Towards Better Generalization: Model, Data, and Explicit Knowledge

Hessam Bagherinezhad

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2020

Reading Committee:

Ali Farhadi, Chair

Dieter Fox

Luke Zettlemoyer

Jeffrey Bilmes

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Towards Better Generalization:
Model, Data, and Explicit Knowledge

Hessam Bagherinezhad

Chair of the Supervisory Committee:
Professor Ali Farhadi
Computer Science and Engineering

In this dissertation, I explore three ways to make models more generalizable. 1) Through explicit knowledge extraction. Explicit knowledge enables models to correct their predictions, and in some cases to break a complex task into smaller pieces where each can be trained with less amount of data. 2) Through reducing model complexity. It is known that over-parameterized complex Convolutional Neural Networks (CNNs) often overfit to the given training set, and are therefore less generalizable. In this dissertation, I explore redesigning convolutional layers that outperform standard CNNs under few shot training scenario. 3) Through making labels more informative. I study the current data labeling paradigm, and present how labels for a simple image classification task are noisy. Noisy labels contribute to less generalizability. This is due to the fact that our over-parameterized models overfit to the noisy signal that is specific to that training set; therefore, they act poorly on an unseen test set. For explicit knowledge extraction, I first explore estimating and modeling Newtonian physics of a scene, and then explore extracting information about sizes of objects without any supervision required. For reducing model complexity, I explore redesigning Convolutional layers to reduce their complexity by sharing a dictionary of vectors among different convolutions. For label noise reduction, I explore making the training more accurate by refining the labels of a dataset with a dynamic label generator, called Label Refinery.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

This thesis absolutely could not exist without the guidance and support of Ali Farhadi. Ali, you are a big part of how I see the world and the scope of problems. I have learned from you to ask critical questions to learn about the core part of the problems at hand; to find what needs to be focused on, and what needs to be ignored. Your personal support through these years has been a firm ground that has given me the confidence and the courage to explore and learn.

My time during PhD years was partially spent at XNOR AI startup company. I would like to thank all my colleagues from whom I learned a great deal. My years at XNOR AI taught me a lot about reality of engineering an idea, and the reality of productionizing and delivering the technology to the mass. I would like to express my special gratitude to Dmitry Blenko, who taught me about the art of simplicity in designing a complex engineering product.

I would like to acknowledge my collaborators: Yejin Choi, Kiana Ehsani, Hannaneh Hajishirzi, Max Horton, Roozbeh Mottaghi, Mohammad Rastegari, and Joseph Redmon. All of whom have been great friends and made the process of exploring and failing more fun. I would like to especially thank Roozbeh and Mohammad who pushed me to explore new areas of ideas and have been a great sounding board for crazy ideas. I would also like to thank Shayan Oveis Gharan, who was my initial PhD advisor; when I was pursuing a PhD in theoretical computer science. During that time, I learned a bit more on how to think about high dimensions, and probability.

My time at UW would have been incomplete without the close friendships that we built. The friendships that transformed me as a person. My good friends Koosha Khalvati and

Hossein Daraei transformed my perspective, in different but complimentary ways. They helped me to see and feel that the world can be seen through many different lenses. I would like to thank my cousin, Shayan Mirjafari; whose support for me has been consistent and reliable.

Finally, I would like to thank my family, who defined who I am. I want to thank my dad, Bagher, who shaped my critical thinking at an early age. I want to thank my mom, Azam, whose care and love has given me the confidence to exist. I want to thank my oldest brother, Saber, who lead me to the path that I took. I want to thank my middle brother, Hanif, from whom I learned how to follow my own self and have fun. I want to thank my youngest brother, Sina, who was a childhood friend and a shield to threats.

DEDICATION

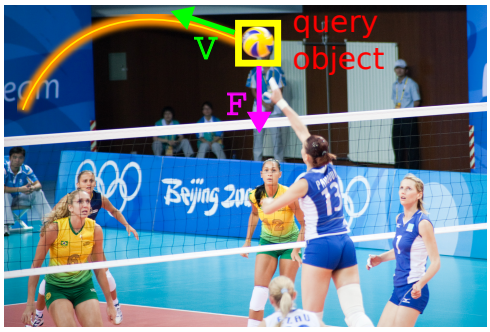
a small contribution towards Substantial Motion

Chapter 1

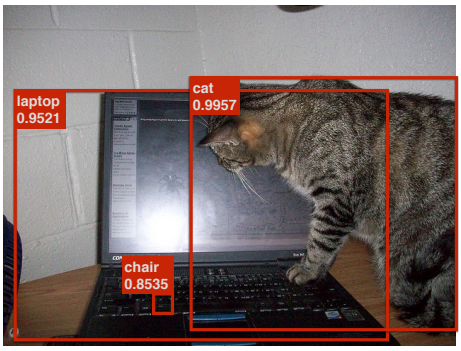
INTRODUCTION

With an infinite amount of training data, a complex enough model can learn the true distribution of data and perform accurately at test time. However, neither infinite amount of training data exists nor too complex models are realistic assumptions because of memory and timing constraints. With the more realistic assumption of finite data, the statistical models often *overfit* to the training set and don't generalize to the unseen test set. This issue is more dramatic for more complex tasks. Let's take the example of an imaginary self driving car system. Imagine that we want to build an end-to-end system that statistically learns how to model user's actions (pedals and steering wheel) from input (RGB, LiDAR, etc.). This is extremely challenging because 1) massive amount of data is needed to model every possible combination of input domain, and 2) collecting data for such complex tasks is often very expensive. In the smart self-driving car example, we're also facing the issue of a resource constraint platform.

One way to reduce the need for data in complex tasks is to break them into smaller pieces. Take the smart self-driving car example. Making a complex end-to-end model that goes directly from the stream of pixels to the action space of driving a car requires a massive amount of data since the model needs to learn about the state of the world by just looking at the pixel values. The learning process would need to learn the concept of *objects*, and that the car should not drive into them through many samples of sequenced pixel data and the actions taken accordingly. Also the model to learn such complexity would need to have a massively large capacity. The need for data could be reduced, however, if we explicitly train an object detector that detects objects, and inject the *explicit knowledge* that the car should not drive to objects. The task of object detection requires significantly less data since the



(a) Humans can infer the forces and the next movements of the objects from the context and pose of the players. That's a key capability in human perception that allows them to interact with the environment.



(b) The top 3 predictions of Fast-RCNN on an example image from MS-coco dataset. The false positive prediction of chair could be avoided if the system knew about the typical sizes of objects.

Figure 1.1: Incorporating explicit knowledge to the models reduces the need for excess data.

problem would reduce to a pattern recognition of pixel values, which convolutional neural networks are shown to be good at.

In the smart self-driving car example, if our system knows about the physics of a scene (i.e. the forces and the initial velocity that is being applied to objects), it can formulate the movements of the objects and proactively predict what happens next in the scene. Humans reliably use these predictions for planning their actions, making everyday decisions, and even correcting visual interpretations [51]. Examples include predictions involved in passing a busy street, catching a Frisbee, or hitting a tennis ball with a racket. Figure 1.1(a) shows an example from a volleyball scene where humans can reliably predict the forces and the next movements of the ball, only based on the context and the pose of the player. That *explicit knowledge*, if available, could enable the model to predict what happens next in a scene and incorporate that to act accordingly. In Chapter 2, we show that models can be trained to obtain the same common sense of the scene's physics.

Take the concrete and less complex task of object detection. The task of object detection is to given a still image of a scene, draw bounding boxes over the objects that exist in that

scene. Making an object detector that can detect objects in all of their view angles and deformations requires collecting a massive amount of data. However, if we have injected the *explicit knowledge* about the sizes of objects, a lot of false positives could be pruned out, and therefore the need for more data would be reduced. Figure 1.1(b) shows the top 3 predictions of Fast-RCNN [39], a widely used object detector, on an image drawn from MSCOCO validation set. If the model knew about the sizes of objects, the false positive detection of a small chair could have been avoided. In computer vision, size information manually extracted from furniture catalogs, has shown to be effective in indoor scenes understanding and reconstruction [107]. However, size information is not playing a major role in mainstream computer vision tasks yet. This might be due to the fact that there is no unified and comprehensive resource for objects sizes. In Chapter 3 we show that size information can be captured from the images freely available on the web.

Overfitting to the finite training set can be witnessed even for the basic task of image classification with the massive dataset of ImageNet. Early successful deep learning models had a big accuracy gap between the training and the test sets. Researchers have tried to reduce this gap by introducing different regularization components for the model such as Dropout [125] and Batch Normalization [64], and introducing different regularization training techniques such as weight decay and data augmentation. Convolutional Neural Networks, however, still suffer from over parameterization. The over parameterization has both made these models overfit to the training set (therefore inaccurate in absence of massive datasets), and slow in practice. In Chapter 4, we study the over-parameterization of convolutional neural networks and show that convolutional layers of these models can be replaced with a dictionary and lookup based solution. We show that by sharing information across different convolutions, these models can become dramatically faster and more generalizable.

Another way that we can fight against the overfitting problem, is to provide a more accurate training set. With a more accurate training set, the models overfit to a more accurate signal and therefore they become more accurate. For instance, Figure 1.2 shows a sample from ImageNet's training set. Although the label of the image is a one hot vector

(a) (b)

Figure 1.2: A sample image and its random crop from the "persian cat" category of ImageNet's training set. The image contains both a persian cat and a ball; however, the label is not reflecting the ball. This would result in penalizing the model even if it gives a little bit of a score to the ball. This problem is more significant with the presence of data augmentation. With data augmentation, the image level label may no longer be the main object of interest, if it still exist at all.

of "persian cat", the content of the image has visual signal of another ImageNet's category: "golf ball". The problem of inaccurate label is much more significant in the presence of data augmentation. Data augmentation is used to synthetically extend the size of the training set. The data is augmented through getting random crops and randomly flipping the images. The labels, however, are borrowed exactly the same from the image level labels. This sometimes results in crops that the image label is no longer the main object of interest in the crop, if it exists at all. In Chapter 5 we study the inaccurate dataset label problems, and show that we can design a *Label Refinery* that goes over all of the dataset and can generate more accurate labels for the training set and its augmented patches. We show that image classifiers can gain significant improvements can be achieved from refining the labels of an image classification dataset.

Chapter 2

NEWTONIAN IMAGE UNDERSTANDING: UNFOLDING THE DYNAMICS OF OBJECTS IN STATIC IMAGES

In this chapter, we study the challenging problem of predicting the dynamics of objects in static images. Given a query object in an image, our goal is to provide a physical understanding of the object in terms of the forces acting upon it and its long term motion as response to those forces. Direct and explicit estimation of the forces and the motion of objects from a single image is extremely challenging. We define intermediate physical abstractions called Newtonian scenarios and introduce Newtonian Neural Network (N^3) that learns to map a single image to a state in a Newtonian scenario. Our evaluations show that our method can reliably predict dynamics of a query object from a single image. In addition, our approach can provide physical reasoning that supports the predicted dynamics in terms of velocity and force vectors. To spur research in this direction we compiled Visual Newtonian Dynamics (VIND) dataset that includes more than 6000 videos aligned with Newtonian scenarios represented using game engines, and more than 4500 still images with their ground truth dynamics.

2.1 Introduction

A key capability in human perception is the ability to proactively predict what happens next in a scene [11]. Humans reliably use these predictions for planning their actions, making everyday decisions, and even correcting visual interpretations [51]. Examples include predictions involved in passing a busy street, catching a frisbee, or hitting a tennis ball with a racket. Performing these tasks require a rich understanding of the dynamics of objects moving in a scene. For example, hitting a tennis ball with a racket requires knowing the

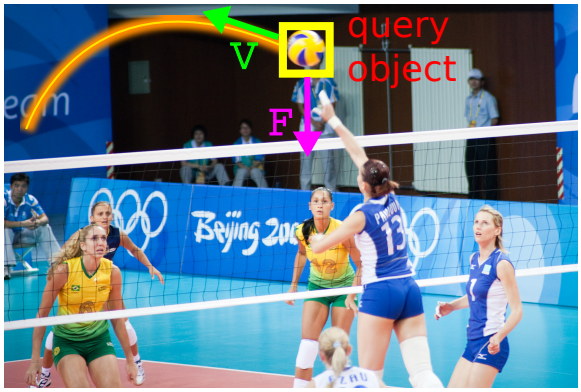


Figure 2.1: Given a static image, our goal is to infer the dynamics of a query object (forces that are acting upon the object and the expected motion of the object as a response to those forces). In this chapter, we show an algorithm that learns to map an image to a state in a physical abstraction called a Newtonian scenario. Our method provides a rich physical understanding of an object in an image that allows prediction of long term motion of the object and reasoning about the direction of net force and velocity vectors.

dynamics of the ball, when it hits the ground, how it bounces back from the ground, and what form of motion it follows.

Rich physical understanding of human perception even allows predictions of dynamics on only a single image. Most people, for example, can reliably predict the dynamics of the volleyball shown in Figure 2.1. Theories in perception and cognition attribute this capability, among many explanations, to previous experience [19] and existence of an underlying physical abstraction [46].

In this chapter, we address the problem of physical understanding of objects in images in terms of the forces actioning upon them and their long term motions as their responses to those forces. Our goal is to unfold the dynamics of objects in still images. Figure 2.1 shows an example of a long term motion predicted by our approach along with the physical reasoning that supports the predicted dynamics.

Motion of objects and its relations to various physical quantities (mass, friction, external

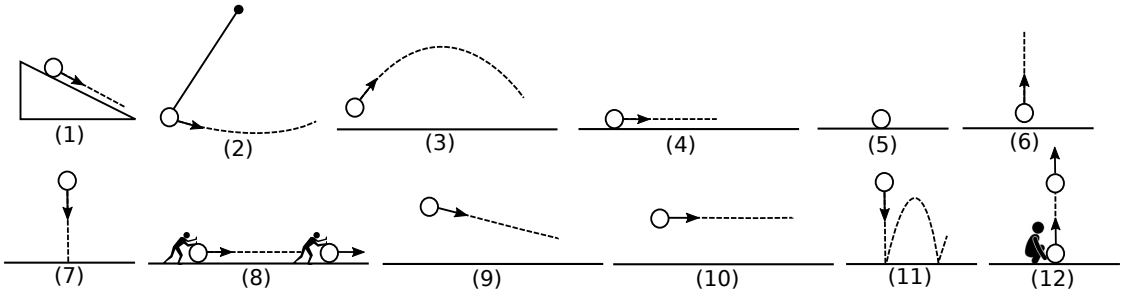


Figure 2.2: **Newtonian Scenarios** are defined according to different physical quantities: direction of motion, forces, etc. We use 12 scenarios that are depicted here. The circle represents the object, and the arrow shows the direction of its motion.

forces, geometry, etc.) has been extensively studied in Mechanics. In schools, classical mechanics is taught using basic Newtonian scenarios that explain a large number of simple motions in real world: inclined surfaces, falling, swinging, external forces, projectiles, etc. To infer the dynamics of an object, students need to figure out the Newtonian scenario that explains the situation, find the physical quantities that contribute to the motion, and then plug them into the corresponding equations that relate contributing physical quantities to the motion.

Estimating physical quantities from an image is an extremely challenging problem. For example, computer vision literature does not provide a reliable solution to direct estimation of mass, friction, the angle of an inclined plane, etc. from an image. Instead of direct estimation of the physical quantities from images, we formulate the problem of physical understanding as a mapping from an image to a physical abstraction. We follow the same principles of classical Mechanics and use Newtonian scenarios as our physical abstraction. These scenarios are depicted in Figure 2.2. We chose to learn this mapping in the visual space and thus render the Newtonian scenarios using game engines.

Mapping a single image to a state in a Newtonian scenario allows us to borrow the rich Newtonian interpretation offered by game engines. This enables predicting the long term

motion of the object along with rich physical reasoning that supports the predicted motion in terms of velocity and force vectors[1]. Learning such a mapping requires reasoning about subtle visual and contextual cues, and common knowledge of motion. For example, to predict the expected motion of the ball in Figure 2.1 one needs to rely on previous experience, visual cues (subtle hand posture of the player on the net, the line of sight of other players, their pose, scene configuration), and the knowledge about how objects move in a volleyball scene. To perform this mapping, we adopt a data driven approach and introduce Newtonian Neural Networks (N^3) that learns the complex interplay between visual cues and motions of objects.

To facilitate research in this challenging direction, we compiled *VIND*, VIsual Newtonian Dynamics dataset, that contains 6806 videos, with the corresponding game engine videos for training and 4516 still images with the predicted motions for testing.

Our experimental evaluations show promising results in Newtonian understanding of objects in images and enable prediction of long-term motions of objects backed by abstract Newtonian explanations of the predicted dynamics. This allows us to unfold the dynamics of moving objects in static images. Our experimental evaluations also show the benefits of using an intermediate physical abstraction compared to competitive baselines that make direct predictions of the motion.

2.2 Related Work

Cognitive studies: Recent studies in computational cognitive science show that humans approximate the principles of Newtonian dynamics and simulate the future states of the world using these principles [46, 12]. Our use of Newtonian scenarios as an intermediate representation is inspired by these studies.

Motion prediction: The problem of predicting future movements and trajectories has been tackled from different perspectives. Data-driven approaches have been proposed in [155, 91] to predict motion field in a single image. Future trajectories of people are inferred

[1]Throughout this chapter we refer to force and velocity vector as normalized unit vectors that show the direction of force or velocity.

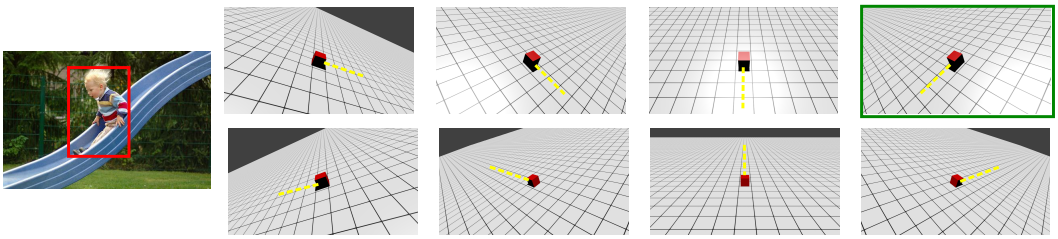


Figure 2.3: **Viewpoint annotation.** We ask the annotators to choose the game engine video (among 8 different views of the Newtonian scenario) that best describes the view of the object in the image. The object in the game engine video is shown in red, and its direction of movement is shown in yellow. The video with a green border is the selected viewpoint. These videos correspond to Newtonian scenario (1).

in [72]. [138] proposed to infer the most likely path for objects. In contrast, our method focuses on the physics of the motion and estimates a 3D long-term motion for objects. There are recent methods that address prediction of optical flow in static images [109, 139]. Flow does not carry semantics and represents very short-term motions in 2D whereas our method can infer long term 3D motions using force and velocity information. Physic-based human motion modeling was studied by [15, 13, 14, 136]. They employed human movement dynamics to predict future pose of humans. In contrast, we estimate the dynamics of objects.

Scene understanding: Reasoning about the stability of a scene has been addressed in [68] that use physical constraints to reason about the stability of objects that are modeled by 3D volumes. Our work is different in that we reason about the dynamics of stable and moving objects. The approach of [158] computes the probability that an object falls based on inferring disturbances caused naturally or by human actions. In contrast, we do not explicitly encode physics equations and we rely on images and direct perception. The early work of Mann *et. al.*[93] studies the perception of scene dynamics to interpret image sequences. Their method, unlike ours, requires complete geometric specification of the scene. A rich set of experiments are performed by [147] on *sliding* motion in the lab settings to estimate

object mass and friction coefficients. Our method is not limited to *sliding* and works on a wide range of physical scenarios in various types of scenes.

Action Recognition: Early prediction of activities has been discussed in [118, 105, 56, 81]. Our work is quite different since we estimate long-term motions as opposed to the class of actions.

Human object interaction: Prediction of human action based on object interactions has been studied in [76]. Prediction of the behavior of humans based on functional objects in a scene has been explored in [151]. Relative motion of objects in a scene are inferred in [37]. Our work is related to this line of thought in terms of predicting future events from still images. But our objective is quite different. We do not predict the next action, we care about understanding the underlying physics that justifies future motions in still images.

Tracking: Note that our approach is quite different from tracking [65, 23, 22] since tracking methods are not destined for single image reasoning. [136] incorporates simulations to properly model human motion and prevent physically impossible hypotheses during tracking.

2.3 Problem Statement & Overview

Given a static image, our goal is to reason about the expected long-term motion of a query object in 3D. To this end, we use an intermediate physical abstraction called Newtonian scenarios (Figure 2.2) rendered by a game engine. We learn a mapping from a single image to a state in a Newtonian scenario by our proposed Newtonian Neural Network (N^3). A state in a Newtonian scenario corresponds to a specific moment in the video generated by the game engine and includes a set of rich physical quantities (force, velocity, 3D motion) for that moment. Mapping to a state in a Newtonian scenario allows us to borrow the corresponding physical quantities and use them to make predictions about the long term motion of the query object in a single image.

Mapping from a single image to a state in a Newtonian scenario involves solving two problems: (a) figuring out which Newtonian scenario explains the dynamics of the image

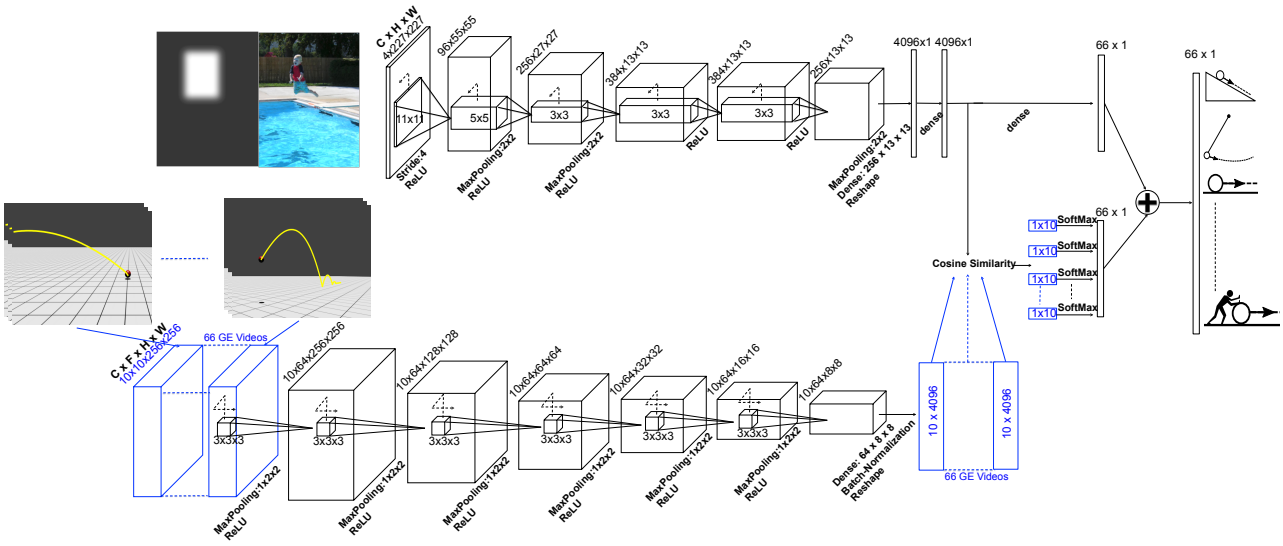


Figure 2.4: **Newtonian Neural Network (N^3):** This figure illustrates a schematic view of our proposed neural network model. The first row (referred to as *image row*), processes the static image augmented by an extra channel that shows the localization of the query object with a Gaussian-smoothed binary mask. Image row has the same architecture as AlexNet [77] for image classification. The larger cubes in the row indicate the convolutional outputs. The dimensions for convolutional outputs are **C**hannels, **H**eight, **W**idth. The smaller cubes inside them indicate 2D convolutional filters, which are convolved across Width and Height. The second row (referred to as *motion row*), processes the video inputs from game engine. This row has similar architecture to C3D [132]. The dimensions for convolutional outputs in this row are **C**hannels, **F**rames, **H**eight, **W**idth. The filters in the motion row are convolved across Frames, Width and Height. These two rows meet by a cosine similarity layer that measures the similarities between the input image and each frame in the game engine videos. The maximum value of these similarities, in each Newtonian scenario is used as the confidence score for that scenario describing the motion of the object in the input image.

best; (b) finding the correct moment in the scenario that matches the state of the object in motion. There are strong contextual and visual cues that can help to solve the first problem. However, the second problem involves reasoning about subtle visual cues and is even hard for human annotators. For example, to predict the expected motion and the current state of the ball in Figure 2.1 one needs to reason from previous experiences, visual cues, and knowledge about the motion of the object. N^3 adopts a data driven approach to use visual cues and the abstract knowledge of motion to learn (a) and (b) at the same time. To encode the visual cues N^3 uses 2D Convolutional Neural Networks (CNN) to represent the image. To learn about motions N^3 uses 3D CNNs to represent game engine videos of Newtonian scenarios. By joint embedding N^3 learns to map visual cues to exact states in Newtonian scenarios.

2.4 VIND Dataset

We collect VIsual Newtonian Dynamics (VIND) dataset, which contains game engine videos, natural videos and static images corresponding to the Newtonian scenarios. The Newtonian scenarios that we consider are inspired by the way Mechanics is taught in school and cover commonly seen simple motions of objects (Figure 2.2). Few factors distinguish these scenarios from each other: (a) the path of the object, *e.g.,*scenario (3) describes a projectile motion, while scenario (4) describes a linear motion, (b) whether the applied force is continuous or not, *e.g.,*, in scenario (8), the external force is continuously applied, while in scenario (4) the force is applied only in the beginning. (c) whether the object has contact with a support surface or not, *e.g.,*, this is the factor that distinguishes scenario (10) from scenario (4).

Newtonian Scenarios: Representing a Newtonian scenario by a natural video is not ideal due to the noise caused by camera motion, object clutter, irrelevant visual nuisances, *etc.*. To abstract away the Newtonian dynamics from noise and clutter in real world, we construct the Newtonian scenarios (shown in Figure 2.2) using a game engine. A game engine takes a scene configuration as input (*e.g.,*a ball above the ground plane) and simulates it forward in time according to laws of motion in physics. For each Newtonian scenario, we render its

corresponding game engine scenario from different viewpoints. In total, we obtain 66 game engine videos. For each game engine video, we store its depth map, surface normals and optical flow information in addition to the RGB image. In total each frame in the game engine video has 10 channels.

Images and Videos: We also collect a dataset of natural videos and images depicting moving objects. The current datasets for action or object recognition are not suitable for our task as they either show complicated movements that go beyond classical dynamics (*e.g.,head massage* or *make up* in UCF-101 [123], HMDB-51 [78]) or they show no motion (most images in PASCAL [34] or COCO [86]).

Annotations. We provide three types of annotations for each image/frame: (1) bounding box annotations for the objects that are described by at least one of our Newtonian scenarios, (2) viewpoint information *i.e.,*which viewpoint of the game engine videos best describes the direction of the movements in the image/video, (3) state annotations. By state, we mean how far the object has moved on the expected scenario (*e.g.,*is it at the beginning of the projectile motion? or is it at the peak point?). More details about the collection of the dataset and the annotation procedure can be found in Section 2.6 and the supplementary material. Example game engine videos corresponding to Newtonian scenario (1) are shown in Figure 2.3.

2.5 Newtonian Neural Network

N^3 is shaped by two parallel convolutional neural networks (CNNs); one to encode visual cues and another to represent Newtonian motions. The input to N^3 is a static image with four channels (RGBM; where M is the object mask channel that specifies the location of the query object by a bounding-box mask smoothed with a Gaussian kernel) and 66 videos of Newtonian scenarios[2](as described in Section 2.4) where each video has 10 frames (equally-spaced frames sampled from the entire video) and each frame has 10 channels (RGB, flow,

[2]From now on, we refer to the game engine videos rendered for Newtonian scenarios as Newtonian scenarios.

depth, and surface normal). The output of N^3 is a 66 dimensional vector where each dimension shows the confidence of the input image being assigned to a viewpoint of a Newtonian scenario. N^3 learns the mapping by enforcing similarities between the vector representations of static images and that of video frames corresponding to Newtonian scenarios. The state prediction is achieved by finding the most similar frame to the static image in the Newtonian space.

Figure 2.4 depicts a schematic illustration of N^3. The first row resembles the standard CNN architecture for image classification introduced by [77]. We refer to this row as *image row*. Image row has five 2D CONV layers (convolutional layers) and two FC layers (fully connected layers). The second row is a volumetric convolutional neural network inspired by [132]. We refer to this row as *motion row*. Motion row has six 3D CONV layers and one FC. The input to the motion row is a batch of 66 videos (corresponding to 66 Newtonian scenarios rendered by game engines). The motion row generates a 4096x10 matrix as output for each video, where a column in this matrix can be seen as a descriptor for a frame in the video. To preserve the same number of frames in the output, we eliminate MaxPooling over the temporal dimension for all CONV layers in the motion row. The two rows are joined by a matching layer that uses cosine similarity as a matching measure. The input to the image row is an RGBM image and the output is a 4096 dimensional vector (values after FC7 layer). This vector can be seen as a visual descriptor for the input image.

The matching layer takes the output of the image row and the output of the motion row as input and computes the cosine similarity between the image descriptors and all of the 10 frames' descriptors in each video in the batch. Therefore, the output of matching layer are 66 vectors where each vector has 10 dimensions. The dimension with maximum similarity value indicates the state of dynamics for each Newtonian scenario. For example, if the third dimension has the maximum value, it means, the input image has maximum similarity with the third frame of the game engine video, thus it must have the same state as that of the third frame in the corresponding game engine video. SoftMax layers are appended after the cosine similarity layer to pick the maximum similarity as a confidence score for each

Newtonian scenario. This enables N^3 to learn the state prediction without any state level annotations. This is an advantage for N^3 that can implicitly learn the state of the motion by directly optimizing for the prediction of Newtonian scenarios. These confidence scores are linearly combined with the confidence scores from the image row to produce the final scores. This linear combination is controlled by a parameter $\lambda \in [0, 1]$ that weights the effect of motion for the final score.

Training: In order to train N^3, we feed the input by picking a batch of random images from the training set and a batch of game engine videos that cover all Newtonian scenarios (66 videos). Each iteration involves a forward and a backward pass through the network. We use cross entropy as our loss function:$E = -\frac{1}{n}\sum_{i=1}^{n}[p_i \log \hat{p}_i + (1-p_i)\log(1-\hat{p}_i)]$, where p_i is the ground truth label of the input image (the value is 1 for the ground truth class and 0 otherwise) and $\hat{p}_i$ is the predicted probability obtained by taking SoftMax over the output of N^3. In each iteration, we feed a random batch of images to the network, but a fixed batch of videos across all iterations. This enables N^3 to penalize the error over all of the Newtonian scenarios at each iteration. The other option could be passing a pair of a random image and a game engine video, then predicting a binary output showing whether the image corresponds to the Newtonian scenario or not. This requires a lot more iterations to see all the possible positive and negative pairings for an image and has shown to be less effective for our problem.

Testing: At test time, the 4096x10 descriptors for abstract motions can be pre-computed from the motion row of N^3 after CONV6 layer. For each test, we only feed a single RGBM image as input and obtain the underlying Newtonian scenario h and its matching state s_h. The predicted scenario (h) is the scenario with maximum confidence in the output. The matching state s_h is achieved by

$$s_h = \arg\max_i \{Sim(\mathbf{x}, \mathbf{v}_h^i)\} \tag{2.1}$$

where $\mathbf{x}$ is the 4096x1 image descriptor, $\mathbf{v}_h^i$ is the 4096x10 video descriptor for Newtonian scenario h and $i \in \{1, 2, .., 10\}$ indicates the frame index in the video. $Sim(.,.)$ is the

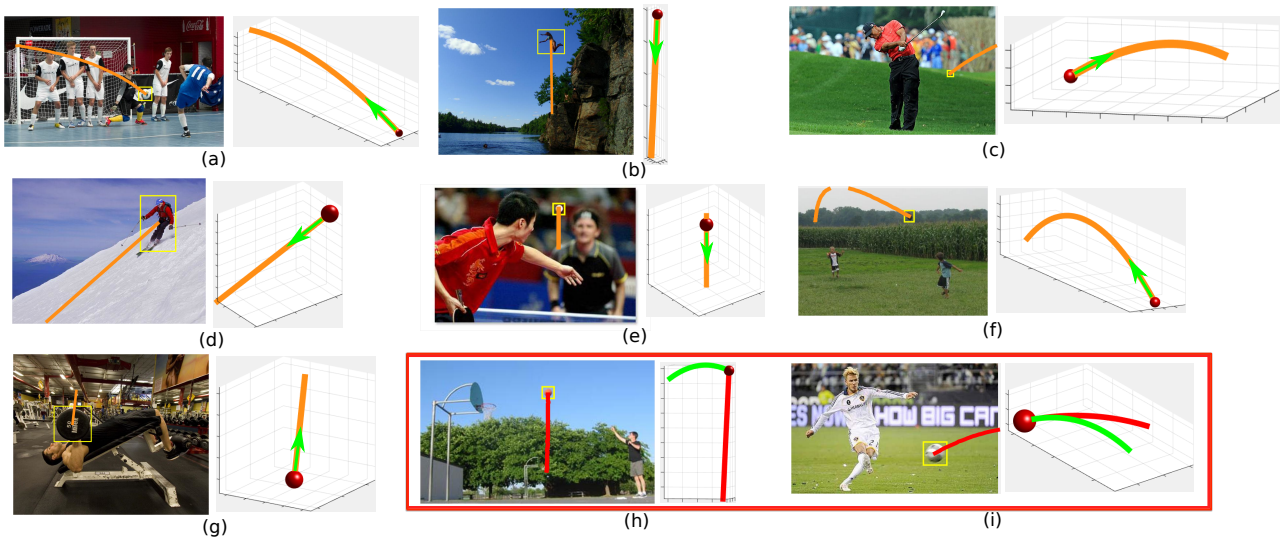


Figure 2.5: The expected motion of the object in the static image is shown in orange. We have visualized the 3D motion of the object (red sphere) and its superposition on the image (left image). We also show failure cases in the red box, where the red and green curves represent our prediction and ground truth, respectively.

standard cosine similarity between two vectors. Given h and s_h, a long-term 3D motion path can be drawn for the query object by borrowing the game engine parameters (*e.g.*,direction of velocity and force, 3D motion, and camera view point) from the state s_h of Newtonian scenario h.

2.6 Experiments

We compare our method with a number of baselines in predicting the motion of a query object in an image and provide an ablation study that examines the utility of different components in our method. We further show qualitative results for motion prediction and estimation of force and velocity directions. We also show the benefits of estimating optical flow from our long term motions predicted by our method. Additionally, we show the generalization to unseen scene types.

2.6.1 Settings

Network: We implemented our proposed neural network N^3 in Torch [2]. We use a machine with a 3.5GHz Intel Xeon CPU and GeForce TITAN X GPU to train and test our model. To train N^3, we initialized the image row (refer to Figure 2.4) by a publicly available [3] pre-trained CNN model. We initialize the fourth channel (M) by random values drawn from a Gaussian distribution ($\mu = 0$,$\sigma = \frac{10}{filter\,size}$). The motion row was initialized randomly, where the random parameters came from a Gaussian distribution ($\mu = 0$,$\sigma = \frac{10}{filter\,size}$). For training, we use batches of 128 input images in the image row and 66 videos in the motion row. We run the forward and backward passes for 5000 iterations[4]. We started by the learning rate of 10^{-1} and gradually decreased it down to 10^{-4}. In order to prevent the numerical instability of the cosine similarity function, we use the smooth version of cosine similarity, which is defined as: $S(x, y) = \frac{x.y}{|x||y|+\epsilon}$, where $\epsilon = 10^{-5}$.

Dataset details: We use Blender [1] game engine to render the game engine videos corresponding to the 12 Newtonian scenarios. We factor out the effect of force magnitude and camera distance.

The Newtonian scenarios are rendered from 8 different azimuth angles. Scenarios 6, 7, and 11 in Figure 2.2 are symmetric across different azimuth angles and we therefore render them from 3 different elevations of the camera. The Newtonian scenarios 2 and 12 are the same across viewpoints with 180° azimuth difference. We consider four views for those scenarios. For *stability* (scenario (5)), we consider only 1 viewpoint (there is no motion). In total, we obtain 66 videos for all 12 Newtonian scenarios.

Our new dataset (VIND) contains more than 6000 video clips in natural scenes. These videos contain more than 200,000 frames in total. For training, we use frames randomly sampled from these video clips. To train our model, we use bounding box information of query objects and viewpoint annotations for the corresponding Newtonian scenario (the

[3]https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet

[4]In our experiments the loss values start converging after 5K iterations.

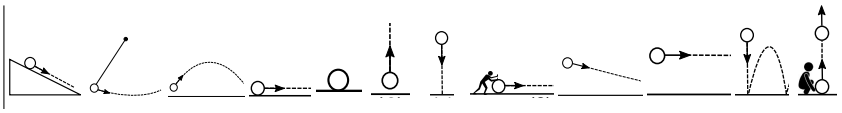
													Avg.
Direct Regression	32.7	59.9	12.4	16.1	84.6	48.8	8.2	20.2	1.6	13.8	49.0	16.4	30.31
Direct Regression - Nearest	52.7	38.4	17.3	23.5	64.9	**69.2**	18.1	36.2	3.2	20.4	**76.5**	24.2	37.05
N^3 (ours)	**60.8**	**64.7**	**39.4**	**37.6**	**95.4**	54.1	**50.3**	**76.9**	**9.4**	**38.1**	72.1	**72.4**	**55.96**

Table 2.1: Estimation of the motion of the objects in 3D. F-measure is used as the evaluation metric.

procedure for viewpoint annotations is shown in Figure 2.3).

The image portion of our dataset includes 4516 images that are divided into 1458 and 3058 images for validation and testing, respectively. We tune our parameters using the validation set and report our results on the test subset. For evaluation, each image has bounding box, viewpoint and state annotations. The details of the annotation and collection process is described in the supplementary material.

2.6.2 Estimating the motion of query objects

Given a single image and a query object, we evaluate how well our method can estimate the motion of the object. We compare the resulting 3D curves from our method with that of the ground truth.

Evaluation Metric. We use an evaluation metric which is similar to the F-measure used for comparing contours (*e.g.,*[6]). The 3D curve of groundtruth and the estimated motion are in XYZ space. However, the two curves do not necessarily have the same length. We slide the shorter curve over the longer curve to find an alignment with the minimum distance. We then compute precision and recall by thresholding the distance between corresponding points on the curves.

We also report results using the Modified Hausdorff Distance (MHD), however the F-measure is more interpretable since it is a number between 0 and 100.

Baselines. A set of comparisons with a number of baselines are presented in Table 2.1. The first baseline, called *Direct Regression*, is a direct regression from images to the trajecto-

ries in the 3D space (groundtruth curves are represented by B-splines with 1200 knots). For this baseline, we modify AlexNet architecture to regress each image to its corresponding 3D curve. More specifically, we replace the classification loss layer with a Mean Squared Error (MSE) loss layer. Table 2.1 shows that N^3 significantly outperforms this baseline that aims at directly regressing the motion from visual data. We postulate that this is mainly due to the dimensionality of the output and the complex interplay between subtle visual cues and the 3D motion of objects. To further probe that if the direct regression can even roughly estimate the shape of the trajectory we build an even stronger baseline. For this new baseline, called *Direct Regression-Nearest*, we use the output of the direct regression baseline above to find the most similar 3D curve among Newtonian scenarios (based on normalized Euclidean distance between the B-spline representations). Table 2.1 shows that N^3 also outperforms this competitive baseline. In terms of the MHD metric, N^3 also outperforms the baselines (5.59 versus 5.97 and 7.32 for the baseline methods; lower is better).

Figure 2.5 shows qualitative results in estimating the expected motion of the object in still images. When N^3 predicts a 3D curve for an image it also estimates the viewpoint. This allows us to project the 3D curve back onto the image. Figure 2.5 shows examples of these estimated motions. For example, N^3 correctly predicts the motion of the football thrown (Figure 2.5(f)), and estimates the right motion for the ping pong ball falling (Figure 2.5(e)). Note that N^3 cannot reason about possible future collisions with other elements in the scene. For example Figure 2.5(a) shows a predicted motion that goes through soccer players. This figure also shows some examples of failures. The mistake in Figure 2.5(h) can be attributed to the large distance between the player and the basketball. Note that when we project 3D curves to images we need to make assumptions about the distance to the camera and the 2D projected curves might have inconsistent scales.

Ablation studies. To study our method in further details, we test two variations of our method. In the first variation, λ (defined in Section 2.5) is set to 1, which means that we are ignoring the motion row in the network. We refer to this variation as $N^3 - NV$ in Table 2.2. N^3 outperforms $N^3 - NV$, indicating that the motion abstraction is an important

Ablations	$N^3 - NV$	N^3	$N^3 + SS$
F-measure	52.67	55.96	56.10

Table 2.2: Ablation study of 3D motion estimation. The average across 12 Newtonian scenarios is reported.

factor in N^3. To study the effectiveness of N^3 in state prediction, in the second variation, we measure the utility of providing state supervision for training N^3. We modified the output layer of N^3 to learn the exact state of the motion from the groundtruth augmented by state level annotations. This case is referred to as $N^3 + SS$ in Table 2.2. The small gap between the results in N^3 and $N^3 + SS$ shows that N^3 can reliably predict the correct state without state supervision. The procedure of annotating states (*i.e.,*specifying which frame of the game engine video corresponds to the state of the object in the image) is explained in the supplementary material.

Another ablation is to study the effectiveness of N^3 in classifying images into 66 classes corresponding to 12 Newtonian scenarios rendered from different viewpoints. In this ablation, shown in Table 2.3, we compare N^3 to $N^3 - NV$ with and without state supervision (SS) in a classification setting (not prediction of the motion). Also, our experiments show that N^3 and $N^3 - NV$ make different types of mistakes since fusing these variations in an optimal way (by an oracle) results in an improvement in classification (25.87).

Ablations	$N^3 - NV$	$N^3 - NV + SS$	N^3	$N^3 + SS$
Avg. Accuracy	20.37	19.32	21.71	21.94

Table 2.3: Estimation of Newtonian scenario and viewpoint (no state estimation).

Short-term flow estimation. Our method is designed to predict long-term motions in 3D, yet it can estimate short term motions by projecting the long term 3D motion onto the image. We compare the effectiveness of N^3 in estimating the flow with the state of the art methods explicitly trained to predict short-term flow from a *single image*. In particular, we

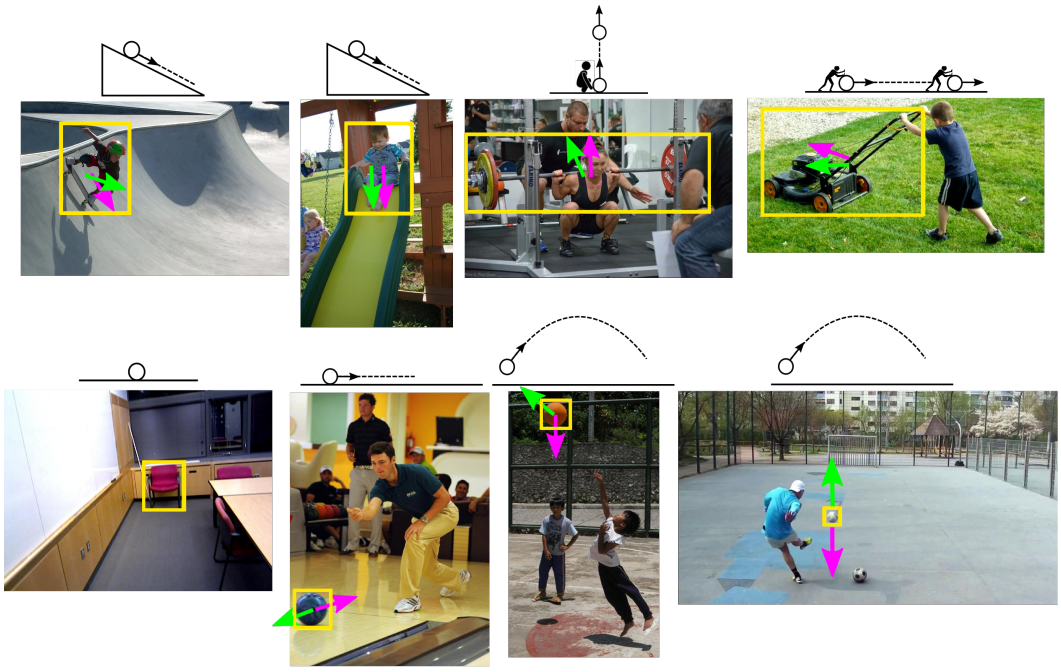


Figure 2.6: Visualization of the *direction* of net force and object velocity. The velocity is shown in green and the net force is shown in magenta. The corresponding Newtonian scenario is shown above each image.

compare with the recent method of Predictive-CNN [139]. For each query object, we average the dense flow predicted by [139] over the pixels in the object box and obtain a single flow vector. The evaluation metric is angular error (we do not compute flow magnitude). As shown in Table 2.4, our method outperforms [139] on our dataset.

Method	Angular Err.
Predictive-CNN [139]	1.53
N^3 (ours)	**1.29**

Table 2.4: Short-term flow prediction in a single image. The evaluation metric is angular error.

Force and velocity estimation. It is interesting to see that N^3 can predict the *direction* of the net force and velocity in a static image for a query object! Figure 2.6 shows qualitative examples. For example, it is exciting to show that N^3 can predict the friction in the bowling example, and the gravity in the basketball example. The net force applied to the chair in the bottom row (left) is zero since the normal force from the floor cancels the gravity.

Generalization to unseen scene types. We also evaluate how well our model generalizes to unseen scene types. We remove all images that represent the same scene type (*e.g.*,, all images that show a *billiard* scene in scenario (4)) from our training data and test how well we can estimate the motion of the object in images that show those scene types. Our method outperforms the baseline method (Table 2.5). The reported result is the average over 12 Newtonian scenarios, where we remove one scene type from each Newtonian scenario during training. The complete list of the removed scene types is available in the supp. material.

2.7 Conclusions

In this chapter we address the challenging problem of Newtonian understanding of objects in static images. Numerous physical quantities contribute to shaping the dynamics of objects in a scene. Direct estimation of those quantities is extremely challenging. In this chapter,

Method	F-measure
Direct Regression	25.76
N^3 (ours)	**36.40**

Table 2.5: Generalization to unseen scene types.

we assume intermediate physical abstractions, Newtonian scenarios and introduce a model that can map from a single image to a state in a Newtonian scenario. This mapping needs to learn subtle visual and contextual cues to be able to reason about the correct Newtonian scenario, state, viewpoint, etc. Rich physical predictions about the dynamics of objects in an images can then be made by borrowing information through the established correspondences to Newtonian scenarios. This allows us to predict the motion and reason about it in terms of velocity and force directions for a query object in a still image.

Our current solution can only reason about simple motions of rigid bodies and cannot handle complex and compound motions, specially when it is affected by other external elements in the scene (e.g. the motion of thrown ball would change if there is a wall in front of it in the scene). In addition, our method does not provide estimates for magnitude of the force and velocity vectors. We postulate that there might be very subtle visual cues that can contribute tho those estimates.

Rich physical understanding of images is an important building block towards deeper understanding of images, enables visual reasoning, and opens several new and exciting research directions in scene understanding. Reasoning about how objects move in an image is tightly coupled with semantic and geometric scene understanding. Explicit joint reasoning about these interactions is an exciting research direction.

Chapter 3

ARE ELEPHANTS BIGGER THAN BUTTERFLIES? REASONING ABOUT SIZES OF OBJECTS

Human vision greatly benefits from the information about sizes of objects. The role of size in several visual reasoning tasks has been thoroughly explored in human perception and cognition. However, the impact of the information about sizes of objects is yet to be determined in computer vision. We postulate that this is mainly attributed to the lack of a comprehensive repository of size information. In this chapter, we introduce a method to automatically infer object sizes, leveraging visual and textual information from web. By maximizing the joint likelihood of textual and visual observations, our method learns reliable relative size estimates, with no explicit human supervision. We introduce the relative size dataset and show that our method outperforms competitive textual and visual baselines in reasoning about size comparisons.

3.1 Introduction

Human visual system has a strong prior knowledge about physical sizes of objects in the real world [66] and can immediately retrieve size information as it recognizes objects [75]. Humans are often very sensitive to discrepancies in size estimates (size constancy [58]) and draw or imagine objects in canonical sizes, despite significant variations due to a change in viewpoint or distance [74]. Considering the importance of size information in human vision, it is counter-intuitive that most of the current object recognition and scene understanding methods in computer vision are agnostic to the sizes of the objects. We postulate that this is mainly due to the lack of a comprehensive resource that can provide information about object sizes. In this chapter, we introduce a method to automatically provide such information by

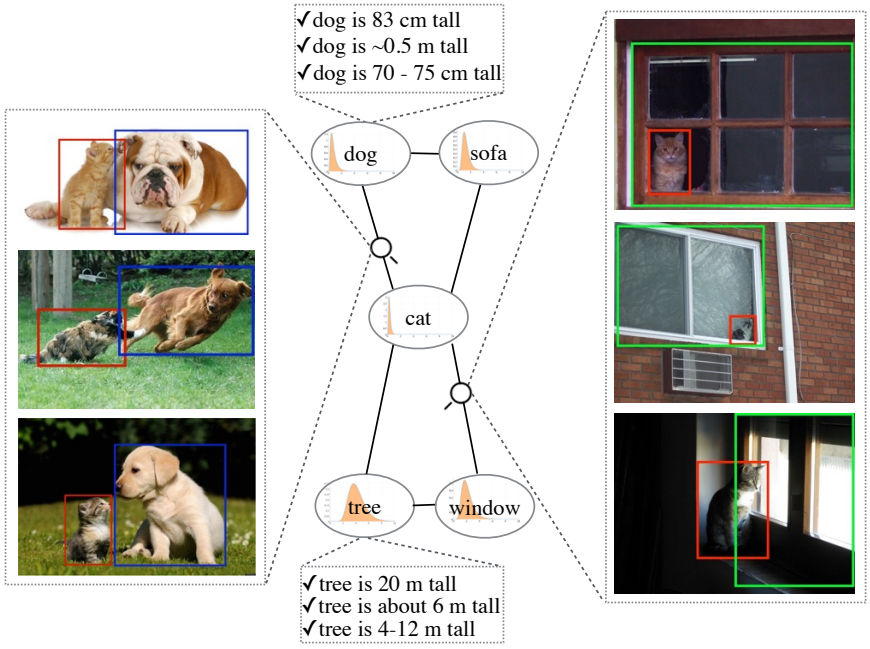


Figure 3.1: In this chapter we study the problem of inferring sizes of objects using visual and textual data available on the web. With no explicit human supervision, our method can produce reliable (83.5% accurate) relative size estimates. We use size graph, shown above, to represent both absolute size information (from textual web data) and relative ones (from visual web data). The size graph allow us to leverage the transitive nature of size information by maximizing the likelihood of both visual and textual observations.

representing and inferring object sizes and their relations. To be comprehensive, our method does not rely on explicit human supervision and only uses web data.

Identifying numerical properties of physical objects, such as size, has been recently studied in Natural Language Processing [5, 26] and shown to be helpful for question answering and information extraction [20, 26]. The core idea of the state-of-the-art methods is to design search queries in the form of manually defined templates either looking for absolute size of objects (e.g. "the size of a car is * unit") or specific relations (e.g. "wheel of a car"). These methods result in promising but noisy and incomplete set of estimates. For example, these methods predict a relatively small size for a 'car' because search queries discover more

frequent relations about the size of a 'toy car' rather than a regular 'car' [5]. In this chapter, we argue for incorporating visual information in estimating sizes of objects.

In images, estimating the absolute sizes of objects requires information about the camera parameters and accurate depth estimates which are not available at scale. Visual data, however, can provide informative cues about relative sizes of objects. For example, consider the 'cat' that is sitting by the 'window' in Figure 3.1. The relative size of the 'cat' and the 'window' can be computed using their detection boxes, adjusted by their coarse depth. A probability distribution over relative sizes of 'cats' and 'windows' can then be computed by observing several images in which 'cats' and 'windows' co-occur. However, not all pairs of objects appear in large enough number of images. Collecting visual observations for some pairs like 'sofa' and 'tree' is not possible. Furthermore, it is not scalable to collect visual observations for all pairs of objects.

In this chapter, we introduce a method to learn to estimate sizes of objects, with no explicit human supervision, leveraging both textual and visual observations. Textual resources provide noisy estimates of the absolute sizes of objects. Visual resources allow for noisy estimates of relative sizes of objects. Our approach is to couple these noisy estimates and use the transitive nature of size information to reason about objects that don't co-occur frequently. For example in Figure 3.1, our method can establish inferences about the relative size of 'sofa' and 'tree' through a set of intermediate relations between 'sofa'-'cat' and 'cat'-'tree'.

We introduce *size graph* as our representation to model object sizes and their relations. The nodes in the size graph correspond to the log-normal distribution of the sizes of objects and edges correspond to relative sizes of pairs of objects that co-occur frequently. The topology of the size graph provides guidance on how to collect enough textual and visual observations to deal with the noise and sparsity of the observations. We formulate the problem of learning the size of the objects as optimizing for a set of parameters that maximize the likelihood of both textual and visual observations.

Our experimental evaluations show strong results. On our dataset of about 500 relative size comparisons, our method achieves 83.5% accuracy, compared to 63.4% of a competitive

NLP baseline. Our results show that textual and visual data are complementary, and optimizing for both outperforms individual models. If available, our model can benefit from reliable information about the actual sizes of a limited number of object categories.

Our contributions are three-fold: (a) We present a scalable, novel method that learns to estimate object sizes by combining visual and textual information, with no explicit human supervision; (b) we present a scalable, compact representation for modeling distributions of object sizes (size graph); (c) we introduce a new dataset of relative size comparisons, and report on a series of experiments showing high efficacy in estimating relative sizes of object.

3.2 Related Work

Human understanding of the sizes of objects has attracted many researchers in perception and cognition. There is evidence to support that when human observers recognize an object, they also know how big it is [75]. The real-world size of objects is among fundamental properties of intermediate visual representations of objects [75]. Despite significant variance in the perceived size of the objects, due to viewpoint and distance, human observers often draw or imagine objects at consistent visual size, called canonical size [74]. Human observers have a prior knowledge about the sizes of objects, called assumed size [66, 10], and the visual size of the objects is correlated with the logarithm of the assumed size of objects [74].

There is some evidence to support that size information may also help computer vision systems to better understand the world. Size information manually extracted from furniture catalogs has shown to be effective in indoor scenes understanding and reconstruction [107]. However, size information is not playing a major role in mainstream computer vision tasks yet. This might be due to the fact that there is no unified and comprehensive resource for objects sizes. The visual size of the objects depends on multiple factors including the distance to the objects and the viewpoint. Single image depth estimation has been an active topic in computer vision for estimating the depth of each pixel either in indoor scenes using geometric cues [27], or layouts [54], or using geometric context in outdoor images [57]. Most beneficial to our task are methods that can produce absolute depth values [90, 119]. Depth estimation

and segmentation has also been addresses jointly resulting in the improved performance of both tasks [79]. In this chapter, we use the deep learning based method of [33] for single image depth estimation that produces state-of-the-art results.

Identifying numerical attributes about objects has been addressed in NLP recently for textual entailment, question answering, and information extraction. A few researchers [110, 20] use commonsense knowledge bases such as CYC as a resource for collecting numerical information. Unlike our method, these are not scalable. The common theme in the recent work [5, 26] is to use search query templates, collect numerical values, and model sizes as a normal distribution. There has been a few work [63, 101] combining query templates with other textual cues (e.g., more than, at least, as many as, etc). There have been recent approaches [129, 26] to extracting comparative knowledge from text, but the quality and scale of such extraction has been somewhat limiting. This is in part because most trivial commonsense knowledge is rarely stated explicitly in natural language text, e.g., it is unlikely to find a sentence that says a car is bigger than an orange. In addition, comparative statements in text, if found, rarely provide precisely how much one object is bigger than the other. In this chapter, we show that visual and textual observations are complementary, and a successful size estimation method will take advantage of both modalities. In particular, our experiments show that textual observations about the relative sizes of objects are very limited, and relative size comparisons are better collected through visual data. In addition, we show that log-normal distribution is a better model for representing sizes than normal distributions.

3.3 Overview of Our Method

Problem Overview: In this chapter, we address the problem of identifying sizes of physical objects using visual and textual information. Our goals are to (a) collect visual observation about the relative sizes of objects, (b) collect textual observations about the absolute sizes of objects, and (c) devise a method to make sense of vast amount of visual and textual observations and estimate object sizes. We evaluate our method by answering queries about

the size comparisons. In particular, we are interested in identifying if the object `A` is bigger than the object `B` for every two objects `A` and `B` in our dataset.

Algorithm 1 The overview of our method.

Representation: Construct Size Graph (Section 3.4.1).
▷ Collect Visual observations (Section 3.5.1)
for all edges (v, u) in the Size Graph **do**
    Get images from Flickr in which v and u are tagged.
    Run object detectors of v and u on all images.
    Observe the depth adjusted ratio of bounding box areas.
end for
▷ Collect Textual observations (Section 3.5.1)
for all nodes v in the Size Graph **do**
    Execute search engine patterns for each object.
    Observe the sizes found for objects.
end for
Model the size of each object with a log-normal.
Learning: Find the optimal parameters maximizing the likelihood (Section 3.5.2).

Overview of Our Method: We devise a method (Algorithm 1) that learns probability distributions over object sizes based on the observations gathered from both visual and textual web, with no explicit human supervision. Unfortunately, both visual and textual observations are noisy and incomplete. We introduce *size graph* that represents object sizes (nodes) and their relations (edges) in a connected, yet sparse graph representation (Section 3.4).

We use textual web data to extract information about the absolute sizes of objects through search query templates. We use web images to extract information about the relative sizes of objects if they co-occur in an image. With scalablity in mind, we incorporate webly-supervised object detectors [32] to detect the objects in the image and compute the depth adjusted ratio of the areas of the detected bounding boxes for objects (Section 3.5.1).

We formulate the problem of estimating the size as maximizing the likelihood of textual and visual observations to learn distributions over object sizes (Section 3.5.2). Finally, we incorporate an inference algorithm to answer queries in the form of "Which object is bigger?" (Section 3.5.3).

3.4 Representation: Size Graph

It is not scalable to collect visual observations for all pairs of objects. In addition, for some pairs like 'aeroplane' and 'apple', it is noisy (if at all possible) to directly collect visual observations. We introduce *size graph* as a compact, well-connected, sparse graph representation (Section 3.4.1) whose nodes are distributions over the actual sizes of the objects (Section 3.4.2). The properties of the size graph allows us to collect enough visual and textual data suitable for modeling the size distributions.

3.4.1 Graph Construction

We first describe the properties of the size graph and then show how we construct the topology of the graph.

Size Graph Properties: Given a list of objects $V = \{O_1, O_2, \cdots, O_n\}$, we want to construct a graph $G = (V, E)$ such that there is one node for every object and there exists an edge $(O_i, O_j) \in E$ only if O_i and O_j co-occur frequently in images. In particular, the size graph should have the following properties:(a) Connectivity, which allows us to take advantage of the transitivity of size and propagate any size information throughout the graph. In addition, we require that there are at least k disjoint paths between every two nodes in the graph in order to reduce the effect of noisy edges in the graph. (b) Sparsity, which allows us to collect enough visual data since it is not feasible (both computationally and statistically) to connect every two nodes in the graph. Adding an edge between two unrelated objects like 'apple' and 'bicycle' not only increases the computational cost, but also increases the noise of the observations.

Therefore our goal is to select a subgraph G of the complete graph with the above properties.

Modeling Co-occurrence: We approximate the likelihood of co-occurrence of two objects in images using the tag lists of images in Flickr 100M dataset. Every image in Flickr is accompanied with a list of tags including names of objects. We use the co-occurrence of two objects in tag lists of Flickr images as a proxy for how much those objects are likely to co-occur in images. We observed that not all co-occurrences are equally important and shorter tag lists are more descriptive (compared to longer lists). We first define the descriptiveness of a tag list as the inverse of the length of the list. Then, we compute co-occurrence of objects O_i and O_j by summing over the descriptiveness of the tag lists in which both objects O_i and O_j co-occur.

We define the cost ϵ_{ij} of an edge $e_{i,j} = (O_i, O_j)$ in the complete graph as the inverse of the co-occurrence of O_i and O_j. Therefore, if two objects co-occur frequently in a short list of tags, the cost of an edge is small. Let L_l be the tag list of the l_{th} image in Flickr 100M dataset, the following equation formulates the cost of an edge (O_i, O_j):

$$\epsilon_{ij} = \begin{cases} \frac{1}{\sum_{l:\{O_i,O_j\}\subseteq L_l} \frac{1}{|L_l|}}, & \text{if } \exists k : \{O_i, O_j\} \subseteq L_l \\ \infty, & \text{otherwise} \end{cases} \tag{3.1}$$

Constructing Size Graph: Let D be the weighted complete graph of objects, with edge costs define by equation 3.1. According to the properties of the size graph, our goal is to find a minimum cost subgraph of D in which there are multiple disjoint paths between every two nodes. Such subgraph would be less susceptible to the noise of visual observations across edges. As a corollary to Menger's theorem [97], there are at least k disjoint paths between every two nodes of an arbitrary graph G if and only if G is k-edge-connected (if we remove any $k-1$ edges, the graph is still connected). Therefore, our goal here is to find the minimum k-edge-connected subgraph.

Finding minimum spanning tree (MST) of the complete graph results in the 1-edge-connected subgraph with the minimum total cost. Finding MST is well studied and solvable

in polynomial time. However, in MST there is only one path between every pairs of nodes, and a noisy visual observation would affect all the size estimations. In order to make the subgraph more reliable, we expect k to be greater than one. The problem of finding the minimum k-edge-connected subgraph, however, is shown to be NP-hard for $k > 1$ [38].

Here, we introduce our algorithm to find a k-edge-connected subgraph whose cost is an approximation of the optimal cost. Our approximation algorithm is to iteratively find an MST $T_1 \subseteq D$, and remove its edges from D, and then continue with finding another MST of the remaining graph. Repeating this iteration for k times results in k disjoint spanning trees $T_1, T_2, \cdots, T_k$. The final subgraph $G = T_1 \cup \cdots \cup T_k$ is then derived by combining all these spanning trees together. The subgraph G is k-edge-connected, and its cost is an approximation of the optimal cost.

Lemma 1. *Every graph $H = T_1 \cup \cdots \cup T_k$ which is a union of k disjoint spanning trees is k-edge-connected.*

Proof. In order to make H disconnected, at least one edge should be removed from each spanning tree. Since spanning trees are disjoint, at least k edge removals are required to disconnect the graph H. □

Lemma 2. *Given a graph $G = (V, E)$, and the subgraph $H = T_1 \cup \cdots \cup T_k$ where T_i is the i_{th} MST of G. The total cost of H is at most $\frac{2M}{m}$ times the cost of the optimal k-edge-connected subgraph, where m and M are the minimum and the maximum of edge costs, respectively.*

Proof. Let OPT denote the optimal k-edge-connected subgraph. The minimum degree of OPT should be at least k. Hence, OPT must have at least $\frac{nk}{2}$ edges, each of which with the cost of at least m. Therefore $\frac{nkm}{2} \leq cost(OPT)$. On the other hand, the subgraph H has exactly $k(n-1)$ edges, each of which with the cost of at most M, so $cost(H) \leq kM(n-1)$.

$$\begin{aligned} cost(H) \leq kM(n-1) < kMn &= \frac{2M}{m} \times \frac{nkm}{2} \\ &\leq \frac{2M}{m} cost(OPT) \end{aligned}$$

□

3.4.2 Log-normal Sizes

There are many instances of the same object in the world, which vary in size. For example, size of a car varies from the size of smallest mini cars to the size of biggest SUVs. Therefore, we need to model the size of cars with a probability distribution. In this chapter, we argue that the sizes of object instances are taken from a log-normal distribution specific to the object type i.e., the logarithm of sizes are taken from a normal distribution. This is different from what has been used in the previous work in NLP [26, 5] where the sizes of objects are from a normal distribution. Our log-normal representation is more intuitive and works better in practice.

Let's assume the actual size of an apple comes from a normal distribution with $\mu = 5$ and $\sigma = 1$. There are two problems with this representation. First, the pdf is non-zero for $x \leq 0$, but physical objects cannot have negative sizes (probability mass leakage). Second, with this representation, the probability of finding an apple with a size less than 0.1 ($\frac{1}{50}$ of an average apple) is greater than finding an apple with a size greater than 10 (twice as big as an average apple). This is intuitively incorrect since we may find an apple twice as big as an average apple in grocery stores, but it is very unlikely to discover an apple with the size of $\frac{1}{50}$ of an average apple.

Using log-normal sizes would resolve both issues. Assume size of an apple comes from a log-normal distribution with parameters $\mu = \ln 5$ and $\sigma = 1$. With this assumption, the probability of finding an apple of negative size is zero. Also, the probability of finding an apple twice as big as an average apple is equal to seeing an apple whose size is half of an average apple. It is very interesting to see that the log-normal representation is aligned well with recent work in psychology that shows the visual size of the objects correlates with the log of their assumed size [74]. In addition, our experimental results demonstrate that the log-normal representation improves the previous work.

3.5 Learning Object Sizes

3.5.1 Collecting Observations

Visual Observations: We collect visual data to observe instances of relative sizes of objects. For each edge $e = (O_i, O_j)$ in the size graph, we download multiple images from Flickr that are tagged with both O_i and O_j and run the corresponding object detectors. These detectors are trained by a webly-supervised algorithm [32] to maintain scalability. Let box_1 and box_2 be the top predicted bounding boxes for the first and the second objects respectively. If the score of both predictions are above the default threshold of each detector, we record $r = \frac{area(box_1)}{area(box_2)} \times \frac{depth(box_1)^2}{depth(box_2)^2}$ as an observation for the relative size $\frac{size(O_i)}{size(O_j)}$, where $depth(box_i)$ is the average depth of box_i computed from the depth estimation of [33].

Textual Observations: We collect textual data to observe instances of absolute sizes of objects. In particular, we collect numerical values for the size of each object by executing search queries with the patterns of "[object] * x * [unit]", "[object] is * [unit] tall", and "[object] width is * [unit]". These patterns are taken from previous works in the NLP community [26, 5]. Each search result might contain multiple numerical results. We compute the geometric mean of the multiple numerical values within each search result. After scaling numerical results with respect to the unit used in each pattern we record them as observations for $size(O_i)$.

3.5.2 Learning

As discussed in section 3.4.2, we assume that log of object sizes comes from a normal distribution i.e., $g_i = \log size(O_i) \sim N(\mu_i, \sigma_i^2)$. The goal of the learning step is to find parameters μ_i and σ_i for every object O_i that maximizes the likelihood of the observations.

Let $x_{ij}^{(r)}$ denote the r_{th} binary visual observation for the relative size $\frac{size(O_i)}{size(O_j)}$, and let $x_i^{(r)}$ denote the r_{th} unary textual observation for $size(O_i)$. We define variables $y_{ij}^{(r)} = \log x_{ij}^{(r)}$ and $y_i^{(r)} = \log x_i^{(r)}$ as the logarithms of the observations $x_{ij}^{(r)}$ and $x_i^{(r)}$, respectively. This implies $y_i \sim g_i$ and $y_{ij} \sim g_i - g_j$. Assuming that the observations are independent, the likelihood of

all observations is as follows:

$$\prod_{(i,j)\in E}\prod_{r=1}^{n_{ij}} f(g_i - g_j = y_{ij}^{(r)} | g_i \sim N(\mu_i, \sigma_i^2), g_j \sim N(\mu_j, \sigma_j^2)) \times \prod_{i=1}^{n}\prod_{r=1}^{n_i} f(g_i = y_i^{(r)} | g_i \sim N(\mu_i, \sigma_i^2)) \tag{3.2}$$

where n is the total number of objects, n_i is the number of textual observations for the i'th node, n_{ij} is the total number of visual observations for the edge (O_i, O_j), and E is the set of edges in size graph. The first and the second production terms of equation 3.2 refer to the likelihood of the visual and textual observations, respectively. Recall that any linear combination of normal random variables is also a normal variable whose mean and variance are also linear combinations of the original means and variances. We use this property of normal distributions and define $g_{ij} = g_i - g_j \sim N(\mu_i - \mu_j, \sigma_i^2 + \sigma_j^2)$. Substituting $g_i - g_j$ by g_{ij}, the log likelihood is:

$$\sum_{(i,j)\in E}\sum_{r=1}^{n_{ij}} \log f(g_{ij} = y_{ij}^{(r)} | g_{ij} \sim N(\mu_i - \mu_j, \sigma_i^2 + \sigma_j^2)) + \sum_{i=1}^{n}\sum_{r=1}^{n_i} \log f(g_i = y_i^{(r)} | g_i \sim N(\mu_i, \sigma_i^2)) \tag{3.3}$$

We solve the above optimization by coordinate ascent. At each step we update parameters μ_i and σ_i from the values of other parameters, assuming all the other parameters are fixed. For μ_i there is a closed form update rule; however, there is no closed form update for σ_i. To update σ_i, we do gradient ascent with the learning rate η. The update rule for μ_i and σ_i, assuming all the other parameters are fixed are:

$$\mu_i = \frac{\sum_{j:(i,j)\in E}\sum_{r=1}^{n_{ij}} \frac{y_{ij}^{(r)} + \mu_j}{\sigma_i^2 + \sigma_j^2} + \sum_{r=1}^{n_i} \frac{y_i^{(r)}}{\sigma_i^2}}{\sum_{j:(i,j)\in E} \frac{n_{ij}}{\sigma_i^2 + \sigma_j^2} + \frac{n_i}{\sigma_i^2}} \tag{3.4}$$

$$\sigma_i^{(t+1)} = \sigma_i^{(t)} + \eta\Bigg(\sum_{j:(i,j)\in E} \Big(\sum_{r=1}^{n_{ij}} \frac{\sigma_i^{(t)}(y_{ij}^{(r)} + \sigma_j - {\sigma_i^{(t)}}^2)}{({\sigma_i^{(t)}}^2 + \sigma_j^2)} - \frac{n_{ij}\sigma_i^{(t)}}{{\sigma_i^{(t)}}^2 + \sigma_j^2} \Big) + \sum_{r=1}^{n_i} \frac{(y_i^{(r)} - \mu_i)^2}{{\sigma_i^{(t)}}^3} - \frac{n_i}{\sigma_i^{(t)}} \Bigg) \tag{3.5}$$

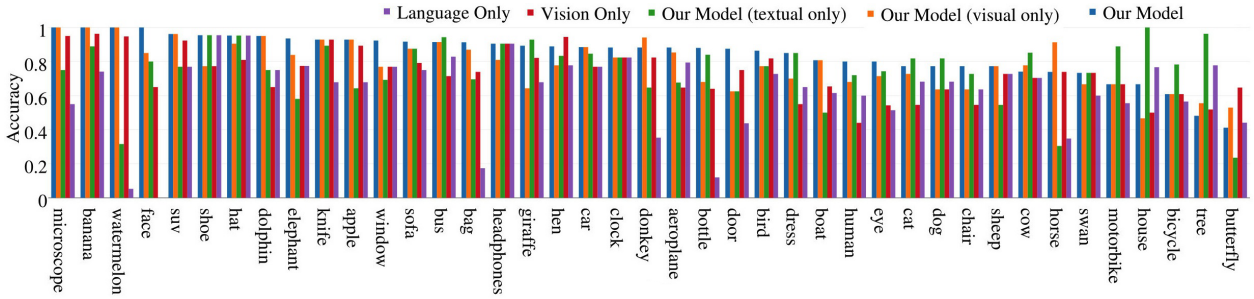


Figure 3.2: The accuracy of models for objects in our dataset. Objects are sorted by the accuracy of our model.

The log likelihood (equation 3.3) is not convex. As a result, the coordinate ascent converges to a local optima depending on the initialization of the parameters. The non-convexity is due to the first production term; the second production term is convex. In practice, we initialize μ_i and σ_i with the mean and the standard deviation of $Y_i = \{y_i^{(r)} | 1 \leq r \leq n_i\}$. These values maximizes the second production term.

3.5.3 Inference

After learning the parameters μ_i and σ_i for all objects in our test set, we are able to infer if object O_i is bigger than O_j from the probability distributions of object sizes.

$$\begin{aligned} P(size(O_i) > size(O_j)) &= P(\log size(O_i) > \log size(O_j)) \\ &= P(\log size(O_i) - \log size(O_j) > 0) \end{aligned}$$

Any linear combination of normal distributions is also a normal distribution; hence, $P(\log size(O_i) - \log size(O_j) > 0) = P(g_{ij} > 0 | g_{ij} \sim N(\mu_i - \mu_j, \sigma_i^2 + \sigma_j^2)) = 1 - \Phi(\frac{\mu_j - \mu_i}{\sqrt{\sigma_i^2 + \sigma_j^2}})$, where $\Phi(x)$ is the cumulative distribution function of the standard normal distribution and can be approximated numerically [3, 50, 21, 94].

3.6 Experimental Setup

We use Flickr 100M dataset [130] as the source of tag lists needed to construct the size graph (Section 3.4.1). We model size graph as a 2-edge-connected subgraph since it is still sparse, the total cost of edges is small, and it does not get disconnected with the removal of an edge. For each edge (O_i, O_j) in the size graph, we retrieve a maximum of 100 images from Flickr. We collect visual observations from the retrieved images and prune the outliers. To collect textual observations for the nodes, we execute our set of patterns on Google Custom Search Engine (Section 3.5.1).

3.6.1 Dataset

We compile a dataset of size comparisons among different physical objects. The dataset includes annotations for a set of object pairs $(object_i, object_j)$ for which people agree that $size(object_i) > size(object_j)$. The list of objects are selected from the 4869 detectors in LEVAN [32] that correspond to 41 physical objects. To annotate the size comparisons, we deployed a webpage and asked annotators to answer queries of the form "Which one is bigger, $object_i$ or $object_j$?" and possible answers include three choices of $object_i$, $object_j$, or 'not obvious'. Annotators selected 'not obvious' for non-trivial comparisons such as "Which one is bigger, *bird* or *microscope*?".

We generated comparison surveys and asked each annotator 40 unique comparison questions. The annotators have shown to be consistent with each other on most of the questions (about 90% agreement). We only kept the pairs of objects that annotators have agreed and pruned out the comparisons with 'not obvious' answers. In total, there are 11 batches of comparison surveys and about 350 unique comparisons. To complete the list of annotated comparisons, we created a graph of all the available physical objects and added a directed edge from $object_i$ to $object_j$ if and only if people has annotated $object_i$ to be bigger than $object_j$. We verified that the generated graph is acyclic. We finally augmented the test set by adding all pairs of objects $(object_i, object_j)$ where there's a path from $object_i$ to $object_j$

Model	Accuracy
Chance	0.5
Language only	0.634
Vision only	0.724
Our model (textual only)	0.753
Our model (visual only)	0.784
Our model	**0.835**

Table 3.1: The accuracy of our model against baselines and ablations on estimating relative size comparisons. Our model outperforms competitive language-based and vision-based baselines by large margins. Our model benefits from both visual and textual information and outperforms language-only and vision-only ablations.

in the graph.

Our final dataset includes a total of 486 object pairs between 41 physical objects. On average, each object appears in about 24 comparison pairs where '*window*' with 13 pairs has the least, and '*eye*' with 35 pairs has the most number of pairs in the dataset.

3.6.2 Comparisons

Language-only baseline: This baseline is inspired by recent work in the NLP community [26, 5]. We re-implement the previous work by forming and executing search engine queries with the size patterns mentioned in section 3.5.1. For every query, we record a size value after scaling the numerical results with respect to their units. The size of each object is then modeled with a normal distribution over observations. Our experiments have shown that textual observations about the relative sizes of physical objects are very limited. It is unlikely to find a sentence that says a car is bigger than an orange. In addition, comparative statements in text, if found, rarely provide precisely how much one object is bigger than the other.

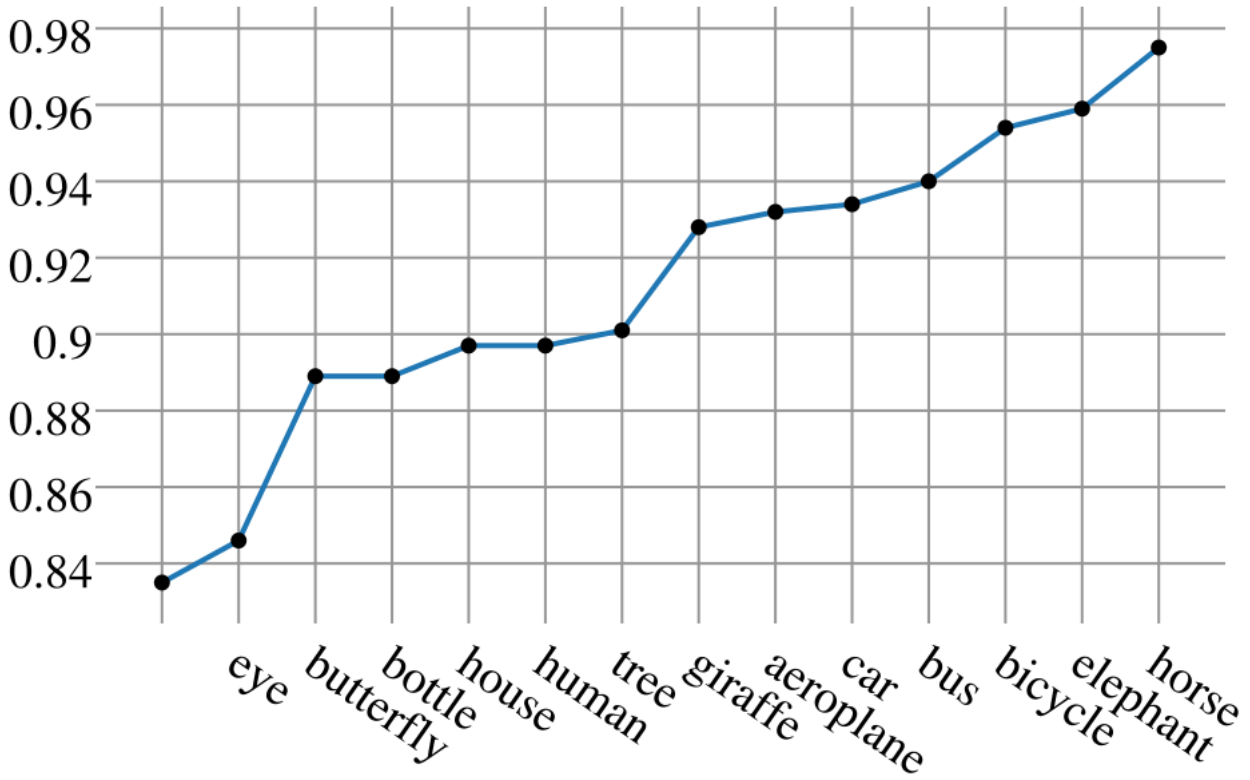


Figure 3.3: Our model can propagate information about true size of objects, if available. This figure shows an example case, where adding true estimates of the size information for about 10 objects results in near perfect size estimates.

Our model (textual only): This is a variant of our model that only uses textual observations. This model maximizes the second production term of log likelihood (equation 3.3).

Vision-only baseline: This baseline is built on using the relative size comparisons directly taken from the visual data. To compare the size of two objects, this model finds the shortest path in the complete graph between the two objects (with edge costs defined in Section 3.4.1). For each edge in the path, we collect visual observations (Section 3.5.1) and set the final relative size of two objects as the geometric mean of all the observations. To compute the relative size between pairs objects, we multiply all the relative sizes of object pairs in the shortest path between them. Doing geometric mean is according to the fact that sizes

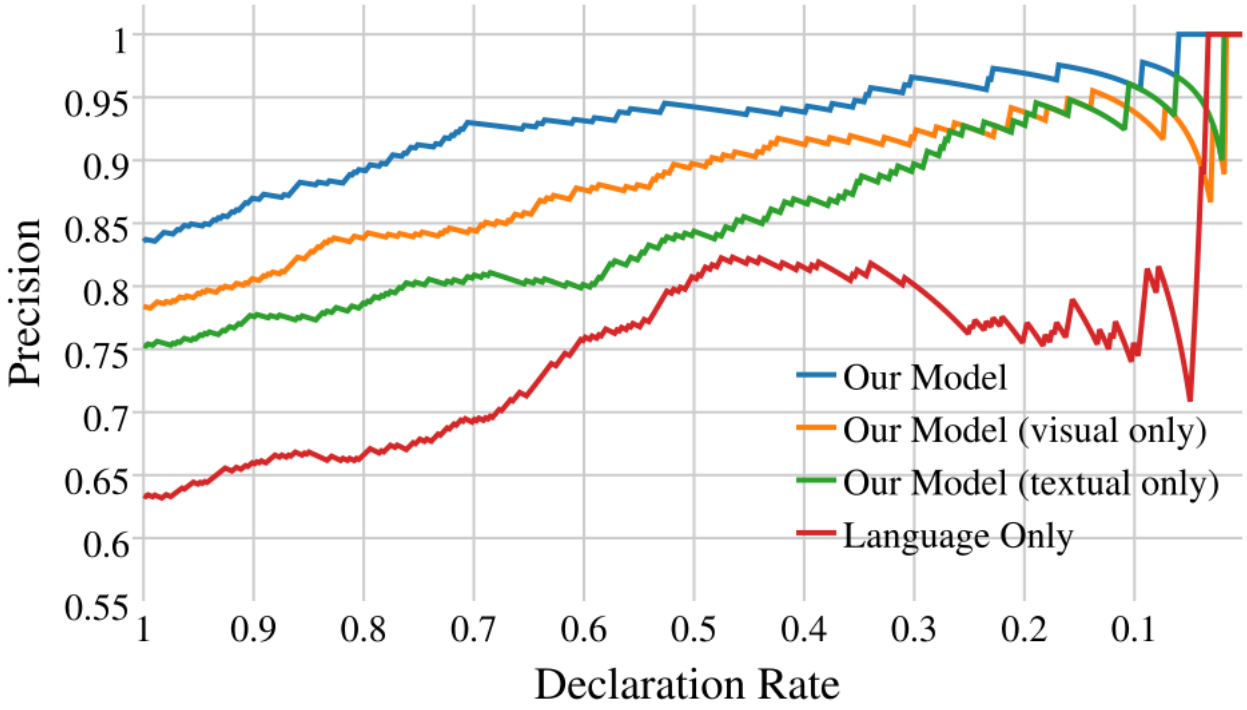


Figure 3.4: Precision vs. declaration rate in estimating the relative size information in our dataset. The curves are traced out by thresholding on $|P(A > B) - 0.5|$. Our model outperforms baselines in all declaration rates.

(and therefore ratios) are taken from log-normal distributions.

Our model (visual only): This is a variant of our model that only uses visual observations. This model maximizes the first production term of log likelihood (equation 3.3). The difference between this model and vision-only baseline is on the representation (using size graph instead of complete graph) and also maximizing the likelihood, which involves observations altogether to estimate the objects' size distributions, instead of relying only on the shortest path edges.

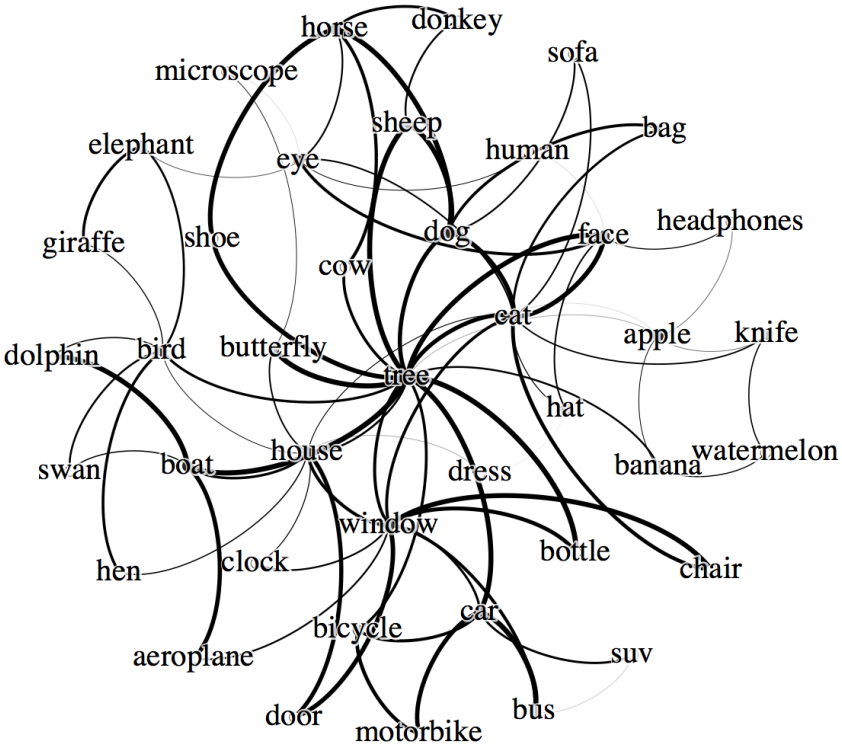


Figure 3.5: The size graph: The thickness of each edge represents the number of images in which both objects are detected successfully (the more, the bolder). The topology of the size graph reveals interesting properties about transitivity of the size information. For example, the size of chairs would be mainly affected by the estimates of the size of cats while the size of trees are affected by several other objects.

3.7 Results

Overall Accuracy in Size Comparisons: We report the accuracy of our model in inferring relative size comparisons in our dataset in Figure 3.1. For inference, we compute $P(size(A) > size(B))$ (Section 3.5.3) and infer A is bigger than B if and only if $P(size(A) > size(B)) > 0.5$. The accuracy is the number of correctly inferred pairs over all the pairs in the dataset. Figure 3.1 shows the results of our model versus the baselines and our model's variants defined in section 3.6.2.

Our model achieves significant improvement over all the other models. The results con-

firm that visual and textual information are complementary and our model can take advantage of both modalities. In addition, our model (textual only) achieves significantly higher performance compared to the language-only baseline. This supports the superiority of our representation that sizes are represented with log-normal distributions. Finally, our model (visual only) achieves significantly higher accuracy compared to the vision-only baseline. This confirms that maximizing the likelihood removes the noise and inaccuracies that exist in individual visual observations.

Per-object Accuracy in Size Comparisons: Figure 3.2 shows the accuracy of our model in inferring relative sizes for all objects. These results show that for most objects our model achieves higher accuracy than the baselines, confirming that visual and textual observations are complementary. For some objects like *giraffe*, *motorbike*, and *house* the textual data are less noisy and contribute more to the accuracy of our model. However, for some other objects like *watermelon*, *apple*, and *donkey* the visual data is more informative about the objects sizes.

To summarize, the accuracy of the joint optimization is higher than the accuracy of optimizing the likelihood for visual or textual data alone for most of the objects. Moreover, our model (visual only) outperforms the vision-only baseline, and our model (textual only) outperforms the language-only baseline in most of the objects.

Precision vs. Declaration Rate: All the models listed in Figure 3.1 (except the vision-only model) estimate the size of objects with a probability distribution. We infer A is bigger than B if and only if $P(size(A) > size(B)) > 0.5$. The difference between the probability $P(size(A) > size(B))$ and 0.5 represents the confidence of the estimation i.e., for a query pair (A, B), the confidence is $|P(size(A) > size(B)) - 0.5|$.

Figure 3.4 shows the precision of the models vs. declaration rate [156]. Declaration rate is the proportion of the of the test queries on which the model outputs a decision. To calculate precision at a specific declaration rate dr, we first sort the queries in ascending order of each model's confidence, and then report precision over top dr proportion of the test queries and

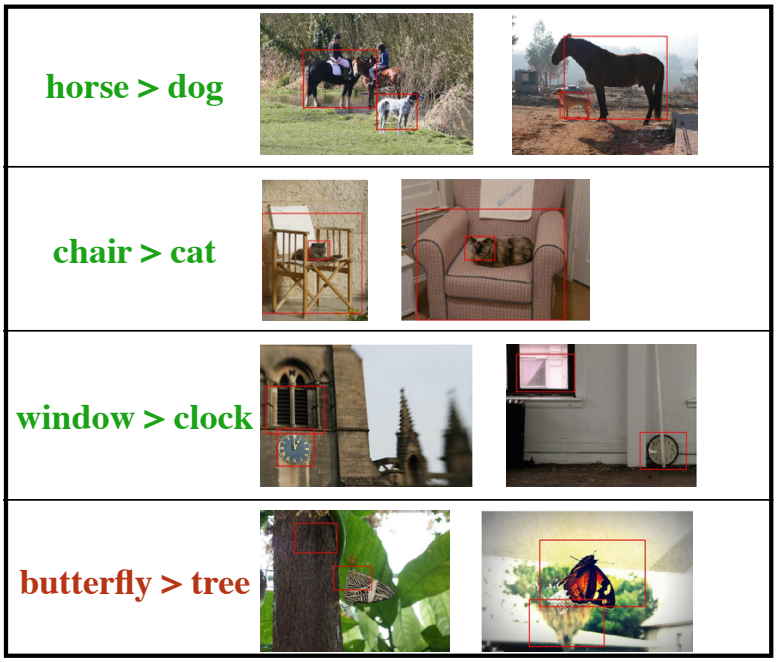


Figure 3.6: Examples of visual observations for the edges of the size graph. Co-occurrence of objects in images provide visual signal to estimate relative sizes of objects. Examples of relative size comparisons is shown in this figure. Erroneous detections (e.g. the tree in the bottom row) results in wrong relative size estimates.

discard the rest. Our results show that our model consistently outperforms other models at all declaration rates. It is worth mentioning that the precision of the language-only model drops at high confidence region ($dr > 0.5$), suggesting that the probabilistic model of this baseline is inaccurate.

Sparse Supervision from True Object Sizes: For a small number of objects, one might posses reliable information about their size. Our model can incorporate these information by fixing the size estimates for those objects and optimize the log-likelihood (equation 3.3) with respect to other objects' parameters. Our model is able to propagate information about the true object sizes to the uncertain nodes. Figure 3.3 shows the growth of accuracy when the true values of few objects are provided. It is very interesting to see that the accuracy of

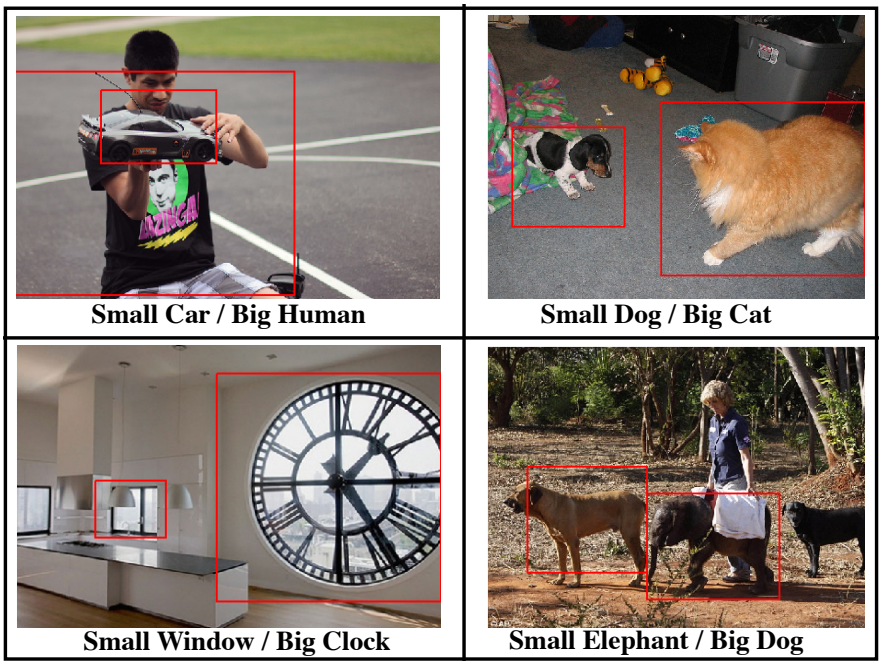


Figure 3.7: Relative size estimates can lead to rich inferences about deep semantics in images. For example, our model can produce statements such as big dog/small elephant, or big clock/small window to identify interesting events in images. Such capability is beneficial for both image captioning and referring expressions.

our model goes up by more than 5% by just adding the true size information of two noisy objects, "eye" and "butterfly".

Qualitative Results: Figure 3.5 shows the size graph constructed using our method (Section 3.4.1); the thickness of each edge represents the number of collected visual observations for that edge. It is interesting to see that the best way to visually estimate the size of a sofa is through dogs and cats. Figure 3.6 shows some examples of detection on the edges of the size graph (e.g., the 'horse' and the 'dog' detections used to establish their relative size). This figure also shows examples of erroneous detections that cause wrong relative size estimates (e.g. 'butterfly' and 'tree').

Size information is among important attributes for referring expressions [99] and can lead to inferences about what is worth mentioning for an image. Deviations from the expected

relative sizes of objects can be identified using our size estimates. For example, Figure 3.7 shows examples of objects with unexpected relative size estimates. Rich statements, such as big person/small car or big clock/small window in Figure 3.7 can be used in generating descriptions for image or even pruning false positives in object detection.

3.8 Conclusion

In this chapter, we introduce a fully automated method to infer information about sizes of objects using both visual and textual information available on the web. With scalability in mind, our approach does not require any explicit human supervision. We evaluate our method on estimates of relative sizes of objects and show significant gain over competitive textual and visual baselines. We introduced size graph and show its benefits in leveraging transitive nature of the size problem.

The visual size of objects is a function of different cues including the distance, camera parameters, view point, etc. An ideal solution to the size problem involves reasoning in 3D space where all the unknown parameters are given. Unfortunately, such information is not available at scale. Per image reasoning cancels out the effects of camera parameters, and noisy depth estimates seem to be reliable for size estimation. We hypothesize that the canonical viewpoint phenomena among pictures on the web, specially Flickr, relaxes the need for accurate estimation of viewpoint.

Future work involves application of inferred size information in mainstream vision tasks, specifically object detection and single image depth estimation. This chapter is a step toward the important problem of inferring the size information and can confidently declare that, yes, *elephants are bigger than butterflies!*

Chapter 4

LOOKUP-BASED CONVOLUTIONAL NEURAL NETWORKS

Porting state of the art deep learning algorithms to resource constrained compute platforms (e.g. VR, AR, wearables) is extremely challenging. We propose a fast, compact, and accurate model for convolutional neural networks that enables efficient learning and inference. We introduce LCNN, a lookup-based convolutional neural network that encodes convolutions by few lookups to a dictionary that is trained to cover the space of weights in CNNs. Training LCNN involves jointly learning a dictionary and a small set of linear combinations. The size of the dictionary naturally traces a spectrum of trade-offs between efficiency and accuracy. Our experimental results on ImageNet challenge show that LCNN can offer 3.2× speedup while achieving 55.1% top-1 accuracy using AlexNet architecture. Our fastest LCNN offers 37.6× speed up over AlexNet while maintaining 44.3% top-1 accuracy. LCNN not only offers dramatic speed ups at inference, but it also enables efficient training. In this chapter, we show the benefits of LCNN in few-shot learning and few-iteration learning, two crucial aspects of on-device training of deep learning models.

4.1 Introduction

In recent years convolutional neural networks (CNN) have played major roles in improving the state of the art across a wide range of problems in computer vision, including image classification [77, 122, 127, 52], object detection [40, 39, 115], segmentation [108, 92], etc. These models are very expensive in terms of computation and memory. For example, AlexNet[77] has 61M parameters and performs 1.5B high precision operations to classify a single image. These numbers are even higher for deeper networks, *e.g.,*VGG [122]. The computational burden of learning and inference for these models is significantly higher than what most

compute platforms can afford.

Recent advancements in virtual reality (VR by Oculus) [103], augmented reality (AR by HoloLens) [43], and smart wearable devices increase the demand for getting our state of the art deep learning algorithm on these portable compute platforms. Porting deep learning methods to these platforms is challenging mainly due to the gap between what these platforms can offer and what our deep learning methods require. More efficient approaches to deep neural networks is the key to this challenge.

Recent work on efficient deep learning have focused on model compression and reducing the computational precision of operations in neural networks [18, 47, 111]. CNNs suffer from over-parametrization [30] and often encode highly correlated parameters [67], resulting in inefficient computation and memory usage[30]. Our key insight is to leverage the correlation between the parameters and represent the space of parameters by a compact set of weight vectors, called dictionary. In this chapter, we introduce LCNN, a lookup-based convolutional neural network that encodes convolutions by few lookups to a dictionary that is trained to cover the space of weights in CNNs. Training LCNN involves jointly learning a dictionary and a small set of linear combinations. The size of the dictionary naturally traces a spectrum of trade-offs between efficiency and accuracy. Our experimental results using AlexNet on ImageNet challenge show that LCNN can offer 3.2× speedup while achieving 55.1% top-1 accuracy. Our fastest LCNN offers 37.6× speed up over CNN while maintaining 44.3% top-1 accuracy. In the ResNet-18, the most accurate LCNN offers 5× speedup with 62.2% accuracy and the fastest LCNN offers 29.2× speedup with 51.8% accuracy

In addition, LCNN enables efficient training; almost all the work in efficient deep learning have focused on efficient inference on resource constrained platforms [111]. Training on these platforms is even more challenging and requires addressing two major problems: i. **few-shot learning:** the settings of on-device training dictates that there won't be enough training examples for new categories. In fact, most training needs to be done with very few training examples; ii. **few-iteration learning:** the constraints in computation and power require the training to be light and quick. This imposes hard constraints on the number of iterations in

training.LCNN offers solutions for both of these problems in deep on-device training.

Few-shot learning, the problem of learning novel categories from few examples (sometimes even one example), have been extensively studies in machine learning and computer vision[36]. The topic is, however, relatively new for deep learning[49], where the main challenge is to avoid overfitting. The number of parameters are significantly higher than what can be learned from few examples. LCNN, by virtue of having fewer parameters to learn (only around 7% of parameters of typical networks), offers a simple solution to this challenge. Our dictionary can be learned offline from training data where enough training examples per category exists. When facing new categories, all we need to learn is the set of sparse reconstruction weights. Our experimental evaluations show significant gain in few-shot learning; 6.3% in one training example per category.

Few-iteration learning is the problem of getting highest possible accuracy in few iterations that a resource constrained platform can offer. In a typical CNN, training often involves hundreds of thousands of iterations. This number is even higher for recent deeper architectures. LCNN offers a solution: dictionaries in LCNN are architecture agnostic and can be transferred across architectures or layers. This allows us to train a dictionary using a shallow network and transfer it to a deeper one. As before, all we need to learn are the few reconstruction weights; dictionaries don't need to be trained again. Our experimental evaluations on ImageNet challenge show that using LCNN we can train an 18-layer ResNet with a pre-trained dictionary from a 10-layer ResNet and achieve 16.2% higher top-1 accuracy on $10K$ iterations.

In this chapter, we 1) introduce LCNN; 2) show state of the art efficient inference in CNNs using LCNN; 3) demonstrate possibilities of training deep CNNs using as few as one example per category 4) show results for few iteration learning .

4.2 Related Work

A wide range of methods have been proposed to address efficient training and inference in deep neural networks. Here, we briefly study these methods under the topics that are related

to our approach.

Weight compression: Several attempts have been made to reduce the number of parameters of deep neural networks. Most of such methods [42, 154, 18, 47, 124] are based on compressing the fully connected layers, which contain most of the weights. These methods do not achieve much improvement on speed. In [62], a small DNN architecture is proposed which is fully connected free and has 50x fewer parameters in compare to AlexNet [77]. However, their model is slower than AlexNet. Recently [48, 47] reduced the number of parameters by pruning. All of these approaches update a pre-trained CNN, whereas we propose to train a compact structure that enables faster inference.

Low Rank Assumption: Approximating the weights of convolutional layers with low-rank tensor expansion has been explored by [67, 30]. They only demonstrated speedup in the case of large convolutions. [31] uses SVD for tensor decomposition to reduce the computation in the lower layers on a pre-trained CNN. [157] minimizes the reconstruction error of the nonlinear responses in a CNN, subject to a low-rank constraint which helps to reduce the complexity of filters. Notably, all of these methods are a post processing on the weights of a trained CNN, and none of them train a lower rank network from scratch.

Low Precision Networks: A fixed-point implementation of 8-bit integer was compared with 32-bit floating point activations in [133, 61]. Several network quantization methods are proposed by [42, 4, 87, 87, 60]. Most recently, binary networks has shown to achieve relatively strong result on ImageNet [111]. They have trained a network that computes the output with mostly binary operations, except for the first and the last layer. [25] uses the real-valued version of the weights as a key reference for the binarization process. [24] is an extension of [25], where both weights and activations are binarized. [70] retrains a previously trained neural network with binary weights and binary inputs. Our approach is orthogonal to this line of work. In fact, any of these methods can be applied in our model to reduce the precision.

Sparse convolutions: Recently, several attempts have been made to sparsify the weights of convolutional layers [89, 148, 142]. [89] shows how to reduce the redundancy in parameters

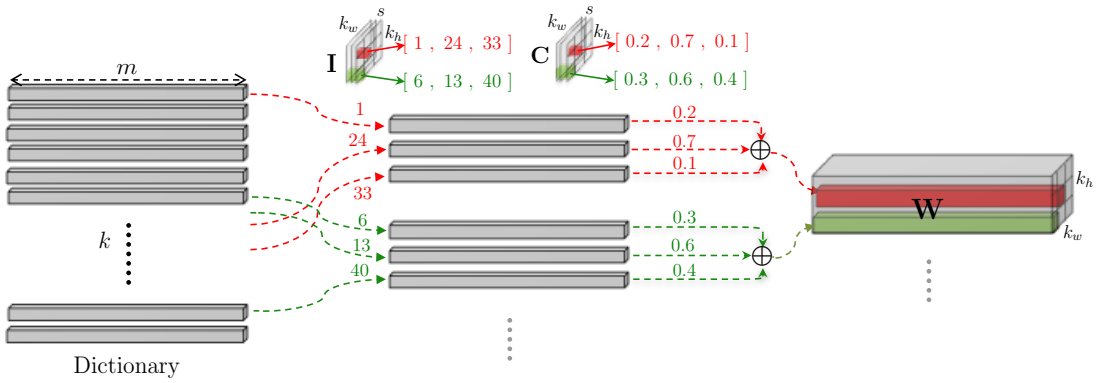


Figure 4.1: This figure demonstrates the procedure for constructing a weight filter in LCNN. A vector in the weight filter (the long colorful cube in the gray tensor **W**) is formed by a linear combination of few vectors, which are looked up from the dictionary **D**. Lookup indices and their coefficients are stored in tensors **I** and **C**.

of a CNN using a sparse decomposition. [148] proposed a framework to simultaneously speed up the computation and reduce the storage of CNNs. [142] proposed a Structured Sparsity Learning (SSL) method to regularize the structures (i.e., filters, channels, filter shapes, and layer depth) of CNNs. Only in [142] a sparse CNN is trained from scratch which makes it more similar to our approach. However, our method provides a rich set of dictionary that enables implementing convolution with lookup operations.

Few-Shot Learning: The problem of learning novel categories has been studied in [131, 7, 88]. Learning from few examples per category explored by [49]. [36, 134, 73] proposed a method to learn from one training example per category, known as one-shot learning. Learning without any training example, zero-shot learning, is studied by [80, 83].

4.3 Our Approach

Overview: In a CNN, each convolutional layer consists of n cubic weight filters of size $m \times k_w \times k_h$, where m and n are the number of input and output channels, respectively,

and k_w and k_h are the width and the height of the filter. Therefore, the weights in each convolutional layer is composed of nk_wk_h vectors of length m. These vectors are shown to have redundant information[30]. To avoid this redundancy, we build a relatively small set of vectors for each layer, to which we refer as dictionary, and enforce each vector in the weight filter to be a linear combination of a few elements from this set. Figure 4.1 shows an overview of our model. The gray matrix at the left of the figure is the dictionary. The dashed lines show how we lookup a few vectors from the dictionary and linearly combine them to build up a weight filter. Using this structure, we devise a fast inference algorithm for CNNs. We then show that the dictionaries provide a strong prior on the visual data and enables us to learn from few examples. Finally, we show that the dictionaries can be transferred across different network architectures. This allows us to speedup the training of a deep network by transferring the dictionaries from a shallower model.

4.3.1 LCNN

A convolutional layer in a CNN consists of four parts: 1) the input tensor $\mathbf{X} \in^{m \times w \times h}$; where m, w and h are the number of input channels, the width and the height, respectively, 2) a set of n weight filters, where each filter is a tensor $\mathbf{W} \in^{m \times k_w \times k_h}$, where k_w and k_h are the width and the height of the filter, 3) a scalar bias term $b \in$ for each filter, and 4) the output tensor $\mathbf{Y} \in^{n \times w' \times h'}$; where each channel $\mathbf{Y}_{[i,:,:]} \in^{w' \times h'}$ is computed by $\mathbf{W} * \mathbf{X} + b$. Here $*$ denotes the discrete convolution operation[1].

For each layer, we define a matrix $\mathbf{D} \in^{k \times m}$ as the shared dictionary of vectors. This is illustrated in figure 4.1, on the left side. This matrix contains k row vectors of length m. The size of the dictionary, k, might vary for different layers of the network, but it should always be smaller than nk_wk_h, the total number of vectors in all weight filters of a layer. Along with the dictionary $\mathbf{D}$, we have a tensor for lookup indices $\mathbf{I} \in \mathbb{N}_{\leq k}^{s \times k_w \times k_h}$, and a tensor for lookup coefficients $\mathbf{C} \in^{s \times k_w \times k_h}$ for each layer. For a pair (r, c), $\mathbf{I}_{[:,r,c]}$ is a vector of length s whose

[1]The (:) notation is borrowed from NumPy for selecting all entries in a dimension.

entries are indices of the rows of the dictionary, which form the linear components of $\mathbf{W}_{[:,r,c]}$. The entries of the vector $\mathbf{C}_{[:,r,c]}$ specify the linear coefficients with which the components should be combined to make $\mathbf{W}_{[:,r,c]}$ (illustrated by a long colorful cube inside the gray cub in Figure 4.1-right). We set s, the number of components in a weight filter vector, to be a small number. The weight tensor can be constructed as follows:

$$\mathbf{W}_{[:,r,c]} = \sum_{t=1}^{s} \mathbf{C}_{[t,r,c]} \cdot \mathbf{D}_{[\mathbf{I}_{[t,r,c]},:]} \quad \forall r, c \tag{4.1}$$

This procedure is illustrated in Figure 4.1. In LCNN, instead of storing the weight tensors $\mathbf{W}$ for convolutional layers, we store $\mathbf{D}$, $\mathbf{I}$ and $\mathbf{C}$, the building blocks of the weight tensors. As a result, we can reduce the number of parameters in a convolutional layer by reducing k, the dictionary size, and s, the number of components in the linear combinations. In the next section, we will discuss how LCNN uses this representation to speedup the inference.

Fast Convolution using a Shared Dictionary

A forward pass in a convolutional layer consists of n convolutions between the input $\mathbf{X}$ and each of the weight filters $\mathbf{W}$. We can write a convolution between an $m \times k_w \times k_h$ weight filter and the input $\mathbf{X}$ as a sum of $k_w k_h$ separate (1×1)-convolutions:

$$\mathbf{X} * \mathbf{W} = \sum_{r,c}^{k_h,k_w} \text{shift}_{r,c}(\mathbf{X} * \mathbf{W}_{[:,r,c]}) \tag{4.2}$$

, where $\text{shift}_{r,c}$ is the matrix shift function along rows and columns with zero padding relative to the filter size. Now we use the LCNN representation of weights (equation 4.1) to rewrite each 1×1 convolution:

$$\begin{aligned} \mathbf{X} * \mathbf{W} &= \sum_{r,c} \text{shift}_{r,c}(\mathbf{X} * (\sum_{t=1}^{s} \mathbf{C}_{[t,r,c]} \cdot \mathbf{D}_{[\mathbf{I}_{[t,r,c]},:]})) \\ &= \sum_{r,c} \text{shift}_{r,c}(\sum_{t=1}^{s} \mathbf{C}_{[t,r,c]} (\mathbf{X} * \mathbf{D}_{[\mathbf{I}_{[t,r,c]},:]})) \end{aligned} \tag{4.3}$$

Equation 4.3 suggests that instead of reconstructing the weight tensor $\mathbf{W}$ and convolving with the input, we can convolve the input with all of the dictionary vectors, and then compute the output according to $\mathbf{I}$ and $\mathbf{C}$. Since the dictionary $\mathbf{D}$ is shared among all weight filters

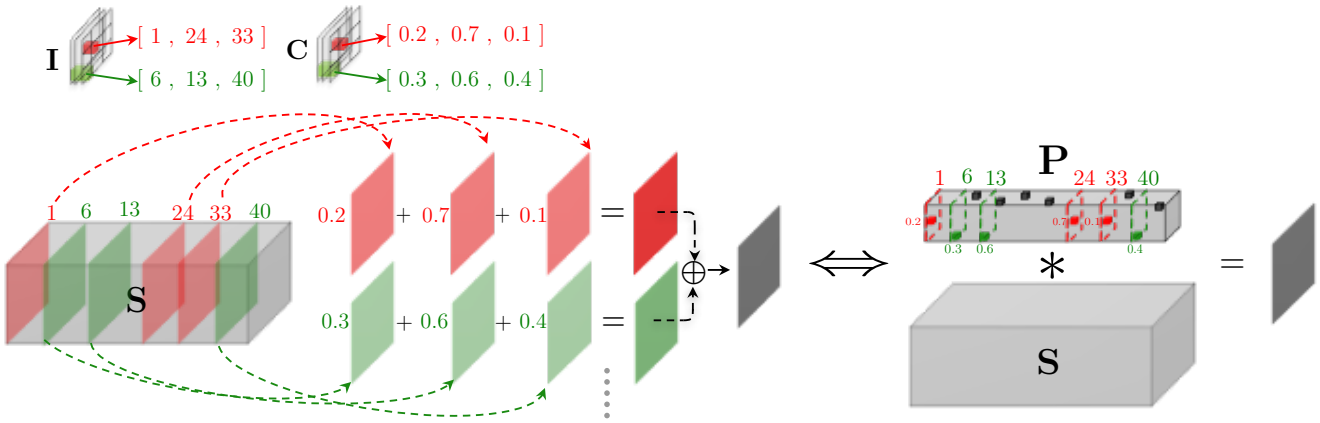


Figure 4.2: **S** is the output of convolving the dictionary with the input tensor. **The left side** of this figure illustrates the inference time forward pass. The convolution between the input and a weight filter is carried out by lookups over the channels of **S** and a few linear combinations. Direct learning of tensors **I** and **C** reduces to an intractable discrete optimization. **The right side** of this figure shows an equivalent computation for training based on sparse convolutions. Parameters **P** can be trained using SGD. The tiny cubes in **P** denote the non-zero entries.

in a layer, we can pre-compute the convolution between the input tensor **X** and all the dictionary vectors. Let $\mathbf{S} \in^{k \times w \times h}$ be the output of convolving the input **X** with all of the dictionary vectors **D**, i.e.,

$$\mathbf{S}_{[i,:,:]} = \mathbf{X} * \mathbf{D}_{[i,:]} \quad \forall 1 \leq i \leq k \tag{4.4}$$

Once the values of **S** are computed, we can reconstruct the output of convolution by *lookups* over the channels of **S** according to **I**, then *scale* them by the values in **C**:

$$\mathbf{X} * \mathbf{W} = \sum_{r,c}^{k_h,k_w} \text{shift}_{r,c}(\sum_{t=1}^{s} \mathbf{C}_{[t,r,c]} \mathbf{S}_{[\mathbf{I}_{[t,r,c]},:,:]}) \tag{4.5}$$

This is shown in Figure 4.2 (left). Reducing the size of the dictionary k lowers the cost of computing **S** and makes the forward pass faster. Since **S** is computed by a dense matrix

multiplication, we are still able to use OpenBlas [141] for fast matrix multiplication. In addition, by pushing the value of s to be small, we can reduce the number of lookups and floating point operations.

Training LCNN

So far we have discussed how LCNN represents a weight filter by linear combinations of a subset of elements in a shared dictionary. We have also shown that how LCNN performs convolutions efficiently in two stages: 1- *Small convolutions*: convolving the input with a set of 1×1 filters (equation 4.4). 2- *Lookup and scale*: few lookups over the channels of a tensor followed by a linear combination (equation 4.5) . Now, we explain how one can jointly train the dictionary and the lookup parameters, $\mathbf{I}$ and $\mathbf{C}$. Direct training of the proposed lookup based convolution leads to a combinatorial optimization problem, where we need to find the optimal values for the integer tensor $\mathbf{I}$. To get around this, we reformulate the lookup and scale stage (equation 4.5) using a standard convolution with sparsity constraints.

Let $\mathbf{T} \in^{k \times k_w \times k_h}$ be a one hot tensor, where $\mathbf{T}_{[t,r,c]} = 1$ and all other entries are zero. It is easy to observe that convolving the tensor $\mathbf{S}$ with $\mathbf{T}$ will result in $\text{shift}_{r,c}(\mathbf{S}_{[t,:,:]})$. We use this observation to convert the lookup and scale stage (equation 4.5) to a standard convolution. Lookups and scales can be expressed by a convolution between the tensor $\mathbf{S}$ and a sparse tensor $\mathbf{P}$, where $\mathbf{P} \in^{k \times w \times h}$, and $\mathbf{P}_{[:,r,c]}$ is a s-sparse vector (*i.e.* it has only s non-zero entries) for all spatial positions (r, c). Positions of the non-zero entries in $\mathbf{P}$ are determined by the index tensor $\mathbf{I}$ and their values are determined by the coefficient tensor $\mathbf{C}$. Formally, tensor $\mathbf{P}$ can be expressed by $\mathbf{I}$ and $\mathbf{C}$:

$$\mathbf{P}_{j,r,c} = \begin{cases} \mathbf{C}_{t,r,c}, & \exists t : \mathbf{I}_{t,r,c} = j \\ 0, & \text{otherwise} \end{cases} \tag{4.6}$$

Note that this conversion is reversible, *i.e.,*we can create $\mathbf{I}$ and $\mathbf{C}$ from the position and the values of the non-zero entries in $\mathbf{P}$. With this conversion, the lookup and scale stage

(equation 4.5) becomes:

$$\sum_{rc} \text{shift}_{(r,c)}(\sum_{t=1}^{s} \mathbf{C}_{[t,r,c]} \mathbf{S}_{[\mathbf{I}_{[t,r,c]},:,:]}) = \mathbf{S} * \mathbf{P} \tag{4.7}$$

This is illustrated in Figure 4.2-right. Now, instead of directly training $\mathbf{I}$ and $\mathbf{C}$, we can train the tensor $\mathbf{P}$ with ℓ_0-norm constraints ($\|\mathbf{P}_{[:,r,c]}\|_{\ell_0} = s$) and then construct $\mathbf{I}$ and $\mathbf{C}$ from $\mathbf{P}$. However, ℓ_0-norm is a non-continuous function with zero gradients everywhere. As a workaround, we relax it to ℓ_1-norm. At each iteration of training, to enforce the sparsity constraint for $\mathbf{P}_{[:,r,c]}$, we sort all the entries by their absolute values and keep the top s entries and zero out the rest. During training, in addition to the classification loss L we also minimize $\sum_{[r,c]} \|\mathbf{P}_{[:,r,c]}\|_{\ell_1} = \|\mathbf{P}\|_{\ell_1}$, by adding a term $\lambda\|\mathbf{P}\|_{\ell_1}$ to the loss function. The gradient with respect to the values in $\mathbf{P}$ is computed by:

$$\frac{\partial(L + \lambda \|\mathbf{P}\|_{\ell_1})}{\partial \mathbf{P}} = \frac{\partial L}{\partial \mathbf{P}} + \lambda \text{ sign}(\mathbf{P}) \tag{4.8}$$

where $\frac{\partial L}{\partial \mathbf{P}}$ is the gradient that is computed through a standard back-propagation. λ is a hyperparameter that adjusts the trade-off between the CNN loss function and the ℓ_1 regularizer. We can also allow s, the sparsity factor, to be different at each spatial position (r, c), and be determined automatically at training time. This can be achieved by applying a threshold function,

$$\delta(x) = \begin{cases} x, & |x| > \epsilon \\ 0, & \text{otherwise} \end{cases} \tag{4.9}$$

over the values in $\mathbf{P}$ during training. We also backpropagate through this threshold function to compute the gradients with respect to $\mathbf{P}$. The derivative of the threshold function is 1 everywhere except at $|x| < \epsilon$, which is 0. Hence, if any of the entries of $\mathbf{P}$ becomes 0 at some iteration, they stay 0 forever. Using the threshold function, we let each vector to be a combination of arbitrary vectors. At the end of the training, the sparsity parameter s at each spatial position (r, c) is determined by the number of non-zero values in $\mathbf{P}[:, r, c]$.

Although the focus of our work is to speedup convolutional layers where most of the computations are, our lookup based convolution model can also be applied on fully connected (FC) layers. An FC layer that goes from m inputs to n outputs can be viewed as a convolutional layer with input tensor $m \times 1 \times 1$ and n weight filters, each of size $m \times 1 \times 1$. We take the same approach to speedup fully connected layers.

After training, we convert $\mathbf{P}$ to the indices and the coefficients tensors $\mathbf{I}$ and $\mathbf{C}$ for each layer. At test time, we follow equation 4.5 to efficiently compute the output of each convolutional layer.

4.3.2 Few-shot learning

The shared dictionary in LCNN allows a neural network to learn from very few training examples on novel categories, which is known as few-shot learning[49]. A good model for few-shot learning should have two properties: a) strong priors on the data, and b) few trainable parameters. LCNN has both of these properties. An LCNN trained on a large dataset of images (e.g. ImageNet [28]) will have a rich dictionary $\mathbf{D}$ at each convolutional layer. This dictionary provides a powerful prior on visual data. At the time of fine-tuning for a new set of categories with few training examples, we only update the coefficients in $\mathbf{C}$. This reduces the number of trainable parameters significantly.

In a standard CNN, to use a pre-trained network to classify a set of novel categories, we need to reinitialize the classification layer randomly. This introduces a large number of parameters, on which we don't have any prior, and they should be trained solely by a few examples. LCNN, in contrast, can use the dictionary of the classification layer of the pre-trained model, and therefore only needs to learn $\mathbf{I}$ and $\mathbf{C}$ from scratch, which form a much smaller set of parameters. Furthermore, for all other layers, we only fine-tune the coefficients $\mathbf{C}$, *i.e.,*only update the non-zero entries of $\mathbf{P}$. Note that the dictionary $\mathbf{D}$ is fixed across all layers during the training with few examples.

4.3.3 Few-iteration learning

Training very deep neural networks are computationally expensive and require hundreds of thousands of iterations. This is mainly due to the complexity of these models. In order to constrain the complexity, we should limit the number of learnable parameters in the network. LCNN has a suitable setting that allows us to limit the number of learnable parameters without changing the architecture. This can be done by transferring the shared dictionaries $\mathbf{D}$ from a shallower network to a deeper one.

Not only we can share a dictionary $\mathbf{D}$ across layers, but we can also share it across different network architectures of different depths. A dictionary $\mathbf{D} \in^{m \times k}$ can be used in any convolutional layer with input channel size m in any CNN architecture. For example, we can train our dictionaries on a shallow CNN and reuse in a deeper CNN with the same channel size. On the deeper CNN we only need to train the indices and coefficients tensors $\mathbf{I}$ and $\mathbf{C}$.

4.4 Experiments

We evaluate the accuracy and the efficiency of LCNN under different settings. We first evaluate the accuracy and speedup of our model for the task of object classification, evaluated on the standard image classification challenge of ImageNet, ILSRVC2012 [28]. We then evaluate the accuracy of our model under few-shot setting. We show that given a set of novel categories with as small as 1 training example per category, our model is able to learn a classifier that is both faster and more accurate than the CNN baseline. Finally we show that the dictionaries trained in LCNN are generalizable and can be transferred to other networks. This leads to a higher accuracy in small number of iterations compared to standard CNN.

---

[2]They have not reported the overall speedup on AlexNet, but only per layer speedup. 3.1× is the weighted average of their per layer speedups.

[3]XNOR-Net gets 32× layer-wise speedup on a 32 bit machine. However, since they haven't binarized the first and the last layer (which has 9.64% of the computation), their overall speedup is 8.0×.

	AlexNet		
Model	**speedup**	**top-1**	**top-5**
CNN	1.0×	56.6	80.2
Wen *et. al.*[142]	3.1×[2]	**55.4**	N/A
XNOR-Net[111]	8.0×[3]	44.2	69.2
LCNN-fast	**37.6**×	44.3	68.7
LCNN-accurate	3.2×	**55.1**	**78.1**

Table 4.1: Comparison of different efficient methods on AlexNet. The accuracies are classification accuracy on the validation set of ILSVRC2012.

4.4.1 Implementation Details

We follow the common way of initializing the convolutional layers by Gaussian distributions introduced in [41], including for the sparse tensor $\mathbf{P}$. We set the threshold in equation 4.9 for each layer in such a way that we maintain the same initial sparsity across all the layers. That is, we set the threshold of each layer to be $\epsilon = c \cdot \sigma$, where c is constant across layers and σ is the standard deviation of Gaussian initializer for that layer. We use $c = 0.01$ for AlexNet and $c = 0.001$ for ResNet. Similarly, to maintain the same level of sparsity across layers we need a λ (equation 4.8) that is proportional to the standard deviation of the Gaussian initializers. We use $\lambda = \lambda'\epsilon$, where λ' is constant across layers and ϵ is the threshold value for that layer. We try $\lambda' \in \{0.1, 0.2, 0.3\}$ for both AlexNet and ResNet to get different sparsities in $\mathbf{P}$.

The dictionary size k, the regularizer coefficient λ, and threshold value ϵ are the three important hyperparameters for gaining speedup. The larger the dictionary is, the more accurate (but slower) the model becomes. The size of the the dictionary for the first layer does not need to be very large as it's representing a 3-dimensional space. We observed that for the first layer, a dictionary size as small as 3 vectors is sufficient for both AlexNet and

	ResNet-18		
Model	**speedup**	**top-1**	**top-5**
CNN	1.0×	69.3	90.0
XNOR-Net[111]	10.6×	51.2	73.2
LCNN-fast	**29.2**×	51.8	76.8
LCNN-accurate	5×	**62.2**	**84.6**

Table 4.2: Comparison of LCNN and XNOR-Net on ResNet-18. The accuracies are classification accuracy on the validation set of ILSVRC2012.

ResNet. In contrast, fully connected layers of AlexNet are of higher dimensionality and a relatively large dictionary is needed to cover the input space. We found dictionary sizes 512 and 1024 to be proper for fully connected layers. In AlexNet we use the same dictionary size across other layers, which we vary from 100 to 500 for different experiments. In ResNet, aside from the very first layer, all the other convolutional layers are grouped into 4 types of ResNet blocks. The dimensionality of input is equal between same ResNet block types, and is doubled for consecutive different block types. In a similar way we set the dictionary size for different ResNet blocks: equal between the same block types, and doubles for different consecutive block types. We vary the dictionary size of the first block from 16 to 128 in different experiments.

4.4.2 Image Classification

In this section we evaluate the efficiency and the accuracy of LCNN for the task of image classification. Our proposed lookup based convolution is general and can be applied on any CNN architecture. We use AlexNet [77] and ResNet [52] architectures in our experiments. We use ImageNet challenge ILSVRC2012 [28] to evaluate the accuracy of our model. We report standard top-1 and top-5 classification accuracy on 1K categories of objects in natural scenes. To evaluate the efficiency, we compare the number of floating point operations as

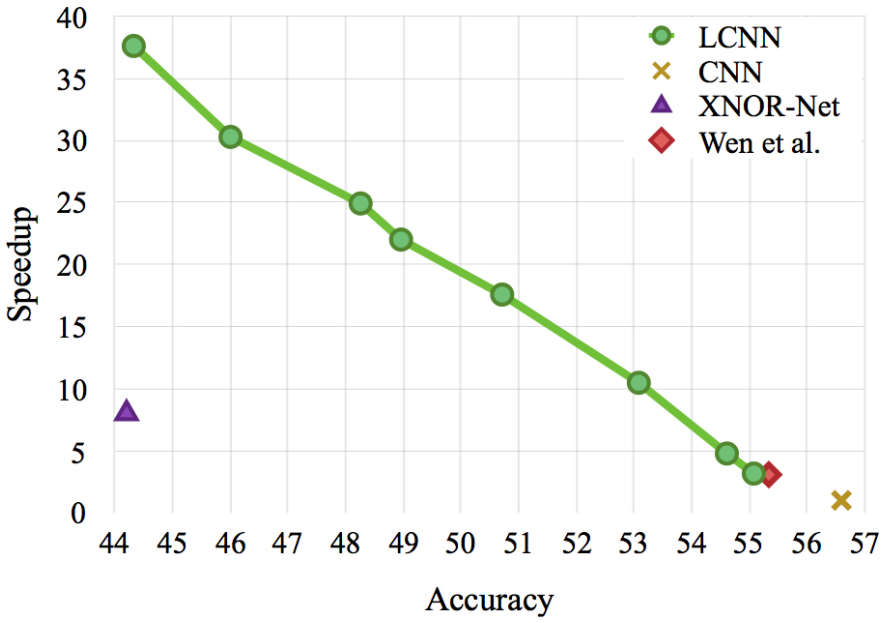
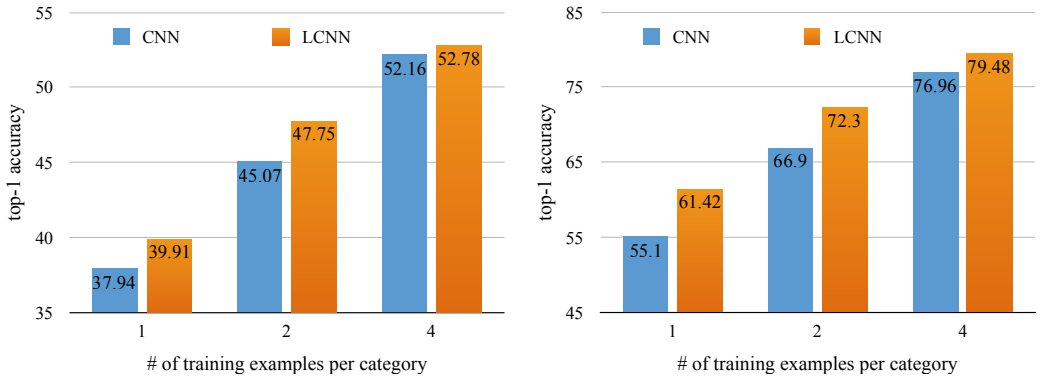


Figure 4.3: Accuracy vs. speedup. By tuning the dictionary size, LCNN achieves a spectrum of speedups.



(a) cats, sofas and bicycles excluded (b) 10 random categories excluded

Figure 4.4: Comparison between the performance of LCNN and CNN baseline on few-shot learning, for $\{1, 2, 4\}$ examples per category. In (a) all cats (7 categories), sofas (1 category) and bicycles (2 categories) are held out for few-shot learning. In (b), 10 random categories are held out for few-shot learning. We repeat sampling the 10 random categories 5 times to avoid over-fitting to a specific sampling.

a representation for speedup. The speed and the accuracy of our model depend on two hyperparameters: 1) k, the dictionary size and 2) λ, which controls the sparsity of $\mathbf{P}$; *i.e.,*the average number of dictionary components in the linear combination . One can set a trade-off between the accuracy and the efficiency of LCNN by adjusting these two parameters. We compare our model with several baselines: 1- XNOR-Net [111], which reduces the precision of weights and outputs to 1-bit, and therefore multiplications can be replaced by binary operations. In XNOR-Net, all the layers are binarized except the first and the last layer (in AlexNet, they contain 9.64% of the computation). 2- Wen *et. al.* [142], which speeds up the convolutions by sparsifying the weight filters.

Table 4.1 compares the top-1 and top-5 classification accuracy of LCNN with baselines on AlexNet architecture. It shows that with small enough dictionaries and sparse linear combinations, LCNN offers 37.6× speedup with the accuracy of XNOR-Net. On the other hand, if we set the dictionaries to be large enough, LCNN can be as accurate as slower

models like Wen *et. al.*. In LCNN-fast, the dictionary size of the mid-layer convolutions is 30 and for the fully connected layers is 512. In LCNN-accurate, the mid-layer convolutions have a dictionary of size 500 and the size of dictionary in fully connected layers is 1024. The reguralizer constant (Section 4.4.1) λ' for LCNN-fast and LCNN-accurate is 0.3 and 0.1, respectively.

Depending on the dictionary size and λ', LCNN can achieve various speedups and accuracies. Figure 4.3 shows different accuracies vs. speedups that our model can achieve. The accuracy is computed by top-1 measure and the speedup is relative to the original CNN model. It is interesting to see that the trend is nearly linear. The best fitted line has a slope of -3.08, *i.e.,*for each one percent accuracy that we sacrifice in top-1, we gain 3.08 more speedup.

We also evaluate the performance of LCNN on ResNet-18 architecture. ResNet-18 is a compact architecture, which has $5\times$ fewer parameters in compare to AlexNet while it achieves 12.7% higher top-1 accuracy. That makes it a much more challenging architecture for further compression. Yet we show that we can gain large speedups with a few points drop in the accuracy. Table 4.2 compares the accuracy of LCNN, XNOR-Net [111], and the original model (CNN). LCNN-fast is getting the same accuracy as XNOR-Net while getting a much larger speedup. Moreover, LCNN-accurate is getting a much higher accuracy yet maintaining a relatively large speedup. LCNN-fast has dictionaries of size 16, 32, 64, and 128 for different block types. LCNN-accuracte has larger dictionaries: 128, 256, 512 and 1024 for different block types.

4.4.3 Few-shot Learning

In this section we evaluate the performance of LCNN on the task of few-shot learning. To evaluate the performance of LCNN on this task, we split the categories of ImageNet challenge ILSVRC2012 into two sets: i) base categories, a set of 990 categories which we use for pre-training, and ii) novel categories, a set of 10 categories that we use for few-shot learning.We do experiments under 1, 2, and 4 samples per category. We take two strategies for splitting

the categories. One is random splitting, where we randomly split the dataset into 990 and 10 categories. We repeat the random splitting 5 times and report the average over all. The other strategy is to hold out all cats (7 categories), bicycles (2 categories) and sofa (1 category) for few-shot learning, and use the other 990 categories for pre-training. With this strategy we make sure that base and novel categories do not share similar objects, like different breeds of cats. For each split, we repeat the random sampling of 1, 2, and 4 training images per category 20 times, and get the average over all. Repeating the random sampling of the few examples is crucial for any few-shot learning experiment, since a model can easily overfit to a specific sampling of images.

We compare the performance of CNN and LCNN on few-shot learning in Figure 4.4. We first train an original AlexNet and an LCNN AlexNet on all training images of base categories (990 categories, 1000 images per category). We then replace the 990-way classification layer with a randomly initialized 10-way linear classifier. In CNN, this produces 10×4096 randomly initialized weights, on which we don't have any prior. These parameters need to be trained merely from the few examples. In LCNN, however, we transfer the dictionary trained in the 990-way classification layer to the new 10-way classifier. This reduces the number of randomly initialized parameters by at least a factor of 4. We use AlexNet LCNN-accurate model (same as the one in Table 4.1) for few-shot learning. At the time of fine-tuning for few-shot categories, we keep the dictionaries in all layers fixed and only fine-tune the sparse $\mathbf{P}$ tensor. This reduces the total number of parameters that need to be fine-tuned by a factor of $14\times$. We use different learning rates η and η' for the randomly initialized classification layer (which needs to be fully trained) and the previous pre-trained layers (which only need to be fine-tuned). We tried $\eta' = \eta$, $\eta' = \frac{\eta}{10}$, $\eta' = \frac{\eta}{100}$ and $\eta' = 0$ for both CNN and LCNN, then picked the best for each configuration.

Figure 4.4 shows the top-1 accuracies of our model and the baseline in the two splitting strategies of our few-shot learning experiment. In Figure 4.4 (a) we are holding out all cat, sofa, and bicycle categories (10 categories in total) for few-shot learning. LCNN is beating the baseline consistently in $\{1, 2, 4\}$ examples per category. Figure 4.4 (b) shows the comparison

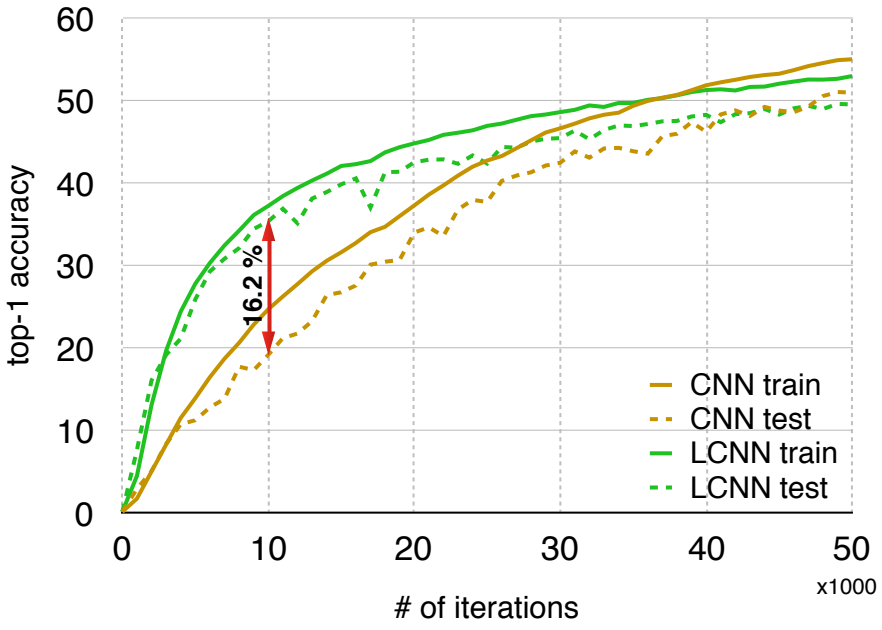


Figure 4.5: LCNN can obtain higher accuracy on few iterations by transferring the dictionaries **D** from a shallower architecture. This figure illustrates the learning curves on top-1 accuracy for both LCNN and standard CNN. The accuracy of LCNN is 16.2% higher than CNN at iteration $10K$.

in the random splitting strategy. We repeat randomly splitting the categories into 990 and 10 categories 5 times, and report the average over all. Here LCNN gets a larger improvement in the top-1 accuracy compared to the baseline for $\{1, 2, 4\}$ images per category.

4.4.4 Few-iteration Learning

In section 4.3.3 we discussed that the dictionaries in LCNN can be transferred from a shallower network to a deeper one. As a result, one can train fewer parameters–only **I** and **C**–in the deeper network with few iterations obtaining a higher test accuracy compared to a standard CNN. In this experiment we train a ResNet with 1 block of each type, 10 layers total. We then transfer the dictionaries of each layer to its corresponding layer of ResNet-18 (with 18 layers). After transfer, we keep the dictionaries fixed. We show that we get higher accuracy in small number of iterations compared to standard CNN. Figure 4.5 illustrates

the learning curves on top-1 accuracy for both LCNN and standard CNN. The test accuracy of LCNN is 16.2% higher than CNN at iteration $10K$. The solid lines denote the training accuracy and the dashed lines denote the test accuracy.

4.5 Conclusion

With recent advancements in virtual reality, augmented reality, and smart wearable devices, the need for getting the state of the art deep learning algorithms onto these resource constrained compute platforms increases. Porting state of the art deep learning algorithms to resource constrained compute platforms is extremely challenging. We introduce LCNN, a lookup-based convolutional neural network that encodes convolutions by few lookups to a dictionary that is trained to cover the space of weights in CNNs. Training LCNN involves jointly learning a dictionary and a small set of linear combinations. The size of the dictionary naturally traces a spectrum of trade-offs between efficiency and accuracy.

LCCN enables efficient inference; our experimental results on ImageNet challenge show that LCNN can offer 3.2× speedup while achieving 55.1% top-1 accuracy using AlexNet architecture. Our fastest LCNN offers 37.6× speed up over AlexNet while maintaining 44.3% top-1 accuracy. LCNN not only offers dramatic speed ups at inference, but it also enables efficient training. On-device training of deep learning methods requires algorithms that can handle few-shot and few-iteration constrains. LCNN can simply deal with these problems because our dictionaries are architecture agnostic and transferable across layers and architectures, enabling us to only learn few linear combination weights. Our future work involves exploring low-precision dictionaries as well as compact data structures for the dictionaries.

Chapter 5

LABEL REFINERY

Among the three main components (data, labels, and models) of any supervised learning system, data and models have been the main subjects of active research. However, studying labels and their properties has received very little attention. Current principles and paradigms of labeling impose several challenges to machine learning algorithms. Labels are often incomplete, ambiguous, and redundant. In this chapter we study the effects of various properties of labels and introduce the *Label Refinery*: an iterative procedure that updates the ground truth labels after examining the entire dataset. We show significant gain using refined labels across a wide range of models. Using a Label Refinery improves the state-of-the-art top-1 accuracy of (1) AlexNet from 59.3 to 67.2, (2) MobileNet[1] from 70.6 to 73.39, (3) MobileNet$^{0.25}$ from 50.6 to 55.59, (4) VGG19 from 72.7 to 75.46, and (5) Darknet19 from 72.9 to 74.47.

5.1 Introduction

There are three main components in the typical pipeline of supervised learning systems: the *data*, the *model*, and the *labels*. Sources of data have expanded drastically in past several years. We have observed the impact of large-scale datasets for several visual tasks. A variety of data augmentation methods [140, 144, 153, 121] have effectively expanded these datasets and improved the performance of learning systems. Models have also been extensively studied in the literature. Recognition systems have shown improvements by increasing the depth of the architectures [53, 122], introducing new activation and normalization layers [64, 77], and developing optimization techniques and loss functions [71, 152]. In contrast to the improvements in data and models, little effort has focused on improving labels.

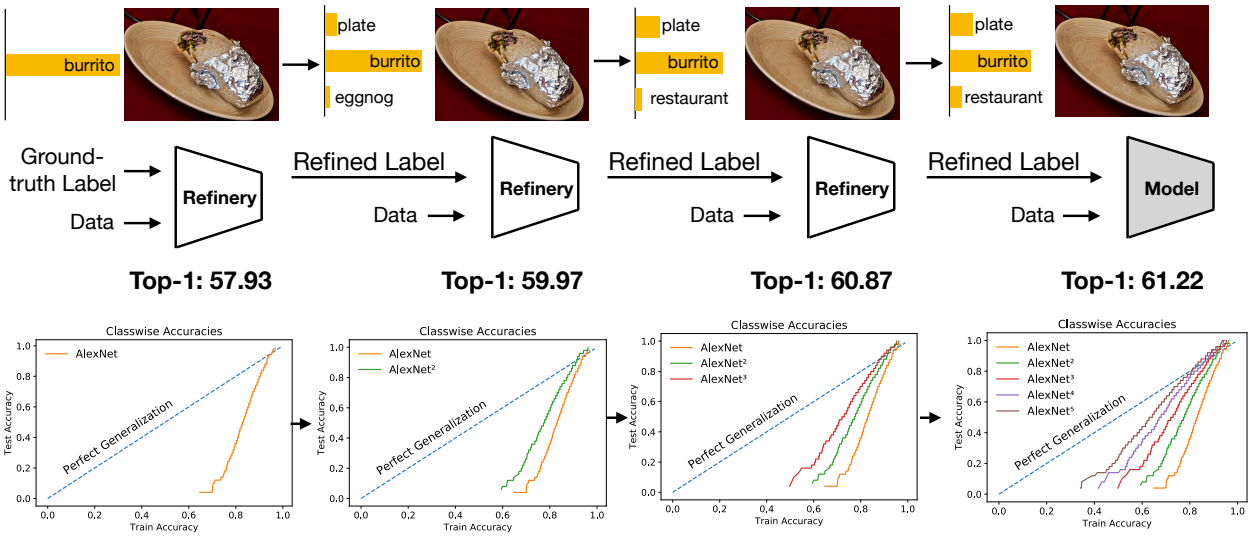


Figure 5.1: Current labeling principles impose challenges for machine learning models. We introduce the *Label Refinery*, an iterative procedure to update ground truth labels using a visual model trained on the entire dataset. The Label Refinery produces soft, multi-category, dynamically-generated labels consistent with the visual signal. The training image shown is labelled with the single category "burrito". After a few iterations of label refining, the labels from which the final model is trained are informative, unambiguous, and smooth. This results in major improvements in the model accuracy during successive stages of refinement as well as improved model generalization. These plots show that as models proceed through successive stages of refinement, the gaps between train and test results and approach ideal generalization.

Current labeling principles and practices impose specific challenges on our learning algorithms. 1) Incompleteness: A natural image of a particular category will contain other object categories as well. For example, Figure 5.2(a) shows an example from ImageNet that is labeled "cat" but the image contains a "ball" as well. This problem is rooted in the nature of how researchers define and collect labels, and is not unique to a specific dataset. 2) Taxonomy Dependency: Categories that are far from each other in the taxonomy structure can be very similar visually. 3) Inconsistency: To prevent overfitting, various loss functions and regularization techniques have been introduced into the training process. Data augmentation [140] is one of the most effective methods employed to prevent neural networks from memorizing the training data. Most modern state-of-the-art architectures for image classification are trained with crop-level data augmentation, in which crops of the image used for training can be as small as 8% of the area of the original image [53]. For many categories, such small crops will frequently result in patches in which the object of interest is no longer visible (Figure 5.2), resulting in an inconsistency with the original label.

To address the aforementioned shortcomings, we argue that several characteristics should apply to ideal labels. Labels should be *soft* to provide more coverage for co-occurring and visually-related objects. Traditional one-hot vector labels introduce challenges in the modeling stage. Labels should be *informative* of the specific image, meaning that they should not be identical for all the images in a given class. For example, an image of a "dog" that has similar appearance to a "cat" should have a different label than an image of a "dog" that has similar appearance to a "fox". This also suggest that labels should be defined at the instance-level rather than the category-level. Determining the best label for each instance may require observing the entire data to establish intra- and inter-category relations, suggesting that labels should be *collective* across the whole dataset. Labels should also be consistent with the image content when crops are taken. Therefore, labels should be *dynamic* in the sense that the label for a crop should depend on the content of the crop.

In this chapter we introduce *Label Refinery*, a solution that uses a neural network model and the data to modify crop labels during training. Refining the labels while training enables

(a) (b)

Figure 5.2: Figure 5.2(a) shows a sample image from the "persian cat" category of ImageNet's training set. The standard technique to train modern state-of-the-art architectures is to crop patches as small as 8% area of the original image, and label them with the original image's label. This will often result in inaccurate labels for the augmented data. Figure 5.2(b) shows a sample crop of the original image where the "persian cat" is no longer in the crop. A trained ResNet-50 labels Figure 5.2(a) by "persian cat", and labels Figure 5.2(b) by "golf ball". We claim that using a model to generate labels for the patches results in more accurate labels and therefore more accurate models.

us to generate soft, informative, collective, and dynamic labels. Figure 5.1 depicts an example of a label refinery. As models go through the stages of the refinery labels are updated based on the previous models. This results in major improvements in the accuracy and generalization. The output of the label refinery is a set of labels from which one can learn a model. The model trained from the produced labels are much more accurate and more robust to over-fitting.

Our experiments show that Label Refining consistently improves the accuracy of object classification networks by a large margin across a variety of popular network architectures. Our improvements in Top-1 accuracy on the ImageNet validation set include: AlexNet from 59.3% to 67.2%, VGG19 from 72.7% to 75.46%, ResNet18 from 69.57% to 72.52%, ResNet50 from 75.7% to 76.5%, DarkNet19 from 72.9% to 74.47%, $\text{MobileNet}_{0.25}$ from 50.65% to 55.59%, and MobileNet_{1} from 70.6% to 73.39%. Collective and dynamic labels enable standard models to generalize better, resulting in significant improvements in image classification. Figure 5.1 Plots the train versus test accuracies as models go through the label refinery pro-

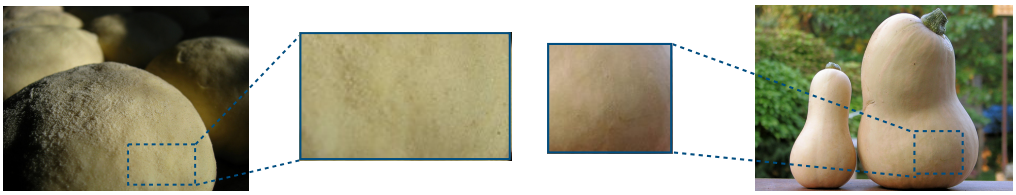
cedure. The gap between train and test accuracies is getting smaller and closer to an ideal generalization.

We further demonstrate that a trained model can serve as a Label Refinery for another model of the same architecture. For example, we iterate through several successions of training a new AlexNet model by using the previously trained AlexNet model as a Label Refiner. Our results show major improvements (from 59.3% to 61.2%) on using AlexNet to refine labels for another AlexNet. Note that the final AlexNet has not seen the actual ground-truth labels in the past few stages. The final AlexNet models demonstrate greatly reduced overfitting compared to the original models (Figure 5.4(a) and Figure 5.5). We also experiment with using a model of one architecture as a Label Refiner for a model of another architecture. Further, we have also shown that adversarially modifying image examples improves the accuracy when using label refinery.

Our contributions include: (1) introducing the Label Refinery for crop-level label augmentation, (2) improving state-of-the-art accuracy on ImageNet for a variety of existing architectures, (3) demonstrating the ability of a network to improve accuracy by training from labels generated by another network of the same architecture, and (4) generating adversarial examples to improve the performance of the Label Refinery method.

5.2 Related Work

Label Smoothing and Regularization: Softening labels has been used to improve generalization. [128] uniformly redistributes 10% of the weight from the ground-truth label to other classes to help regularize during training. DisturbLabel [152] replaces some of the labels in a training batch with random labels. This helps regularize training by preventing overfitting to ground-truth labels. [114] augments noisy labels using other models to improve label consistency. [100] introduces a notion of local distributional smoothness in model outputs based on the smoothness of the model's outputs when inputs are perturbed. The smoothness criterion is enforced with the purpose of regularizing models. The work of [106] explores penalizing networks by regularizing the entropy of the model outputs. Unlike



(a) (b)

Figure 5.3: Sample training examples from "dough" and "butternut squash" categories of ImageNet. While the two sample images are visually distinctive, their random crops are quiet similar. A trained ResNet-50 labels both cropped patches softly over categories of "dough", "butternut squash", "burrito", "french loaf", and "spaghetti squash". We claim that labelling the crops softly by a trained model makes the training of the same model more stable, and therefore results in more accurate models.

our method, these approaches can not address the inconsistency of the labels.

Incorporating Taxonomy: Several methods have explored using taxonomy to improve label and model quality. [85] uses cross-category relationships from knowledge graphs to mitigate the issues caused by noisy labels. [146] designs a hierarchical loss to reduce the penalty for predictions that are close in taxonomy to the ground-truth. [145] investigates learning multi-label classification with missing labels. They incorporate instance-level information as well as semantic hierarchies in their solution. Incorporating taxonomic information directly into the model's architecture is explored in [17]. [96] uses the output of existing binary classifiers to address the problem of training models on single-label examples that contain multiple training categories. These methods fail to address the incompleteness of the labels. Instead of directly using taxonomy, our model collectively infer the visual relations between categories to impose these knowledge into the training while capturing a complete description of the image.

Data Augmentation: To preserve generalization, several data augmentations such as cropping, rotating, and flipping input images have been applied in training models [77, 122, 53, 126]. [144] proposes data warping and synthetic over-sampling to generate additional

training data. [140] and [121] explore using GANs to generate training examples. Most of such augmentation techniques further confuse the model with inconsistent labels. For example, a random crop of an image might not contain the main object the that image We propose augmenting the labels alongside with the data by refining them during training when augmenting the data.

Teacher-Student Training: Using another network or an ensemble of multiple networks as a teacher model to train a student model has been explored in [16, 8, 84, 120, 55, 117]. [8] explores training a shallow student network from a deeper teacher network. A teacher model is used in [117, 120] to train a compressed student network. Most similar to our work is [55], where they introduce distillation loss for training a model from an ensemble of its own. We show that Label Refinery must be done at the crop level, it benefits from being performed iteratively, and models benefit by learning off of the labels generated by the exact same model.

5.3 Label Refinery

Previous works have shown that data augmentation using cropping significantly improves the performance of classification models [77, 127]. Given a dataset $\mathcal{D} = \{(X_i, Y_i)\}$, we can formalize data augmentation by defining a new dataset $\tilde{\mathcal{D}} = \{(f(X_i), Y_i)\}$, where f is a stochastic function that generates crops on-the-fly for the image X_i. The image labels assigned to the augmented crops are often not accurate (Figure 5.2 and Figure 5.3). We address this problem by passing the dataset through multiple Label Refiners. The first Label Refinery network C_{θ_1} is trained over the dataset $\tilde{\mathcal{D}}$ with the inaccurate crop labels. The second Label Refinery network C_{θ_2} is trained over the same set of images, but uses labels generated by C_{θ_1}. More formally, we can view this procedure as training C_{θ_2} on a new augmented dataset $\tilde{\mathcal{D}}_1 = \{(f(X_i), C_{\theta_1}(f(X_i)))\}$. Once C_{θ_2} is trained, we can similarly use it to train a subsequent network C_{θ_3}.

We train the first Label Refinery network C_{θ_1} using the cross-entropy loss against the image-level ground-truth labels. We train all subsequent Label Refinery models C_{θ_t} for $t > 1$

by minimizing the KL-divergence between its output and the soft label generated by the previous Label Refinery $C_{\theta_{t-1}}$. Letting $p_c^t(z) \triangleq C_{\theta_t}(z)[c]$ be the probability assigned to class c in the output of model C_{θ_t} on some crop z, our loss function for training model C_{θ_t} is:

$$L_t(f(X_i)) = -\sum_c p_c^{t-1}(f(X_i)) \log \left(\frac{p_c^t(f(X_i))}{p_c^{t-1}(f(X_i))} \right) \tag{5.1}$$

$$= -\sum_c p_c^{t-1}(f(X_i)) \log p_c^t(f(X_i)) + \sum_c p_c^{t-1}(f(X_i)) \log p_c^{t-1}(f(X_i))$$

The second term is the entropy of the soft labels, and is constant with respect to C_{θ_t}. We can remove it and instead minimize the cross entropy loss:

$$\tilde{L}_t(f(X_i)) = -\sum_c p_c^{t-1}(f(X_i)) \log p_c^t(f(X_i)) \tag{5.2}$$

Note that training C_{θ_1} using cross entropy loss can be viewed as a special case of our sequential training method using KL-divergence in which C_{θ_1} is trained from the original image-level labels. It's worth emphasizing that the subsequent models do not see the original ground truth labels Y_i. The information in the original labels is propagated by the sequence of Label Refinery networks.

If any of the Label Refinery networks have Batch Normalization [64], we put them in training mode even at the label generation step. That is, their effective mean and standard deviation to be computed from the current training batch as opposed to the saved running mean and running variance. We have observed that this results in more accurate labels and, therefore, more accurate models. We believe that this is due to the fact that the Label Refinery has been trained with the Batch Normalization layers in the training mode. Hence it produces more accurate labels *for the training set* if it's in the same mode.

It is possible to use the same network architecture for some (or all) of the Label Refinery networks in the sequence. We have empirically observed that the dataset labels improve iteratively even when the same network architecture is used multiple times (Section 5.4). This is because the same Label Refinery network trained on the new refined dataset becomes more accurate that its previous versions over each pass. Thus, subsequent networks are trained with more accurate labels.

The accuracy of a trained model heavily depends on the consistency of the labels provided to it during training. Unfortunately, assessing the quality of crop labels quantitatively is not possible because there crop level labels are not provided. Asking human annotators to evaluate individual crops is infeasible both due to the number of possible crops and due to the difficulty of evaluating soft labels to a large number of categories for a crop in which there may not be a single main object. We can use a network's validation set accuracy as a measure of its ability to produce correct labels for crops. Intuitively, this measurement serves as an indication of the quality of a Label Refinery network. However, we observe that models with higher validation accuracy do not always produce better crop labels if the model with higher validation accuracy is severely overfit to the training set. Intuitively, this is because the model will reproduce the ground-truth image labels for training set images. We explore this more in Section 5.4.1.

One popular way to augment ImageNet data is to crop patches as small as 8% of the area of the image [127]. In the presence of such aggressive data augmentation, the original image label is often very inaccurate for the given crop. Whereas traditional methods only augment the image input data through cropping, we additionally augment the labels using Label Refinery networks to produce labels for the crops. Smaller networks such as MobileNet [59] usually aren't trained with such small crops. Yet, we observe that such networks can benefit from small crops if a Label Refinery is used. This demonstrates that a primary cause in accuracy degradation of such networks is inaccurate labels on small crops.

5.3.1 Adversarial Jittering

Using a Label Refinery network allows us to generate labels for any set of images. Our training dataset $\tilde{\mathcal{D}}_t = \{(f(X_i), C_{\theta_t}(f(X_i)))\}$ depends only on the input images X_i, and labels are generated on-the-fly by the Refinery network C_{θ_t}. This means that we are no longer limited to using images in the training set $\mathcal{D}$. We could use another unlabeled image dataset as a source of X_i. We could even use synthetic images. We experiment with using the Label Refinery in conjunction with the network being trained in order to generate adversarial

examples on which the two networks disagree.

Let $C_{\theta_{t-1}}$ and C_{θ_t} be two of the networks in a sequence of Label Refinery networks. Given a crop $f(X_i)$, we define $\alpha_t(f(X_i))$ to be a modification of $f(X_i)$ for which $C_{\theta_{t-1}}$ and C_{θ_t} output different probability distributions. Following the practice of [102] for generating adversarial examples, we define α_t as

$$\alpha_t(X) = X + \eta \frac{\partial L_t}{\partial X} \tag{5.3}$$

, where L_t is the KL-divergence loss defined in Equation 5.1. This update performs one step of gradient ascent in the direction of increasing the KL-divergance loss. In other words, the input is modified to exacerbate the discrepancy between the output probability distributions. In order to prevent the model being trained from becoming confused by the unnatural inputs $\alpha_t(f(X_i))$, we batch the adversarial examples with their corresponding natural crops $f(X_i)$.

5.4 Experiments

We evaluate the effect of label refining for a variety of network architectures on the standard ImageNet, ILSRVC2012 [29] classification challenge. We first explore the effect of label refining when the Label Refinery network architecture is identical to the architecture of the network being trained. We then evaluate the effect of label refining when the Label Refinery uses a more accurate network architecture. Finally, we present some ablation studies and analysis to investigate the source of the improvements. Note that all experiments are done with a single model over a single validation crop.

Implementation Details: All models are trained using PyTorch [104] on 4 GPUs for 200 epochs to ensure convergence. The learning rate is constant for the first 140 epochs. It is divided by 10 after epoch 140 and again divided by 10 after epoch 170. We use an initial learning rate of 0.01 to train AlexNet and an initial learning rate of 0.1 for all other networks. We use image cropping and horizontal flipping to augment the training set. When cropping, we follow the data augmentation practice of [127] in which the crop areas are chosen uniformly from 8% to 100% of the area of the image. We use a batch size of 256 for

Model	**Top-1**	**Top-5**
AlexNet	57.93	79.41
AlexNet^2	59.97	81.44
AlexNet^3	60.87	82.13
AlexNet^4	61.22	**82.56**
AlexNet^5	**61.37**	**82.56**

Model	**Top-1**	**Top-5**
ResNet50	75.7	92.81
ResNet50^2	**76.5**	**93.12**

Model	**Top-1**	**Top-5**
MobileNet	68.51	88.13
MobileNet^2	**69.52**	**88.7**

Model	**Top-1**	**Top-5**
VGG16	70.1	88.54
VGG16^2	71.85	90.07
VGG16^3	**72.49**	**90.76**

Model	**Top-1**	**Top-5**
VGG19	71.39	89.44
VGG19^2	72.66	90.75
VGG19^3	**73.32**	**91.30**

Model	**Top-1**	**Top-5**
Darknet19	70.6	89.13
Darknet19^2	72.74	90.73
Darknet19^3	**73.01**	**90.92**

Table 5.1: Self-Refining results on the ImageNet 2012 validation set. Each model is trained using labels refined by the model right above it. That is, AlexNet^3 is trained by the labels refined by AlexNet^2, and AlexNet^2 is trained by the labels refined by AlexNet. The first row models are trained using the image level ground-truth labels.

all models except the MobileNet variations, for which we use batch size of 512. Except for adversarial inputs experiments, we train models from refined labels starting from a random initialization. Our source code is available at `http://github.com/hessamb/label-refinery`.

Self-Refinement: We first explore using a Label Refinery to train another network with the same architecture. Table 5.1 shows the results for self-refinement on various architectures. Each row represents a randomly-initialized instance of the network architecture trained with labels refined by the model directly one row above it in the table. All six network architectures improve their accuracy through self-refinement. For AlexNet the self-refining

process must be repeated 4 times before convergence, whereas MobileNet and ResNet-50 converge much faster. We argue that this is because AlexNet is more overfit to the training set. Therefore, it takes more training iterations to forget the information that it has memorized from training examples. One might argue that the accuracy improvements are due to the extended training time of models. However, we experimented with training models for an equal number of total epochs and the model accuracies did not improve further. This is discussed further in Section 5.4.1.

Cross-Architecture Refinement: The architecture of a Label Refinery network can be different from that of the trained network. A high-quality Label Refinery should not overfit on training data even if its validation accuracy is high. In other words, under the same validation accuracy, a network with lower training accuracy is a better Label Refinery. Intuitively, this property allows the refinery to generate high-quality crop labels that are reflective of the true content of the crops. This property prevents the refinery from simply predicting the training labels. We observe that a ResNet-50 model trained to 75.7% top-1 validation accuracy on ImageNet can serve as a high-quality refinery. Table 5.2 shows that a variety of network architectures benefit significantly from training with refined labels. All network architectures that we tried using Label Refineries gained significant accuracy improvement over their previous state-of-the-art. AlexNet and ResNetXnor-50[1] achieve more than a 7 point improvement in top-1 accuracy. Efficient and compact models such as MobileNet benefit significantly from cross-architecture refinement. VGG networks have a very high capacity and they overfit to the training set more than the other networks. Providing more accurate training set labels helps them to fit to more accurate signals and perform better at validation time. Darknet19, the backbone architecture of YOLOv2 [113], improves almost 4 points when trained with refined labels.

Adversarial Inputs: As discussed in Section 5.3.1 we can adversarially augment our training set with patches on which the refinery network and the trained model disagree. We

[1]ResNetXnor-50 is the XNOR-net [112] version of ResNet-50 in which layers are binary.

Model	Paper Number		Our Impl.		Label Refinery	
	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
AlexNet [77]	59.3	81.8	57.93	79.41	**66.28**†	**86.13**†
MobileNet [59]	70.6	N/A	68.53	88.14	**73.39**	**91.07**
MobileNet0.75 [59]	68.4	N/A	65.93	86.28	**70.92**	**89.68**
MobileNet0.5 [59]	63.7	N/A	63.03	84.55	**66.66**†	**87.07**†
MobileNet0.25 [59]	50.6	N/A	50.65	74.42	**54.62**†	**77.92**†
ResNet-50 [53]	N/A	N/A	75.7	92.81	**76.5**	**93.12**
ResNet-34 [53]	N/A	N/A	73.39	91.32	**75.06**	**92.35**
ResNet-18 [53]	N/A	N/A	69.7	89.26	**72.52**	**90.73**
ResNetXnor-50 [112]	N/A	N/A	63.1	83.61	**70.34**	**89.18**
VGG16 [122]	73	91.2	70.1	88.54	**75**	**92.22**
VGG19 [122]	72.7	91	71.39	89.44	**75.46**	**92.52**
Darknet19 [113]	72.9	91.2	70.6	89.13	**74.47**	**91.94**

Table 5.2: Using refined labels improves the accuracy of a variety of network architectures to new state-of-the-art accuracies. The Label Refinery used in these experiments is a ResNet-50 model trained with weight decay. † These models can be further improved by training with adversarial inputs (Table 5.3).

used a gradient step of $\eta = 1$, as defined in Equation 5.3 to augment the dataset. We batch each adversarially modified crop with the original crop during training. This helps to ensure the trained model does not drift too far from natural images. We observe in Table 5.3 that smaller models further improve beyond the improvements from using a Label Refinery alone.

Model	GT Labels		Label Refinery		Adversarial	
	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
AlexNet	57.93	79.41	66.28	86.13	**67.2**	**86.92**
MobileNet0.5	63.03	84.55	66.66	87.07	**67.33**	**87.4**
MobileNet0.25	50.65	74.42	54.62	77.92	**55.59**	**78.58**

Table 5.3: Smaller models are further improved by training over adversarial inputs. The Adversarial Label Refinery is ResNet-50.

Model	Top-1	Top-5
AlexNet – no refinery	57.93	79.41
AlexNet – soft static refinery	63.55	84.16
AlexNet – hard dynamic refinery	64.41	84.53
AlexNet – soft dynamic refinery	66.28	86.13

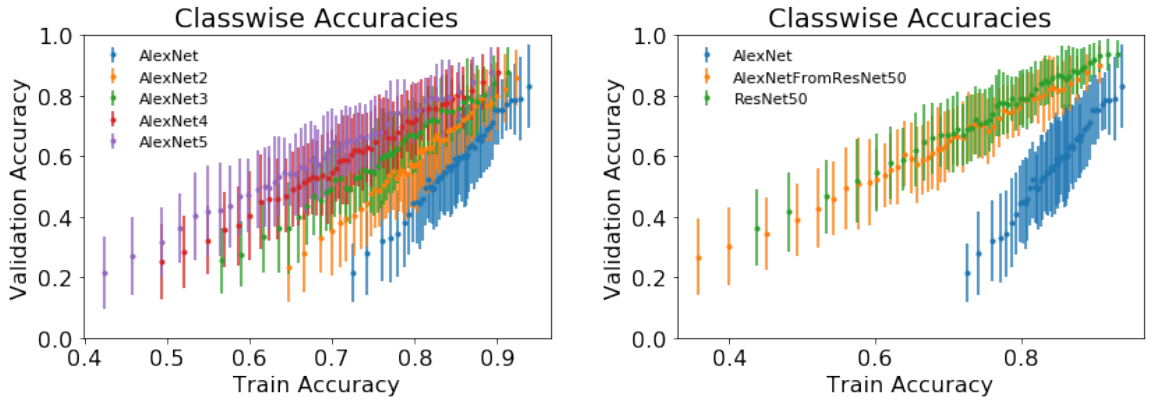
Table 5.4: AlexNet benefits from both soft labeling and dynamic labeling. When combined the improvement is increased over both, suggesting that they capture different aspects of label errors. Label Refinery is ResNet-50.

5.4.1 Analysis

We explore the characteristics of models trained using a Label Refinery. We first explore how much of the improvement comes from the dynamic labeling of the image crops and how much of it comes from softening the target labels. We then explore the overfitting characteristics of models trained with a Label Refinery. Finally, we explore using various loss functions to train models against the refined labels. Most of the analyses are performed on AlexNet architecture because it trains relatively fast ($\sim$ 1 day) on the ImageNet dataset.

Model	Top-1	Top-5
AlexNet – no refinery	57.93	79.41
AlexNet – taxonomy based refined categories	56.73	77.69
AlexNet – visually refined categories	58.54	80.77
AlexNet – visually refined images	62.69	83.46

Table 5.5: Comparing refining labels at category level vs. image level. Note that "AlexNet – visually refined images" is trained over image level refined labels as opposed to crop level. For fairness, we fixed the batch normalization layers of label refinery (which harms the quality of label refinery) in all visually refined labels experiments. Label Refinery is ResNet-50.



(a) (b)

Figure 5.4: Per category train and test accuracy. For each model, labels were sorted according to training set accuracies and divided into bins. each point in the plot shows the average validation set accuracy and the associated standard deviation for each bin. These figures show that training with a refinery results in models with less overfitting.

Dynamic Labels vs. Soft Labels: The benefits of using a label refinery are twofold: (1) Each crop is dynamically re-labeled with more accurate labels for the crop (Figure 5.2), and (2) images are softly labeled according to the distribution of visually similar objects in the crop (Figure 5.3). We find that both aspects of the refinement process improve performance. To assess the improvement from dynamic labeling alone, we perform label refinement with hard dynamic labels. Specifically, we assign a one-hot label to each crop by passing the crop to the Label Refinery and choosing the most-likely category from the output. To observe the improvement from soft labeling alone, we perform label refinement with soft static labels. To compute these labels for a given crop, we pass a center crop of the original image to the refiner rather than using the training crop. We compare the results for soft static labels and hard dynamic labels in Table 5.4. Both dynamic labeling and soft labeling significantly improve the accuracy of AlexNet. When they are combined we observe an additional improvement, suggesting that they address different issues with labels in the dataset.

Category Level Refining vs. Image Level Refining: Labels can be refined at the category level. That is, all images in a class can be assigned a unique soft label that models intra-category similarities. At the category level, labels can be refined either by visual cues (based on the visual similarity between the categories) or by semantic relations (based on the taxonomic relationship between the categories). Since ImageNet categories are drawn from WordNet, we can use taxonomy-based distances to refine the labels. We experiment with using the Wu-Palmer similarity [149] of the WordNet [98] categories to refine the category labels. Table 5.5 compares refining labels at the category level with refining at the image level. We observe larger improvements when the labels are refined at the image level. Our experiment shows that taxonomy-based refinement does not improve training. We believe this is because WordNet similarities do not correlate well with visual similarities in the image space. Refining category labels based off of their WordNet distance can confuse the target model.

Model Generalization: Figure 5.4(a) and 5.4(b) show the per-category train and val-

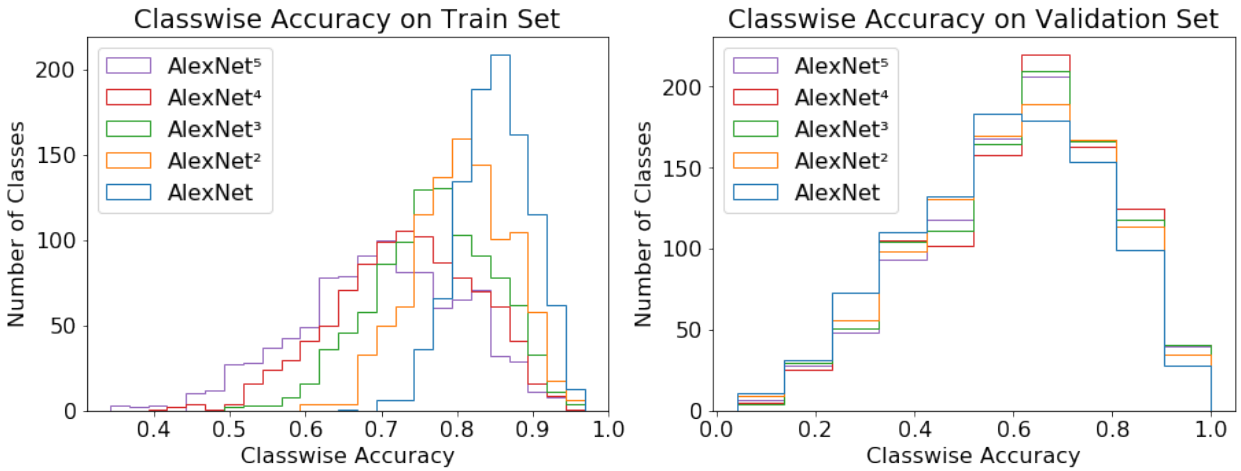


Figure 5.5: The train and validation accuracy distribution of AlexNet models trained sequentially. AlexNet is trained off of the ground-truth labels, and the successive models AlexNet^{i+1} are trained off of the labels generated by AlexNet^{i}.

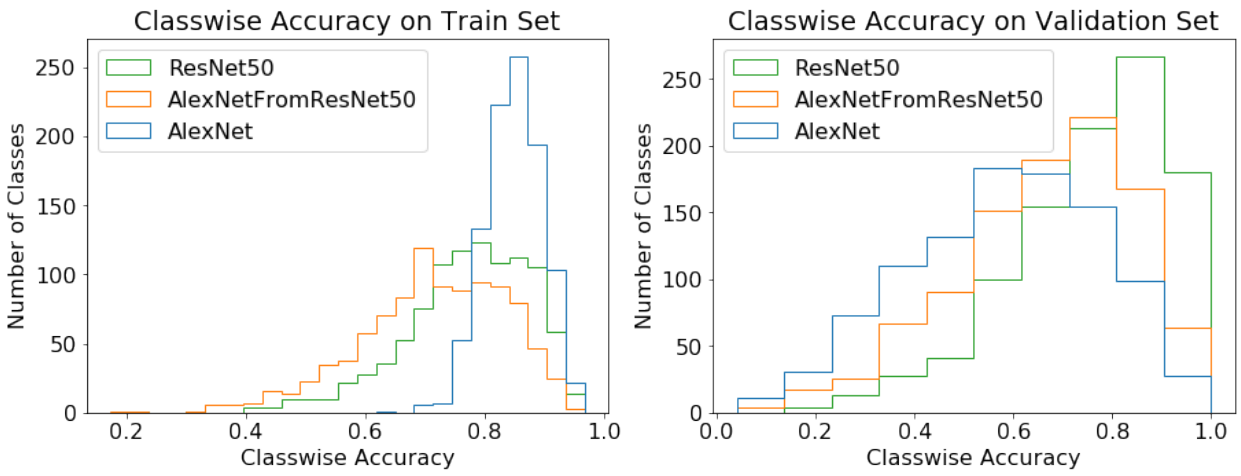


Figure 5.6: The train and validation accuracies for AlexNet, ResNet, and AlexNet trained off of labels generate by ResNet50. `AlexNetFromResNet50` has a train accuracy profile that more closely resembles ResNet50 than AlexNet.

idation accuracies of ImageNet categories for models trained with a Label Refinery. Each point in the plot shows the average and standard deviation of the accuracies for a set of categories. Figure 5.4(a) shows the accuracies of a sequence of AlexNet models (in different colors). AlexNet trained using the ground-truth labels has much higher train accuracy. Successive models demonstrate less overfitting as shown by the decrease in the ratio between

Model	**Refinery**		**AlexNet**	
	Top-1	Top-5	Top-1	Top-5
AlexNet – no refinery	`N/A`	`N/A`	57.93	79.41
AlexNet – refinery: VGG16	70.1	88.54	60.78	81.80
AlexNet – refinery: MobileNet	68.53	88.14	65.22	85.69
AlexNet – refinery: ResNet-50	75.7	92.81	66.28	86.13

Table 5.6: Different architecture choices for the refinery network.

Model	Top-1	Top-5
AlexNet – no refinery	57.93	79.41
AlexNet – l_2 loss	63.16	85.56
AlexNet – KL-divergence from output to label	65.36	85.41
AlexNet – KL-divergence from label to output	66.28	86.13

Table 5.7: Different loss function choices. Label Refinery is ResNet-50.

train accuracy and validation accuracy. Figure 5.4(b) shows the per-category accuracies of AlexNet and ResNet-50, as well as an AlexNet model trained with a ResNet50 Label Refinery. ResNet-50 trained with weight decay generalizes better compared to AlexNet, which has two fully connected layers. Intuitively, the generalization of ResNet-50 enables it to generate accurate per-crop labels for the training set. Thus, training AlexNet with a ResNet-50 Label Refinery allows AlexNet to perform well on the test set without overfiting to the original ground-truth labels.

Figure 5.5 shows the training set and validation set accuracies of a sequence of AlexNet models trained with a Label Refinery. The AlexNet trained with ground-truth labels achieves $\sim 86\%$ training accuracy for the majority of classes, but achieves much lower validation set accuracies. By contrast, AlexNet[5] has a training accuracy profile more closely resembling its validation accuracy profile. Figure 5.6 shows a similar phenomena training AlexNet with a ResNet-50 refinery. It's interesting to note that the training and validation profiles of AlexNet trained with a ResNet50 Label Refinery more closely resemble the refinery than the original AlexNet.

Choice of Label Refinery Network: A good Label Refinery network should generate accurate labels for the *training set* crops. A Label Refinery's validation accuracy is an informative signal of its quality. However, if the Label Refinery network is heavily overfitted on the training set, it will not be helpful during training because it will produce the same ground-truth label for all image crops. Table 5.6 compares different architecture choices for

refinery network. VGG16 is a worse choice of Label Refinery than MobileNet, even though VGG16 is more accurate. This is because VGG16 severely overfits to the training set and therefore produces labels too similar to the ground-truth.

Choice of Loss Function: We can use a variety of loss functions to train our target networks to match the soft labels. The KL-divergence loss function that we use is a generalization of the standard cross-entropy classification loss. Note that KL-divergence is not a symmetric function (*i.e.*,$D_{KL}(P||Q) \neq D_{KL}(Q||P)$). Table 5.7 shows the model accuracy if other standard loss functions are used.

Qualitative Results: Using a refinery to produce crop labels reduces overfitting by providing more accurate labels during training. In Figure 5.7, we see an example in which a training image crop does not contain enough information to identify the image category as "barbershop". In spite of this, AlexNet assigns the crop a label of barbershop with high

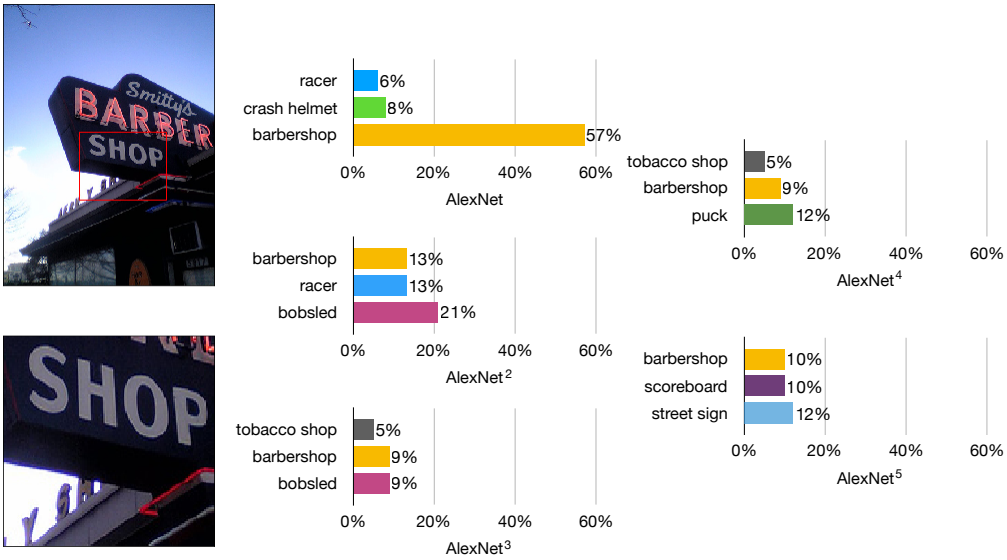


Figure 5.7: The top three predictions for a crop of an image labelled "barber shop" in the ImageNet training set. AlexNet trained on the ground-truth labels is overfit towards the image level label. Successive AlexNet models overfit less, reducing the weight of the "barber shop" category and eventually assigning more probability to other plausible categories such as "street sign" and "scoreboard".

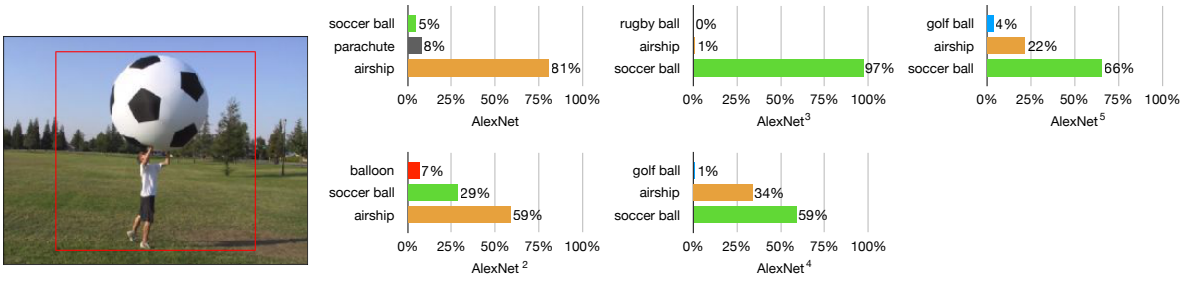


Figure 5.8: The top three predictions for an image labelled "soccer ball" in the ImageNet validation set. Successive models learn to avoid overfitting an object surrounded by patches of sky to the "airship" category.

confidence. This is due to overfitting on the training set. By using an AlexNet as a refinery, AlexNet^2 learns to generalize better. It produces a lower score for "barbershop", and a higher score for other categories. Generalization behavior improves with successive rounds of label refining until AlexNet^5 produces a smooth distribution over plausible categories. In Figure 5.8, we see an example of a "soccer ball" from the validation set of ImageNet. AlexNet incorrectly predicts "airship" with high confidence. This prediction is most likely because the main object is surrounded by blue sky, which is common for an airship but uncommon for a soccer ball. By using AlexNet as a refinery to train another AlexNet model we achieve a reduced score for "airship" and a higher score for "soccer ball". After several rounds of successive refining we achieve an AlexNet model that makes the correct prediction without completely forgetting the similarities between the soccer ball in the sky and an airship.

5.5 Conclusion

In this chapter we address shortcomings commonly found in the labels of supervised learning pipelines. We introduce a solution to refine the labels during training in order to improve the generalization and the accuracy of learning models. The proposed Label Refinery allows us to dynamically label augmented training crops with soft targets. Using a Label Refinery, we achieve a significant gain in the classification accuracy across a wide range of network archi-

tectures. Our experimental evaluation shows improvement in the state-of-the-art accuracy for popular architectures including AlexNet, VGG, ResNet, MobileNet, and XNOR-Net.

BIBLIOGRAPHY

[1] Blender. `http://www.blender.org/`.

[2] Torch7. `http://torch.ch`.

[3] Milton Abramowitz and Irene Stegan. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover Publications, 1964.

[4] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Fixed point optimization of deep convolutional neural networks for object recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 1131–1135. IEEE, 2015.

[5] Eiji Aramaki, Takeshi Imai, Kengo Miyo, and Kazuhiko Ohe. Uth: Svm-based semantic relation classification using physical sizes. In *SemEval Workshop*. Association for Computational Linguistics, 2007.

[6] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *PAMI*, 2011.

[7] Hossein Azizpour, Ali Sharif Razavian, Josephine Sullivan, Atsuto Maki, and Stefan Carlsson. From generic to specific deep representations for visual recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 36–45, 2015.

[8] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014.

[9] R. Baillargeon. The acquisition of physical knowledge in infancy: A summary in eight lessons. In U. Goswami, editor, *Blackwell handbook of childhood cognitive development*. 2007.

[10] JC Baird. Retinal and assumed size cues as determinants of size and distance perception. *Journal of Experimental Psychology*, 1963.

[11] Moshe Bar. The proactive brain: memory for predictions. *Royal Society of London. Series B, Biological sciences*, 2009.

[12] P. Battaglia, J. Hamrick, and J. B. Tenenbaum. Simulation as an engine of physical scene understanding. *PNAS*, 2013.

[13] Marcus A Brubaker and David J Fleet. The kneed walker for human pose tracking. In *CVPR*, 2008.

[14] Marcus A Brubaker, David J Fleet, and Aaron Hertzmann. Physics-based person tracking using simplified lower-body dynamics. In *CVPR*, 2007.

[15] Marcus A Brubaker, Leonid Sigal, and David J Fleet. Estimating contact dynamics. In *ICCV*, 2009.

[16] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 535–541, New York, NY, USA, 2006. ACM.

[17] Lijuan Cai and Thomas Hofmann. Exploiting known taxonomies in learning overlapping concepts. In *IJCAI*, volume 7, pages 708–713, 2007.

[18] Wenlin Chen, James T Wilson, Stephen Tyree, Kilian Q Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *ICML*, 2015.

[19] O.S. Cheung and M. Bar. Visual prediction and perceptual expertise. *Intl. J. of Psychophysiology*, 2012.

[20] Jennifer Chu-carroll, David Ferrucci, John Prager, and Christopher Welty. C.: Hybridization in question answering systems. In *Directions in Question Answering*. AAAI Press, 2003.

[21] W. J. Cody. Rational chebyshev approximations for the error function. *Mathematics of Computation*, 1969.

[22] Robert Collins, Yanxi Liu, and Marius Leordeanu. On-line selection of discriminative tracking features. *PAMI*, 2005.

[23] D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *PAMI*, 2003.

[24] Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, 2016.

[25] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pages 3105–3113, 2015.

[26] Dmitry Davidov and Ari Rappoport. Extraction and approximation of numerical attributes from the web. In *ACL*, 2010.

[27] Erick Delage, Honglak Lee, and Andrew Y Ng. A dynamic bayesian network model for autonomous 3d reconstruction from a single indoor image. In *CVPR*, 2006.

[28] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR09*, 2009.

[29] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[30] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In *NIPS*, 2013.

[31] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, 2014.

[32] Santosh K Divvala, Ali Farhadi, and Carlos Guestrin. Learning everything about anything: Webly-supervised visual concept learning. In *CVPR*, 2014.

[33] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *NIPS*, 2014.

[34] Mark Everingham, Luc Gool, Christopher K. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *IJCV*, 2010.

[35] Clément Farabet, Yann LeCun, Koray Kavukcuoglu, Eugenio Culurciello, Berin Martini, Polina Akselrod, and Selcuk Talay. Large-scale fpga-based convolutional networks. *Scaling up Machine Learning: Parallel and Distributed Approaches*, pages 399–419, 2011.

[36] Li Fei-Fei, Rob Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence*, 28(4):594–611, 2006.

[37] David F. Fouhey and C.L. Zitnick. Predicting object dynamics in scenes. In *CVPR*, 2014.

[38] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.

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

[40] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[41] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[42] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[43] ML Gottmer. Merging reality and virtuality with microsoft hololens. 2015.

[44] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. In *arXiv*, 2014.

[45] Abhinav Gupta, Scott Satkin, Alexei A. Efros, and Martial Hebert. From 3d scene geometry to human workspace. In *CVPR*, 2011.

[46] J. Hamrick, P. Battaglia, and J. B. Tenenbaum. Internal physics models guide probabilistic judgments about object dynamics. *Annual Meeting of the Cognitive Science Societ*, 2011.

[47] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*, 2015.

[48] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[49] Bharath Hariharan and Ross Girshick. Low-shot visual object recognition. *arXiv preprint arXiv:1606.02819*, 2016.

[50] John F. Hart. *Computer Approximations*. Krieger Publishing Co., Inc., 1978.

[51] J. Hawkins and S. Blakeslee. *On Intelligence*. Times Books, 2004.

[52] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, 2015.

[53] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[54] Varsha Hedau, Derek Hoiem, and David Forsyth. Recovering the spatial layout of cluttered rooms. In *CVPR*, 2009.

[55] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[56] Minh Hoai and Fernando De la Torre. Max-margin early event detectors. In *CVPR*, 2012.

[57] Derek Hoiem, Alexei A Efros, and Martial Hebert. Recovering surface layout from an image. *International Journal of Computer Vision*, 2007.

[58] Alfred H Holway and Edwin G Boring. Determinants of apparent visual size with distance variant. *The American Journal of Psychology*, 1941.

[59] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[60] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv preprint arXiv:1609.07061*, 2016.

[61] Kyuyeon Hwang and Wonyong Sung. Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1. In *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*, pages 1–6. IEEE, 2014.

[62] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡1mb model size. *CoRR*, abs/1602.07360, 2016.

[63] Adrian Iftene and Mihai-Alex Moruz. UAIC participation at rte-6. In *TAC*, 2010.

[64] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456, 2015.

[65] M. Isard and A. Blake. Condensation conditional density propagation for visual tracking. *IJCV*, 1998.

[66] William H Ittelson. Size as a cue to distance: Static localization. *The American journal of psychology*, 1951.

[67] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *British Machine Vision Conference (BMVC)*, 2014.

[68] Zhaoyin Jia, Andy Gallagher, Ashutosh Saxena, and Tsuhan Chen. 3d-based reasoning with blocks, support, and stability. In *CVPR*, 2013.

[69] G. Johansson. Visual perception of biological motion and a model for its analysis. *Perception and Psychophysics*, 1973.

[70] Minje Kim and Paris Smaragdis. Bitwise neural networks. *arXiv preprint arXiv:1601.06071*, 2016.

[71] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[72] Kris M Kitani, Brian D. Ziebart, J. Andrew (Drew) Bagnell, and Martial Hebert. Activity forecasting. In *ECCV*, 2012.

[73] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, 2015.

[74] Talia Konkle and Aude Oliva. Canonical visual size for real-world objects. *Journal of Experimental Psychology: Human Perception and Performance*, 2011.

[75] Talia Konkle and Aude Oliva. A familiar-size stroop effect: real-world size is an automatic property of object representation. *Journal of Experimental Psychology: Human Perception and Performance*, 2012.

[76] H. Koppula and A. Saxena. Anticipating human activities using object affordances for reactive robotic response. In *RSS*, 2013.

[77] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[78] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. Hmdb: a large video database for human motion recognition. In *ICCV*, 2011.

[79] Lubor Ladicky, Jianbo Shi, and Marc Pollefeys. Pulling things out of perspective. In *CVPR*, 2014.

[80] Christoph H Lampert, Hannes Nickisch, and Stefan Harmeling. Attribute-based classification for zero-shot visual object categorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(3):453–465, 2014.

[81] T. Lan, T. Chen, and S. Savarese. A hierarchical representation for future action prediction. In *ECCV*, 2014.

[82] Andrew Lavin. Fast algorithms for convolutional neural networks. *arXiv preprint arXiv:1509.09308*, 2015.

[83] Jimmy Lei Ba, Kevin Swersky, Sanja Fidler, et al. Predicting deep zero-shot convolutional neural networks using textual descriptions. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4247–4255, 2015.

[84] Jinyu Li, Rui Zhao, Jui-Ting Huang, and Yifan Gong. Learning small-size dnn with output-distribution-based criteria. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[85] Yuncheng Li, Jianchao Yang, Yale Song, Liangliang Cao, Jiebo Luo, and Jia Li. Learning from noisy labels with distillation. *arXiv preprint arXiv:1703.02391*, 2017.

[86] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.

[87] Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. Neural networks with few multiplications. *arXiv preprint arXiv:1510.03009*, 2015.

[88] Etai Littwin and Lior Wolf. The multiverse loss for robust transfer learning. *arXiv preprint arXiv:1511.09033*, 2015.

[89] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *CVPR*, 2015.

[90] Beyang Liu, Stephen Gould, and Daphne Koller. Single image depth estimation from predicted semantic labels. In *CVPR*, 2010.

[91] C. Liu, J. Yuen, and A. Torralba. Sift flow: Dense correspondence across scenes and its applications. *PAMI*, 2011.

[92] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[93] R. Mann, A.D. Jepson, and J.M. Siskind. The computational perception of scene dynamics. *CVIU*, 1997.

[94] George Marsaglia. Evaluating the normal distribution. *Journal of Statistical Software*, 2004.

[95] Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. In *ICLR*, 2014.

[96] Julian J McAuley, Arnau Ramisa, and Tibério S Caetano. Optimization of robust loss functions for weakly-labeled image taxonomies: An imagenet case study.

[97] Karl Menger. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 1927.

[98] George A Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J Miller. Introduction to wordnet: An on-line lexical database. *International journal of lexicography*, 3(4):235–244, 1990.

[99] Margaret Mitchell, Kees van Deemter, and Ehud Reiter. On the use of size modifiers when referring to visible objects. In *Annual Conference of the Cognitive Science Society*, 2011.

[100] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. Distributional smoothing by virtual adversarial examples. *stat*, 1050:2, 2015.

[101] Katsuma Narisawa, Yotaro Watanabe, Junta Mizuno, Naoaki Okazaki, and Kentaro Inui. Is a 204 cm man tall or small ? acquisition of numerical common sense from the web. In *ACL*, 2013.

[102] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 427–436, 2015.

[103] VR Oculus. Oculus rift-virtual reality headset for 3d gaming. *URL: http://www. oculusvr. com*, 2012.

[104] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[105] M. Pei, Y. Jia, and S.-C. Zhu. Parsing video events with goal inference and intent prediction. In *ICCV*, 2011.

[106] Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548*, 2017.

[107] Luca Del Pero, Joshua Bowdish, Daniel Fried, Bonnie Kermgard, Emily Hartley, and Kobus Barnard. Bayesian geometric modeling of indoor scenes. In *CVPR*, 2012.

[108] Pedro O Pinheiro, Ronan Collobert, and Piotr Dollar. Learning to segment object candidates. In *Advances in Neural Information Processing Systems*, pages 1990–1998, 2015.

[109] Silvia L. Pintea, Jan C. van Gemert, and Arnold W. M. Smeulders. Déjà vu: - motion prediction in static images. In *ECCV*, 2014.

[110] John M. Prager, Jennifer Chu-Carroll, Krzysztof Czuba, Christopher A. Welty, Abraham Ittycheriah, and Ruchi Mahindru. Ibm's piquant in trec2003. In *TREC*, 2003.

[111] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

[112] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[113] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 6517–6525. IEEE, 2017.

[114] Scott Reed, Honglak Lee, Dragomir Anguelov, Christian Szegedy, Dumitru Erhan, and Andrew Rabinovich. Training deep neural networks on noisy labels with bootstrapping. *arXiv preprint arXiv:1412.6596*, 2014.

[115] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[116] Raymond D. Rimey and Christopher M. Brown. Control of selective perception using bayes nets and decision theory. *IJCV*, 1994.

[117] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[118] M. S. Ryoo. Human activity prediction: Early recognition of ongoing activities from streaming videos. In *ICCV*, 2011.

[119] Ashutosh Saxena, Sung H Chung, and Andrew Y Ng. Learning depth from single monocular images. In *NIPS*, 2005.

[120] Jonathan Shen, Noranart Vesdapunt, Vishnu N Boddeti, and Kris M Kitani. In teacher we trust: Deep network compression for pedestrian detection.

[121] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, and Russell Webb. Learning from simulated and unsupervised images through adversarial training. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[122] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[123] K. Soomro, A. R. Zamir, and M. Shah. UCF101: A dataset of 101 human action classes from videos in the wild. Technical Report CRCV-TR-12-01, 2012.

[124] Suraj Srinivas and R Venkatesh Babu. Data-free parameter pruning for deep neural networks. In *British Machine Vision Conference (BMVC)*, 2015.

[125] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[126] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.

[127] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. Going deeper with convolutions. Cvpr, 2015.

[128] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[129] Niket Tandon, Gerard de Melo, and Gerhard Weikum. Acquiring comparative commonsense knowledge from the web. In *AAAI*, 2014.

[130] Bart Thomee, David A. Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. The new data and new challenges in multimedia research. *arXiv preprint arXiv:1503.01817*, 2015.

[131] Sebastian Thrun. Is learning the n-th thing any easier than learning the first? *Advances in neural information processing systems*, pages 640–646, 1996.

[132] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*, 2015.

[133] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. Improving the speed of neural networks on cpus. In *Deep Learning and Unsupervised Feature Learning NIPS Workshop*, 2011.

[134] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. *arXiv preprint arXiv:1606.04080*, 2016.

[135] Oriol Vinyals, Charles Blundell, Timothy P. Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *NIPS*, 2016.

[136] M. Vondrak, L. Sigal, and O. C. Jenkins. Physical simulation for probabilistic motion tracking. In *CVPR*, 2008.

[137] Marek Vondrak, Leonid Sigal, and Odest Chadwicke Jenkins. Physical simulation for probabilistic motion tracking. In *CVPR*, 2008.

[138] Jacob Walker, Abhinav Gupta, and Martial Hebert. Patch to the future: Unsupervised visual prediction. In *CVPR*, 2014.

[139] Jacob Walker, Abhinav Gupta, and Martial Hebert. Dense optical flow prediction from a static image. In *ICCV*, 2015.

[140] Jason Wang and Luis Perez. The effectiveness of data augmentation in image classification using deep learning. Technical report, Technical report, 2017.

[141] Qian Wang, Xianyi Zhang, Yunquan Zhang, and Qing Yi. Augem: automatically generate high performance dense linear algebra kernels on x86 cpus. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013.

[142] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *NIPS*, 2016.

[143] Jeremy HM Wong and Mark John Gales. Sequence student-teacher training of deep neural networks. 2016.

[144] Sebastien C Wong, Adam Gatt, Victor Stamatescu, and Mark D McDonnell. Understanding data augmentation for classification: when to warp? In *Digital Image Computing: Techniques and Applications (DICTA), 2016 International Conference on*, pages 1–6. IEEE, 2016.

[145] Baoyuan Wu, Siwei Lyu, and Bernard Ghanem. Ml-mg: Multi-label learning with missing labels using a mixed graph. In *Proceedings of the IEEE international conference on computer vision*, pages 4157–4165, 2015.

[146] Cinna Wu, Mark Tygert, and Yann LeCun. Hierarchical loss for classification. *arXiv preprint arXiv:1709.01062*, 2017.

[147] Jiajun Wu, Ilker Yildirim, Joseph J. Lim, William T. Freeman, and Joshua B. Tenenbaum. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. In *NIPS*, 2015.

[148] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. *arXiv preprint*, 2015.

[149] Zhibiao Wu and Martha Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 133–138. Association for Computational Linguistics, 1994.

[150] J. Xiao, A. Owens, and A. Torralba. Sun3d: A database of big spaces reconstructed using sfm and object labels. In *ICCV*, 2013.

[151] D. Xie, S. Todorovic, and S.-C. Zhu. Inferring dark matter and dark energy from videos. In *ICCV*, 2013.

[152] Lingxi Xie, Jingdong Wang, Zhen Wei, Meng Wang, and Qi Tian. Disturblabel: Regularizing cnn on the loss layer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4753–4762, 2016.

[153] Yan Xu, Ran Jia, Lili Mou, Ge Li, Yunchuan Chen, Yangyang Lu, and Zhi Jin. Improved relation classification by deep recurrent neural networks with data augmentation. *arXiv preprint arXiv:1601.03651*, 2016.

[154] Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alex Smola, Le Song, and Ziyu Wang. Deep fried convnets. In *ICCV*, 2015.

[155] Jenny Yuen and Antonio Torralba. A data-driven approach for event prediction. In *ECCV*, 2010.

[156] Peng Zhang, Jiuling Wang, Ali Farhadi, Martial Hebert, and Devi Parikh. Predicting failures of vision systems. In *CVPR*, 2014.

[157] Xiangyu Zhang, Jianhua Zou, Xiang Ming, Kaiming He, and Jian Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *CVPR*, 2015.

[158] Bo Zheng, Yibiao Zhao, Joey C. Yu, Katsushi Ikeuchi, and Song-Chun Zhu. Detecting potential falling objects by inferring human action and natural disturbance. In *ICRA*, 2014.

Appendix A

NEWTONIAN IMAGE UNDERSTANDING: DATASET COLLECTION DETAILS

Here we describe how we collected natural images/videos of the VIND dataset. For each Newtonian scenario, we queried on YouTube keywords that involve those scenarios. For example, for scenario (4), which represents *rolling* dynamics, we queried variations of *billiard*, *bowling*, *soccer pass*, and *golf rolling* and downloaded top 200 videos for each query. Then, we pruned out videos that were irrelevant. For each remaining video, we segmented out at most 8 clips that contained the Newtonian scenario of interest. This procedure resulted in more than 6000 video clips that contain more than 200K frames.

To collect our static images, we used a similar set of queries on Google Images. We removed low-quality and duplicate images and ended up with 4516 images for all Newtonian scenarios. We considered one third of images (randomly sampled) as the validation set and the remaining images as our test set.

For scenario (5), which represents *stability*, we augment our dataset with frames from 8 annotated sequences of the SUN3D dataset [150], which show stable objects in office/hotel environments. We use 4 sequences for training and the other 4 for testing. In Figure A.3, we show some example images for each scenario.

We provide three types of annotations for each frame/image. First, we provide bounding box annotations for the objects that are described by at least one of our Newtonian scenarios. For video clips, we choose 5 frames randomly and annotate bounding boxes in those frames. Then, we find the location of the objects in other frames by interpolation. Second, we provide viewpoint annotations. We show the annotators the game engine videos that are rendered from different viewpoints of the same Newtonian scenario as that of the image/video clip

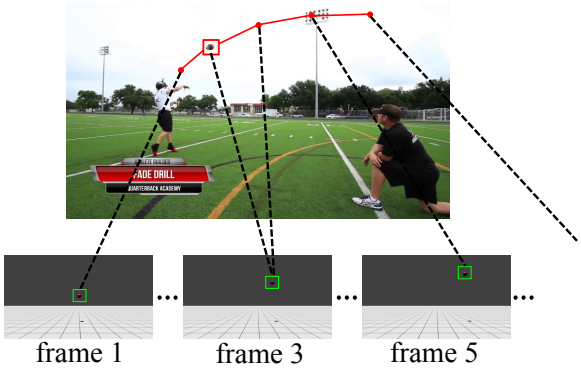


Figure A.1: **State annotation**. We match the movement of the object in video (red curve) with the 2D projection of the movement of the object in the game engine video. Using Dynamic Time Warping, we infer which frame of the natural video corresponds to which frame of the game engine video.

and ask them which viewpoint better represents the scenario in the image/video clip (refer to Figure 3 of the chapter). Finally, we provide *state* annotations for the objects. By *state*, we mean how far the object has moved on the expected scenario (*e.g.,*is the object in the beginning of the projectile? or is it at the peak point?). For each video clip, we sample 10 equally spaced frames from its corresponding game engine video (the first frame is the first frame of the game engine video and the tenth frame is the last frame of the game engine video). For video clips, we have bounding box annotations across all frames (as mentioned above). We also know the 2D location of the object in the game engine video. For annotation, we need to solve an optimization problem that finds the correspondence between the projected 2D movement of the object in the game engine video and the frames in the natural videos. To solve this problem, we use Dynamic Time Warping (DTW). DTW provides the best assignment *i.e.,*it specifies which video frame corresponds to which of the 10 frames of the game engine video. Figure A.1 shows an example for this process. The annotation procedure is different for images since we do not know the movement of objects in images. We show the 10 frames of the game engine video to annotators and ask them to

specify which frame (out of 10 frames) best shows the state of the object. Note that we do not use this type of annotation for training N^3. It is just used for our ablation study and also to evaluate how well we can approximate the state of the object.

A.1 Unseen scene types (Section 6.2)

To test the generalization of our method, we removed one scene type per Newtonian scenario from our training set and evaluated our method on images that represented those scene types. The list of removed scene types for each Newtonian scenario is as follows:

- Scenario (1): Playground scene
- Scenario (2): Scenes showing swinging with a rope
- Scenario (3): Soccer scene
- Scenario (4): Bowling scene
- Scenario (6): Table tennis scene
- Scenario (7): Diving scene
- Scenario (8): Scenes including cars
- Scenario (9): Volleyball scene
- Scenario (10): Rugby scene
- Scenario (11): Tennis scene
- Scenario (12): Weightlifting scene

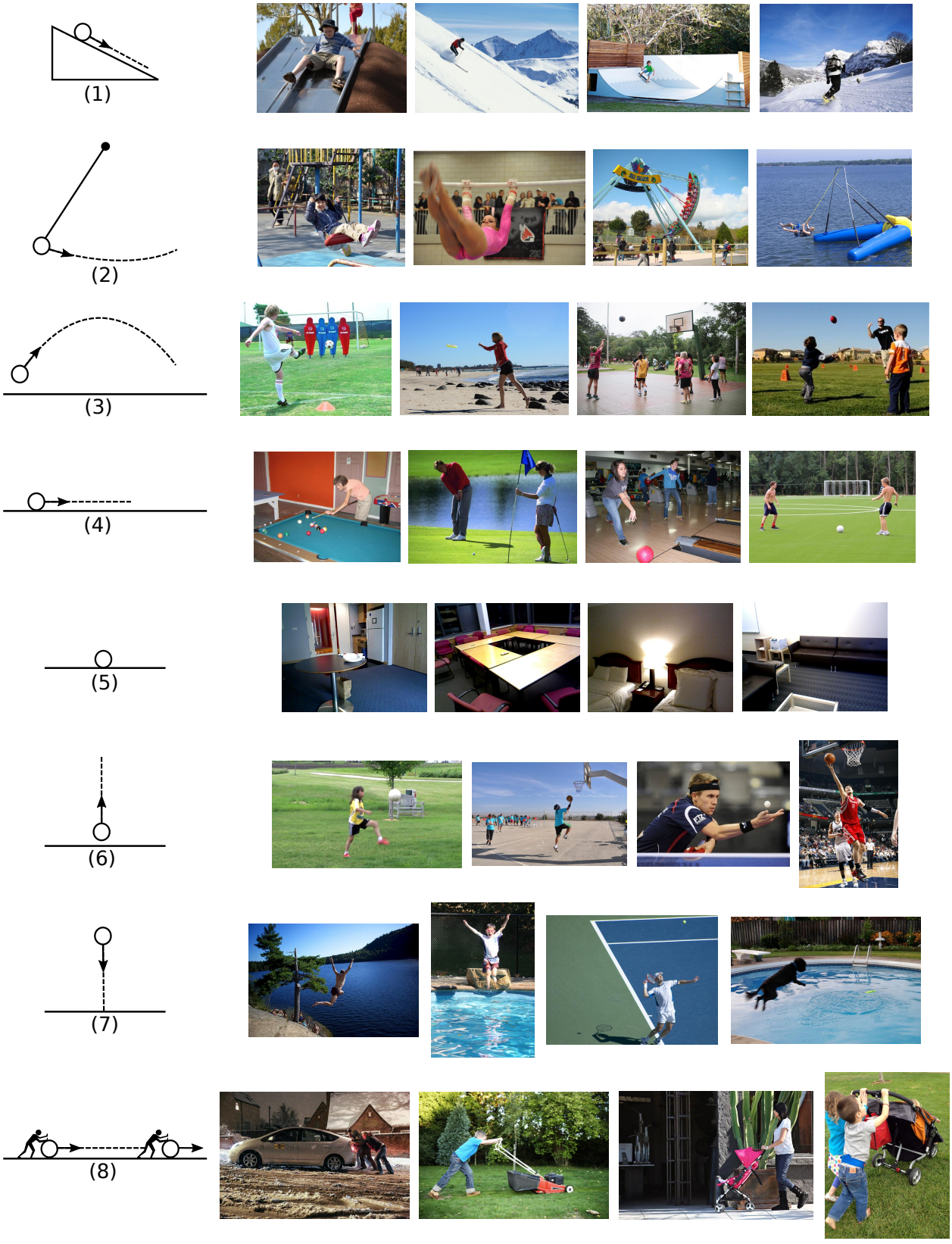


Figure A.2: (left) The Newtonian scenarios. (right) Four example images that show the Newtonian scenario.

Figure A.3: (left) The Newtonian scenarios. (right) Four example images that show the Newtonian scenario.

Appendix B

LOOKUP-BASED CONVOLUTIONAL NEURAL NETWORKS: EXPERIMENT DETAILS

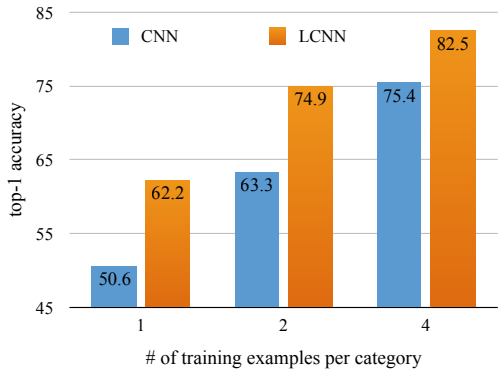
B.1 Layer-wise speedup

In this section we compare the layer-wise speedup of LCNN with the baselines. In AlexNet most of the computation is done in the early layers, where the input size is still large. Table B.1 shows the percentage of the computation in each layer of AlexNet and the speedup gain of each model on each layer. XNOR-Net [111] gets 32× speedup on 32-bit machines, and it can be higher for 64-bit or 128-bit machines. However, since they don't binarize the first layer, where 9.29% of computation is done, their speedup is bounded by $\frac{1}{9.29\%} = 10.8\times$. This is still much lower than LCNN-fast speedup, which gets about the same accuracy. Wen *et. al.* [142] gets good speedup on conv2-5, yet their speedup is much lower on the first layer. We think this is because they're sparsifying the convolution tensors. The convolution tensor in the first layer cannot become very sparse as they are performing on the input itself, which has only 3 channels. LCNN-accurate, however, is speeding up the first layer by representing the convolution tensor by a sparse combination of a set of vectors. This allows a more compact representation, and therefore larger speedup in that layer.

B.2 Few-example trials

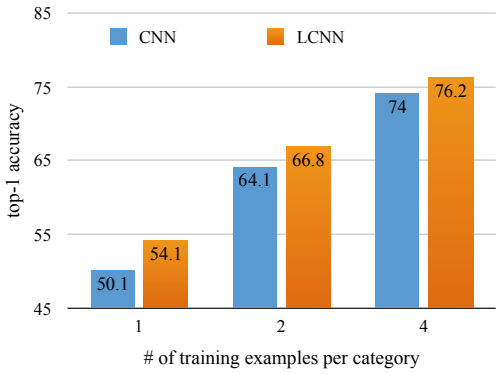
We do the few-example experiment under two settings: 1) Try 5 random samplings of 10 random categories for few-example training and report the average over all. 2) Set aside all cats (7 categories), bicycles (2 categories) and sofa (1 category). For the latter setting, the categories listed in Figure B.4 are excluded:

In each of the trials, we repeat the random sampling of the few examples (1, 2 or 4



(a) Trial #1 categories:

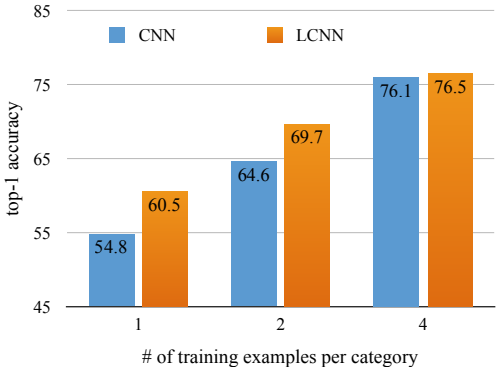
1-	n01514859	hen.n.02
2-	n01773549	barn_spider.n.01
3-	n01978287	dungeness_crab.n.02
4-	n02099429	curly-coated_retriever.n.01
5-	n02669723	academic_gown.n.01
6-	n03888257	parachute.n.01
7-	n03995372	power_drill.n.01
8-	n04005630	prison.n.01
9-	n04467665	trailer_truck.n.01
10-	n13133613	ear.n.05



(b) Trial #2 categories:

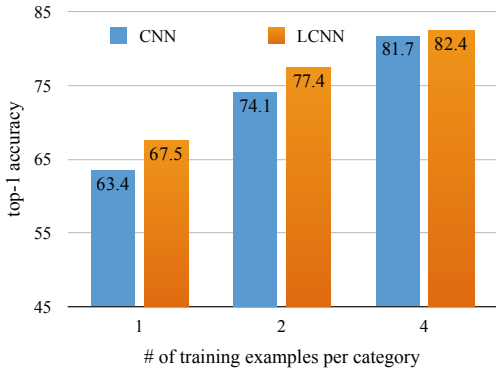
1-	n01983481	american_lobster.n.02
2-	n02091467	norwegian_elkhound.n.01
3-	n02444819	otter.n.02
4-	n02607072	anemone_fish.n.01
5-	n02817516	bearskin.n.02
6-	n02879718	bow.n.04
7-	n03530642	honeycomb.n.02
8-	n03908618	pencil_box.n.01
9-	n04286575	spotlight.n.02
10-	n04554684	washer.n.03

Figure B.1: Comparing LCNN and standard CNN on few-example training. LCNN beats standard CNN in all samplings.



(a) Trial #3 categories:

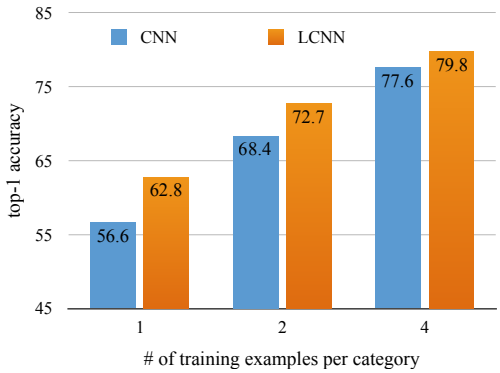
1- n02110063 malamute.n.01
2- n02111277 newfoundland.n.01
3- n03724870 mask.n.01
4- n03775546 mixing_bowl.n.01
5- n03782006 monitor.n.05
6- n03929660 pick.n.05
7- n04201297 shoji.n.01
8- n04487081 trolleybus.n.01
9- n07753113 fig.n.04
10- n07930864 cup.n.06



(b) Trial #4 categories:

1- n01669191 box_turtle.n.01
2- n01773157 black_and_gold_garden_spider.n.01
3- n02106662 german_shepherd.n.01
4- n03733131 maypole.n.01
5- n03929855 pickelhaube.n.01
6- n04116512 rubber_eraser.n.01
7- n04389033 tank.n.01
8- n04590129 window_shade.n.01
9- n04592741 wing.n.02
10- n07836838 chocolate_sauce.n.01

Figure B.2: Comparing LCNN and standard CNN on few-example training. LCNN beats standard CNN in all samplings.



(a) Trial #5 categories:

1- `n01774384` black_widow.n.01
2- `n02090379` redbone.n.01
3- `n02113023` pembroke.n.01
4- `n02138441` meerkat.n.01
5- `n02444819` otter.n.02
6- `n02917067` bullet_train.n.01
7- `n03016953` chiffonier.n.01
8- `n03180011` desktop_computer.n.01
9- `n03207941` dishwasher.n.01
10- `n03476684` hair_slide.n.01

Figure B.3: Comparing LCNN and standard CNN on few-example training. LCNN beats standard CNN in all samplings.

1-	n02123045	tabby.n.01
2-	n02123159	tiger_cat.n.02
3-	n02123394	persian_cat.n.01
4-	n02123597	siamese_cat.n.01
5-	n02124075	egyptian_cat.n.01
6-	n02125311	cougar.n.01
7-	n02127052	lynx.n.02
8-	n02835271	bicycle-built-for-two.n.01
9-	n03792782	mountain_bike.n.01
10-	n04344873	studio_couch.n.01

Figure B.4: Excluded categories under the setting when semantically and visually similar categories are excluded together. The first 7 categories are cats, categories 8 and 9 are bicycles, and category 10 is a sofa.

examples) 20 times. We evaluate the performance of LCNN and CNN on each random sampling and get the average over all. Figure B.3 shows the categories that have been excluded and the performance of LCNN and the CNN baseline in each trial. Notably, LCNN is consistently getting higher accuracy in all trials.

AlexNet	conv1	conv2	conv3	conv4	conv5	fc6	fc7	fc8	overall
computation %	9.29%	39.45%	13.17%	19.76%	13.17%	3.33%	1.48%	0.36%	100%
Wen *et. al.* [142]	1.05×	3.37×	6.27×	9.73×	4.93×	1×	1×	1×	3.1×
XNOR-Net [111]	1×	32×	32×	32×	32×	32×	32×	1×	8.0×
LCNN-fast	16.66×	80.24×	83.23×	75.47×	61.99×	7.73×	7.91×	1×	37.6×
LCNN-accurate	6.97×	2.57×	3.51×	3.75×	3.21×	3.14×	3.83×	1×	3.2×

Table B.1: Comparing the layer-wise speedup of each model on AlexNet. The accuracy of each model is reported in the chapter.