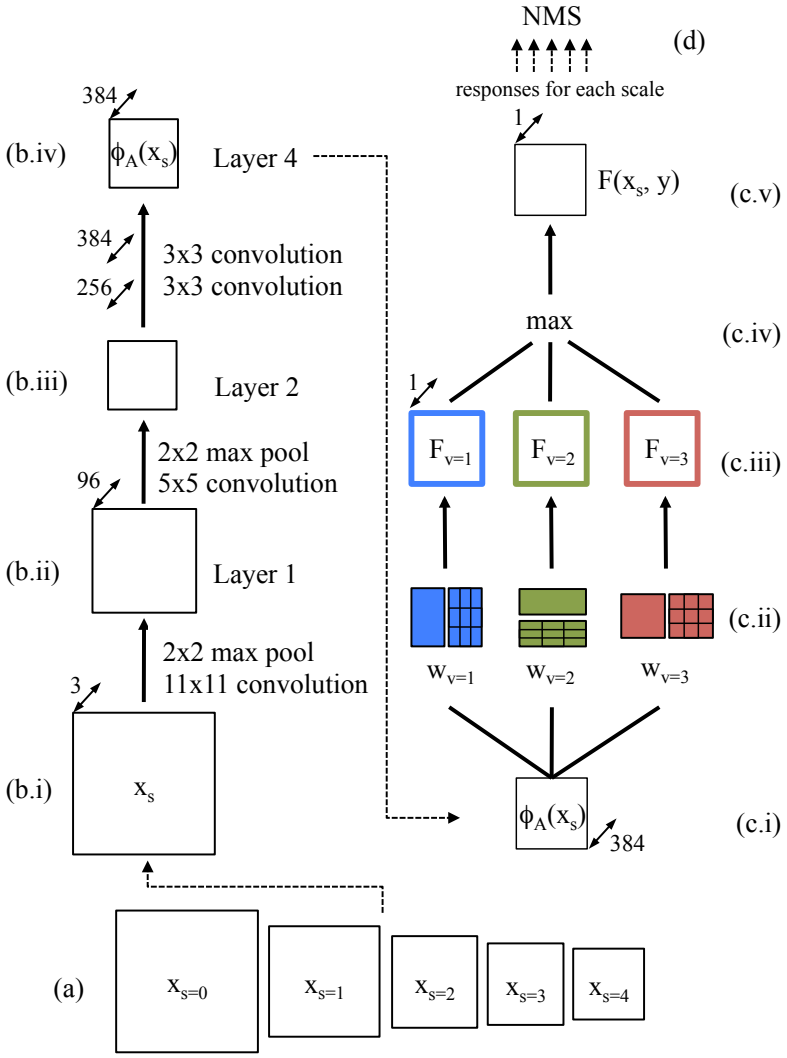


Figure 6.3: model architecture, with Convolutional Network (left), Deformable Parts Model (right) and non-maximum suppression (top) components. An input x is first repeatedly downsampled to create an image pyramid (a). We run the convolutional network on each scale, by performing four layers of convolution and max-pooling operations (b.ii - b.iv). This produces a set of appearance features $\phi_A(x_s)$ at each scale, which are used as input to a DPM (c.i). Each object class model has three views of object templates (c.ii), each of which is composed of a root filter and nine parts filters. These produce a response map F_v for each view (c.iii), which are then combined using a pixel-wise max (c.iv) to form a final activation map for the object class, $F(x_s, y)$. We then perform NMS (d) across responses for all scales. To generate bounding boxes, we trace the activation locations back to their corresponding boxes in the input.

problems that must be addressed. The first is that there is a total of 16x subsampling between the input and the fifth layer; that is, each pixel in ϕ_A corresponds to 16 pixels

of input — this is insufficient for detection, as it effectively constrains detected bounding boxes to a lie on a 16-pixel grid. The second is that the ImageNet classifier was trained on objects that are fairly large, taking up much of the 224x224 image area. By contrast, many target objects in PASCAL are significantly smaller.

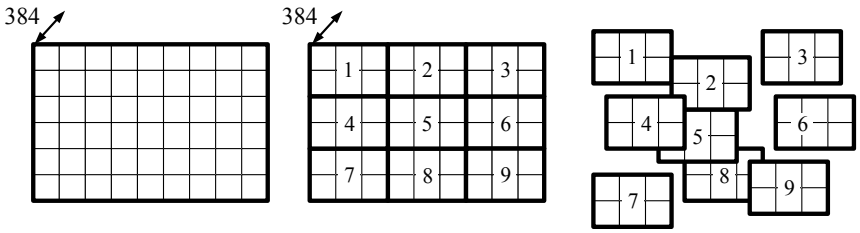
To address these, we simply apply the first convolution layer with a stride of 1 instead of 4 when combining with the DPM (however, we also perform 2x2 pooling after the top ConvNet layer due to speed issues in training, making the net resolution increase only a factor of 2). This addresses both scale issues simultaneously. The feature resolution is automatically increased by elimination of the stride. Moreover, the scale of objects presented to the network at layers 2 and above is increased by a factor of 4, better aligning the PASCAL objects to the ImageNet expected size This is due to the fact that when the second layer is applied to the output of the stride-1 maps, their field of view is 4x smaller compared to stride-4, effectively increasing the size of input objects.

Note that changing the stride of the first layer is effectively the same as upsampling the input image, but preserves resolution in the convolutional filters (if the filters were downsampled, these would be equivalent operations; however we found this to work well without changing the filter size, as they are already just 11x11).

6.2.2 Deformable Parts Model

Part Responses

The first step in the DPM formulation is to convolve the appearance features with the root and parts filters, producing appearance responses. Each object view has both a root filter and nine parts filters; the parts are arranged on a 3x3 grid relative to the root, as illustrated in Figure 6.4. (This is similar to [78], who find this works as well as the more complex placements used by [15]). Note that the number of root and parts filters is the same for all classes, but the size of each root and part may vary between classes



ht

(a) $w^{\text{root}}_{A,v}$ (b) $w^{\text{part}}_{A,v}$ (c) $w^{\text{part}}_{A,v}, w^{\text{part}}_{D,v}$

Figure 6.4: Root and parts filter setup for our DPM. (a) Each view v has a root filter with a different pre-defined aspect ratio. (b) Part filters are aligned on a 3x3 grid relative to the root. (c) Parts may deform relative to the root position at a cost, parameterized by w^{part}_{D}.

and views.

Given appearance filters $w^{\text{root}}_{A,y,v}$ for each class y and view v, and filters $w^{\text{part}}_{A,y,v,p}$ for each part p, the appearance scores are:

$$\begin{aligned} F^{\text{root}}_{v}(x_s, y) &= w^{\text{root}}_{A,y,v} * \phi_A(x_s) \\ F^{\text{part}}_{v,p}(x_s, y) &= w^{\text{part}}_{A,y,v,p} * \phi_A(x_s) \end{aligned}$$

Part responses are then fed to the deformation layer.

Deformation Layer

The deformation layer finds the optimal part locations, accounting for both apperance and a deformation cost that models the spatial relation of the part to the root. Given appearance scores $F^{\text{part}}_{v,p}$, part location p relative to the root, and deformation parameters $w_{D,v,p}$ for each part, the deformed part responses are the following (input variables (x_s, y) omitted):

$$F^{\text{def}}_{v,p} = \max_{\delta_i, \delta_j} \; F^{\text{part}}_{v,p}[p_i + \delta_i, p_j + \delta_j] + w^{\text{part}}_{D,v,p} \phi_D(\delta_i, \delta_j) \tag{6.1}$$

where $F^{\text{part}}_{v,p}[p_i + \delta_i, p_j + \delta_j]$ is the part response map $F^{\text{part}}_{v,p}(x_s, y)$ shifted by spatial offset $(p_i + \delta_i, p_j + \delta_j)$, and $\phi_D(\delta_i, \delta_j) = [|\delta_i|, |\delta_j|, \delta_i^2, \delta_j^2]^T$ is the shape deformation feature. $w^{\text{part}}_{D,y,v} \geq 0$ are the deformation weights.

Note the maximum in Eq. 6.1 is taken independently at each output spatial location: i.e. for each output location, we find the max over possible deformations (δ_i, δ_j). In practice, searching globally is unnecessary, and we constrain to search over a window $[-s, s] \times [-s, s]$ where s is the spatial size of the part (in feature space). During training, we save the optimal $(\hat{\delta}_i, \hat{\delta}_j)$ at each output location found during forward-propagation to use during back-propagation.

The deformation layer extends standard max-pooling over (δ_i, δ_j) with (i) a shift offset (p_i, p_j) accounting for the part location, and (ii) deformation cost $w_D^T \phi_D(\delta_i, \delta_j)$. Setting both of these to zero would result in standard max-pooling.

AND/OR Layer

Combining the scores of root, parts and object views is done using an AND-like accumulation over parts to form a score F_v for each view v, followed by an OR-like maximum over views to form the final object score F:

$$\begin{aligned} F_v(x_s, y) &= F_v^{\text{root}}(x_s, y) + \sum_{p \in \text{parts}} F_{v,p}^{\text{def}}(x_s, y) \\ F(x_s, y) &= \max_{v \in \text{views}} F_v(x_s, y) \end{aligned}$$

$F(x_s, y)$ is then the final score map for class y at scale s, given the image x as shown in Figure 6.5.

6.2.3 Bounding Box Prediction

After obtaining activation maps for each class at each scale of input, we trace the activation locations back to their corresponding bounding boxes in input space. Detection locations in $F(x_s, y)$ are first projected back to boxes in $\phi_A(x_s)$, using the root and parts filter sizes and inferred parts offsets. These are then projected back into input space through the convolutional network. As shown in Figure 6.6(right), each pixel in $\phi_A(x_s)$

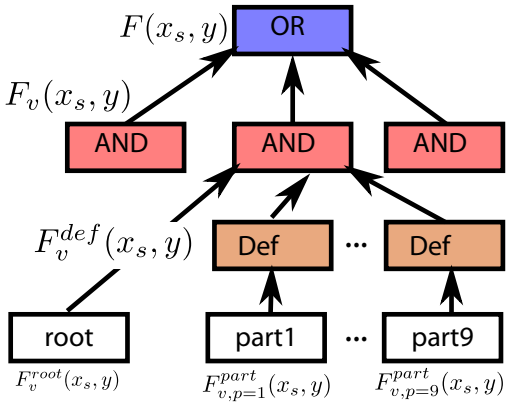


Figure 6.5: Overview of the top part of our network architecture: (i) the root and part layers are convolution layers with different sizes of filter but same input size; (ii) OR/AND/Def layer preserve the size of the input to the output; (iii) each AND layer represents an object view which contains a root and 9 parts.

has a field of view of 35 pixels in the input, and moving by 1 pixel in ϕ_A moves by 8 pixels in the input (due to the 8x subsampling of the convolutional model). Each bounding box is obtained by choosing the input region that lies between the field of view centers of the box's boundary. This means that 17 pixels on all sides of the input field of view are treated as context, and the bounding box is aligned to the interior region.

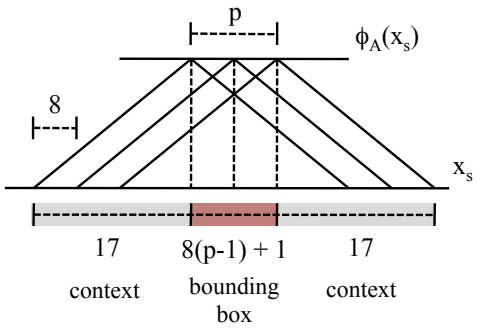


Figure 6.6: Aligning a bounding box from DPM prediction through the convolutional network (see Section 6.2.3).

6.2.4 Non-Maximal Suppression (NMS)

The procedure above generates a list of label assignments $A_0 = \{(b_i, y_i, r_i)_{i=1...|B|}\}$ for the image, where b_i is a bounding box, and y_i and r_i are its associated class label and

network response score, *i.e.*r_i is equal to $F(x, y_i)$ at the output location corresponding to box b_i. B is the set of all possible bounding boxes in the search.

The final detection result is a subset of this list, obtained by applying a modified version of non-maximum suppression derived from [78]. If we label location b_i as object type y_i, some neighbors of b_i might also have received a high scores, where the neighbors of b are defined as:

$$neigh(b) = \{b' | overlap(b, b') \geq \theta\}$$

However, $neigh(b_i) \backslash b_i$ should not be labeled as y_i to avoid duplicate detections. Applying this, we get a subset of $A = \{(b_i, y_i, r_i)_{i=1...n}\}$ as the final detection result; usually $n \ll |B|$.

When calculating $overlap(b, b')$, we use a symmetric form when the bounding boxes are for different classes, but an asymmetric form when the boxes are both of the same class. For different-class boxes:

$$overlap(b, b') = \frac{Area(b \cap b')}{Area(b \cup b')}$$

and threshold $\theta = 0.75$.

For same-class boxes, *eg*boxes of different views or locations,

$$overlap(b, b') = \max\left(\frac{Area(b \cap b')}{Area(b)}, \frac{Area(b \cap b')}{Area(b')}\right)$$

with theshold $\theta = 0.5$.

6.3 Final Prediction Loss

6.3.1 Motivation

Our second main contribution is the use of a final-prediction loss that takes into account the NMS step used in inference. In contrast to bootstrapping with a hard negative

Figure 6.7: Three possible bounding boxes: blue, green and red (blue closest to the ground truth). However, green and red should not be considered negative instances (since they may be positive in other images where the person is occluded). Thus, we want
$r(blue) > r(red)$
$r(blue) > r(green)$

pool, such as in [78] [15], we consider each image individually when determining positive and negative examples, accounting for NMS and the views present in the image itself. Consider the example in Figure 6.7: A person detector may fire on three object views: red, green, and blue. The blue (largest in this example) is closest to the ground truth, while green and red are incorrect predictions. However, we cannot simply add the green or red boxes to a set negative examples, since they are indeed present in other images as occluded people. This leads to a situation where the red view has a higher inference score than blue or green, i.e. $r(red) > r(blue)$ and $r(red) > r(green)$, because red is never labeled as negative in the bootstrapping process. After NMS, blue response will be suppressed by red, causing a NMS error. Such an error can only be avoided when we have a global view on each image: if $r(blue) > r(red)$, then we would have a correct final prediction.

6.3.2 Loss Function

Recall that the NMS stage produces a set of assignments predicted by the model $A = \{(b_i, y_i, r_i)_{i=1...n}\}$ from the set B of all possible assignments. We compose the loss using two terms, $C(A)$ and $C(A')$. The first, $C(A)$, measures the cost incurred by the assignment currently predicted by the model, while $C(A')$ measures the cost incurred by an

assignment close to the ground truth. The current prediction cost $C(A)$ is:

$$C(A) = \underbrace{\sum_{(b_i,y_i,r_i)\in A} H(r_i, y_i)}_{C^P(A)} + \underbrace{\sum_{(b_j,y_j,r_j)\in S(A)} H(r_j, 0)}_{C^N(A)} \tag{6.2}$$

where $H(r, y) = I(y > 0)\max(0, 1 - r)^2 + I(y = 0)\max(0, r + 1)$ i.e. a squared hinge error. [3] $S(A)$ is the set of all bounding boxes predicted to be in the background ($y = 0$): $S(A) = B \setminus neigh(A)$ with

$$neigh(A) = \bigcup_{(b_i,y_i,r_i)\in A} neigh(b_i)$$

. $C_P(A)$ and $C_N(A)$ are the set of positive predicted labels and the set of background labels, respectively.

The second term in the loss, $C(A')$, measures the cost incurred by the ground truth bounding boxes under the model. Let the ground truth bounding box set be $A^{gt} = \{(b_i^{gt}, y_i^{gt})_{i=1...m}\}$. We construct a constrained inference assignment A' close to A^{gt} by choosing for each b_i^{gt} the box $(b'_i, y'_i, r'_i) = \text{argmax}_{(b,y,r)}\, r$, where the argmax is taken over all $overlap(b, b_i^{gt}) \geq \theta'$; that is, the box with highest response out of those with sufficient overlap with the ground truth. ($\theta' = 0.7$ in our experiments.) Similarly to before, the cost $C(A')$ is:

$$C(A') = \sum_{(b'_i,y'_i,r'_i)\in A'} H(r'_i, y'_i) + \sum_{(b'_j,y'_j,r'_j)\in S(A')} H(r'_j, 0) \tag{6.3}$$

Thus we measure two costs: that of the current model prediction, and that of an assignment close to the ground truth. The final discriminative training loss is difference between these two:

$$L(A, A') = C(A') - C(A) \geq 0 \tag{6.4}$$

[3] where I is an indicator function that equals 1 iff the condition holds

Note this loss is always greater than 0 because the constrained assignment always has cost at least as large as the unconstrained one, and $L(A, A') = 0$ when $A = A'$, i.e. when we produce detection results which are consistent with the ground truth A^{gt}.

Combining Equations Eq. 6.2 and Eq. 6.3 leads to

$$
\begin{aligned}
L(A, A') &= L^P(A, A') + L^N(A, A') \qquad (6.5)\\
L^P(A, A') &= \sum_{(b',y',r') \in A'} H(r', y') - \sum_{(b,y,r) \in A} H(r, y)\\
L^N(A, A') &= \sum_{(b',y',r') \in S(A')} H(r', 0) - \sum_{(b,y,r) \in S(A)} H(r, 0)\\
&= \sum_{(b,y,r) \in N \setminus N'} H(r, 0) - \sum_{(b',y',r') \in N' \setminus N} H(r', 0)
\end{aligned}
$$

where $N = neigh(A)$ and $N' = neigh(A')$. The last line comes from the fact that most of the boxes included in $S(A)$ and $S(A')$ are shared, and cancel out (see Figure 6.8); thus we can compute the loss looking only at these neighborhood sets.

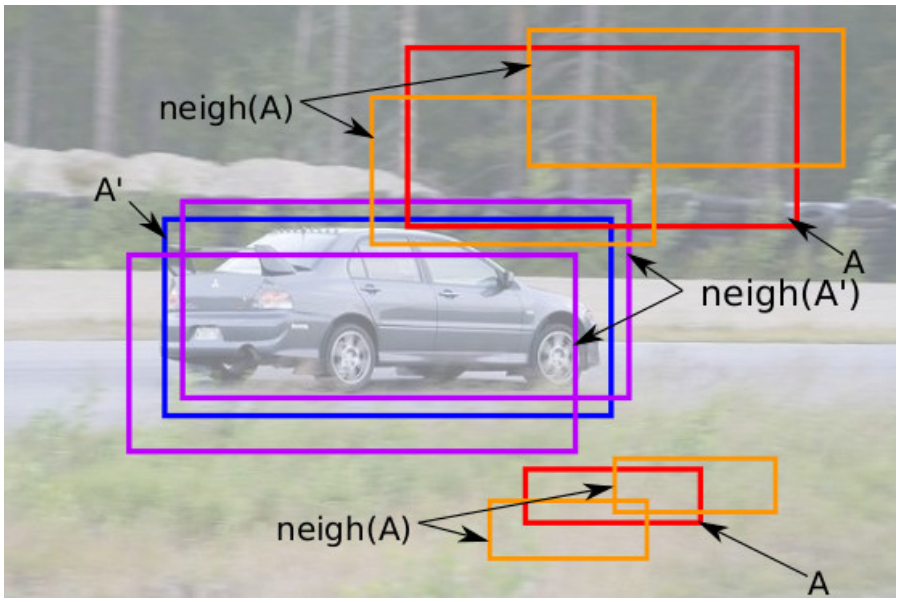


Figure 6.8: Illustration of ground-truth-constrained assignment A' and unconstrained assignments A from the model, along with associated neighborhoods. Note neighborhoods are actually dense, and we show only a few boxes for illustration.

6.3.3 Interpretation and Effect on NMS Ordering

As mentioned earlier, a key benefit to training on the final predictions as we describe is that our loss accounts for the NMS inference step. In our example in Figure 6.7, if the response $r(red) > r(blue)$, then $red \in A$ and $blue \in A'$. Thus $L^P(A, A')$ will decrease $r(red)$ and increase $r(blue)$. This ensures the responses r are in an appropriate order when NMS is applied. Once $r(blue) > r(red)$, the mistake will be fixed.

The term L^N in the loss is akin to an online version of hard negative mining, ensuring that the background is not detected as a positive example.

6.3.4 Soft Positive Assignments

When training jointly with the ConvNet, it is insufficient to measure the cost using only single positive instances, as the network can easily overfit to the individual examples. We address this using soft positive assignments in place of hard assignments; that is, we replace the definition of C^P in Eq. 6.2 used above with one using a weighted average of neighbors for each box in the assignment list:

$$C^P(A) = \sum_{b_i \in A} \frac{\sum_{b_j \in neigh(b_i)} \alpha_{ij} H(r_j, y_j)}{\sum_j \alpha_{ij}}$$

where $\alpha_{ij} = 2(Area(b_i \cap b_j)/Area(b_i)) - 1$, and similarly for $C^P(A')$.

Note a similar strategy has been tried in the case of HoG features before, but was not found to be beneficial [15]. By contrast, we found this to be important for integrating the ConvNet. We believe this is because the ConvNet has many more parameters than can easily overfit, whereas HoG is more constrained.

6.4 Training

Our model is trained in online fashion with SGD, with each image being forward propagated through the model and then the resulting error backpropagated to update the parameters. During the fprop, the position of the parts in the DPM are computed and then used for the subsequent bprop. Training in standard DPM models [15] differs in two respects: (i) a fixed negative set is mined periodically (we have no such set, instead processing each image in turn) and (ii) part positions on this negative set are fixed for many subsequent parameter updates.

We first pretrain the DPM root and parts filters without any deformation, using a fixed set of 20K random negative examples for each class. Note that during this stage, the ConvNet weights are fixed to their initialization from ImageNet. Following this, we perform end-to-end joint training of the entire system, including ConvNet, DPM and NMS (via the final prediction loss). During joint training, we use inferred part locations in the deformation layer.

The joint training phase is outlined in Algorithm 6. For each training image sample, we build an image pyramid, and fprop each scale of the pyramid through the ConvNet and DPM to generate the assignment list A_0. Note A_0 is represented using the output response maps. We then apply NMS to get the final assignments A, as well as construct the ground-truth constrained assignments A'. Using the final prediction loss $L(A, A')$ from Eq. 6.5, we find the gradient and backpropagate through the network to update the model weights. We repeat this for 15 epochs through the training set with a learning rate $\eta = 10^{-3}$, then another 15 using $\eta = 10^{-4}$.

At test time, we simply forward-propagate the input pyramid through the network (ConvNet and DPM) and apply NMS.

Algorithm 6 Training algorithm for each image

Input: Image X with ground truth assignment A^{gt}
Build image pyramid $X \rightarrow X_1, X_2, \ldots, X_s$
$A_0 = \{\}$
for $X_i \in X_1, X_2, \ldots X_s$ **do**
    $A_0 = A_0 \cup$ assignments from responses $F(X_i; w)$
end for
find $A = NMS(A_0)$
find A' using A_0 and A^{gt}
for $X_i \in X_1, X_2, \ldots X_s$ **do**
    find gradient at scale i: $g_i = \frac{\partial L(A,A')}{\partial F(X_i;w)} \frac{\partial F(X_i;w)}{\partial w}$
end for
$w \leftarrow w + \eta \sum_i g_i$

6.5 Experiments

We apply our model to the PASCAL VOC 2007 and VOC 2011/2012 object detection tasks [12]. Table Table 6.1 shows how each component in our system improves performance on the PASCAL 2007 dataset. Our baseline implementation of HoG DPM with bootstrap training achieves 30.7 mAP. Switching HoG for a fixed pretrained ConvNet results in a large 32% relative performance gain to 40.8 mAP, corroborating the finding of [20] that such features greatly improve performance. On top of this, training using our online post-NMS procedure improves substantially improves performance to 43.3 mAP, and jointly training all components (ConvNet + DPM + NMS) further improves to 46.5 mAP. In addition, we can train different models to produce detections for each class, or

	Bootstrap	NMS loss	NMS loss+FT
HoG-root	22.8	23.9	N/A
HoG-root+part	30.7	33.2	N/A
conv-root	38.7	40.3	43.1
conv-root+part	40.8	43.3	46.5

Table 6.1: A performance breakdown of our approach. Columns show different training methods and loss functions. Rows show different feature extractors and DPM with/without parts. Note: (i) conv features give a significant boost; (ii) our new NMS loss consistently improves performance, irrespective of features/model used and (iii) fine-tuning (FT) of the entire model gives further gains.

train all classes at once using a single model with shared ConvNet feature extractor (but different DPM components). Training all classes together further boosts performance to

46.9% mAP. Note that this allows the post-NMS loss to account for objects of different classes as well as locations and views within classes, and also makes inference faster due to the shared features. We call this model "conv-dpm+FT-all", and the separate-class set of models "conv-dpm+FT".

Comparisons with other systems are shown in Tables Table 6.2 (VOC 2007) and Table 6.3 (VOC 2011/2012). For VOC 2007 (Table 6.2), our results are very competitive, beating all other methods except the latest version of R-CNN trained on fc_7 ("R-CNN(v4)FT fc_7"). Notably, we outperform the DP-DPM method (45.2% vs. our 46.9%), due to our integrated joint training and online NMS loss. In addition, our final model achieves comparible performance to R-CNN [19] with a similar feature extractor using $pool_5$ features (46.9% vs. 47.3%). Recent version of R-CNN achieve a better performance 54.2% using a more complex network which includes fully connected layers (fc_7); extending our model to use deeper networks may also provide similar gains from better feature representations.

Table 6.3 shows our system performance on VOC2011. Here our system outperforms comparison methods, and in particular DP-DPM, achieving 43.7% mAP versus 29.6% for HoG-DPM [15] and 41.6% for DP-DPM [20].

Finally, we provide examples of detections from our model in Figures Figure 6.9, Figure 6.10 and Figure 6.11. Detection results are either show in green or red with ground truth bounding box in blue. Figure Figure 6.11 illustrates training with our new loss function helps model fix problem for both inter-class and intra-class NMS. Our loss allows the larger view of the train to be selected in (a), rather than the more limited view that appears in more images. However, the gains are not limited to selecting larger views: In (c), we see a cat correctly selected at a smaller scale. Finally, there are also examples of inter-class correction in (f), *eg* "train" being selected over "bus".

Figure Figure 6.9 shows the effect of using a DPM with parts over just the root-only model. Figures Figure 6.10 shows correct and incorrect detection examples.

VOC2007	aero	bike	bird	boat	botl	bus	car	cat	chair	cow	table	dog	horse	mbike	pers	plant	sheep	sofa	train	tv	mAP
DetectorNet [65]	29.2	35.2	19.4	16.7	3.7	53.2	50.2	27.2	10.2	34.8	30.2	28.2	46.6	41.7	26.2	10.3	32.8	26.8	39.8	47.0	30.5
HoG-dpm(v5) [15]	33.2	60.3	10.2	16.1	27.3	54.3	58.2	23.0	20.0	24.1	26.7	12.7	58.1	48.2	43.2	12.0	21.1	36.1	46.0	43.5	33.7
HSC-dpm [53]	32.2	58.3	11.5	16.3	30.6	49.9	54.8	23.5	21.5	27.7	34.0	13.7	58.1	51.6	39.9	12.4	23.5	34.4	47.4	45.2	34.3
Regionlets [71]	54.2	52.0	20.3	24.0	20.1	55.5	68.7	42.6	19.2	44.2	49.1	26.6	57.0	54.5	43.4	16.4	36.6	37.7	59.4	52.3	41.7
DP-DPM [20]	44.6	65.3	32.7	24.7	35.1	54.3	56.5	40.4	26.3	49.4	43.2	41.0	61.0	55.7	**53.7**	25.5	47.0	39.8	47.9	**59.2**	45.2
R-CNN [19]fc_7	56.1	58.8	34.4	29.6	22.6	50.4	58.0	52.5	18.3	40.1	41.3	**46.8**	49.5	53.5	39.7	23.0	46.4	36.4	50.8	59.0	43.4
R-CNN(v1)FT $pool_5$	55.6	57.5	31.5	23.1	23.2	46.3	59.0	49.2	16.5	43.1	37.8	39.7	51.5	55.4	40.4	23.9	46.3	37.9	49.7	54.1	42.1
R-CNN(v4)FT $pool_5$	58.2	63.3	**37.9**	27.6	26.1	54.1	**66.9**	**51.4**	**26.7**	**55.5**	43.4	43.1	57.7	59.0	45.8	**28.1**	**50.8**	40.6	53.1	56.4	**47.3**
R-CNN(v1)FT fc_7	60.3	62.5	41.4	37.9	29.0	52.6	61.6	56.3	24.9	52.3	41.9	48.1	54.3	57.0	45.0	26.9	51.8	38.1	56.6	62.2	48.0
R-CNN(v4)FT fc_7	**64.2**	**69.7**	**50.0**	**41.9**	32.0	**62.6**	**71.0**	**60.7**	**32.7**	**58.5**	46.5	**56.1**	60.6	**66.8**	**54.2**	**31.5**	**52.8**	**48.9**	**57.9**	**64.7**	**54.2**
HoG	29.3	55.5	9.3	13.3	25.2	43.1	53	20.4	18.5	25.1	23.3	10.3	55.4	44.2	40.8	10.5	19.8	34.3	43.3	39.5	30.7
HoG+	32.8	58.5	10.3	16.0	27.1	46.1	56.9	21.9	20.6	27.2	26.4	13.0	57.8	47.5	44.2	11.0	22.7	36.5	45.8	42.1	33.2
conv-root	38.1	60.9	21.9	17.8	29.3	51.4	58.5	26.7	16.5	31.1	33.2	24.2	65.0	58.0	44.4	21.7	35.4	36.8	49.5	54.1	38.7
conv-dpm	45.3	64.5	21.1	21.0	34.2	54.4	59.0	32.6	20.0	31.0	34.5	25.3	63.8	60.1	45.0	23.2	36.0	38.4	51.5	56.2	40.8
conv-dpm+	48.9	67.3	25.3	25.1	35.7	58.3	60.1	35.3	22.7	36.4	37.1	26.9	64.9	62.0	47.0	24.1	37.5	40.2	54.1	57.0	43.3
conv-dpm+ FT	**50.9**	68.3	31.9	28.2	38.1	61.0	61.3	39.8	25.4	46.5	47.3	29.6	67.5	**63.4**	46.1	25.2	39.1	45.4	57.0	57.9	46.5
conv-dpm+ FT-all	49.3	**69.5**	31.9	**28.7**	**40.4**	**61.5**	61.5	41.5	25.5	44.5	**47.8**	32.0	**67.5**	61.8	46.7	25.9	40.5	**46.0**	**57.1**	58.2	**46.9**

Table 6.2: Mean AP on PASCAL VOC 2007

VOC2011/2012	aero	bike	bird	boat	botl	bus	car	cat	chair	cow	table	dog	horse	mbike	pers	plant	sheep	sofa	train	tv	mAP
HoG-DPM [15]	45.6	49.0	11.0	11.6	27.2	50.5	43.1	23.6	17.2	23.2	10.7	20.5	42.5	44.5	41.3	8.7	29.0	18.7	40.0	34.5	29.6
conv-root	56.0	45.4	20.6	12.7	29.5	49.2	38.6	38.1	16.4	28.2	22.9	28.8	48.3	52.1	47.7	17.0	39.1	29.6	41.2	48.6	35.5
conv-dpm	56.9	53.2	26.6	17.6	29.9	51.4	42.5	42.4	16.5	31.6	25.0	37.7	52.7	56.7	49.9	16.5	41.0	30.9	44.4	49.7	38.4
conv-dpm+	59.6	56.6	29.8	20	31.1	55.8	42.8	43.3	18.3	35.6	28.5	39.7	56.3	59.7	51.1	19.6	42.1	33.1	49.1	50.3	41.1
conv-dpm+FT-all	63.3	60.2	33.4	24.4	33.6	60	44.7	49.3	19.4	36.6	30.2	40.7	57.7	61.4	52.3	21.2	44.4	37.9	51.1	52.2	**43.7**

Table 6.3: Mean AP on PASCAL VOC 2011

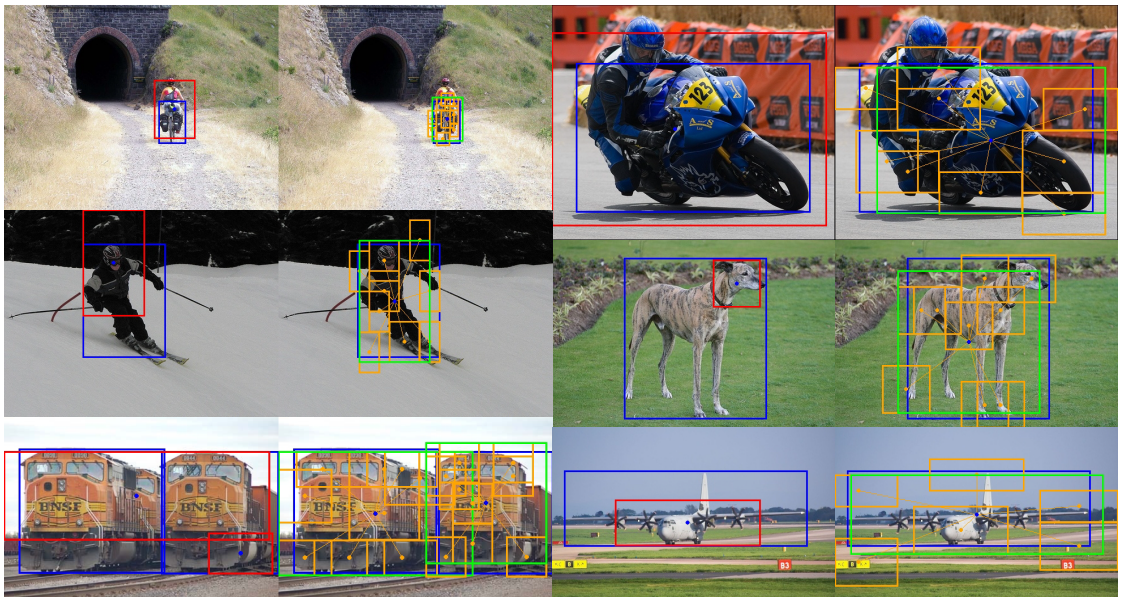


Figure 6.9: Examples of detections using root filter only (left half of each example; red) and the DPM with both root and part filters (right halves; green+orange).

6.6 Discussion

We have described an object detection system that integrates a Convolutional Network, Deformable Parts model and NMS loss in an end-to-end fashion. This fuses together aspects from both structured learning and deep learning: object structures are modeled by a composition of parts and views, while discriminative features are leveraged for appearance comparisons. Our evaluations show that our model achieves competitive performance on PASCAL VOC 2007 and 2011 datasets, and achieves substantial gains from integrating both ConvNet features as well as NMS, and training all parts jointly.

Figure 6.10: Examples of correct (green) and incorrect (red) detections found by our model.

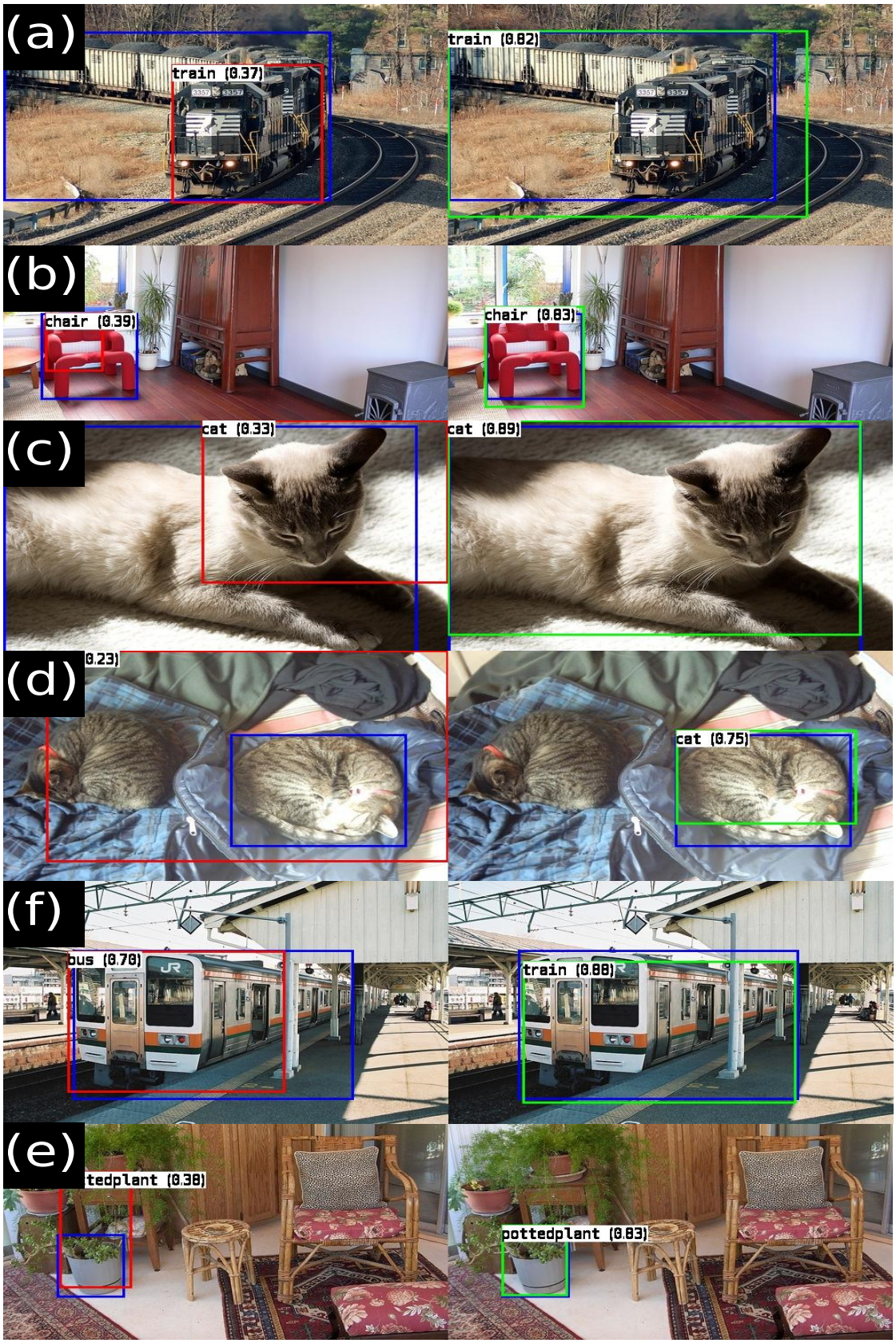


Figure 6.11: Examples of model with (green) and without (red) NMS loss (parts location are ommited)

Chapter 7

Conclusion

Structured models are good at capturing high-level information into model and neural networks are effectively learning good feature representation. Throughout this thesis, we have explored hybrid models combine structured model and neural network for both image classification and object detection.

In Chapter 3, we build hybrid model that combines neural networks with topical model for image classification and shown the advantages of such end-to-end trained hybrid model over individual components. In Chapter 4, we proposed neural network regularization technique called DropConnect to overcome overfitting problem in training. Both theoretical and empirical results clearly shown the advantages of DropConnect. In Chapter 5, we build a detection system based on deformable part model with handcraft features. An iterative non-maximal suppression algorithm also proposed for post-processing of detector result. To overcome the disadvantages of handcraft features, we build a hybrid model that combines ConvNets and part-based model for object detection (In Chapter 6). This system also integrates post-processing step which was treated as a separate step before. A single network represents object contains root and parts with ConvNet feature extractor plus end-to-end training achieves good performance. Extensive experiment results shown performance again of such integration.

Bibliography

[1] R. P. Adams, H. M. Wallach, and Z. Ghahramani. Learning the structure of deep sparse graphical models. *Journal of Machine Learning Research - Proceedings Track*, 9:1–8, 2010.

[2] R. Baur, A. Efros, and M. Hebert. Statistics of 3d object locations in images. Technical report, 2008.

[3] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, U. D. Montral, and M. Qubec. Greedy layer-wise training of deep networks. In *In NIPS*. MIT Press, 2007.

[4] M. B. Blaschko and C. H. Lampert. Learning to localize objects with structured output regression. In *ECCV*, 2008.

[5] D. Blei and J. McAuliffe. Supervised topic models. In *NIPS*, 2007.

[6] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[7] Y. Chen, L. Zhu, and A. L. Yuille. Active mask hierarchies for object detection. In *ECCV 2010*, volume 6315 of *Lecture Notes in Computer Science*, pages 43–56. Springer, 2010.

[8] D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Proceedings of the 2012 IEEE Conference on Computer*

Vision and Pattern Recognition (CVPR), CVPR '12, pages 3642–3649, Washington, DC, USA, 2012. IEEE Computer Society.

[9] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005.

[10] C. Dance, J. Willamowski, L. Fan, C. Bray, and G. Csurka. Visual categorization with bags of keypoints. In *ECCV International Workshop on Statistical Learning in Computer Vision.*, Prague, 2004.

[11] C. Desai, D. Ramanan, and C. Fowlkes. Discriminative models for multi-class object layout. *International Journal of Computer Vision*, 2011.

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

[13] L. Fei-Fei, R. Fergus, and P. Perona. A Bayesian approach to unsupervised one-shot learning of object categories. In *Proceedings of the 9th International Conference on Computer Vision, Nice, France*, pages 1134–1141, Oct. 2003.

[14] L. Fei-Fei and P. Perona. A bayesian hierarchical model for learning natural scene categories. *CVPR*, pages 524–531, 2005.

[15] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(9):1627–1645, Sept. 2010.

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

[17] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 264–271, June 2003.

[18] C. Galleguillos, A. Rabinovich, and S. Belongie. Object categorization using co-occurrence, location and appearance. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, June 2008.

[19] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[20] R. B. Girshick, F. N. Iandola, T. Darrell, and J. Malik. Deformable part models are convolutional neural networks. *CoRR*, abs/1409.5403, 2014.

[21] I. J. Goodfellow, Q. V. Le, A. M. Saxe, H. Lee, and A. Y. Ng. Measuring invariances in deep networks, 2009.

[22] X. He, R. Zemel, and M. Carreira-Perpindn. Multiscale conditional random fields for image labeling. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–695–II–702 Vol.2, June 2004.

[23] G. E. Hinton and S. Osindero. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:2006, 2006.

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

[25] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[26] T. Hofmann. Unsupervised learning by probabilistic latent semantic analysis. In *Machine Learning*, page 2001, 2001.

[27] D. Hoiem, A. A. Efros, and M. Hebert. Putting objects in perspective. *Int. J. Comput. Vision*, 80(1):3–15, Oct. 2008.

[28] K. Kavukcuoglu, P. Sermanet, Y. Boureau, K. Gregor, M. Mathieu, and Y. LeCun. Learning convolutional feature hierachies for visual recognition. In *Advances in Neural Information Processing Systems (NIPS 2010)*, 2010.

[29] K. Kavukcuoglu, P. Sermanet, Y. lan Boureau, K. Gregor, M. Mathieu, and Y. Lecun. Learning convolutional feature hierarchies for visual recognition, 2010.

[30] V. Koltchinskii and D. Panchenko. Empirical margin distributions and bounding the generalization error of combined classifiers. *Annals of Statistics*, 30:2002, 2000.

[31] A. Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Master's thesis, University of Toront, 2009.

[32] A. Krizhevsky. cuda-convnet. `http://code.google.com/p/cuda-convnet/`, 2012.

[33] A. Krizhevsky. cuda-convnet. `http://code.google.com/p/cuda-convnet/`, 2012.

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. pages 1106–1114, 2012.

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1106–1114. 2012.

[36] S. Kumar and M. Hebert. A hierarchical field framework for unified context-based classification. In IEEE, editor, *Tenth IEEE International Conference on Computer Vision (ICCV '05)*, volume 2, pages 1284 – 1291, October 2005.

[37] S. Lacoste-Julien, F. Sha, and M. Jordan. Disclda: Discriminative learning for dimensionality reduction and classification. In *NIPS*, 2009.

[38] 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.

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

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

[41] Y. LeCun, F. J. Huang, and L. Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the 2004 IEEE computer society conference on Computer vision and pattern recognition*, CVPR'04, pages 97–104, Washington, DC, USA, 2004. IEEE Computer Society.

[42] M. Ledoux and M. Talagrand. *Probability in Banach Spaces*. Springer, New York, 1991.

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

[44] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[45] D. J. C. Mackay. Probable networks and plausible predictions - a review of practical bayesian methods for supervised neural networks. In *Bayesian methods for backpropagation networks*. Springer, 1995.

[46] K. Murphy, A. Torralba, and W. T. Freeman. Using the forest to see the trees: A graphical model relating features, objects, and scenes, 2003.

[47] V. Nair and G. E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *ICML*, 2010.

[48] Y. Netzer, T. Wang, C. A., A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[49] J. Ngiam, Z. Chen, P. W. Koh, and A. Ng. Learning deep energy models. In L. Getoor and T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 1105–1112, New York, NY, USA, June 2011. ACM.

[50] T. Ojala, M. Pietikäinen, and T. Mäenpää. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(7):971–987, July 2002.

[51] D. Parikh and C. L. Zitnick. Human-debugging of machines. In *In NIPS WCSSWC*, 2011.

[52] M. Ranzato and G. Hinton. Modeling pixel means and covariances using factorized third-order Boltzmann Machines. In *CVPR*, 2010.

[53] X. Ren and D. Ramanan. Histograms of sparse codes for object detection. *2013 IEEE Conference on Computer Vision and Pattern Recognition*, 0:3246–3253, 2013.

[54] M. Rosen-Zvi, T. Griffiths, M. Steyvers, and P. Smyth. The author-topic model for authors and documents. In *Uncertainty in Artificial Intelligence 20*, pages 487–494, 2004.

[55] R. Salakhutdinov and G. E. Hinton. Using deep belief nets to learn covariance kernels for gaussian processes. In *NIPS*, 2008.

[56] R. Salakhutdinov, J. Tenenbaum, and A. Torralba. Learning to learn with compound hd models. In *NIPS*, 2011.

[57] C. Schmid. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *In CVPR*, pages 2169–2178, 2006.

[58] P. Schnitzspan, M. Fritz, S. Roth, and B. Schiele. Discriminative structure learning of hierarchical representations for object detection. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2238–2245, June 2009.

[59] P. Sermanet, S. Chintala, and Y. LeCun. Convolutional neural networks applied to house numbers digit classification. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 3288–3291, Nov 2012.

[60] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229, 2013.

[61] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *Proceedings of the International Conference on Computer Vision*, volume 2, pages 1470–1477, Oct. 2003.

[62] J. Snoek, H. Larochelle, and R. A. Adams. Practical bayesian optimization of machine learning algorithms. In *Neural Information Processing Systems*, 2012.

[63] E. Sudderth and M. Jordan. Shared segmentation of natural scenes using dependence Pitman-Yor processes. In *Proc. Neural Information Processing Systems*, 2008.

[64] E. Sudderth, A. Torralba, W. Freeman, and A. Willsky. Describing visual scenes using transformed objects and parts. *Intl. Journal of Computer Vision*, 77, March 2008.

[65] C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. *NIPS*, 2013.

[66] J. Tompson, A. Jain, Y. LeCun, and C. Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. *CoRR*, abs/1406.2984, 2014.

[67] A. Torralba, K. P. Murphy, and W. T. Freeman. Contextual models for object detection using boosted random fields. In L. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1401–1408. MIT Press, 2005.

[68] J. Uijlings, K. Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013.

[69] S. Walk, N. Majer, K. Schindler, and B. Schiele. New features and insights for pedestrian detection. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1030–1037, June 2010.

[70] L. Wan, M. Zeiler, S. Zhang, Y. Lecun, and R. Fergus. Regularization of neural networks using dropconnect. In *In ICML*, 2013.

[71] X. Wang, M. Yang, S. Zhu, and Y. Lin. Regionlets for generic object detection. *IEEE 14th International Conf. on Computer Vision*, 2013.

[72] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Generalization by weight-elimination with application to forecasting. In *NIPS*, 1991.

[73] Y. N. Wu, Z. Si, H. Gong, and S. chun Zhu. Active basis for modeling, learning and recognizing deformable templates, 2008.

[74] A. L. Yuille and A. Rangarajan. The concave-convex procedure. *Neural Comput.*, 15(4):915–936, Apr. 2003.

[75] M. D. Zeiler and R. Fergus. Stochastic pooling for regualization of deep convolutional neural networks. In *ICLR*, 2013.

[76] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[77] L. Zhu, Y. Chen, A. Torralba, W. Freeman, and A. Yuille. Part and appearance sharing: Recursive compositional models for multi-view. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1919–1926, June 2010.

[78] L. L. Zhu, Y. Chen, A. Yuille, and W. Freeman. Latent hierarchical structural learning for object detection. *2010 IEEE Conference on Computer Vision and*

Pattern Recognition, 0:1062–1069, 2010.