Building and Leveraging Category Hierarchies for Large-scale Image Classification

Hao Zhang
July 31, 2016

Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Eric P. Xing, Chair
Abhinav Gupta
Wei Dai

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

CMU-RI-TR-16-38

Keywords: Image Categorization, Taxonomy Induction, Convolutional Neural Networks, Probabilistic Inference

Abstract

In image classification, visual separability between different object categories is highly uneven, and some categories are more difficult to distinguish than others. Such difficult categories demand more dedicated classifiers. However, existing deep convolutional neural networks (CNN) are trained as flat N-way classifiers, and few efforts have been made to leverage the hierarchical structure of categories. Naturally, incorporating external knowledge from category hierarchies presents a major opportunity to improve the task. However, traditional methods of manually constructing category hierarchies by experts (e.g. WordNet, ImageNet) and interest communities (e.g. Wikipedia) are either knowledge or time intensive, and the results have limited coverage.

In this report, we study the problem of automatically learning and utilizing category hierarchies (taxonomies) for large-scale image classification. First, we present a probabilistic model for taxonomy induction by jointly leveraging text corpus and images from the web. The model is discriminatively trained given a small set of existing ontologies and is capable of building full category hierarchies from scratch for a collection of unseen conceptual labels with associated images. Then, we introduce hierarchical deep convolutional neural networks (HD-CNNs), which embeds deep convolutional neural networks into a category hierarchy. An HD-CNN separates easy classes using a coarse category classifier while distinguishing difficult classes using fine category classifiers. We reported state-of-the-art results on both taxonomy induction and image classification tasks.

Acknowledgments

I wish to thank my advisor at Carnegie Mellon, Prof. Eric P. Xing, for giving me so much freedom to discover and explore new subjects in machine learning, computer vision and distributed systems. An M.S. under his mentorship is the experience of a lifetime.

My other committee members have also been very supportive. Prof. Abhinav Gupta has been a warm supporter on my endeavor in computer vision. Wei has been a great friend who gave me a lot of support on both research, study and life.

Further, I would like to thank my friends and colleagues at Carnegie Mellon with whom I have had the pleasure of working over the years. Their encouragement and friendship and their help have brought me incredible joy during my study at CMU.

I am especially thankful to my girlfriend, Luona Yang, for her constant support and encouragement on my work with love, patience and understanding. She has been always standing on my side during my up and downs. She is the inner peace in my heart. Finally, I want to thank my parents for always supporting me to pursue my dreams and being there whenever I needed them. Without them, I would not have come this far.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Image classification is one of the central tasks in computer vision and has been studied for decades. State-of-the-art results in image classification are mostly established by deep convolutional neural networks (CNNs) [30]. Deep CNNs are well suited for large-scale supervised visual recognition tasks because of their huge model capacity and highly scalable training algorithm.

However, in CNN-based image classification, one of the complications that arise in large datasets with large number of categories is that the visual separability of object categories is highly uneven. Some categories are much harder to distinguish than others. Take the categories in CIFAR100 [29] as an example. It is easy to tell an *Apple* from a *Bus*, but harder to tell an *Apple* from an *Orange*. In fact, both *Apples* and *Oranges* belong to the same coarse category *fruit and vegetables* while *Buses* belong to another coarse category *vehicles 1*, as defined within CIFAR100. Nonetheless, most deep CNN models nowadays are flat N-way classifiers, which share a set of fully connected layers. This makes us wonder whether such a flat structure is adequate for distinguishing all the difficult categories. A very natural and intuitive alternative organizes classifiers in a hierarchical manner according to the category hierarchies, so that the external knowledge from hierarchies could be utilized for the classifier to distinguish difficult categories. Although hierarchical classification has been proven effective for conventional linear classifiers [14, 34, 60, 62], few attempts have been made to exploit category hierarchies [12, 46] in deep CNN models.

Since deep CNN models are large models themselves, organizing them hierarchically imposes the following challenges. First, traditional methods of manually constructing category hierarchies by experts (e.g. WordNet) and interest communities (e.g. Wikipedia) are either knowledge or time intensive, and the results have limited coverage. Instead of handcrafting a hierarchy with extensive labor, how can we automatically induce taxonomic structure from data? Second, a hierarchical CNN classifier consists of multiple CNN models at different levels. How can we leverage the commonalities among these models and effectively train them all? Third, it would also be slower and more memory-consuming to run a hierarchical CNN classifier on a novel testing image. How can we alleviate such limitations?

In this report, we address the aforementioned problems from two aspects. On one hand, we propose a novel probabilistic framework (Chapter 3) for taxonomy induction by jointly leveraging textual and visual data. Our observation is that both textual and visual information provide

important cues for taxonomy induction. For example, in a semantically meaningful taxonomy, the parent category *seafish* and its two child categories *shark* and *ray* are closely related as: (1) there is a hypernym-hyponym (*is-a*) relation between the words "seafish" and "shark"/"ray" through text descriptions like "...seafish, such as shark and ray...", "...shark and ray are a group of seafish..."; (2) images of the close neighbors, e.g., *shark* and *ray* are usually visually similar and images of the child, e.g. *shark/ray* are similar to a subset of images of *seafish*. To effectively capture these patterns, in contrast to previous works that rely on various hand-crafted features [2, 7], we extract features by leveraging the *distributed representations* that embed images [41] and words [37] as compact vectors, based on which the semantic closeness is directly measured in vector space. Further, we develop a Bayesian model that integrates the rich multi-modal features to induce "is-a" relations between categories, encouraging *local semantic consistency* that each category should be visually and textually close to its parent and siblings. The model is discriminatively trained and can be directly applied to build a category hierarchy from scratch for a collection of semantic labels. We demonstrated superior performance of our model and feature on automatical taxonomy induction tasks.

Given the pre-built hierarchies, on the other hand, we propose a generic and principled hierarchical architecture, *Hierarchical Deep Convolutional Neural Network* (HD-CNN), that embeds a convolutional neural network classifier onto the category hierarchy. It thus decomposes an image classification task into two steps. First, easy classes are separated by a coarse category CNN classifier which corresponds to the intermediate nodes in the category hierarchy. Second, more challenging classes are routed downstream to fine category classifiers (leaf nodes). HD-CNN is modular and is built upon a *building block* CNN, which can be chosen to be any of the state-of-the-art single CNN. An HD-CNN follows the coarse-to-fine classification paradigm and probabilistically integrates predictions from fine category classifiers. We show that HD-CNN can achieve lower error than the corresponding building block CNN, at the cost of a manageable increase in memory footprint and classification time.

In summary, this report has the following contributions. First, we propose a novel probabilistic Bayesian model (Chapter 3) for taxonomy induction by jointly leveraging textual and visual data. The model is discriminatively trained and can be directly applied to build a taxonomy from scratch for a collection of semantic labels. We design novel features (Section 3.3) based on general-purpose distributed representations of text and images to capture both textual and visual relations between labels. We empirically evaluate our model and features on the ImageNet hierarchies with two different taxonomy induction tasks (Section 5.1.2). We achieve superior performance on both tasks and improve the F_1 score by 2x in the *taxonomy construction* task, compared to previous approaches. Second, we introduce HD-CNN, a novel hierarchical CNN architecture for image classification. We develop a scheme for learning the two-level organization of coarse and fine categories based on the pre-built category hierarchies, and demonstrate that various components of an HD-CNN can be independently pretrained. We show state-of-the-art performance on the image classification task on both CIFAR100 and ImageNet dataset.

The rest of this thesis is organized as follows. In chapter 2 we review related works. Chapter 3 elaborates the proposed model and features for taxonomy induction, and chapter 4 details the design and architecture of HDCNN. In chapter 5 we demonstrate empirical results on both taxonomy induction and image classification. Chapter 6 concludes the thesis.

Chapter 2

Related Work

2.1 Taxonomy Induction

Building taxonomies using textual data. Many approaches have been recently developed that build hierarchies purely by identifying either lexical patterns or statistical features in text corpora [2, 15, 27, 28, 38, 43, 50, 51, 56, 61]. The approaches in [56] and [43] assume a starting incomplete hierarchy and try to extend it by inserting new terms. [28] and [38] first find leaf nodes and then use lexical patterns to find intermediate terms and all the attested hypernymy links between them. In [50], syntactic contextual similarity is exploited to construct the taxonomy, while [51] go one step further to consider trustiness and collective synonym/contrastive evidence. Different from them, our model is discriminatively trained with multi-modal data. The works of [15] and [2] use similar language-based features as ours. Specifically, in [15], linguistic regularities between pretrained word vectors [37] are modeled as projection mappings. The trained projection matrix is then used to induce pairwise hypernym-hyponym relations between words. Our features are partially motivated by [15], but we jointly leverage both textual and visual information. In [27], both textual and visual evidences are exploited to detect pairwise lexical entailments. Our work is significantly different as our model is optimized over the whole taxonomy space rather than considering only word pairs separately. In [2], a structural learning model is developed to induce a globally optimal hierarchy. Compared with this work, we exploit much richer features from both text and images, and leverage distributed representations instead of hand-crafted features.

Building taxonomies using visual data. Several approaches [3, 21, 36] have also been proposed to construct visual hierarchies from image collections. In [3], a nonparametric Bayesian model is developed to group images based on low-level features. In [21] and [36], a visual taxonomy is built to accelerate image categorization. In [7], only binary object-object relations are extracted using co-detection matrices. Our work differs from all of these as we integrate textual with visual information to construct taxonomies. Also of note are several works that integrate text and images as evidence for knowledge base autocompletion [5] and zero-shot recognition [16, 17, 44]. Our work is different because our task is to accurately construct multi-level hyponym-hypernym hierarchies from a set of (seen or unseen) categories.

Evaluate the visual semantic relatedness. There are also considerable efforts in designing vi-

sual features to evaluate the visual semantic relatedness [7, 39]. In [7], to extract binary relations between categories, they have to pre-construct a co-detection matrix. By contrast, we directly extract end-to-end vector features by exploiting distributed representations of images, avoiding any feature engineering and detector training. In [39], the visual semantic relatedness of categories is indirectly measured depending on linguistic patterns extracted from WordNet, Wikipedia, etc. In contrast, we measure the visual relatedness using our image-image features, which are directly extracted from images themselves, thus our model not only considers linguistic patterns, but encode perceptual semantics underlying the images themselves.

2.2 Convolutional Neural Networks

CNN-based models hold state-of-the-art performance in various computer vision tasks, including image classification [30], object detection [19, 23], and image parsing [13]. Recently, there has been considerable interest in enhancing CNN components, including pooling layers [58], activation units [20, 45], and nonlinear layers [33]. These enhancements either improve CNN training [58], or expand the network learning capacity. BY contrast, HDCNN boosts CNN performance from an orthogonal angle and does not redesign a specific part within any existing CNN model. Instead, we design a novel generic hierarchical architecture that uses an existing CNN model as a building block. We embed multiple building blocks into a larger hierarchical deep CNN.

2.3 Category Hierarchy for Visual Recognition

In visual recognition, there is a vast literature exploiting category hierarchical structures [49]. For classification with a large number of classes using linear classifiers, a common strategy is to build a hierarchy or taxonomy of classifiers so that the number of classifiers evaluated given a testing image scales sub-linearly in the number of classes [4, 18]. The hierarchy can be either predefined [26, 35, 52] or learnt by top-down and bottom-up approaches [1, 10, 21, 32, 36, 40, 42]. In [11], the predefined category hierarchy of ImageNet dataset is utilized to achieve trade-offs between classification accuracy and specificity. In [34], a hierarchical label tree is constructed to probabilistically combine predictions from leaf nodes. Such hierarchical classifier achieves significant speedup at the cost of certain accuracy loss.

One of the earliest attempts to introduce a category hierarchy in CNN-based methods is reported in [46] but their main goal is transferring knowledge between classes to improve the results for classes with insufficient training examples. In [12], various label relations are encoded in a hierarchy. Improved accuracy is achieved only when a subset of training images are relabeled with internal nodes in the hierarchical class tree. They are not able to improve accuracy in the original setting where all training images are labeled with leaf nodes. In [54], a hierarchy of CNNs is introduced but they experimented with only two coarse categories mainly due to scalability constraints. HD-CNN exploits the category hierarchy in a novel way that we embed deep CNNs into the hierarchy in a scalable manner and achieves superior classification results over standard CNN.

Chapter 3

Taxonomy Induction

In this chapter, we introduce a probabilistic model for taxonomy induction based on features extracted from both text and images. Our model is motivated by the key observation that in a semantically meaningful taxonomy, a category tends to be closely related to its children as well as its siblings. Figure 3.1 illustrates such an example. For instance, there exists a hypernym-hyponym relation between the name of category *shark* and that of its parent *seafish*. Besides, images of *shark* tend to be visually similar to those of *ray*, both of which are seafishes. Our model is thus designed to encourage such local semantic consistency; and by jointly considering all categories in the inference, a globally optimal structure is achieved. A key advantage of the model is that we incorporate both visual and textual features induced from distributed representations of images and text (Section 3.3). These features capture the rich underlying semantics and facilitate taxonomy induction. We further distinguish the relative importance of visual and textual features that could vary in different layers of a taxonomy. Intuitively, visual features would be increasingly indicative in the deeper layers, as sub-categories under the same category of specific objects tend to be visually similar. In contrast, textual features would be more important when inducing hierarchical relations between the categories of general concepts (i.e. in the near-root layers) where visual characteristics are not necessarily similar.

The rest of this chapter is organized as follows. Section 3.1 formally defines the problem and provides the notations. Section 3.2 induces our proposed model, and section 3.3 elaborates the features we used for taxonomy induction. Empirical studies of our model could be found in chapter 5.

3.1 Problem Definition

Assume a set of N categories $\boldsymbol{x} = \{x_1, x_2, \dots, x_N\}$, where each category x_n consists of a text term t_n as its name, as well as a set of images $\boldsymbol{i}_n = \{i_1, i_2, \dots\}$. Our goal is to construct a taxonomy tree T over these categories[1], such that categories of specific object types (e.g. shark) are grouped and assigned to those of general concepts (e.g. seafish). As the categories in $\boldsymbol{x}$ may

[1]We assume T to be a tree. Most existing taxonomies are modeled as trees [2], since a tree helps simplify the construction and ensures that the learned taxonomy is interpretable. With minor modifications, our model also works on non-tree structures.

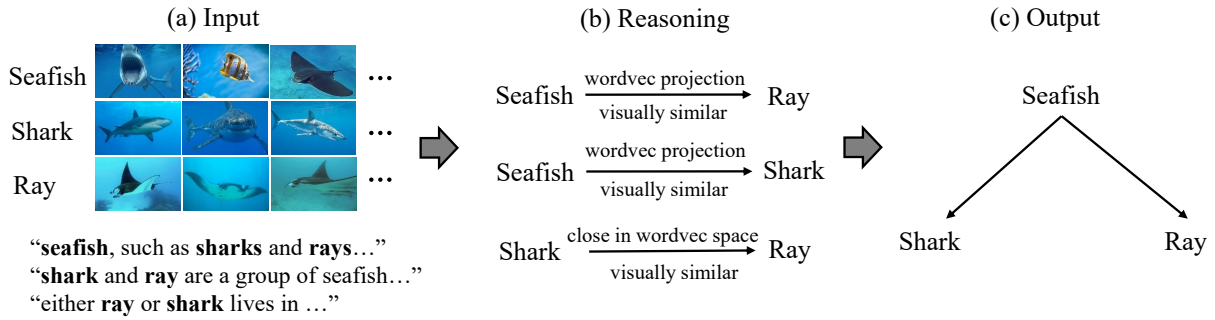


Figure 3.1: An overview of our system. Input: a collection of label items, represented by text and images; Output: we build a taxonomy from scratch by extracting features based on distributed representations of text and images.

be from multiple disjoint taxonomy trees, we add a *pseudo* category x_0 as the hyper-root so that the optimal taxonomy is ensured to be a single tree. Let $z_n \in \{1, \ldots, N\}$ be the index of the parent of category x_n, i.e. x_{z_n} is the hypernymic category of x_n. Thus the problem of inducing a taxonomy structure is equivalent to inferring the conditional distribution $p(\boldsymbol{z}|\boldsymbol{x})$ over the set of (latent) indices $\boldsymbol{z} = \{z_1, \ldots, z_n\}$, based on the images and text.

3.2 Model

We formulate the distribution $p(\boldsymbol{z}|\boldsymbol{x})$ through a model which leverages rich multi-modal features. Specifically, let $\boldsymbol{c}_n$ be the set of child nodes of category x_n in a taxonomy encoded by $\boldsymbol{z}$. Our model is defined as

$$p_w(\boldsymbol{z}, \boldsymbol{\pi}|\boldsymbol{x}, \boldsymbol{\alpha}) \propto p(\boldsymbol{\pi}|\boldsymbol{\alpha}) \prod_{n=1}^{N} \prod_{x_{n'} \in \boldsymbol{c}_n} \pi_n g_w(x_n, x_{n'}, \boldsymbol{c}_n \backslash x_{n'}) \tag{3.1}$$

where $g_w(x_n, x_{n'}, \boldsymbol{c}_n \backslash x_{n'})$, defined as

$$g_w(x_n, x_{n'}, \boldsymbol{c}_n \backslash x_{n'}) = \exp\{\boldsymbol{w}_{d(x_{n'})}^\top \boldsymbol{f}_{n,n',\boldsymbol{c}_n \backslash x_{n'}}\},$$

measures the semantic consistency between category $x_{n'}$, its parent x_n as well as its siblings indexed by $\boldsymbol{c}_n \backslash x_{n'}$. The function $g_w(\cdot)$ is loglinear with respect to $\boldsymbol{f}_{n,n',\boldsymbol{c}_n \backslash x_{n'}}$, which is the feature vector defined over the set of relevant categories $(x_n, x_{n'}, \boldsymbol{c}_n \backslash x_{n'})$, with $\boldsymbol{c}_n \backslash x_{n'}$ being the set of child categories excluding $x_{n'}$ (Section 3.3). The simple exponential formulation can effectively encourage close relations among nearby categories in the induced taxonomy. The function has combination weights $\boldsymbol{w} = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_L\}$, where L is the maximum depth of the taxonomy, to capture the importance of different features, and the function $d(x_{n'})$ to return the depth of $x_{n'}$ in the current taxonomy. Each layer l $(1 \leq l \leq L)$ of the taxonomy has a specific $\boldsymbol{w}_l$ thereby allowing varying weights of the same features in different layers. The parameters are learned in a *supervised* manner. In eq 3.1, we also introduce a weight π_n for each node x_n, in order to capture the varying popularity of different categories (in terms of being a parent category). For example, some categories like *plant* can have a large number of sub-categories,

while others such as *stone* have less. We model $\boldsymbol{\pi}$ as a multinomial distribution with Dirichlet prior $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_N)$ to encode any prior knowledge of the category popularity[2]; and the conjugacy allows us to marginalize out $\boldsymbol{\pi}$ analytically to get

$$\begin{aligned} p_w(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{\alpha}) &\propto \int p(\boldsymbol{\pi}|\boldsymbol{\alpha}) \prod_{n=1}^{N} \prod_{x_{n'} \in \boldsymbol{c}_n} \pi_n g_w(x_n, x_{n'}, \boldsymbol{c}_n \backslash x_{n'}) d\boldsymbol{\pi} \\ &\propto \prod_n \Gamma(q_n + \alpha_n) \prod_{x_{n'} \in \boldsymbol{c}_n} g_w(x_n, x_{n'}, \boldsymbol{c}_n \backslash x_{n'}) \end{aligned} \tag{3.2}$$

where q_n is the number of children of category x_n.

Next, we describe our approach to infer the expectation for each z_n, and based on that select a particular taxonomy structure for the category nodes $\boldsymbol{x}$. As $\boldsymbol{z}$ is constrained to be a tree (i.e. cycle without loops), we include with eq A.1, an indicator factor $\mathbf{1}(\boldsymbol{z})$ that takes 1 if $\boldsymbol{z}$ corresponds a tree and 0 otherwise. We modify the inference algorithm appropriately to incorporate this constraint.

Inference. Exact inference is computationally intractable due to the normalization constant of eq A.1. We therefore use Gibbs Sampling, a procedure for approximate inference. Here we present the sampling formula for each z_n directly, and defer the details to Appendix A. The sampling procedure is highly efficient because the normalization term and the factors that are irrelevant to z_n are cancelled out. The formula is

$$\begin{aligned} p(z_n = m|\boldsymbol{z} \backslash z_n, \cdot) \propto \mathbf{1}(z_n = m, \boldsymbol{z} \backslash z_n) \cdot \left(q_m^{-n} + \alpha_m\right) \cdot \\ \frac{\prod_{x_{n'} \in \boldsymbol{c}_m \cup \{x_n\}} g_w(x_m, x_{n'}, \boldsymbol{c}_m \cup \{x_n\})}{\prod_{x_{n'} \in \boldsymbol{c}_m \backslash x_n} g_w(x_m, x_{n'}, \boldsymbol{c}_m \backslash x_n)}, \end{aligned} \tag{3.3}$$

where q_m is the number of children of category m; the superscript $-n$ denotes the number excluding x_n. Examining the validity of the taxonomy structure (i.e. the tree indicator) in each sampling step can be computationally prohibitive. To handle this, we restrict the candidate value of z_n in eq A.1.2, ensuring that the new z_n is always a tree. Specifically, given a tree T, we define a *structure operation* as the procedure of detaching one node x_n in T from its parent and appending it to another node x_m which is not a descendant of x_n.

Proposition 1. *(1) Applying a structure operation on a tree T will result in a structure that is still a tree. (2) Any tree structure over the node set $\boldsymbol{x}$ that has the same root node with tree T can be achieved by applying structure operation on T a finite number of times.*

The proof is straightforward and we omit it due to space limitations. We also add a pseudo node x_0 as the fixed root of the taxonomy. Hence by initializing a tree-structured state rooted at x_0 and restricting each updating step as a structure operation, our sampling procedure is able to explore the whole valid tree space.

Output taxonomy selection. To apply the model to discover the underlying taxonomy from a given set of categories, we first obtain the marginals of $\boldsymbol{z}$ by averaging over the samples generated through eq A.1.2, then output the optimal taxonomy $\boldsymbol{z}^*$ by finding the maximum spanning tree (MST) using the Chu-Liu-Edmonds algorithm [2, 8].

[2] $\boldsymbol{\alpha}$ could be estimated using training data.

Training. We need to learn the model parameters $\boldsymbol{w}_l$ of each layer l, which capture the relative importance of different features. The model is trained using the EM algorithm. Let $\ell(x_n)$ be the depth (layer) of category x_n; and $\tilde{\boldsymbol{z}}$ (siblings $\tilde{\boldsymbol{c}}_n$) denote the gold structure in training data. Our training algorithm updates $\boldsymbol{w}$ through maximum likelihood estimation, wherein the gradient of $\boldsymbol{w}_l$ is (see Appendix A for details):

$$\delta \boldsymbol{w}_l = \sum_{n:\ell(x_n)=l} \{\boldsymbol{f}(x_{\tilde{z}_n}, x_n, \tilde{\boldsymbol{c}}_n \backslash x_n) - \mathbb{E}_p[\boldsymbol{f}(x_{z_n}, x_n, \boldsymbol{c}_n \backslash x_n)]\},$$

which is the net difference between gold feature vectors and expected feature vectors as per the model. The expectation is approximated by collecting samples using the sampler described above and averaging them.

3.3 Features

In this section, we describe the feature vector $\boldsymbol{f}$ used in our model, and defer more details in Appendix A. Compared to previous taxonomy induction works which rely purely on linguistic information, we exploit both perceptual and textual features to capture the rich spectrum of semantics encoded in images and text. Moreover, we leverage the *distributed representations* of images and words to construct compact and effective features. Specifically, each image i is represented as an embedding vector $\boldsymbol{v}_i \in \mathbb{R}^a$ extracted by deep convolutional neural networks. Such image representation has been successfully applied in various vision tasks. On the other hand, the category name t is represented by its word embedding $\boldsymbol{v}_t \in \mathbb{R}^b$, a low-dimensional dense vector induced by the Skip-gram model [37] which is widely used in diverse NLP applications too. Then we design $\boldsymbol{f}(x_n, x_{n'}, \boldsymbol{c}_n \backslash x_{n'})$ based on the above image and text representations. The feature vector $\boldsymbol{f}$ is used to measure the local semantic consistency between category $x_{n'}$ and its parent category x_n as well as its siblings $\boldsymbol{c}_n \backslash x_{n'}$.

3.3.1 Image Features

Sibling similarity. As mentioned above, close neighbors in a taxonomy tend to be visually similar, indicating that the embedding of images of sibling categories should be close to each other in the vector space $\mathbb{R}^a$. For a category x_n and its image set $\boldsymbol{i}_n$, we fit a Gaussian distribution $\mathcal{N}(\overline{\boldsymbol{v}}_{\boldsymbol{i}_n}, \Sigma_n)$ to the image vectors, where $\overline{\boldsymbol{v}}_{\boldsymbol{i}_n} \in \mathbb{R}^a$ is the mean vector and $\Sigma_n \in \mathbb{R}^{a \times a}$ is the covariance matrix. For a sibling category x_m of x_n, we define the visual similarity between x_n and x_m as

$$vissim(x_n, x_m) = [\mathcal{N}(\overline{\boldsymbol{v}}_{\boldsymbol{i}_m}; \overline{\boldsymbol{v}}_{\boldsymbol{i}_n}, \Sigma_n) + \mathcal{N}(\overline{\boldsymbol{v}}_{\boldsymbol{i}_n}; \overline{\boldsymbol{v}}_{\boldsymbol{i}_m}, \Sigma_m)]/2$$

which is the average probability of the mean image vector of one category under the Gaussian distribution of the other. This takes into account not only the distance between the mean images, but also the closeness of the images of each category. Accordingly, we compute the visual similarity between $x_{n'}$ and the set $\boldsymbol{c}_n \backslash x_{n'}$ by averaging:

$$vissim(x_{n'}, \boldsymbol{c}_n \backslash x_{n'}) = \frac{\sum_{x_m \in \boldsymbol{c}_n \backslash x_{n'}} vissim(x_{n'}, x_m)}{|\boldsymbol{c}_n| - 1}.$$

We then bin the values of $vissim(x_{n'}, \boldsymbol{c}_n \backslash x_{n'})$ and represent it as an one-hot vector, which constitutes $\boldsymbol{f}$ as a component named as *siblings image-image relation feature* (denoted as *S-V1*[3]).

Parent prediction. Similar to feature S-V1, we also create the similarity feature between the image vectors of the parent and child, to measure their visual similarity. However, the parent node is usually a more general concept than the child, and it usually consists of images that are not necessarily similar to its child. Intuitively, by narrowing the set of images to those that are most similar to its child improves the feature. Therefore, different from S-V1, when estimating the Gaussian distribution of the parent node, we only use the top K images with highest probabilities under the Gaussian distribution of the child node. We empirically show in section 5.1.3 that choosing an appropriate K consistently boosts the performance. We name this feature as *parent-child image-image relation feature* (denoted as *PC-V1*).

Further, inspired by the linguistic regularities of word embedding, i.e. the hypernym-hyponym relationship between words can be approximated by a linear projection operator between word vectors [15, 37], we design a similar strategy to [15] between images and words so that the parent can be "predicted" given the image embedding of its child category and the projection matrix. Specifically, let $(x_n, x_{n'})$ be a parent-child pair in the training data, we learn a projection matrix $\boldsymbol{\Phi}$ which minimizes the distance between $\boldsymbol{\Phi}\overline{\boldsymbol{v}}_{\boldsymbol{i}_{n'}}$ (i.e. the projected mean image vector $\overline{\boldsymbol{v}}_{\boldsymbol{i}_{n'}}$ of the child) and $\boldsymbol{v}_{t_n}$ (i.e. the word embedding of the parent):

$$\boldsymbol{\Phi}^* = \underset{\boldsymbol{\Phi}}{\operatorname{argmin}} \frac{1}{N} \sum_n \|\boldsymbol{\Phi}\overline{\boldsymbol{v}}_{\boldsymbol{i}_{n'}} - \boldsymbol{v}_{t_n}\|_2^2 + \lambda\|\boldsymbol{\Phi}\|_1,$$

where N is the number of parent-child pairs in the training data. Once the projection matrix has been learned, the similarity between a child node $x_{n'}$ and its parent x_n is computed as $\|\boldsymbol{\Phi}\overline{\boldsymbol{v}}_{\boldsymbol{i}_{n'}} - \boldsymbol{v}_{t_n}\|$, and we also create an one-hot vector by binning the feature value. We call this feature as *parent-child image-word relation feature* (*PC-V2*).

3.3.2 Word Features

We briefly introduce the text features employed. More details about the text feature extraction could be found in Appendix A.

Word embedding features. Similar to PC-V1, We induce features using word vectors to measure both sibling-sibling and parent-child closeness in text domain [15]. One exception is that, as each category has only one word, the sibling similarity is computed as the cosine distance between two word vectors. This will produce another two parts of features, *parent-child word-word relation feature* (*PC-T1*) and *siblings word-word relation feature* (*S-T1*).

Word surface features. In addition to the embedding-based features, we further leverage lexical features based on the surface forms of child/parent category names. Specifically, we employ the *Capitalization*, *Ends with*, *Contains*, *Suffix match*, *LCS* and *Length different* features, which are commonly used in previous works in taxonomy induction [2, 56].

[3]S: sibling, PC: parent-child, V: visual, T: textual.

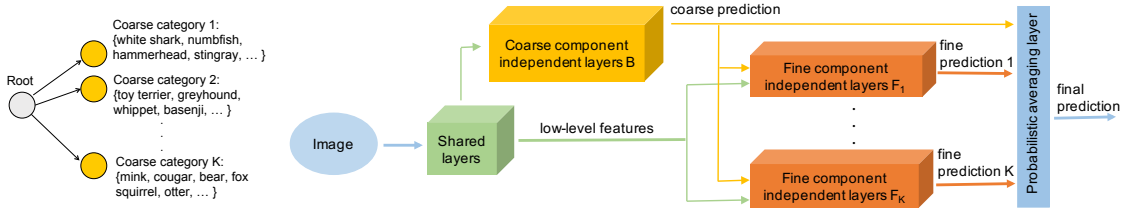
Chapter 4

Hierarchial Deep Convolutional Neural Networks (HDCNN)

In this chapter, we introduce the architecture of hierarchial deep convolutional neural networks in detail [1].

4.1 Notations

Assume the dataset consists of a set of pairs $\{\mathbf{x}_i, y_i\}$, where $\mathbf{x}_i$ is an image and y_i its category label. C denotes the number of fine categories, which will be automatically grouped into K coarse categories depending the built category hierarchies in chapter 3. $\{S_j^f\}_{j=1}^C$ and $\{S_k^c\}_{k=1}^K$ are partitions of image indices based on fine and coarse categories. Superscripts f and c denote fine and coarse categories.



(a) (b)

Figure 4.1: (a) A two-level category hierarchy where the classes are taken from ImageNet 1000-class dataset. (b) Hierarchical Deep Convolutional Neural Network (HD-CNN) architecture.

4.1.1 HD-CNN Architecture

HD-CNN is designed to mimic the structure of category hierarchy where fine categories are grouped into coarse categories as in Fig 4.1(a). It performs end-to-end classification as illustrated in Fig 4.1(b). It mainly comprises four parts (i) shared layers (ii) a single component B to

[1]The notations in this chapter is independent of those in chapter 3

handle coarse categories (iii) multiple components $\{F_k\}_{k=1}^K$ (one for each group) for fine classification and (iv) a single probabilistic averaging layer. Shared layers (left of Fig 4.1 (b)) receive raw image pixel as input and extract low-level features. The configuration of shared layers is set to be the same as the preceding layers in the building block CNN. On the top of Fig 4.1(b) are independent layers of coarse category component B which reuses the configuration of rear layers from the building block CNN and produces an intermediate fine prediction $\{B_{ij}^f\}_{j=1}^C$ for an image $\mathbf{x_i}$. To produce a coarse category prediction $\{B_{ik}\}_{k=1}^K$, we append a fine-to-coarse aggregation layer (not shown in Fig 4.1(b)) which reduces fine predictions into coarse using a mapping $P : [1, C] \mapsto [1, K]$. The coarse category probabilities serve two purposes. First, they are used as weights for combining the predictions made by fine category components $\{F_k\}_{k=1}^K$. Second, when thresholded, they enable conditional execution of fine category components whose corresponding coarse probabilities are sufficiently large. In the bottom-right of Fig 4.1 (b) are independent layers of a set of fine category classifiers $\{F_k\}_{k=1}^K$, each of which makes fine category predictions. As each fine component only excels in classifying a small set of categories, they produce a fine prediction over a partial set of categories. The probabilities of other fine categories absent in the partial set are implicitly set to zero. The layer configurations are mostly copied from the building block CNN except that the number of filters in the final classification layer is set to be the size of partial set instead of the full set of categories.

Both coarse (B) and fine ($\{F_k\}_{k=1}^K$) components share common layers. The reason is threefold. First, it is shown in [57] that preceding layers in deep networks response to class-agnostic low-level features such as corners and edges, while rear layers extract more class-specific features such as dog face and bird's legs. Since low-level features are useful for both coarse and fine classification tasks, we allow the preceding layers to be shared by both coarse and fine components. Second, it reduces both the total floating point operations and the memory footprint of network execution. Both are of practical significance to deploy HD-CNN in real applications. Last but not the least, it can decrease the number of HD-CNN parameters which is critical to the success of HD-CNN training.

On the right side of Fig 4.1 (b) is the probabilistic averaging layer which receives fine as well as coarse category predictions and produces a final prediction based on weighted average

$$p(y_i = j|\mathbf{x}_i) = \frac{\sum_{k=1}^K B_{ik} p_k(y_i = j|\mathbf{x}_i)}{\sum_{k=1}^K B_{ik}}, \tag{4.1}$$

where B_{ik} is the probability of coarse category k for image $\mathbf{x}_i$ predicted by the coarse category component B. $p_k(y_i = j|\mathbf{x}_i)$ is the fine category prediction made by the fine category component F_k. We stress that both coarse and fine category components reuse layer configurations from the building block CNN. This flexible modular design allows us to choose the state-of-the-art CNN as building block, depending on the task at hand.

4.2 HD-CNN Training

As we embed fine category components from a category hierarchy into HD-CNN, the number of parameters in rear layers grows linearly in the number of coarse categories. Given the same amount of training data, this increases the training complexity and the risk of over-fitting. On

Algorithm 1: HD-CNN training algorithm

procedure HD-CNN TRAINING
  Step 1: Pretrain HD-CNN
    Step 1.1: Initialize coarse category component
    Step 1.2: Pretrain fine category components
  Step 2: Fine-tune the complete HD-CNN

the other hand, the training images within the stochastic gradient descent mini-batch are probabilistically routed to different fine category components. It requires larger mini-batch to ensure parameter gradients in the fine category components are estimated by a sufficiently large number of training samples. Large training mini-batch both increases the training memory footprint and slows down the training process. Therefore, we decompose the HD-CNN training into multiple steps instead of training the complete HD-CNN from scratch as outlined in Algorithm 1.

4.2.1 Pretraining HD-CNN

We sequentially pretrain the coarse category component and fine category components.

Initializing the Coarse Category Component. We first pretrain a building block CNN F^p using the training set. Subscript p denotes the CNN is pretrained. As both the preceding and rear layers in coarse category component resemble the layers in the building block CNN, we initialize the coarse category component B with the weights of F^p.

Pretraining the Rear Layers of Fine Category Components. Fine category components $\{F_k\}_{k=1}^K$ can be independently pretrained in parallel. Each F_k should specialize in classifying fine categories within the coarse category k. Therefore, the pretraining of each F_k only uses images $\{\mathbf{x}_i | i \in S_k^c\}$ from the coarse category. The shared preceding layers are already initialized and kept fixed in this stage. For each F_k, we initialize all the rear layers except the last convolutional layer by copying the learned parameters from the pretrained model F^p.

4.2.2 Fine-tuning HD-CNN

After both coarse and fine category components are properly pretrained, we fine-tune the complete HD-CNN. As the category hierarchy as well as the associated mapping P^o are learned, each fine category component focuses on classifying a fixed subset of fine categories. During fine-tuning, the semantics of coarse categories predicted by the coarse category component should be kept consistent with those associated with fine category components. Thus we add a coarse category consistency term to regularize the conventional multinomial logistic loss.

Coarse category consistency. The coarse category consistency term ensures the mean coarse category distribution $\{t_k\}_{k=1}^K$ within the mini-batch is preserved during the fine-tuning. The fine-to-coarse category mapping $P : [1, C] \mapsto [1, K]$ (from the category hierarchy) provides a way to specify the target coarse category distribution $\{t_k\}_{k=1}^K$. Specifically, t_k is set to be the fraction of all the training images within the coarse category k under the assumption that the distribution

over coarse categories across the training set is close to that within a training mini-batch.

$$t_k = \frac{\sum_{j|k \in P(j)} \left| S_j^f \right|}{\sum_{k'=1}^{K} \sum_{j|k' \in P(j)} \left| S_j^f \right|} \quad \forall k \in [1, K] \tag{4.2}$$

The final loss function we use for fine-tuning the HD-CNN is shown below.

$$E = -\frac{1}{n} \sum_{i=1}^{n} log(p_{y_i}) + \frac{\lambda}{2} \sum_{k=1}^{K} (t_k - \frac{1}{n} \sum_{i=1}^{n} B_{ik})^2 \tag{4.3}$$

where n is the size of training mini-batch. λ is a regularization constant and is set to $\lambda = 20$.

4.3 HD-CNN Testing

As we add fine category components into the HD-CNN, the number of parameters, memory footprint and execution time in rear layers, all scale linearly in the number of coarse categories. To ensure HD-CNN is scalable for large-scale visual recognition, we develop conditional execution and layer parameter compression techniques.

Conditional Execution. At test time, for a given image, it is not necessary to evaluate all fine category classifiers as most of them have insignificant weights B_{ik} as in Eqn 4.1. Their contributions to the final prediction are negligible. Conditional execution of top relevant fine components can accelerate the HD-CNN classification. Therefore, we threshold B_{ik} using a parametric variable $B_t = (\beta K)^{-1}$ and reset B_{ik} to zero when $B_{ik} < B_t$. Those fine category classifiers with $B_{ik} = 0$ are not evaluated.

Parameter Compression. In HD-CNN, the number of parameters in rear layers of fine category classifiers grows linearly in the number of coarse categories. Thus we compress the layer parameters at test time to reduce memory footprint. Specifically, we choose Product Quantization [24] to compress the parameter matrix $W \in R^{m \times n}$. We first partition it horizontally into segments of width s such that $W = [W^1, ..., W^{(n/s)}]$. Then k-means clustering is used to cluster the rows in $W^i, \forall i \in [1, (n/s)]$. By only storing the nearest cluster indices in an 8-bit integer matrix $Q \in R^{m \times (n/s)}$ and cluster centers in a single-precision floating number matrix $C \in R^{k \times n}$, we can achieve a compression factor of $(32mn)/(32kn + 8mn/s)$, where s, k are hyperparameters for parameter compression.

Chapter 5

Evaluation

We evaluate the whole framework from two aspects. First, we evaluate our taxonomy induction model by training and testing on WordNet Taxonomies. Then, we evaluate HDCNN on image classification tasks using standard benchmark dataset.

5.1 Taxonomy Induction

We first disclose our implementation details in section 5.1.1 and Appendix A for better reproducibility. We then compare our model with previous state-of-the-art methods [2, 15] with two taxonomy induction tasks. Finally, we provide analysis on the weights and taxonomies induced.

5.1.1 Implementation Details

Dataset. We conduct our experiments on the ImageNet2011 dataset [9], which provides a large collection of category items (synsets), with associated images and a label hierarchy (sampled from WordNet) over them. The original ImageNet taxonomy is preprocessed, resulting in a tree structure with 28231 nodes.
Word embedding training. We train word embedding for synsets by replacing each word/phrase in a synset with a unique token and then using Google's word2vec tool [37]. We combine three public available corpora together, including the latest Wikipedia dump [53], the One Billion Word Language Modeling Benchmark [6] and the UMBC webbase corpus [22], resulting in a corpus with total 6 billion tokens. The dimension of the embedding is set to 200.
Image processing. we employ the ILSVRC12 pre-trained convolutional neural networks [41] to embed each image into the vector space. Then, for each category x_n with images, we estimate a multivariate Gaussian parameterized by $\mathcal{N}_{x_n} = (\mu_{x_n}, \Sigma_{x_n})$, and constrain Σ_{x_n} to be diagonal to prevent overfitting. For categories with very few images, we only estimate a mean vector μ_{x_n}. For nodes that do not have images, we ignore the visual feature.
Training configuration. The feature vector is a concatenation of 6 parts, as detailed in section 3.3. All pairwise distances are precomputed and stored in memory to accelerate Gibbs sampling. The initial learning rate for gradient descent in the M step is set to 0.1, and is decreased by a fraction of 10 every 100 EM iterations.

Trees	Tree A	Tree B	Tree C
Synset ID	12638	19919	23733
Name	consumer goods	animal	food, nutrient
$h = 4$	187	207	572
$h = 5$	362	415	890
$h = 6$	493	800	1166
$h = 7$	524	1386	1326

Table 5.1: Statistics of our evaluation set. The bottom 4 rows give the number of nodes within each height $h \in \{4, 5, 6, 7\}$. The scale of the threes range from small to large, and there is no overlapping among them.

5.1.2 Evaluation

Experimental Settings

We evaluate our model on three subtrees sampled from the ImageNet taxonomy. To collect the subtrees, we start from a given root (e.g. consumer goods) and traverse the full taxonomy using BFS, and collect all descendant nodes within a depth h (number of nodes in the longest path). We vary h to get a series of subtrees with increasing heights $h \in \{4, 5, 6, 7\}$ and various scales (maximally 1326 nodes) in different domains. The statistics of the evaluation sets are provided in Table 5.1. To avoid ambiguity, all nodes used in ILSVRC 2012 are removed as the CNN feature extractor is trained on them.

We design two different tasks to evaluate our model. (1) In the *hierarchy completion* task, we randomly remove some nodes from a tree and use the remaining hierarchy for training. In the test phase, we infer the parent of each removed node and compare it with groundtruth. This task is designed to figure out whether our model can successfully induce hierarchical relations after learning from within-domain parent-child pairs. (2) Different from the previous one, the *hierarchy construction* task is designed to test the generalization ability of our model, i.e. whether our model can learn statistical patterns from one hierarchy and transfer the knowledge to build a taxonomy for another collection of out-of-domain labels. Specifically, we select two trees as the training set to learn $\boldsymbol{w}$. In the test phase, the model is required to build the full taxonomy from scratch for the third tree.

We use *Ancestor* F_1 as our evaluation metric [2, 28, 38]. Specifically, we measure $F_1 = 2PR/(P + R)$ values of predicted "is-a" relations where the precision (P) and recall (R) are:

$$P = \frac{|\text{isa}_{\text{predicted}} \cap \text{isa}_{\text{gold}}|}{|\text{isa}_{\text{predicted}}|}, R = \frac{|\text{isa}_{\text{predicted}} \cap \text{isa}_{\text{gold}}|}{|\text{isa}_{\text{gold}}|}.$$

We compare our method to two previously state-of-the-art models by [15] and [2], which are closest to ours.

Results

Hierarchy completion. In the *hierarchy completion* task, we split each tree into 70% nodes for training and 30% for test, and experiment with different h. We compare the following three

Method	$h = 4$	$h = 5$	$h = 6$	$h = 7$
Hierarchy Completion				
Fu2014	0.66	0.42	0.26	0.21
Ours (L)	0.70	0.49	0.45	0.37
Ours (LV)	**0.73**	**0.51**	**0.50**	**0.42**
Hierarchy Construction				
Fu2014	0.53	0.33	0.28	0.18
Bansal2014	0.67	0.53	0.43	0.37
Ours (L)	0.58	0.41	0.36	0.30
Ours (LB)	0.68	0.55	0.45	0.40
Ours (LV)	0.66	0.52	0.42	0.34
Ours (LVB - E)	0.68	0.55	0.44	0.39
Ours (LVB)	**0.70**	**0.57**	**0.49**	**0.43**

Table 5.2: Comparisons among different variants of our model, [15] and [2] on two tasks. The ancestor-F_1 scores are reported.

systems: (1) *Fu2014*[1] [15]; (2) *Ours (L)*: Our model with only language features enabled (i.e. surface features, parent-child word-word relation feature and siblings word-word relation feature); (3) *Ours (LV)*: Our model with both language features and visual features [2]. The average performance on three trees are reported at Table 5.2. We observe that the performance gradually drops when h increases, as more nodes are inserted when the tree grows higher, leading to a more complex and difficult taxonomy to be accurately constructed. Overall, our model outperforms Fu2014 in terms of the F_1 score, even without visual features. In the most difficult case with $h = 7$, our model still holds an F_1 score of 0.42 ($2\times$ of Fu2014), demonstrating the superiority of our model.

Hierarchy construction. The hierarchy construction task is much more difficult than hierarchy completion task because we need to build a taxonomy from scratch given only a hyper-root. For this task, we use a leave-one-out strategy, i.e. we train our model on every two trees and test on the third, and report the average performance in Table 5.2. We compare the following methods: (1) *Fu2014*, (2) *Ours (L)*, and (3) *Ours (LV)*, as described above; (4) *Bansal2014*: The model by [2] retrained using our dataset; (5) *Ours (LB)*: By excluding visual features, but including other language features from [2]; (6) *Ours (LVB)*: Our full model further enhanced with all semantic features from [2]; (7) *Ours (LVB - E)*: By excluding word embedding-based language features from *Ours (LVB)*.

As shown, on the hierarchy construction task, our model with only language features still outperforms Fu2014 with a large gap (0.30 compared to 0.18 when $h = 7$), which uses similar embedding-based features. The potential reasons are two-fold. First, we take into account not only parent-child relations but also siblings. Second, their method is designed to induce only pairwise relations. To build the full taxonomy, they first identify all possible pairwise relations

[1]We tried different parameter settings for the number of clusters C and the identification threshold δ, and reported the best performance we achieved.

[2]In the comparisons to [15] and [2], we simply set $K = \infty$, *i.e.* we use all available images of the parent category to estimate the PC-V1 feature.

using a simple thresholding strategy and then eliminate conflicted relations to obtain a legitimate tree hierarchy. In contrast, our model is optimized over the full space of all legitimate taxonomies by taking the *structure operation* in account during Gibbs sampling.

When comparing to Bansal2014, our model with only word embedding-based features underperforms theirs. However, when introducing visual features, our performance is comparable (p-value = 0.058).Furthermore, if we discard visual features but add semantic features from [2], we achieve a slight improvement of 0.02 over Bansal2014 (p-value = 0.016), which is largely attributed to the incorporation of word embedding-based features that encode high-level linguistic regularity. Finally, if we enhance our full model with all semantic features from [2], our model outperforms theirs by a gap of 0.04 (p-value < 0.01), which justifies our intuition that perceptual semantics underneath visual contents are quite helpful.

5.1.3 Qualitative Analysis

In this section, we conduct qualitative studies to investigate *how* and *when* the visual information helps the taxonomy induction task.

Contributions of visual features. To evaluate the contribution of each part of the visual features to the final performance, we train our model jointly with textual features and different combinations of visual features, and report the ancestor-F_1 scores. As shown in Table 5.3. When incorporating the feature S-V1, the performance is substantially boosted by a large gap at all heights, showing that visual similarity between sibling nodes is a strong evidence for taxonomy induction. It is intuitively plausible, as it is highly likely that two specific categories share a common (and more general) parent category if similar visual contents are observed between them. Further, adding the PC-V1 feature gains us a better improvement than adding PC-V2, but both minor than S-V1.

Compared to that of siblings, the visual similarity between parents and children does not strongly holds all the time. For example, images of *Terrestrial animal* are only partially similar to those of *Feline*, because the former one contains the later one as a subset. Our feature captures this type of "contain" relation between parents and children by considering only the top-K images from the parent category that have highest probabilities under the Gaussian distribution of the child category. To see this, we vary K while keep all other settings, and plot the F_1 scores in Fig 5.1. We observe a trend that when we gradually increase K, the performance goes up until reaching some maximal; It then slightly drops (or oscillates) even when more images are available, which confirms with our feature design that only top images should be considered in parent-child visual similarity.

Overall, the three visual features complement each other, and achieve the highest performance when combined.

Visual representations. To investigate how the image representations affect the final performance, we compare the ancestor-F1 score when different pre-trained CNNs are used for visual feature extraction. Specifically, we employ both the CNN-128 model (128 dimensional feature with 15.6% top-5 error on ILSVRC12) and the VGG-16 model (4096 dimensional feature with 7.5% top-5 error) by [41], but only observe a slight improvement of 0.01 on the ancestor-F1 score for the later one.

S-V1	PC-V1	PC-V2	h = 4	h = 5	h = 6	h = 7
			0.58	0.41	0.36	0.30
✓			0.63	0.48	0.40	0.32
	✓		0.61	0.44	0.38	0.31
		✓	0.60	0.42	0.37	0.31
✓	✓		0.65	**0.52**	0.41	0.33
✓	✓	✓	**0.66**	**0.52**	**0.42**	**0.34**

Table 5.3: The performance when different combinations of visual features are enabled.

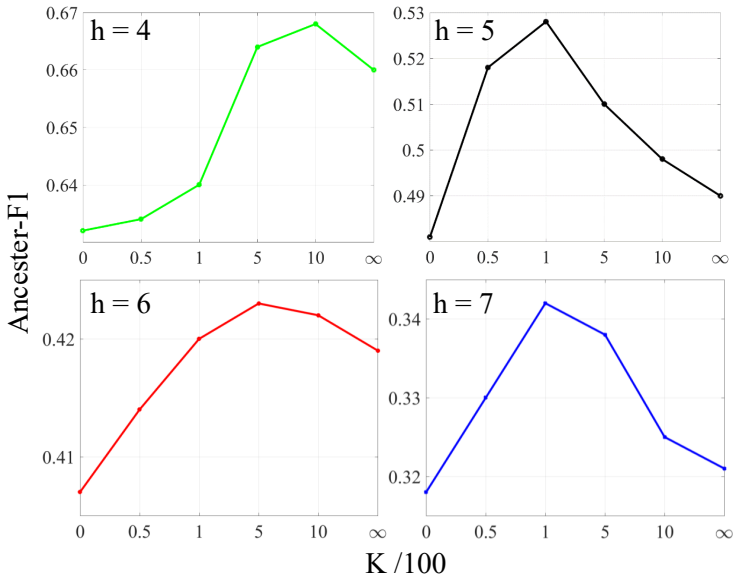


Figure 5.1: The Ancestor-F_1 scores changes over K (number of images used in the PC-V1 feature) at different heights. The values in the x-axis are $K/100$; $K = \infty$ means all images are used.

Relevance of textual and visual features v.s. depth of tree. Compared to [2], a major difference of our model is that different layers of the taxonomy correspond to different weights $\boldsymbol{w}_l$, while in [2] all layers share the same weights. Intuitively, introducing layer-wise $\boldsymbol{w}$ not only extends the model capacity, but also differentiates the importance of each feature at different layers. For example, the images of two specific categories, such as *shark* and *ray*, are very likely to be visually similar. However, when the taxonomy goes from bottom to up (specific to general), the visual similarity is gradually undermined — images of *fish* and *terrestrial animal* are not necessarily similar any more. Hence, it is necessary to privatize the weights $\boldsymbol{w}$ for different layers to capture such variations, i.e. the visual features become more and more evident from shallow to deep layers, while the textual counterparts, which capture more abstract concepts, relatively grow more indicative oppositely from specific to general.

To visualize the variations across layers, for each feature component, we fetch its corresponding block in $\boldsymbol{w}$ as V. Then, we average $|V|$ and observe how its values change with the layer depth h. For example, for the parent-child word-word relation feature, we first fetch its corre-

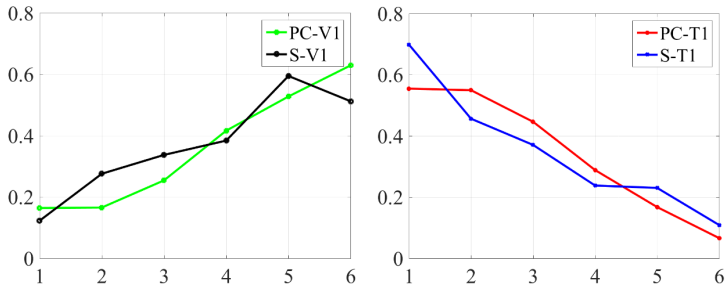


Figure 5.2: Normalized weights of each feature v.s. the layer depth.

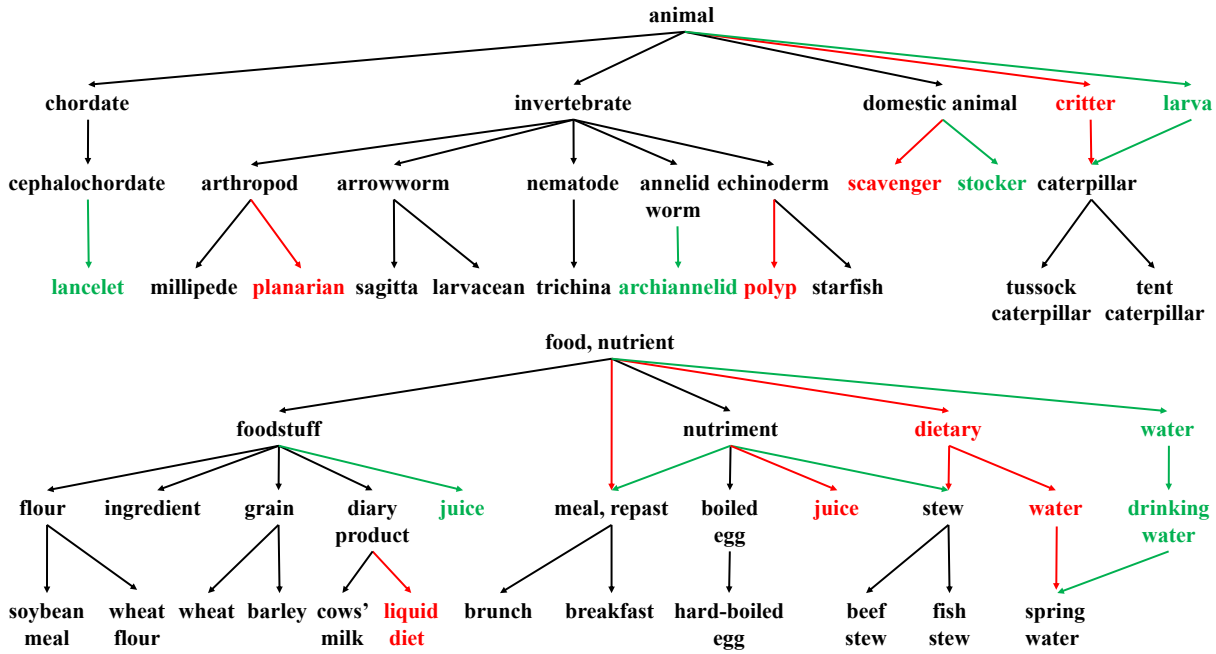


Figure 5.3: Excerpts of the prediction taxonomies, compared to the groundturth. Edges marked as red and green are false predictions and unpredicted groundtruth links, respectively.

sponding weights V from $\boldsymbol{w}$ as a 20×6 matrix, where 20 is the feature dimension and 6 is the number of layers. We then average its absolute values[3] in column and get a vector v with length 6. After ℓ_2 normalization, the magnitude of each entry in v directly reflects the relative importance of the feature as an evidence for taxonomy induction. Fig 5.2(b) plots how their magnitudes change with h for every feature component averaged on three train/test splits. It is noticeable that for both word-word relations (S-T1, PC-T1), their corresponding weights slightly decrease as h increases. On the contrary, the image-image relation features (S-V1, PC-V1) grows relatively more prominent. The results verify our conjecture that when the category hierarchy goes deeper into more specific classes, the visual similarity becomes relatively more indicative as an evidence for taxonomy induction.

[3]We take the absolute value because we only care about the relevance of the feature as an evidence for taxonomy induction, but note that the weight can either encourage (positive) or discourage (negative) connections of two nodes.

Visualizing results. Finally, we visualize some excerpts of our predicted taxonomies, as compared to the groundtruth in Fig 5.3.

5.2 Image Classification

We evaluate HD-CNN on CIFAR100 [29] and ImageNet [9]. HD-CNN is implemented on the widely deployed *Caffe* [25] software. The network is trained by back propagation [30]. We run all the testing experiments on a single NVIDIA Tesla K40c card.

5.2.1 CIFAR100

The CIFAR100 dataset consists of 100 classes of natural images. There are 50K training images and 10K testing images. We follow [20] to preprocess the dataset (e.g. global contrast normalization and ZCA whitening). Randomly cropped and flipped image patches of size 26×26 are used for training. We adopt a *NIN* network [4] with three stacked layers [33]. We denote it as CIFAR100-NIN which will be the HD-CNN building block. Fine category components share preceding layers from *conv1* to *pool1* which accounts for 6% of the total parameters and 29% of the total floating point operations. The remaining layers are used as independent layers. For building the category hierarchy, we pertained a taxonomy induction model using the ImageNet hierarchy and then use it to build the category hierarchy for CIFAR100 categories. Fine categories within the same coarse categories are visually more similar. We pretrain the rear layers of fine category components. The initial learning rate is 0.01 and it is decreased by a factor of 10 every 6K iterations. Fine-tuning is performed for 20K iterations with large mini-batches of size 256. The initial learning rate is 0.001 and is reduced by a factor of 10 once after 10K iterations. The model structure could be found in Appendix B.

For evaluation, we use 10-view testing [30]. We extract five 26×26 patches (the 4 corner patches and the center patch) as well as their horizontal reflections and average their predictions. The CIFAR100-NIN net obtains 35.27% testing error. Our HD-CNN achieves testing error of 32.62% which improves the building block net by 2.65%.

Shared layers. Use of shared layers makes both computational complexity and memory footprint of HD-CNN sublinear in the number of fine category classifiers when compared to the building block net. Our HD-CNN based on CIFAR100-NIN consumes less than three times as much memory as the building block net without parameter compression. We also want to investigate the impact of the use of shared layers on the classification error, memory footprint and the net execution time (Table 5.5). We build another HD-CNN where coarse category component and all fine category components use independent preceding layers initialized from a pretrained building block net. Under single-view testing where only a central cropping is used, we observe a minor error increase from 34.36% to 34.50%. But using shared layers dramatically reduces the memory footprint from 1356 MB to 459 MB and testing time from 2.43 seconds to 0.28 seconds.

Conditional execution. By varying the hyperparameter β, we can effectively affect the number of fine category components that will be executed. There is a trade-off between execution time

[4] `https://github.com/mavenlin/cuda-convnet/blob/master/NIN/cifar-100_def`

Table 5.4: 10-view testing errors on CIFAR100 dataset. Notation **CCC**=coarse category consistency.

Method	Error
Model averaging (2 CIFAR100-NIN nets)	35.13
DSN [31]	34.68
CIFAR100-NIN-double	34.26
dasNet [47]	33.78
Base: CIFAR100-NIN	35.27
HD-CNN, no finetuning	33.33
HD-CNN, finetuning	**32.62**
HD-CNN+CE+PC, finetuning	32.79

and classification error. A larger value of β leads to higher accuracy at the cost of executing more components for fine categorization. By enabling conditional executions with hyperparameter $\beta = 6$, we obtain a substantial 2.8X speed up with merely a minor increase in error from 34.36% to 34.57% (Table 5.5). The testing time of HD-CNN is about 2.5 times as much as that of the building block net.

Parameter compression. As fine category CNNs have independent layers from *conv2* to *cccp6*, we compress them and reduce the memory footprint from 447MB to 286MB with a minor increase in error from 34.57% to 34.73%.

Comparison with a strong baseline. As our HD-CNN memory footprint is about two times as much as the building block model (Table 5.5), it is necessary to compare a stronger baseline of similar complexity with HD-CNN. We adapt CIFAR100-NIN and double the number of filters in all convolutional layers which accordingly increases the memory footprint by three times. We denote it as CIFAR100-NIN-double and obtain error 34.26% which is 1.01% lower than that of the building block net but is 1.64% higher than that of HD-CNN.

Comparison with model averaging. HD-CNN is fundamentally different from model averaging [30]. In model averaging, all models are capable of classifying the full set of the categories and each one is trained independently. The main sources of their prediction differences are different initializations. In HD-CNN, each fine category classifier only excels at classifying a partial set of categories. To compare HD-CNN with model averaging, we independently train two CIFAR100-NIN networks and take their averaged prediction as the final prediction. We obtain an error of 35.13%, which is about 2.51% higher than that of HD-CNN (Table 5.4). Note that HD-CNN is orthogonal to the model averaging and an ensemble of HD-CNN networks can further improve the performance.

Coarse category consistency. To verify the effectiveness of coarse category consistency term in our loss function (4.3), we fine-tune a HD-CNN using the traditional multinomial logistic loss function. The testing error is 33.21%, which is 0.59% higher than that of a HD-CNN fine-tuned with coarse category consistency (Table 5.4).

Comparison with state-of-the-art. Our HD-CNN improves on the current two best methods [31] and [47] by 2.06% and 1.16% respectively and sets new state-of-the-art results on CIFAR100 (Table 5.4).

5.2.2 ImageNet 1000

The ILSVRC-2012 ImageNet dataset consists of about 1.2 million training images, $50,000$ validation images. To demonstrate the generality of HD-CNN, we experiment with two different building block nets. In both cases, HD-CNNs achieve significantly lower testing errors than the building block nets.

Network-In-Network Building Block Net

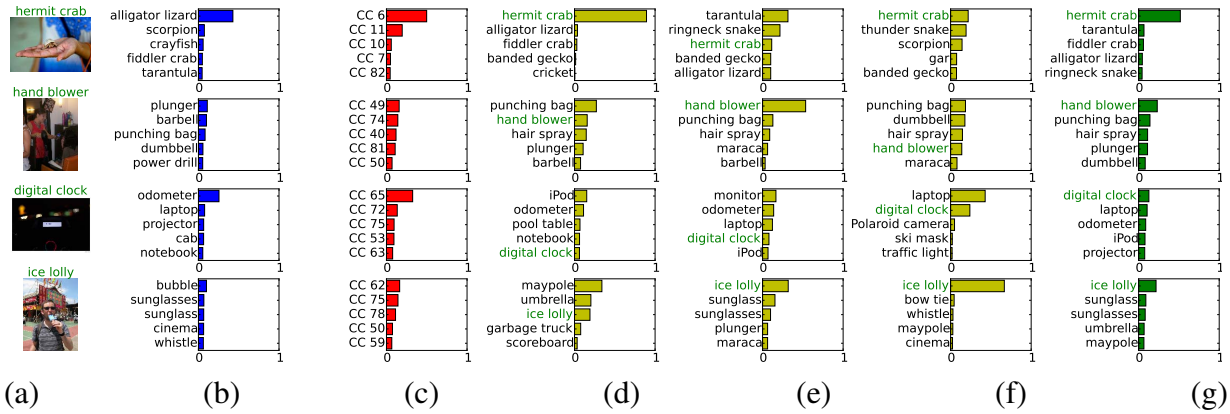


Figure 5.4: Case studies on ImageNet dataset. Each row represents a testing case. **Column (a)**: test image with ground truth label. **Column (b)**: top 5 guesses from the building block net ImageNet-NIN. **Column (c)**: top 5 Coarse Category (**CC**) probabilities. **Column (d)-(f)**: top 5 guesses made by the top 3 fine category CNN components. **Column (g)**: final top 5 guesses made by the HD-CNN. See text for details.

We choose a public 4-layer NIN net[5] as our first building block as it has greatly reduced number of parameters compared to AlexNet [30] but similar error rates. It is denoted as ImageNet-NIN. The model structure could be found in Appendix B. In HD-CNN, various components share preceding layers from *conv1* to *pool3* which account for 26% of the total parameters and 82% of the total floating point operations. We follow the training and testing protocols as in [30]. Original images are resized to 256×256. Randomly cropped and horizontally reflected 224×224 patches are used for training. At test time, the net makes a 10-view averaged prediction. We train ImageNet-NIN for 45 epochs. The top-1 and top-5 errors are 39.76% and 17.71%. To build the category hierarchy, we pretrained the taxonomy induction model on WordNet hierarchy (excluding those used in ImageNet) and induce a taxonomy of the ILSVRC 2012 categories with 89 intermediate nodes. Each fine category CNN is fine-tuned for 40K iterations while the initial

[5] `https://gist.github.com/mavenlin/d802a5849de39225bcc6`

learning rate 0.01 is decreased by a factor of 10 every 15K iterations. Fine-tuning the complete HD-CNN is not performed as the required mini-batch size is significantly higher than that for the building block net. Nevertheless, we still achieve top-1 and top-5 errors of 36.66% and 15.80% and improve the building block net by 3.1% and 1.91%, respectively (Table 5.6). The class-wise top-5 error improvement over the building block net is shown in Fig 5.5 left.

Table 5.5: Comparison of testing errors, memory footprint and testing time between building block nets and HD-CNNs on CIFAR100 and ImageNet datasets. Statistics are collected under *single-view* testing. Three building block nets CIFAR100-NIN, ImageNet-NIN and ImageNet-VGG-16-layer are used. The testing mini-batch size is 50. Notations: **SL**=Shared layers, **CE**=Conditional execution, **PC**=Parameter compression.

Model	top-1, top-5	Memory (MB)	Time (sec.)
Base:CIFAR100-NIN	37.29	188	0.04
HD-CNN w/o SL	34.50	1356	2.43
HD-CNN	**34.36**	459	0.28
HD-CNN+CE	34.57	447	0.10
HD-CNN+CE+PC	34.73	286	0.10
Base:ImageNet-NIN	41.52, 18.98	535	0.19
HD-CNN	**37.92**, **16.62**	3544	3.25
HD-CNN+CE	38.16, 16.75	3508	0.52
HD-CNN+CE+PC	38.39, 16.89	1712	0.53
Base:ImageNet-VGG-16-layer	32.30, 12.74	4134	1.04
HD-CNN+CE+PC	**31.34**, **12.26**	6863	5.28

Case studies We want to investigate how HD-CNN corrects the mistakes made by the building block net. In Fig 5.4, we collect four testing cases. In the first case, the building block net fails to predict the label of the tiny *hermit crab* in the top 5 guesses. In HD-CNN, two coarse categories $\#6$ and $\#11$ receive most of the coarse probability mass. The fine category component $\#6$ specializes in classifying crab breeds and strongly suggests the ground truth label. By combining the predictions from the top fine category classifiers, the HD-CNN predicts *hermit crab* as the most probable label. In the second case, the ImageNet-NIN confuses the ground truth *hand blower* with other objects of close shapes and appearances, such as *plunger* and *barbell*. For HD-CNN, the coarse category component is also not confident about which coarse category the object belongs to and thus assigns even probability mass to the top coarse categories. For the top 3 fine category classifiers, $\#74$ strongly predicts ground truth label while the other two $\#49$

and #40 rank the ground truth label at the 2nd and 4th place respectively. Overall, the HD-CNN ranks the ground truth label at the 1st place. This demonstrates HD-CNN needs to rely on multiple fine category classifiers to make correct predictions for difficult cases.

Conditional executions. By varying the hyperparameter β, we can control the number of fine category components that will be executed. There is a trade-off between execution time and classification error as shown in Fig 5.5 right. A larger value of β leads to lower error at the cost of more executed fine category components. By enabling conditional executions with hyperparameter $\beta = 8$, we obtain a substantial 6.3X speed up with merely a minor increase of single-view testing top-5 error from 16.62% to 16.75% (Table 5.5). With such speedup, the HD-CNN testing time is less than 3 times as much as that of the building block net.

Parameter compression. We compress independent layers *conv4* and *cccp7* as they account for 60% of the parameters in ImageNet-NIN. Their parameter matrices are of size 1024×3456 and 1024×1024 and we use compression hyperparameters $(s, k) = (3, 128)$ and $(s, k) = (2, 256)$. The compression factors are 4.8 and 2.7. The compression decreases the memory footprint from 3508MB to 1712MB and merely increases the top-5 error from 16.75% to 16.89% under single-view testing (Table 5.5).

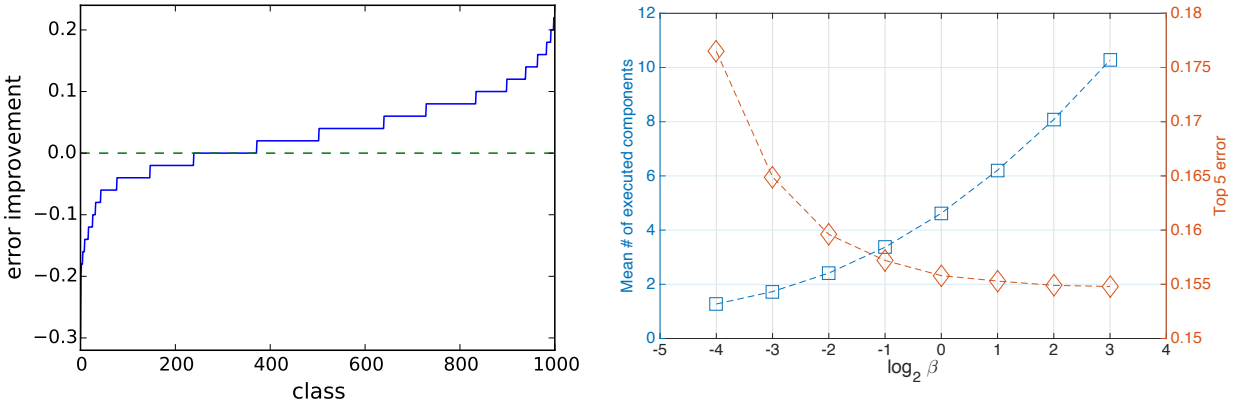


Figure 5.5: **Left**: Class-wise HD-CNN top-5 error improvement over the building block net. **Right**: Mean number of executed fine category classifiers and top-5 error against hyperparameter β on the ImageNet validation dataset.

Comparison with model averaging. As the HD-CNN memory footprint is about three times as much as the building block net, we independently train three ImageNet-NIN nets and average their predictions. We obtain top-5 error 17.11% which is 0.6% lower than the building block but is 1.31% higher than that of HD-CNN (Table 5.6).

VGG-16-layer Building Block Net

The second building block net we use is a 16-layer CNN from [41]. We denote it as ImageNet-VGG-16-layer[6]. The model structure could be found in Appendix B. The layers from *conv1_1*

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

Table 5.6: Comparisons of 10-view testing errors between ImageNet-NIN and HD-CNN. Notation **CC**=Coarse category.

Method	top-1, top-5
Base:ImageNet-NIN	39.76, 17.71
Model averaging (3 base nets)	38.54, 17.11
HD-CNN	**36.66, 15.80**
HD-CNN+CE+PC	36.88, 15.92

Table 5.7: Errors on ImageNet validation set.

Method	top-1, top-5
GoogLeNet,multi-crop [48]	N/A,7.9
VGG-19-layer, dense [41]	24.8,7.5
VGG-16-layer+VGG-19-layer,dense	24.0,7.1
Base:ImageNet-VGG-16-layer,dense	24.79,7.50
HD-CNN+PC,dense	**23.69,6.76**
HD-CNN+PC+CE,dense	23.88,6.87

to *pool4* are shared and they account for 5.6% of the total parameters and 90% of the total floating number operations. The remaining layers are used as independent layers in coarse and fine category classifiers. We follow the training and testing protocols as in [41]. For training, we first sample a size S from the range $[256, 512]$ and resize the image so that the length of short edge is S. Then a randomly cropped and flipped patch of size 224×224 is used for training. For testing, dense evaluation is performed on three scales $\{256, 384, 512\}$ and the averaged prediction is used as the final prediction. Please refer to [41] for more training and testing details. On ImageNet validation set, ImageNet-VGG-16-layer achieves top-1 and top-5 errors 24.79% and 7.50% respectively.

Similar to the previous NIN structure, we build a category hierarchy with 84 intermediate categories. We implement multi-GPU training on Caffe by exploiting data parallelism [41] and train the fine category classifiers on two NVIDIA Tesla K40c cards. The initial learning rate is 0.001 and it is decreased by a factor of 10 every 4K iterations. HD-CNN fine-tuning is not performed. Due to large memory footprint of the building block net (Table 5.5), the HD-CNN with 84 fine category classifiers cannot fit into the memory directly. Therefore, we compress the parameters in layers *fc6* and *fc7* as they account for over 85% of the parameters. Parameter matrices in *fc6* and *fc7* are of size 4096×25088 and 4096×4096. Their compression hyper-

parameters are $(s, k) = (14, 64)$ and $(s, k) = (4, 256)$. The compression factors are 29.9 and 8 respectively. The HD-CNN obtains top-1 and top-5 errors 23.69% and 6.76% on ImageNet validation set and improves over ImageNet-VGG-16-layer by 1.1% and 0.74% respectively.

Comparison with state-of-the-art. Currently, the two best nets on ImageNet dataset are GoogLeNet [48] (Table 5.7) and VGG 19-layer network [41]. Using multi-scale multi-crop testing, a single GoogLeNet net achieves top-5 error 7.9%. With multi-scale dense evaluation, a single VGG 19-layer net obtains top-1 and top-5 errors 24.8% and 7.5% and improves top-5 error of GoogLeNet by 0.4%. Our HD-CNN decreases top-1 and top-5 errors of VGG 19-layer net by 1.11% and 0.74% respectively. Furthermore, HD-CNN slightly outperforms the results of averaging the predictions from VGG-16-layer and VGG-19-layer nets.

Chapter 6

Conclusion

Existing deep convolutional neural networks (CNN) are trained as flat N-way classifiers and few efforts have been made to leverage the hierarchical structure of categories. This thesis builds on the intuition not all categories are equally difficult to distinguish. Based on this, we first study the problem of automatically inducing semantically meaningful category hierarchies from multi-modal data. We propose a probabilistic Bayesian model which leverages distributed representations for images and words [59]. We compare our model and features to previous ones on two different tasks using the ImageNet hierarchies, and demonstrate superior performance of our model, and the effectiveness of exploiting visual contents for taxonomy induction. Given the pre-build category hierarchies, we then combine the category hierarchy with deep CNN and introduce Hierarchical Deep CNN (HD-CNN) [55]. HD-CNN separates easy classes in coarse category classifier while distinguishing the most difficult classes in fine category classifiers. HD-CNN is trained by component-wise pre-training, followed by a global finetuning with a multinomial logistic loss regularized by a temporal sparsity penalty. We demonstrated that HD-CNN is a flexible deep CNN architecture to improve over existing deep CNN models. We showed this empirically on both CIFAR-100 and Image-Net datasets using three different building block nets. As part of future work, we plan to extend HD-CNN architectures to those with more than 2 hierarchical levels and also verify our empirical results in a theoretical framework.

Appendix A

Taxonomy Induction

Appendix A is organized as follows. In section A.1, we provide more details about our proposed model, including an illustration of our model (section A.1.1), the derivation of the Gibbs sampler (section A.1.2) and the gradient descent algorithm (section A.1.3). Section A.2 gives more details on the feature extraction. As supplementary to the paper, section A.3 discloses more implementation details, including the text data processing (section A.3.1) and the implementation efficiency (section A.3.2).

A.1 Model Derivation

A.1.1 Illustration of Our Model

Fig A.1 illustrates the intuition of our model. Each parent-children group (the green boxes in Fig A.1) corresponds to a consistency term which encodes the semantic closeness of all parent-child pairs and sibling pairs within that group. The model encourages the local semantic consistency by factorizing consistency terms of all parent-children groups present in the taxonomy.

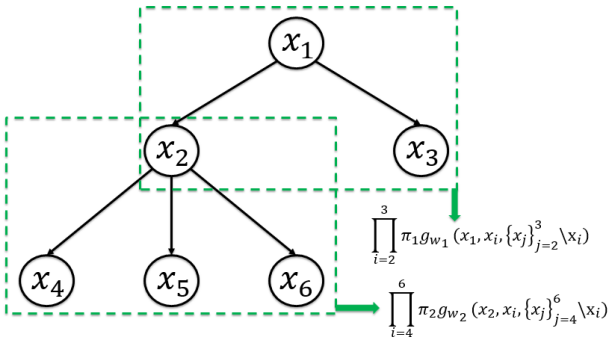


Figure A.1: An illustration of our model, which encourages *local semantic consistency.*

A.1.2 Gibbs Sampling

The probability of a configuration $\boldsymbol{z}$ is defined as

$$p_w(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{\alpha}) \propto \prod_n \Gamma(q_n + \alpha_n) \prod_{x_{n'} \in \boldsymbol{c}_n} g_w(x_n, x_{n'}, \boldsymbol{c}_n \backslash x_{n'}) \cdot \mathbf{1}(\boldsymbol{z}). \tag{A.1}$$

To sample the parent index z_n of category x_n, conditioned on the structure of the rest nodes, we have

$$\begin{aligned} p(z_n = m|\boldsymbol{z}\backslash z_n, \cdot) \propto & \prod_{t \neq m} \Gamma(q_t^{-n} + \alpha_t) \prod_{x_{n'} \in \boldsymbol{c}_t \backslash x_n} g_w(x_t, x_{n'}, \boldsymbol{c}_t \backslash x_n) \\ & \cdot \Gamma(q_m^{-n} + 1 + \alpha_t) \prod_{x_{n'} \in \boldsymbol{c}_m \cup \{x_n\}} g_w(x_m, x_{n'}, \boldsymbol{c}_m \cup \{x_n\}) \\ & \cdot \mathbf{1}(z_n = m, \boldsymbol{z}\backslash z_n), \end{aligned}$$

where q_m is the number of children of category m; the superscript $-n$ denotes the number excluding x_n. To simplify the sampling procedure, we divide $p(z_n = m|\boldsymbol{z}\backslash z_n, \cdot)$ with the "likelihood" of the whole structure excluding x_n, i.e.

$$p(\boldsymbol{z}\backslash z_n|\cdot) \propto \prod_t \Gamma(q_t^{-n} + \alpha_t) \prod_{x_{n'} \in \boldsymbol{c}_t \backslash x_n} g_w(x_t, x_{n'}, \boldsymbol{c}_t \backslash x_n),$$

which is independent of the value of z_n. This leads to our sampling formula as in Eq 3 of the paper

$$p(z_n = m|\boldsymbol{z}\backslash z_n, \cdot) \propto \mathbf{1}(z_n = m, \boldsymbol{z}\backslash z_n) \cdot \left(q_m^{-n} + \alpha_m\right) \cdot \frac{\prod_{x_{n'} \in \boldsymbol{c}_m \cup \{x_n\}} g_w(x_m, x_{n'}, \boldsymbol{c}_m \cup \{x_n\})}{\prod_{x_{n'} \in \boldsymbol{c}_m \backslash x_n} g_w(x_m, x_{n'}, \boldsymbol{c}_m \backslash x_n)}.$$

A.1.3 Gradient Descent

Our training algorithm updates $\boldsymbol{w}$ through maximum likelihood estimation. As we employ an exponential form for the local consistency function $g_w(\cdot) = \exp(\boldsymbol{w}_{d(\cdot)}^\top \boldsymbol{f})$ (where we have simplified the notations to avoid cluttering the notations), the model defined in Eq A.1 can be seen to have a loglinear form with respect to $\boldsymbol{w}_{d(\cdot)}$. For the weights $\boldsymbol{w}_l$ of layer l, all the terms in Eq A.1, except $g_{w_l}(\cdot)$ for the nodes in the lth layer, are independent of $\boldsymbol{w}_l$, and we denote them as $C_{\boldsymbol{z}}$. Thus we have

$$\log p_w(\tilde{\boldsymbol{z}}|\boldsymbol{x}, \boldsymbol{\alpha}) = \log \frac{C_{\tilde{\boldsymbol{z}}} \exp\{\boldsymbol{w}_l^\top \boldsymbol{f}_{\tilde{\boldsymbol{z}},l}\}}{\sum_{\boldsymbol{z}} C_{\boldsymbol{z}} \exp\{\boldsymbol{w}_l^\top \boldsymbol{f}_{\boldsymbol{z},l}\}},$$

where $\tilde{\boldsymbol{z}}$ is the gold taxonomy from training data, and $\boldsymbol{f}_{\boldsymbol{z},l} = \sum_{n:\ell(x_n)=l} \boldsymbol{f}(x_{z_n}, x_n, \boldsymbol{c}_n \backslash x_n)$ is the sum over the node feature vectors of layer l in taxonomy $\boldsymbol{z}$. Take derivative with respect to $\boldsymbol{w}_l$

we obtain the gradient

$$\begin{aligned}\delta \boldsymbol{w}_l &= \boldsymbol{f}_{\tilde{\boldsymbol{z}},l} - \sum_{\boldsymbol{z}} \frac{C_{\boldsymbol{z}} \exp\{\boldsymbol{w}_l^\top \boldsymbol{f}_{\boldsymbol{z},l}\}}{\sum_{\boldsymbol{z}'} C_{\boldsymbol{z}'} \exp\{\boldsymbol{w}_l^\top \boldsymbol{f}_{\boldsymbol{z}',l}\}} \boldsymbol{f}_{\boldsymbol{z},l} + \log C_{\tilde{\boldsymbol{z}}} \\ &= \sum_{n:\ell(x_n)=l} \{\boldsymbol{f}(x_{\tilde{z}_n}, x_n, \tilde{\boldsymbol{c}}_n \backslash x_n) - \mathbb{E}_p[\boldsymbol{f}(x_{z_n}, x_n, \boldsymbol{c}_n \backslash x_n)]\} \\ &\quad + \log C_{\tilde{\boldsymbol{z}}}.\end{aligned}$$

The expectation is approximated by collecting a set of samples using the Gibbs sampler as described above, and then averaging over them.

A.2 Feature Extraction

In this section, we further elaborate the procedures how to extract the features, as complementary to the descriptions in section 4 of our paper.

A.2.1 Parent-child Word-word Relation Feature (PC-T1)

Following [15], we first learn C^{tt} word-word projection matrices $\{\boldsymbol{\Phi}_c^{tt}\}_{c=1}^{C^{tt}}$ using all pairwise relations from the training hierarchies, where C^{tt} is the number of clusters chosen by cross validation [15], and the superscript "tt" denotes text to text (word to word). Then, we compute the distance $d = \|\boldsymbol{\Phi}_c^{tt} \boldsymbol{v}_{t_{n'}} - \boldsymbol{v}_{t_n}\|_2$, where $\boldsymbol{v}_{t_{n'}}$ and $\boldsymbol{v}_{t_n}$ are the word embedding of the child and parent category, respectively, and $\boldsymbol{\Phi}_c^{tt}$ is the projection matrix for the pair $\{\boldsymbol{v}_{t_{n'}}, \boldsymbol{v}_{t_n}\}$, whose index $c \in \{1, 2, \ldots, C^{tt}\}$ is determined by cluster assignment of the pair (see more details in [15]). Then, we quantize d into a histogram lying on $[u, v]$ with k bins, thus produce a k-dimensional vector as the feature vector.

A.2.2 Parent-child Image-word Relation Feature (PC-V2)

We already elaborate how PC-V2 feature is extracted in the paper. Here we provide more detailed implementation notes.

We firstly $\ell 2$-normalize the mean image vector $\bar{\boldsymbol{v}}_{\boldsymbol{i}_{n'}}$ of the child category $x_{n'}$. We then learn the image-word projection matrices $\{\boldsymbol{\Phi}_c^{it}\}_{c=1}^{C^{it}}$ to project $\bar{\boldsymbol{v}}_{\boldsymbol{i}_{n'}}$ to $\boldsymbol{v}_{t_n}$, where $\boldsymbol{v}_{t_n}$ is the word embedding of the parent node x_n, C^{it} is the number of clusters and "it" denotes image to word. Then we use the same quantization strategy to extract a k-dimensional vector as the parent-child image-word relation feature. It is noticeable that for category without $\bar{\boldsymbol{v}}_{\boldsymbol{i}_{n'}}$ (without images), we produce a k-dimensional zero vector instead. As each part of the feature is independent with the other, when multiplying with the weights $\boldsymbol{w}$, the counterpart in $\boldsymbol{w}$ is automatically cancelled by multiplying zeroes, contributing nothing to the local semantic consistency term.

A.2.3 Parent-child Image-image Relation Feature (PC-V1)

As described in section 4.1 of the paper, for image-image relation, we compute the *vissim* which is defined as

$$vissim(x_n, x_m) = \frac{\mathcal{N}(\overline{\boldsymbol{v}}_{\boldsymbol{i}_m}; \overline{\boldsymbol{v}}_{\boldsymbol{i}_n}, \Sigma_n) + \mathcal{N}(\overline{\boldsymbol{v}}_{\boldsymbol{i}_n}; \overline{\boldsymbol{v}}_{\boldsymbol{i}_m}, \Sigma_m)}{2}$$

as the visual similarity between two categories, where the Gaussian of the child category is estimated using all images in that category, yet the Gaussian of the parent category is fit using only the top K images with highest probabilities under the distribution of the child category. This usually results in a relatively small value which is not in the same scale with other features. Hence, we further transform the *vissim* into log scale and rescale it using:

$$d = \frac{s}{\log(vissim(x_n, x_m))}$$

so that a smaller value of d indicates stronger similarity. Then, the visual distance d is quantized into a histogram and a d-dimensional vector is produced as the feature. For nodes without images, a zero vector will be used instead.

A.2.4 Siblings Image-image Relation Feature (S-V1)

Similar to the parent-child image-image relation feature, we first compute pairwise visual similarity between each pair of siblings (but using all images in that category when fitting the Gaussian). Then, their mean value is quantized as a feature vector.

A.2.5 Siblings Word-word Relation Feature (S-T1)

Based on the observation that word vectors with a smaller distance are usually semantically closer, we first compute the *cosine distance* between the word embedding of each pair of siblings, then quantize the mean distance into a histogram to form the feature vector, where the histogram range $[u, v]$ is determined empirically for each feature.

For all features mentioned above, We set the number of bins k to 20 by cross validation.

A.2.6 Surface Features

For two categories x_n and $x_{n'}$ with category name t_n and $t_{n'}$ respectively, we list the surface features we used as below [2].

- **Ends with**, i.e. whether $t_{n'}$ ends with t_n (e.g. $catshark$ is a sub-category of $shark$).
- **Contains**, i.e. whether $t_{n'}$ contains t_n.
- **Capitalization**, whether $t_{n'}$ and t_n are capitalized. Intuitively, if $t_{n'}$ is capitalized while t_n is not, the probability of x_n being a parent of $x_{n'}$ tends to be low.
- **Suffix match**, whether $t_{n'}$ and t_n share a common suffix with length k, where k is ranged as $k = 1, \ldots, 7$.
- **LCS**, the *longest continuous common substring* of $t_{n'}$ and t_n. The value $2\frac{|LCS|}{|t_{n'}|+|t_n|}$ is quantized into a histogram as a feature vector.

- **Length difference**, i.e. the indicator features for rounded-off and binned values of $2\frac{|t_{n'}|-|t_n|}{|t_{n'}|+|t_n|}$.

A.3 Implementation Details

A.3.1 Word Embedding Training

Preprocessing. We first download the entire structure of ImageNet2011 Release. Every category in ImageNet is denoted as a *synset*, and every synset is jointly described by multiple terms [1]. To match every synset against the corpus for word embedding training, we match every descriptive term of the synset, and discard synsets that are not found or those which rarely exist in the corpus.

We implement a tri-tree in order to detect synset terms in the large corpus more efficiently. The time complexity for phrase matching is $O(dn)$, where d is the depth of the tri-tree and n is the number of tokens in the corpus.

Training. Once we determined the mappings between synsets and words in the training corpus, we re-scan the whole corpus and replace the matched words (or phrases) with a unique string *__Synset_id*, where *id* is the ID of the query synset. We use the hierarchical softmax training algorithm [37] to train 15 iterations for 200-dimensional word embedding. Synsets occurring fewer than 5 times in the corpus are removed.

A.3.2 Efficiency

Features described above can be classified as *pairwise features* or *group-wise features*. Specifically, all pairwise features, including the *parent-child word-word relation feature (PC-T1)*, *parent-child image-word relation feature (PC-V2)*, *parent-child image-image relation feature (PC-V1)* and *surface features*, can be obtained in $O(1)$ time by pre-computation. While, the group-wise features, including the *sibling image-image relation feature (S-V1)* and *siblings word-word relation feature (S-T1)*, can be obtained in linear time.

[1] A term could be a single word (e.g. apple), or a phrase represented by multiple words (e.g. thresher shark).

Appendix B

Hierarchical Deep Convolutional Neural Networks (HDCNN)

B.1 CIFAR100 Dataset

B.1.1 HD-CNN Based on CIFAR100-NIN net

The instance of HD-CNN we use for CIFAR100 dataset is built upon a building block net CIFAR100-NIN. The layer configurations in CIFAR100-NIN are listed in Table B.1. The architectures of both CIFAR100-NIN and the corresponding HD-CNN are illustrated in Figure B.1. We use the preceding layers from *conv1* to *pool1* as shared layers.

B.2 ImageNet 1000-class Dataset

We experiment with two different building block nets on ImageNet dataset, namely ImageNet-NIN and ImageNet-VGG-16-layer.

B.2.1 HD-CNN based on ImageNet-NIN

The layer configurations of the building block net ImageNet-NIN are listed in Table B.2. The architectures of ImageNet-NIN and the corresponding HD-CNN are shown in Figure B.2. The preceding layers from *conv1* to *pool3* are shared in HD-CNN.

B.2.2 HD-CNN based on ImageNet-VGG-16-layer

The layer configurations of the building block net ImageNet-VGG-16-layer are listed in Table B.3. The architectures of ImageNet-VGG-16-layer and the corresponding HD-CNN are shown in Figure B.3. Layers from *conv1_1* to *pool4* are shared within HD-CNN.

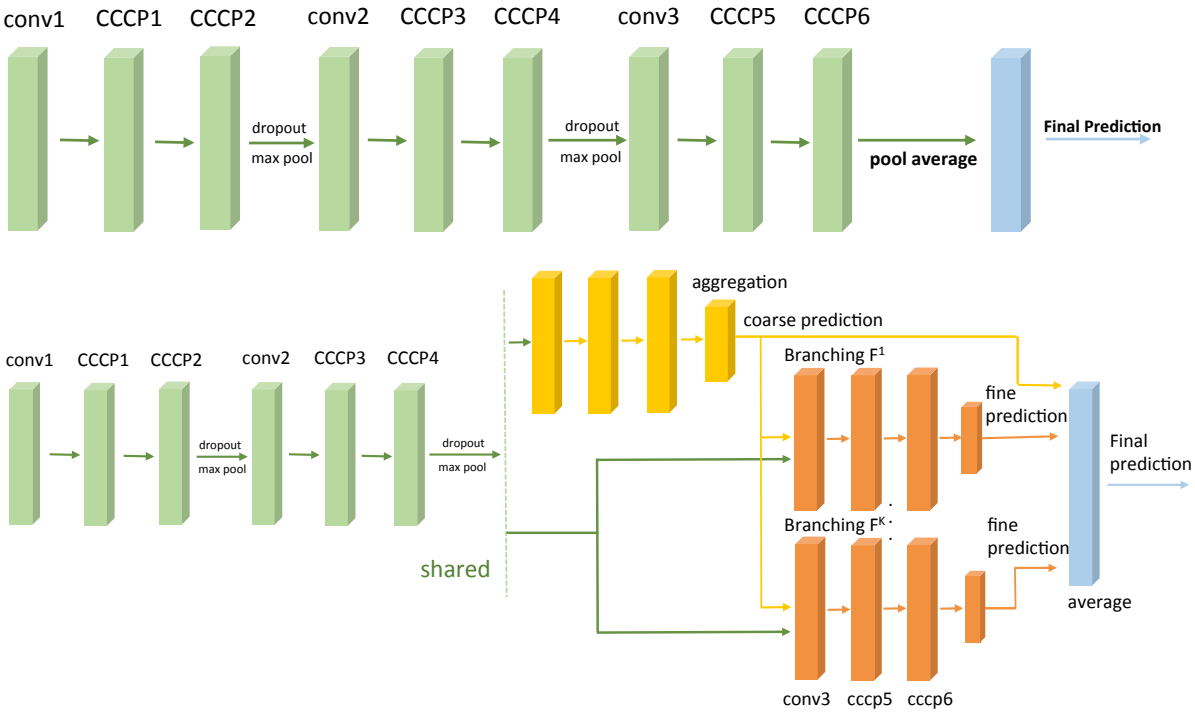


Figure B.1: **Top**: CIFAR100-NIN network. **Bottom**: HD-CNN network using CIFAR100-NIN building block.

LAY	conv1	cccp1	cccp2	pool1	conv2	cccp3	cccp4	pool2	conv3	cccp5	cccp6	pool3	prob
CFG	192,5,5	160,1,1	96,1,1	3,3,2 MAX	192, 5,5	192,1,1	192,1,1	3,3,2 MAX	192,3,3	192,1,1	100,1,1	6,6,1 AVG	SMAX
ACT		ReLU	ReLU			ReLU	ReLU			ReLU	ReLU		
PAR #	1.4e+4	3.1e+4	1.5e+4		4.6e+5	3.7e+4	3.7e+4		3.3e+5	3.7e+4	1.9e+4		
PAR %	1.5	3.1	1.6		46.9	3.8	3.8		33.8	3.8	2.0		
FLOP #	9.7e+6	2.1e+7	1e+7		7.8e+7	6.2e+6	6.2e+6		1.2e+7	1.3e+6	6.9e+5		
FLOP %	6.7	14.3	7.2		53.6	4.3	4.3		8.2	0.9	0.5		

Table B.1: CIFAR100-NIN network. The configuration of convolutional layer is denoted as (filter number, filter height, filter width). The configuration of pooling layer is denoted as (pooling height,pooling width, stride). Notations: **LAY**=Layer. **CFG**=Configuration. **ACT**=Activation. **PAR #**=Parameter number. **PAR %**=Parameter percentage. **FLOP #**=FLoating-point OPerations. **FLOP %**=FLoating-point OPeration percentage. **SMAX**=SOFTMAX.

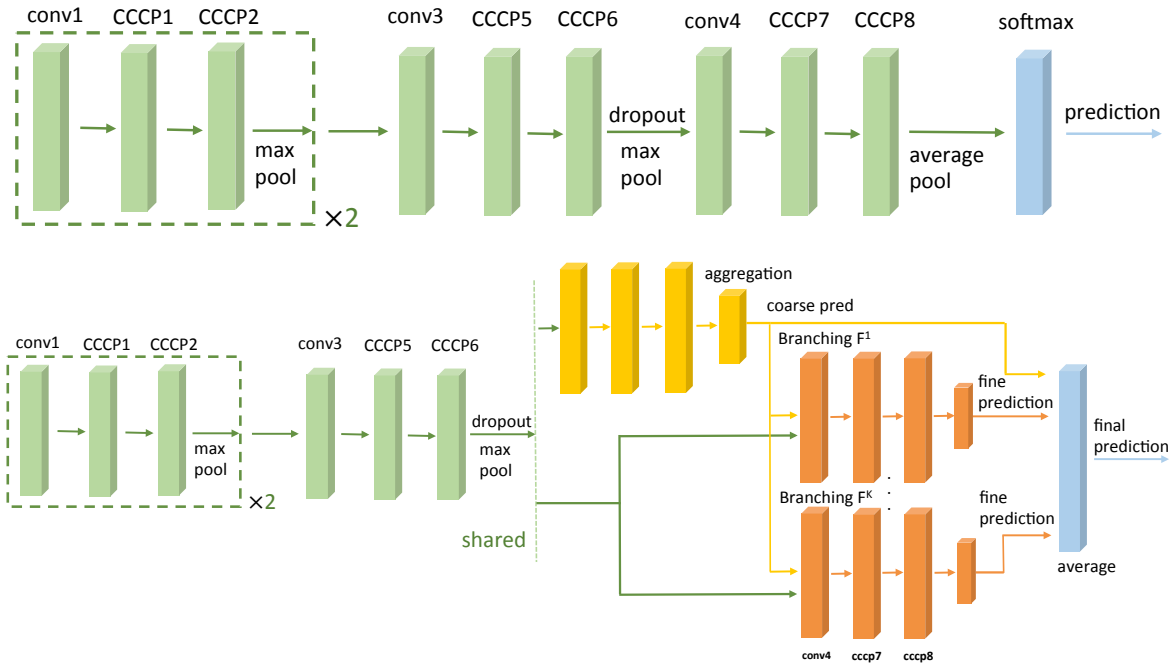


Figure B.2: **Top**: ImageNet-NIN network. **Bottom**: HD-CNN network using ImageNet-NIN building block.

LAY	conv1	cccp1	cccp2	pool0	conv2	cccp3	cccp4	pool2	conv3	cccp5	cccp6	pool3	conv4	cccp7	cccp8	pool4	prob
CFG	96,11,11	96,1,1	96,1,1	3,3,2 MAX	256,5,5	256,1,1	256,1,1	3,3,2 MAX	384,3,3	384,1,1	384,1,1	3,3,2 MAX	1024,3,3	1024,1,1	1000,1,1	6,6,1 AVG	SMAX
ACT	ReLU	ReLU	ReLU		ReLU	ReLU	ReLU		ReLU	ReLU	ReLU		ReLU	ReLU	ReLU		
PAR #	3.5e+4	9.2e+3	9.2e+3		6.1e+5	6.6e+4	6.6e+4		8.9e+5	1.5e+5	1.5e+5		3.5e+6	1.1e+6	1.1e+6		
PAR %	0.5	0.1	0.1		8.1	0.9	0.9		11.7	1.9	1.9		46.6	13.8	13.5		
FLOP #	1e+8	2.7e+7	2.7e+7		4.5e+8	4.8e+7	4.8e+7		1.5e+8	2.5e+7	2.5e+7		1.3e+8	3.8e+7	3.8e+7		
FLOP %	9.2	2.4	2.4		40.7	4.3	4.3		13.6	2.3	2.3		11.6	3.4	3.4		

Table B.2: ImageNet-NIN network.

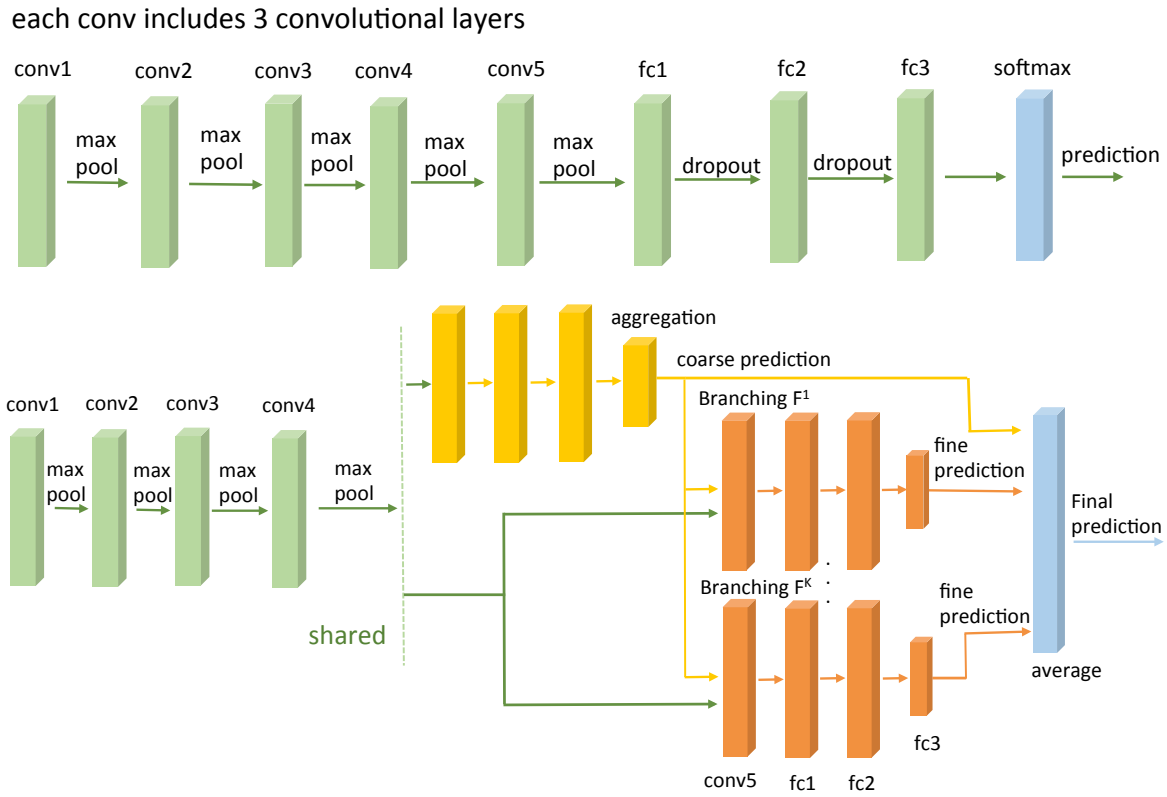


Figure B.3: **Top**: ImageNet-VGG-16-layer network. **Bottom**: HD-CNN network using ImageNet-VGG-16-layer building block.

LAY	conv 1_1	conv 1_2	pool1	conv 2_1	conv2_2	pool2	conv 3_1	conv 3_{2,3}	pool3	conv 4_1	conv 4_{2,3}	pool4	conv 5_{1,2,3}	pool5	fc6	fc7	fc8	prob
CFG	64, 3,3	64, 3,3	2,2,2 MAX	128, 3,3	128, 3,3	2,2,2 MAX	256,3,	3256, 3,3	2,2,2 MAX	512, 3,3	512, 3,3	2,2,2 MAX	512,3,3	2,2,2 MAX	4096	4096	1000	SMAX
ACT	ReLU	ReLU		ReLU	ReLU		ReLU	ReLU		ReLU	ReLU		ReLU		ReLU	ReLU		
PAR #	1.7e3	3.7e4		7.4e4	1.5e5		3.0e5	5.9e5		1.2e6	2.4e6		2.4e6		1.0e8	1.7e7	4.1e6	
PAR %	0.01	0.03		0.1	0.1		0.2	0.4		0.9	1.7		1.7		74.3	12.1	3.0	
FLOP #	8.7e7	1.9e9		9.3e8	1.9e9		9.3e8	1.9e9		9.3e8	1.9e9		4.6e8		1.0e8	1.7e7	4.1e6	
FLOP %	0.6	12.0		6.0	12.0		6.0	12.0		6.0	12.0		3.0		0.7	0.11	0.1	

Table B.3: ImageNet-VGG-16-layer network. For clarity, adjacent layers with the same configuration are merged, such as layers *conv3_2* and *conv3_3*.

Bibliography

[1] Hichem Bannour and Céline Hudelot. Hierarchical image annotation using semantic hierarchies. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, 2012. 2.3

[2] Mohit Bansal, David Burkett, Gerard de Melo, and Dan Klein. Structured learning for taxonomy induction with belief propagation. 2014. (document), 1, 2.1, 1, 3.2, 3.3.2, 5.1, 5.1.2, 5.1.2, 5.2, 5.1.2, 2, 5.1.3, A.2.6

[3] Evgeniy Bart, Ian Porteous, Pietro Perona, and Max Welling. Unsupervised learning of visual taxonomies. In *CVPR*, 2008. 2.1

[4] Samy Bengio, Jason Weston, and David Grangier. Label embedding trees for large multi-class tasks. In *NIPS*, 2010. 2.3

[5] Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. Learning structured embeddings of knowledge bases. In *Conference on Artificial Intelligence*, number EPFL-CONF-192344, 2011. 2.1

[6] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013. 5.1.1

[7] Xinlei Chen, Abhinav Shrivastava, and Abhinav Gupta. Neil: Extracting visual knowledge from web data. In *CVPR*, 2013. 1, 2.1

[8] Yoeng-Jin Chu and Tseng-Hong Liu. On shortest arborescence of a directed graph. *Scientia Sinica*, 1965. 3.2

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 5.1.1, 5.2

[10] Jia Deng, Sanjeev Satheesh, Alexander C Berg, and Fei Li. Fast and balanced: Efficient label tree learning for large scale object recognition. In *Advances in Neural Information Processing Systems*, 2011. 2.3

[11] Jia Deng, Jonathan Krause, Alexander C Berg, and Li Fei-Fei. Hedging your bets: Optimizing accuracy-specificity trade-offs in large scale visual recognition. In *CVPR*, 2012. 2.3

[12] Jia Deng, Nan Ding, Yangqing Jia, Andrea Frome, Kevin Murphy, Samy Bengio, Yuan Li, Hartmut Neven, and Hartwig Adam. Large-scale object classification using label relation graphs. In *ECCV*. 2014. 1, 2.3

[13] Clément Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013. 2.2

[14] Rob Fergus, Hector Bernal, Yair Weiss, and Antonio Torralba. Semantic label sharing for learning with many categories. In *ECCV*. 2010. 1

[15] Ruiji Fu, Jiang Guo, Bing Qin, Wanxiang Che, Haifeng Wang, and Ting Liu. Learning semantic hierarchies via word embeddings. In *ACL*, 2014. (document), 2.1, 3.3.1, 3.3.2, 5.1, 5.1.2, 5.2, 5.1.2, 2, A.2.1

[16] Chuang Gan, Yi Yang, Linchao Zhu, Deli Zhao, and Yueting Zhuang. Recognizing an action using its name: A knowledge-based approach. *International Journal of Computer Vision*, pages 1–17. 2.1

[17] Chuang Gan, Ming Lin, Yi Yang, Yueting Zhuang, and Alexander G Hauptmann. Exploring semantic inter-class relationships (SIR) for zero-shot action recognition. In *AAAI*, 2015. 2.1

[18] Tianshi Gao and Daphne Koller. Discriminative learning of relaxed hierarchy for large-scale visual recognition. In *ICCV*, 2011. 2.3

[19] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 2.2

[20] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In *ICML*, 2013. 2.2, 5.2.1

[21] Gregory Griffin and Pietro Perona. Learning and using taxonomies for fast visual categorization. In *CVPR*, 2008. 2.1, 2.3

[22] Lushan Han, Abhay Kashyap, Tim Finin, James Mayfield, and Jonathan Weese. Umbc ebiquity-core: Semantic textual similarity systems. *Atlanta, Georgia, USA*, 2013. 5.1.1

[23] K He, X Zhang, S Ren, and J Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*. 2014. 2.2

[24] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2011. 4.3

[25] Yangqing Jia. Caffe: An open source convolutional architecture for fast feature embedding, 2013. 5.2

[26] Yangqing Jia, Joshua T Abbott, Joseph Austerweil, Thomas Griffiths, and Trevor Darrell. Visual concept learning: Combining machine vision and bayesian generalization on concept hierarchies. In *NIPS*, 2013. 2.3

[27] Douwe Kiela, Laura Rimell, Ivan Vulic, and Stephen Clark. Exploiting image generality for lexical entailment detection. In *ACL*, 2015. 2.1

[28] Zornitsa Kozareva and Eduard Hovy. A semi-supervised method to learn and construct taxonomies using the web. In *EMNLP*, 2010. 2.1, 5.1.2

[29] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep*, 2009. 1, 5.2

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep

convolutional neural networks. In *NIPS*, 2012. 1, 2.2, 5.2, 5.2.1, 5.2.1, 5.2.2

[31] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. *arXiv preprint arXiv:1409.5185*, 2014. 5.4, 5.2.1

[32] Li-Jia Li, Chong Wang, Yongwhan Lim, David M Blei, and Li Fei-Fei. Building and using a semantivisual image hierarchy. In *CVPR*, 2010. 2.3

[33] M. Lin, Q. Chen, and S. Yan. Network in network. *CoRR*, 2013. 2.2, 5.2.1

[34] Baoyuan Liu, Fereshteh Sadeghi, Marshall Tappen, Ohad Shamir, and Ce Liu. Probabilistic label trees for efficient large scale image classification. In *CVPR*, 2013. 1, 2.3

[35] Marcin Marszalek and Cordelia Schmid. Semantic hierarchies for visual object recognition. In *CVPR*, 2007. 2.3

[36] Marcin Marszałek and Cordelia Schmid. Constructing category hierarchies for visual recognition. In *ECCV*. 2008. 2.1, 2.3

[37] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013. 1, 2.1, 3.3, 3.3.1, 5.1.1, A.3.1

[38] Roberto Navigli, Paola Velardi, and Stefano Faralli. A graph-based algorithm for inducing lexical taxonomies from scratch. In *IJCAI*, 2011. 2.1, 5.1.2

[39] Marcus Rohrbach, Michael Stark, György Szarvas, Iryna Gurevych, and Bernt Schiele. What helps where–and why? semantic relatedness for knowledge transfer. In *CVPR*, 2010. 2.1

[40] Ruslan Salakhutdinov, Antonio Torralba, and Josh Tenenbaum. Learning to share visual appearance for multiclass object detection. In *CVPR*, 2011. 2.3

[41] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1, 5.1.1, 5.1.3, 5.2.2, 5.7

[42] Josef Sivic, Bryan C Russell, Andrew Zisserman, William T Freeman, and Alexei A Efros. Unsupervised discovery of visual object class hierarchies. In *CVPR*, 2008. 2.3

[43] Rion Snow, Daniel Jurafsky, and Andrew Y Ng. Semantic taxonomy induction from heterogenous evidence. In *ACL*, 2006. 2.1

[44] Richard Socher, Milind Ganjoo, Christopher D Manning, and Andrew Ng. Zero-shot learning through cross-modal transfer. In *Advances in neural information processing systems*, pages 935–943, 2013. 2.1

[45] Jost Tobias Springenberg and Martin Riedmiller. Improving deep neural networks with probabilistic maxout units. *arXiv preprint arXiv:1312.6116*, 2013. 2.2

[46] Nitish Srivastava and Ruslan Salakhutdinov. Discriminative transfer learning with tree-based priors. In *NIPS*, 2013. 1, 2.3

[47] Marijn F Stollenga, Jonathan Masci, Faustino Gomez, and Jürgen Schmidhuber. Deep networks with internal selective attention through feedback connections. In *NIPS*, 2014. 5.4, 5.2.1

[48] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir

Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014. 5.7, 5.2.2

[49] Anne-Marie Tousch, Stéphane Herbin, and Jean-Yves Audibert. Semantic hierarchies for image annotation: A survey. *Pattern Recognition*, 2012. 2.3

[50] Luu Anh Tuan, Jung-jae Kim, and Ng See Kiong. Taxonomy construction using syntactic contextual evidence. In *EMNLP*, 2014. 2.1

[51] Luu Anh Tuan, Jung-jae Kim, and Ng See Kiong. Incorporating trustiness and collective synonym/contrastive evidence into taxonomy construction. 2015. 2.1

[52] Nakul Verma, Dhruv Mahajan, Sundararajan Sellamanickam, and Vinod Nair. Learning hierarchical similarity metrics. In *CVPR*, 2012. 2.3

[53] Wikipedia. `https://dumps.wikimedia.org/enwiki/20141208/`, 2014. 5.1.1

[54] T. Xiao, J. Zhang, K. Yang, Y. Peng, and Z. Zhang. Error-driven incremental learning in deep convolutional neural network for large-scale image classification. In *Proceedings of the ACM International Conference on Multimedia*, 2014. 2.3

[55] Zhicheng Yan, Hao Zhang, Robinson Piramuthu, Vignesh Jagadeesh, Dennis DeCoste, Wei Di, and Yizhou Yu. Hd-cnn: Hierarchical deep convolutional neural networks for large scale visual recognition. In *ICCV*, 2015. 6

[56] Hui Yang and Jamie Callan. A metric-based framework for automatic taxonomy induction. In *ACL-IJCNLP*, 2009. 2.1, 3.3.2

[57] M. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV*. 2014. 4.1.1

[58] M. D Zeiler and R. Fergus. Stochastic pooling for regularization of deep convolutional neural networks. In *ICLR*, 2013. 2.2

[59] Hao Zhang, Zhiting Hu, Yuntian Deng, Mrinmaya Sachan, Zhicheng Yan, and Eric P. Xing. Learning concept taxonomies from multi-modal data. In *The Annual Meeting of the Association for Computational Linguistics (ACL)*, 2016. 6

[60] Bin Zhao, Fei Li, and Eric P Xing. Large-scale category structure aware image categorization. In *NIPS*, 2011. 1

[61] Xingwei Zhu, Zhao-Yan Ming, Xiaoyan Zhu, and Tat-Seng Chua. Topic hierarchy construction for the organization of multi-source user generated contents. In *SIGIR*, 2013. 2.1

[62] Alon Zweig and Daphna Weinshall. Exploiting object hierarchy: Combining models from different category levels. In *ICCV*, 2007. 1