University of Nevada, Reno

Deep Convolutional Neural Networks for Multilabel Prediction Using RGBD Data

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

by

Liesl Wigand

Dr. Monica Nicolescu / Thesis Advisor

May 2014

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

LIESL WIGAND

entitled

Deep Convolutional Neural Networks for Multilabel Prediction Using RGBD Data

be accepted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

Monica Nicolescu, Advisor

Mircea Nicolescu, Committee Member

Murat Yuksel, Graduate School Representative

Marsha H. Read, Ph. D., Dean, Graduate School

May, 2014

Deep Convolutional Neural Networks for Multilabel Prediction Using RGBD Data

by

Liesl Wigand

Submitted to the Department of Computer Science and Engineering
on May 18, 2014, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

Robotics relies heavily on the system's ability to perceive the world around the robot accurately and quickly. In a narrow setting as in manufacturing this goal is relatively simple. To make robotics feasible in more dynamic settings we must handle more objects, more attributes, and events that may be out of the scope of what a system has been exposed to previously. To this end, the present work focuses on automatic feature formation from RGB-D data, using deep convolutional neural networks, in order to recognize, not only objects but also attributes which are more applicable across objects, including those objects which have not been seen previously. Progress is shown in relation to more standard systems and near real-time classification of multiple targets is achieved.

Thesis Supervisor: Monica Nicolescu
Title: Associate Professor

Acknowledgments

Thank you very much to my advisor and committee, as well as my labmates.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

For robotic systems to be feasible outside of controlled settings, they need a more advanced understanding of the world, and a level of adaptability to new situations that does not currently exist. The distant goal of a truly autonomous system, seems ever more possible as techniques in vision, machine learning and artificial intelligence continue to develop.

In particular, the use of combinations of neural networks and other learning models have shown state of the art performance in document classification, scene segmentation, and image retrieval. Unfortunately, there are still limitations to these techniques that require further study.

The goal of this work is object and attribute recognition. We make use of feature extraction techniques such as Isomap, and classifiers like convolutional neural networks and support vector machines in different combinations with cross-validated parameter settings to predict objects and attributes of objects.

1.1 Motivations

The majority of robots in the world today are employed in factories and similar environments, where they work in a closed and carefully structured environment, with little variation. Removing robots from this environment introduces the complications

of a complex dynamic environment. Where a robot in a factory may never need to move more than a few feet, a robot working in a home may need to travel from room to room in a variety of house plans. When a robot welds a seam on a car that sits on a conveyor belt, it does not need to identify the car in different positions or in motion the way a robot in a hospital would need to when interacting with a patient. The factory robot may see the exact same product every day, while a robot in a school may see and work with any number of books, papers, students, and activities. To do well in these situations a robot should be able to learn primitive concepts, which can be extended and related to new situations.

This is by no means simple. Recognizing a single object can be difficult. Change the background, the lighting, the size, and an object may become impossible to recognize. People are presented with these issues everyday, and while their abilities are not perfect, they still outperform machines. Creating a single system that solves all these problems is a long term goal; this work focuses on smaller problems which contribute to a complete solution.

The following sections discuss particular goals and difficulties which are addressed in the approaches described in later chapters. The main goal is to identify object attributes, such as *smooth*, *red*, or *round*. This problem leads directly to a need for multiclass and multilabel prediction, as well as a search for data and features in the data which can better represent attributes. As a result, the entire pipeline of the learning process is investigated for potential improvements: data formats, preprocessing, feature extraction, and classification.

1.1.1 Attributes and Concepts

Concept learning is the central problem of machine learning. The problems of detecting objects, recognizing people, and identifying their pose all rely on finding an abstract concept based on features extracted from observations. These observations must be used to develop a model that generalizes the characteristics of the concept. In practice concept learning is difficult. Concepts may be activities, objects, or at-

tributes. They may be associated with things that can be seen or otherwise sensed, or done. Often the information is incomplete or noisy. To simplify this problem we focus on objects, types of objects, or their attributes.

Systems that recognize objects may be able to survive in narrow settings, learning the specific tools they need to use. In more dynamic settings, psychological research has shown that humans make use of attributes and uses of objects when finding, identifying and describing them, and it has been shown that computer systems can do the same [4, 20]. When a person needs to point out a specific object they may say "the red mug" or more generally "it is flat and round, with brown stripes." These attributes are then useful in locating unknown objects, or objects which are similar to others as in the case of several different cups or plates. These references are useful across different objects, and may be used to describe an object which has never been seen before, allowing the system to locate an object without knowing what it is, as long as the system can recognize attributes.

Attribute recognition is an active area of study, and has been shown to enhance object recognition, but continues to perform poorly compared to direct object recognition due to issues of overlapping classes and labels which disagree. For example, while two people may agree that an object is a mug, one may say it is *orange*, another *red*, and they may describe it as a *large brightly colored mug* or as *something for drinking from*. Conflicting labels are often ignored based on the assumption that with sufficiently many examples, there will be a majority which agree.

Attribute learning has the same problems as any classification problem, but these issues arise sooner and more frequently. Essentially an object may only be described as one or two objects, such as "vehicle," "car," and "sedan," which are either synonyms or exist in a hierarchy. The same object may have dozens, or hundreds of attributes related to color, size, shape, quality, or material, and none of these may have anything to do with what the object is (a car may be any color). The problem is immediately broader, requiring more examples and better coverage of the attribute in the training data. Conflicts between these labels are also more likely when attributes blend and

exist in scales and ranges, as in the case of colors and sizes. Reds mix with purples and oranges; large may only be large in a comparison; some features change with perspective and setting.

1.1.2 Multilabel vs Multiclass

Successful work has been done to prepare robots for diverse settings, where many classes are present. Systems exist that can accurately classify images into hundreds of different classes [45]. However, these systems require large amounts of data, and generally assume independent classes and single class membership. For example, most datasets for objects provide a single unique identifier for a given image, so that if an object is labeled as *car* it is unlikely that *vehicle* or *automobile* are also available labels in the set. As mentioned above, attribute classifiers cannot generally make this assumption.

These systems also do not allow separation of objects from their contributing features. For example, a system may have seen a *red book* and a *blue ball*, and could not predict a *red ball*, but more generally, if it had seen a *red book* and a *blue book* it would learn to discount color completely and ignore it except when determining the shape of the object. To avoid losing this information, attributes are a valuable class of labels.

Multiclass prediction is prediction which allows a sample to belong to one of some number of classes which could be either two or possibly thousands. Multilabel, or multitask prediction allows a set of labels to apply to the sample. The simplest way of implementing both of these is to treat the labels separately, implementing a classifier per label. This approach and more complicated methods are explored in this thesis.

1.1.3 Data Representation

The richest source of information for the system is often visual, although this is also the most complex. Video from a camera provides information in 2D, across color

channels, and through time, all of which can contain information that is valuable separately and in combination. The attributes which people use to describe objects are usually visual, and so the data which this work relies on is image data, with text labels. We also use depth data as collected by Kinect cameras, and show its influence on learning.

Generally, text is treated as individual words or groups of words, that is, unigrams, bigrams, trigrams, etc. In this case we only use unigrams as labels and ignore sentence structure and words which have no substantial meaning (stop words). The images are considered directly, except where pre-processing is mentioned. Generally images are either transformed into a different color space, such as YUV or HSV, and the values are sometimes normalized [47]. These changes can substantially change the features and predictions that result from later steps, but vary in usefulness depending on the methods and application. For example YUV can be especially useful when recognizing skin tones.

1.1.4 Features

The training data may be in a format that is ready for immediate classification, as when the data is a simple vector of numbers, but often the data is too complex, as in the case of video. There may be extra information that doesn't matter, or the data may be so complicated that algorithms cannot process it. For example, images are, of course, multidimensional, and classification systems can usually only handle "flat" feature vectors. There is also a great deal of noise, in the form of background, clutter, and slight variations in the image that have little or no influence on the final prediction of a label. Feature extraction and selection process the data by removing unimportant information and compressing the portion of the data which represents the most or best information. Often these features in images are intended to be invariant to rotations, translations, and scaling, so that they may be found anywhere in a scene and still recognized. This is discussed further in the next chapter.

1.1.5 Classifiers

As mentioned above, classifiers can predict single classes, multiple classes, or several classes at a time. Classifiers use functions to map data to labels. This is a supervised learning approach, where labels for each sample are provided and used to adjust the model. Unsupervised learning involves clustering the data, or otherwise improving the model of the data without known labels. The complexity of the classifier should be adjusted to suit the data in order to avoid either over generalizing or over-fitting, which result in poor performance. Classifiers can also include some sort of feature extraction, as in the case of Neural Networks. Several types of classifiers are used in this work, and unsupervised techniques are touched on briefly.

1.2 Contributions and Goals

The final goal is a system that uses RGB-D data to identify objects and their attributes in near real time. To accomplish this goal, the problem was broken down into smaller systems, and protoypes were developed. Initially we began with a system to predict multiple labels using separate classifiers, beginning with multiple classes of objects, then including attributes as well. This system used Isomap to create features, and a separate SVM classifier for each label, and was used on generated simple images, and real color images taken from the Kinect camera (not including depth). The next system used convolutional neural networks in order to improve 2D feature learning, and depth data was added. Camera streaming was used during testing, and prediction approached real time. The final version focused on attributes and the multilabel problem.

The next chapter presents background and related work. Chapters 3 to 7 describe the systems implemented to predict attributes and their performance. Chapter 3 describes the initial test system which makes use of Isomap and Support Vector Machines to predict labels for both generated and real data. Chapter 4 updates the first project, adding additional types of feature extraction, classifiers and preprocessing,

as well as a more complete parameter search. Neural Networks are used in Chapter 5 for multiclass object prediction on video streams. Chapter 6 presents a system that attempts to predict multiple labels at once using a single convolutional neural network. Results across systems are discussed in Chapter 7. The final chapters contain future work and conclusions.

Chapter 2

Related Work

Real-time recognition, whether of objects, faces or attributes, is essential to continuing artificial intelligence and robotics work. Contributing to this work, are techniques from computer vision and general machine learning, with constraints added by the robotics system. Usually this means any technique is feasible if it can be made fast, which is not easy. It may also include memory constraints if the system must run entirely on a single robotic platform. Here, it is assumed that communication between several machines removes any memory limit.

The following sections discuss algorithms which were used and similar systems that use these algorithms or which attempt to solves the same problems.

2.1 Attribute Recognition

Studies specific to the issue of attribute recognition are unusual. Recently a paper was presented which predicts three classes of attributes, color, material, and shape, from the RGB-D dataset [46]. They assumed that there was only one of each of these labels per object, so if an object was two colors, they only accepted the more prominent color. They show object identification based on the three attributes, obtaining 85 to 97% accuracy. Another system uses the PubFig dataset of labeled faces of public figures to identify attributes such as skin and hair color, as well as lighting [32].

This system achieved around 80 to 88% accuracy on most facial features, including attributes like *Masculine*, *White*, *Young*, *Smiling*, and *Chubby*. Both of these systems narrowed the scope of the problem. The first limits the attributes to be of a certain type, and only one per type allowed for an object. The second focuses only on human faces.

2.2 Machine Learning

Machine learning involves the construction of systems that can learn from data. Supervised systems require labeled data and attempt to find the relation between the input data and the resulting label, while unsupervised methods learn from unlabeled data by clustering, or otherwise finding structure. The data may be presented directly to the learning system or preprocessed, often by feature extraction. In this work, the focus is on data representation and supervised learning techniques, where the data is presented as images and paired with sets of English labels.

2.2.1 Feature Extraction

Feature representation is an important part of the learning process. Some classifiers are run directly on the training data, but often the quality of the system is improved by selecting and creating better representations of the data before the classifier. Feature selection is the process of dropping features that are unimportant or redundant, while feature extraction often involves reducing the dimensionality of the data by compressing the information with a constraint on how much data is lost, or how useful the remaining data is. This provides a simpler problem to the classifier, by providing less or lower dimensional data to learn from as well as more relevant data.

The processes of feature selection and extraction may be performed with knowledge of the prediction problem or without. For example, if the goal is to predict the color of a shape, knowledge about the current accuracy given a set of features can be used to select those features. The shape won't matter to a prediction of *red*, but

pixel values within the shape will. However creating the features can require expert knowledge of the application and finding good combinations of features can be time consuming. There are automatic techniques that attempt to measure the amount of information in the data and preserve certain aspects of it. Principal Component Analysis [25], Independent Component Analysis [22], and Isomap [48] are all examples of this type of feature extraction.

Isomap is meant to create a limited set of features from a larger feature space by preserving geodesic distance between samples [48]. This means that points of data that are close together in the original space are close together in the new space. Isomap can be seen as a type of kernel Principal Component Analysis, with the geodesic distances as a kernel. The number of resulting dimensions must be chosen, either automatically based on training error or a measure of data loss, or chosen by a user. If this value is too small or large then the resulting space is inaccurate, and information is lost or poorly represented. This representation can also be poor if the data was noisy to begin with.

Another feature extraction technique which is used in this system is a type of codebook feature creation [7]. This method collects overlapping patches from an image and uses K-Means Clustering to create a set of template patches. Then a vector of k true or false values is created, where each of the entries indicates that a given type of image patch was seen in the image. The choice between these two methods becomes a trade-off between time and space, although for simple data both may reduce to comparable dimensions.

2.2.2 Support Vector Machines

After features are extracted or without such processing, a classifier is used to model the data. Support Vector Machines are maximum margin classifiers that construct a set of hyperplanes to divide the classes [8]. The original space is mapped into a higher or infinite dimensional space in order to make it simpler to separate, and computations are simplified by using the kernel trick to map data into an inner product space,

without having to explicitly calculate the mapping. The learning algorithm would use inner products on data in the higher dimensional space, but instead can find them in the original space with kernel functions.

Support Vector Machines have a special version for single class problems that are meant for outlier detection. If a dataset has only positive examples, or very few negative examples this version creates hyperplanes around the positive data instead of between classes. As a result a margin (how far out to place this boundary) becomes an important parameter. In practice one-class problems are difficult since they require much better coverage of the positive class. That is, if the positive class extends farther than the samples indicate, many false negatives result, and if the negatives do overlap with the positives, prediction fails. If at all possible, negative samples should be used, at the very least to aid in selection of the margin.

2.2.3 Neural Networks

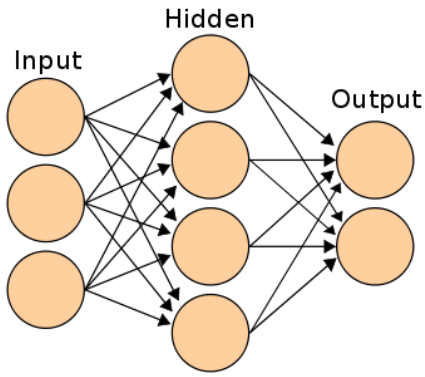


Figure 2-1: Neural Network [51]

Neural Networks are models made of sets of neurons connected in one or more layers, through which inputs are passed to produce outputs. They were devised in the 40's, but have not seen much use until very recently due to their complexity and the difficulty of training [35].

There are many types of neural networks which vary in depth, shape, what connections are permitted between nodes and layers, activation functions and training methods. In its simplest form, a single neuron takes one input and produces one output. More generally it takes some number of inputs and produces an output which may contribute to one or more nodes in the next layer. This is seen in Equation 2.1.

$$\boldsymbol{y} = \boldsymbol{\sigma}(\boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b}) \tag{2.1}$$

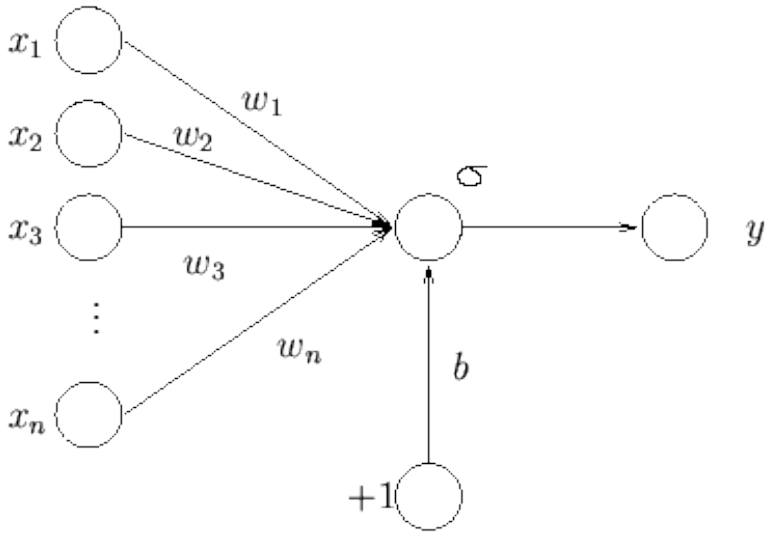


Figure 2-2: Neural Network [18]

This function is shown graphically in Figure 2-2, where x is the input vector, y is the output, W is a matrix of weights, b is a bias term and σ is a non-linear function such as Sigmoid or hyperbolic tangent. Groups of these neurons are then linked together to form larger networks.

Autoencoders

Autoencoders are a special type of unsupervised neural network which learn a compressed representation of the input. They do this by having an encoding and decoding layer, or several sets of such layers. The encoding layers are smaller than the input dimension, forcing a compression, and which is then reversed by the decoding layers. Autoencoders are trained by comparing the input to the output, that is, the decoded values should be the same or as close as possible to the un-encoded values. For classification, the decoder is removed, and the output of the decoder is used as input to either a final network layer or other type of classifier which is then trained using the labeled data.

Denoising Autoencoders have been shown to be more robust to noise, and improve generalization by adding random noise to the training data. This can be thought of as

similar to bootstrapping, and drop-out in other networks. A convolutional denoising autoencoder has been presented recently [34] and shown to have competitive results on MNIST [31] and CIFAR [27].

Convolutional Neural Networks

Convolutional Neural Networks are designed to work with images by using 2D convolutions for one or more layers. Given a simple 2D filter, also called a kernel or convolution matrix, it is applied to an image patch by multiplying each value in the filter by the corresponding pixel value, then adding these values together to produce a pixel value in the next layer. The size of the output depends on the size and shape of the image and filter, as well as whether it is applied in overlapping positions or with padding around the edges.

Shown in Figure 2-3 is a simple convolution, with no padding, and some overlap. Padding refers to adding zero valued pixels around the edges so that the convolution can run over the edge, while overlap indicates that the filter overlaps by some amount with the previous filter position.

In a convolutional neural network, a node becomes a convolutional operation, where the weight matrix is a collection of kernels which are applied to each of the in-

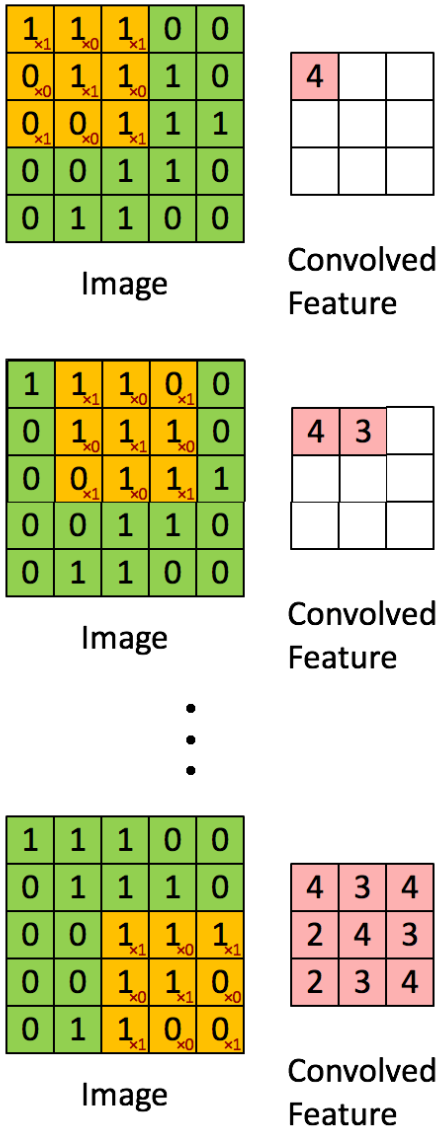


Figure 2-3: Convolution: First and second show initial convolution steps; final is the end result. [10]

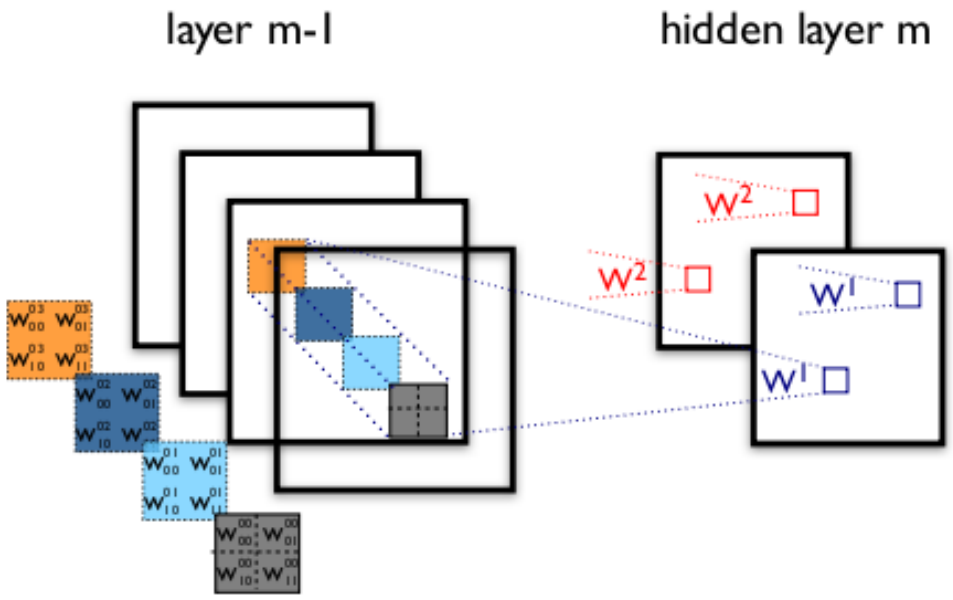


Figure 2-4: Convolutional Neural Network [10]

put layers. In Equation 2.2, h_{ij}^k is the resulting hidden representation of the data at point i, j after a convolution with filter k. Shown in Figure 2-4 are two layers in such a network.

$$h_{ij}^k = tanh((W^k * x)_{ij} + b_k) \tag{2.2}$$

In a color image this means a single convolutional node would be an x by y by 3 kernel which would be applied to each position in the image with additional parameters indicating how much overlap and padding in order to produce a new image of a different dimensionality, where each pixel represents several nearby pixels from the previous layer. Parameters for this system include padding, overlap, size of the filters, which are assumed to be square, the number of these convolution nodes, and whether we consider the channels as separate or connected. Connected channels means that the system sums across channels or share weights across them, or each can be treated as a separate network. These networks are generally much larger than standard networks, especially since the more complicated the data the more filters are needed to represent it.

Convolutional networks have been used recently on ImageNet, CIFAR, and MNIST, and in combination with other classifiers to reach state-of-the-art performance. A large GPU implementation of pure convolutional networks was used in the ILSVRC-2012 competition and achieved top-5 test error rate of 15.3% [28]. Top-5 error rate is the percent of test cases where the correct solution was not in the top five ranked predictions. Following the competition, alterations were made to the network structure and several other reported improvements and extensions [13, 24]. A very recent work extended the system to perform localization and detection of objects [45]. These implementations are the inspiration for much of the current work.

2.2.4 SVM vs NN

SVMs perform as well as Neural Networks on a variety of classification problems, although SVMs perform better than neural networks if these networks are not carefully tuned [6, 36]. An unfortunate problem with neural networks is the number of hyper-parameters that must be selected: the number of hidden layers, the number of nodes within each layer, the choice of non-linear function, to be applied at these nodes, and the cost function as well as types of regularization and learning rate. Several papers have shown that when these parameters are adjusted carefully, neural networks, especially deep networks, outperform SVMs [2]. Some works have used both, either combining the classifications of both models, or using neural networks as feature extractors that feed into SVMs, which get better performance in some cases then either on its own [21, 49].

There are limitations to each, and problems which are handled better by one or the other. SVMs train faster than Neural Networks and may require less data, but if a problem is complex, an SVM will be slower than a comparable Neural Network when making predictions.

2.2.5 Multiclass and Multilabel Learning

An important issue being pursued for many applications is multitask or multilabel learning. In the simplest classifiers, there are two classes, and the result is True, the sample is of the class, or False the sample is not of the class. A step up from that is Multiclass learning, which allows more classes, such as *apples*, *oranges*, and *bananas*, where it is assumed that every class seen is exactly one of these. Multilabel learning, as will be mentioned throughout this work, attempts to classify a sample into several classes, such as *apple*, *fruit*, *round*, and *red*. This complicates the system as discussed previously in 1.1.2.

There are techniques for handling this, but they generally simplify the problem by assuming independence of the labels, or they perform poorly when compared to other classifiers for multiclass or single class problems. The easiest way to handle the problem is to have a separate classifier for each label, but this loses any information that may be gained from other labels. There are several works that use neural networks with multilabel output and cost functions, which are relevant to the current work.

For those classifiers which attempt to predict multiple classes at once, special loss functions and measures of accuracy are used. Traditionally, for neural networks this involves Hamming loss, softmax loss, or ranking loss. Accuracy may be given per class, averaged across all classes, or counted as "all or nothing," which gives no credit to samples that are labeled partially correctly. In most cases it is better to give partial credit, and to use precision, recall, negative rate and F1 score as well to reveal issues related to unbalanced data and over-fitting. Many papers reference accuracy alone, but if there is uneven representation of classes, or negatives and positives of each class, then other scores reveal issues.

Hamming Loss is shown in Equation 2.3. This loss has been popular, and gives value to predictions which are partly correct, by counting those predictions which are not correct using *xor* and averaging across the classes. However, this has been shown to be less useful then other loss functions and inconsistent for certain problems and

types of learning [14, 17, 15].

$$H(x_i, y_i) = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{xor(x_i, y_i)}{|L|} \tag{2.3}$$

where

$|D|$ is the number of samples

$|L|$ is the number of labels

y_i is the ground truth vector

x_i is the prediction vector

Softmax loss is just like traditional softmax, except that more then just the top prediction is accepted. As seen in Equation 2.4, softmax produces probabilities for each class, which sum to one. This is not a good technique for multilabel prediciton: the more labels are present, the smaller all the probabilities become. However, it has been shown to work nearly as well as ranking loss, which is currently one of the best multilabel loss functions [15].

$$\begin{aligned} p_{ij} &= \frac{e^{f_j(x_i)}}{\sum_{k=1}^{C} e^{f_k(x_i)}} \\ J &= -\frac{1}{n} \sum_{i}^{n} \sum_{j}^{c^+} \frac{1}{c^+} log(p_{ij}) \end{aligned} \tag{2.4}$$

where

p_{ij} is the probability of class j given sample x_i

$f_j(x_i)$ is the predicted value for class j at sample x_i

C is the number of classes

c^+ are the positive labels

Ranking loss, and Weighted Approximate Ranking Pairwise loss, shown in Equation 2.5, and 2.6 have been shown to perform slightly better than softmax, although

none achieve very high precision and recall, with the best results being 20 per class precision, and 50 per class recall, with much higher accuracy of 80 to 90% [15]. Ranking loss is more difficult to use with neural networks since the loss is not directly differentiable. Instead sub-gradients must be used to approximate the gradient at discontinuous points.

$$J = \sum_{i}^{n} \sum_{j}^{c^+} \sum_{k}^{c^-} max(0, 1 - f_j(x_i) + f_k(x_i))) \tag{2.5}$$

where

n is the number of samples

$f_j(x_i)$ is the predicted value for class j at sample x_i

c^- are the negative labels

c^+ are the positive labels

$$\begin{aligned} J &= \sum_{i}^{n} \sum_{j}^{c^+} \sum_{k}^{c^-} L(r_j) max(0, 1 - f_j(x_i) + f_k(x_i))) \\ L(r_j) &= \sum_{j=1}^{r} \alpha_j \text{ with } \alpha_1 \geq \alpha_2 \geq ... \geq 0 \\ r_j &= \left\lfloor \frac{c-1}{s} \right\rfloor \end{aligned} \tag{2.6}$$

where

n is the number of samples

$f_j(x_i)$ is the predicted value for class j at sample x_i

c^- are the negative labels

c^+ are the positive labels

r_j is the rank for class j

s is the number of times we sampled for negative labels

To measure the quality of prediction, it is useful to use both per class and across class measures, especially for unbalanced data where some classes are better represented in the data than others. There are several versions of precision and recall used for multilabel problems, and we use the form in 2.7, as well as F1 score to evaluate our system. Overall precision and recall, as shown in 2.8 are also used. Other methods of multilabel prediction were considered, but not implemented, and will be mentioned in section 7.4.

$$\begin{aligned} precision &= \frac{1}{c}\sum_{i}^{c}\frac{N_i^c}{N_i^p} \\ recall &= \frac{1}{c}\sum_{i}^{c}\frac{N_i^c}{N_i^g} \\ F1 &= \frac{2 * precision * recall}{precision + recall} \end{aligned} \tag{2.7}$$

where

c is the number of classes

N_i^c is the number of correctly labeled samples for class i (true positives)

N_i^p is the number of predictions for class i (true positives + false positives)

N_i^g is the number of ground truth labels for class i (true positives + false negatives)

$$\begin{aligned} precision_o &= \frac{\sum_i^c N_i^c}{\sum_i^c N_i^p} \\ recall_o &= \frac{\sum_i^c N_i^c}{\sum_i^c N_i^g} \end{aligned} \tag{2.8}$$

2.2.6 Tricks for Neural Networks

There are several useful additions to neural networks which aid in performance. Generally neural networks are set to have many parameters, more than are needed to

represent the data, which would produce over-fitting, but regularization is added to limit this effect. They also require a large amount of data which can be very difficult to gather, so there are methods of simulating additional data.

There are several standard ways to prevent over-fitting in Neural Networks. The first involves using L1 or L2 regularization, where the L1 or L2 norm of the parameters of the network are added to the cost function. The L1 norm is generally better for very small amounts of data, while L2 performs better on more data, and also on rotationally invariant data. It is best to use a combination and set the coefficients with cross-validation.

$$\begin{aligned} L1 &= \sum_{i=0}^{n} |param_i| \\ L2 &= \sum_{i=0}^{n} (param_i)^2 \\ cost &= loss + \lambda_1 * L1 + \lambda_2 * L2 \end{aligned} \tag{2.9}$$

where

$param_i$ is the ith parameter of the model

$loss$ is the loss function

λ_1 and λ_2 are weights on the regularization terms

$cost$ is the final regularized cost

Early stopping can also prevent over-fitting, by stopping training as errors increase. Generally during training, the error decreases. However this decrease is not smooth, it may decrease for a while, then increase, then decrease, while the overall effect is a decrease. Early stopping attempts to detect whether the error is decreasing overall or not, and stops training if it seems to be increasing instead. There are several versions of early stopping which rely on slightly different measures of the change in error. This work makes use of GL and PQ early stopping, described below. For

more information see [41].

$$\begin{aligned} E_{opt}(t) &= \min_{t' \leq t} E_{va}(t') \\ GL(t) &= 100 * \left(\frac{E_{va}(t)}{E_{opt}(t)} - 1 \right) \end{aligned} \tag{2.10}$$

where

GL is the generalization loss at epoch t

$E_{va}(t)$ is the error on the validation set at epoch t

$E_{opt}(t)$ is the best error up to epoch t

Stop when GL is greater than some α

$$\begin{aligned} P_k(t) &= 1000 * \left(\frac{\sum_{t'=t-k+1}^{t} E_{tr}(t')}{k * \min_{t'=t-k+1}^{t} E_{tr}(t')} - 1 \right) \\ PQ &= \frac{GL(t)}{P_k(t)} \end{aligned} \tag{2.11}$$

where

$E_{tr}(t)$ is the training error during epoch t

k is some number of past epochs

$P_k(t)$ is how much larger the average training error was than the minimum error

PQ is how much generalization loss over training progress

Networks can also be made more robust by adding noise to the data (see autoencoders) or by random drop-out of values in the network [19]. Dropout, and DropConnect are techniques that randomly drop either activations or weights in a network. They simulate an averaging of networks, by in a sense training a single network to represent many architectures: each network with dropped values represents a version of the full network where the dropped values are either dropped nodes or

connections in the graph.

To get the best behavior from neural networks, combinations of the above tricks are necessary.

2.3 Computer Vision Techniques

Computer vision and machine learning have significant overlap, especially when discussing recognition approaches. Here several techniques which have been specially applied to and developed for vision are discussed. The main difference between vision and other machine learning, is the need to represent 2 and 3 dimensional data. Local regions in an image can be transformed into a feature, or the system can act on the pixels directly, but the location of a value and nearby values in an image provide useful information which should not be ignored.

In simpler classification problems the data may be represented by a few numbers which are totally independent, as in the price of a house relating to the number of bedrooms, bathrooms, and floors. In an image, each location contains a set of colors (or possibly luminance, hue, alpha, or depth) and these values are related to those nearby. That is, an object will take up a two or three dimensional space in the image, and the values within this region should be associated in the features we use to classify our image.

There are established ways to do this, as well as several that show it is possible to ignore the structure of the data entirely and still make progress. In this work, feature descriptors are not used in an attempt to find better representations of our relevant features automatically. The methods for feature extraction are discussed in the following section.

2.3.1 Features for Data Representation

Often classifiers do not perform adequate processing of the data which is presented. They assume that the presented features are no more or less than what is necessary

to produce a model of the data. The data presented with a problem is often noisy, not always relevant, often correlated, and otherwise unfit to be directly presented to classifiers. As a result, most classifiers are preceded by some feature creation or selection which transform the data into what is meant to be a simpler, or more accurate representation of the problem. Within vision and more general machine learning, there are many approaches to this process.

Codebooks

In the simplest case, each value in the data is treated independently, and presented to the classifier directly. This means flattening the image and presenting each value from each channel in one large vector. This is not used, but suppose smaller regions are flattened. The smaller vectors now present information relevant to their location and the entire image. One method based on this idea relies on codebooks to represent these flattened regions. Codebooks are sets of features where the vector representing data is a set of true and false values indicating which "codes" occurred in the data, while the codes are either selected, learned, or both. For example, these codes could be standard features mentioned below, or those extracted using dimensionality reduction.

As discussed later, the first portion of this thesis makes use of a codebook of clustered types of image patches which are found based on unlabeled training data in an unsupervised manner. This was a useful compromise which allowed the use of 2D features and a classifier meant for flat data.

Receptive Regions

Most classifiers are designed to take one dimensional vectors of features or data, and are not easily adapted to handling two or three dimensional image data. This is why feature extraction is often separately used to transform the data, not only into a more general or shorter representation, but also to limit the dimensionality so it can be processed by standard predictors.

Traditional methods in computer vision rely on feature descriptors to represent

the image. These include HOG, SIFT, and SURF descriptors. SIFT is usually used for alignment or image stitching, but has also been used for object recognition [33, 26]. HOG feature descriptors are generally used in a sliding window for object detection and recognition [9]. These features have been used effectively in pedestrian detection, object recognition, and scene classification. [40] presents an incremental approach using these features to learn the set of features necessary for visual classification; whenever the system has difficulty classifying an image, it seeks new features that are capable of helping differentiate between the multiple classes. Several classes of features relevant to images are discussed in greater detail below. Our system does not make use of them due to choices discussed in later sections.

Convolutional Neural Networks get around the issue of representing local information by acting as both feature extractors and classifiers. Standard networks also extract features, but for image processing, convolutional networks are, as mentioned in the previous sections, meant for 2 and 3 dimensional data, having receptive regions which scan the image and respond to learned patterns. A well structured neural network learns a representation of images, usually displaying edge, texture, and color detection in the first layer, and more complicated filters in deeper layers. This network style has shown great success on image classification problems, and has the added advantage of being implemented on the GPU, which helps with system speed.

There have been many papers released in the past decade on the use of neural networks as both feature extractors and classifiers. [30] uses support vector machines (SVMs) and convolutional nets together to characterize objects in variable conditions of illumination and from multiple viewpoints. [52] demonstrates how a hierarchical neural network evolves structures invariant to features such as color and orientation, consistent with physiological findings.

2.3.2 Object Detection

Object detection in vision is the process of finding an object within a frame. If we remove the assumption that an object is the main content of an image, then we need

to account for the presence of other objects or more generally background which we want the system to ignore. As was discussed in Section 1.1.1, the attributes of an object may apply to regions of the object rather than the entire surface. The result is that detection of objects and their attributes is important to identifying where an object may be and if attributes contribute to the same object or several, to a portion of an object or the entire thing. This work implements a simplistic detection system which works reasonably well, and a similar system described in a recent paper uses a network to predict the bounding box of objects [45].

2.4 Speed: Cuda and OpenCL

Robotics draws on the above fields as well as others, but the focus of this work is on learning of visual features. To include a prediction system as part of a robotic setup, it is necessary to be careful with time and space taken by the system. It is the case in previous work that much of the processing takes place on a separate machine connected to the robot through a network. It is assumed here that the use of this setup will continue, and place few limits on the space that the system takes up. However, the speed of predictions should be as fast as possible.

To improve speed of classification, many current systems use parallel implementations. If there are separate classifiers, each one may run at the same time. In the case of a neural network, there are GPU implementations. There are several libraries available that make use of the GPU and other optimizations, and of course CUDA or OpenCL may be used to write GPU kernels directly.

Theano provides a python based system for creating highly optimized code which is compiled into C++ for the GPU [3]. It is the most general system available for building neural networks. Several research labs have produced narrower implementations of networks on the GPU, including Cuda-Convnet [28], Decaf and Caffe [13, 24], and Overfeat [45], each with their own framework for neural networks. There have been no comparisons of these implementations when used on identical networks.

Krizhevsky's implementation, convnet, is the most flexible, but relies on python for the initial processing, which can be slow. Decaf and Caffe are based on convnet, but add some additional types of layers, with Decaf written in python and C++, and Caffe as a pure C++ implementation. Overfeat provides only the pre-trained version of a convolutional network trained on the ImageNet dataset, although additional features may be forthcoming.

The issue with GPU systems then becomes the time required to transfer data to the GPU, as well as the specific optimizations that depend on the design of the card, such as how much memory it has, how many processors and the appropriate way to break up networks, data, and processes among these resources. When designing neural networks to be run on the GPU, memory can quickly become an issue. Training is often better when performed on larger batches of images, but GPUs have limited space for storing both the network and the data. The larger the data, the larger the network needed to represent it, and the more memory will be needed for both. Trade offs between size and speed continue, and several systems scale down data and batch size which also reduces the size of the required network. In spite of these complications, results are encouraging.

Chapter 3

Isomap and Support Vector Machines

The initial prototype was designed to show the feasibility of predicting attributes rather than objects, especially when provided with mainly positive examples. Before using either a specialized feature extraction technique or classifiers that were meant for multiple labels, a standard feature extraction technique and set of classifiers were used. This shows the behavior of a standard system on both generated and collected images, and provides a comparison for further methods. The following prototype was developed in Python. This chapter is structured as follows: first the issues this project addresses are discussed, as well as the issues which emerged during development. Then the implementation and results are shown, and followed by analysis. Finally extensions and alterations are described.

3.1 Approach

3.1.1 Single Class Classifier

Using a single classifier for a single label, we can learn to predict yes or no. This is true of both an object and an attribute: so long as the label is something we can see,

it is feasible to learn it based on images and labels for the images. Thus, if we have many *apple* images, we could design a system learn to predict if an image contains an *apple* or not. If our label is an attribute, such as *red*, we may also learn to predict this. In both cases there may be issues with breadth of samples (colors, shapes, sizes of apples; shades of red).

Let us now assume that we want to predict more than a single class at a time. We may see *apples* or *bananas*; we may see *red* or *blue*. There is also a chance that we will see neither. This can also be handled either by having a separate classifier for each label, or by using a multiclass classifier. But now let us assume that we have the objects *mug* and *cup*, and the attributes *flat* and *smooth*. In both cases we may now have overlapping classes. This means we may decide that a *mug* is a type of *cup*, or that if a thing is *flat* it is also *smooth*. If using separate classifiers this is still possible. Each classifier worries about its own label, and any gain we might have by considering relations among the labels is ignored. That is, if something is flat, it may also always be smooth, but this correlation is not used. Multilabel classifiers can work on this problem, but are often much less accurate then single class classifiers or multiclass classifiers which assume a single output is likely.

If the best classifiers are those which have many internal parameters which adapt to the data and take time to train, such as neural networks, we would rather not have a separate classifier per label. However, this is a good place to begin, in order to see the trade-offs between multiclass and single class classifiers.

3.1.2 Unknown and Contradictory labels

Another issue in classification problems is the acceptability of the ground truth labels. In the narrow case, a set of images may be labeled from a small set of labels; that is, a list of possible labels is given, and the labeling is done for each of the samples completely. In this case, completely means that for every possible label, each samples has a true or false value; it is or is not each label, and this value is never unknown. As soon as the labels are changed from objects to attributes we increase the possibilities.

For a given image the labels *red* and *orange* may both be applied. The object may have a *red* part, and an *orange* part, or it may be one solid color which may be *red* or *orange*. Attributes may apply to the whole object or just a portion. An object may be *shiny*, but the feature that makes the object appear *shiny* may be a tiny bright spot on the surface to one side.

This issue may also occur in object labeling, (for example *boat*, *ship*, and *vessel*,) but isn't seen as often, partly because these labels may be collapsed into a single label to eliminate the problem, and partly because object labels generally apply to a whole thing rather then being broken down into parts. There have, however, been some systems which identify objects based on sub-objects, such as using *wheel*, *window*, and *door* to identify a car. These systems rely on detecting sub-objects first, which is a technique visited in Section 5.

If the data is collected in a less structured way, for example when using online sources such as Flickr, then the labels could be anything: sometimes related to what is seen in the image, sometimes related to the mood or thoughts of the person who posted it. These are crowdsourced datasets. In these situations, the assumption is made that the majority of the labels do refer to the contents of the image. The incorrect, or minority labels will be either manually filtered out with rules provided by programmers or will not have a great influence on the model during learning since a slight movement away from the optimum may occur, but will be corrected when better examples are seen. This still means that the values of many labels are unknown for a given image. A picture may just happen to be labeled as a *coat* instead of a *jacket*, and it cannot be assumed that it is or is not a *jacket*. How these unknowns are handled has a huge influence on the quality of the predictions: while dropping unknown samples is the safe course, every dropped example is less training data for the system. A better goal is a system that is robust enough to learn what it can from known labels, without being heavily influenced by unknowns or incorrect labels.

In the first prototype developed, the unknowns are left out, and the classifiers are provided with only positive examples to discover whether it is better to assume that

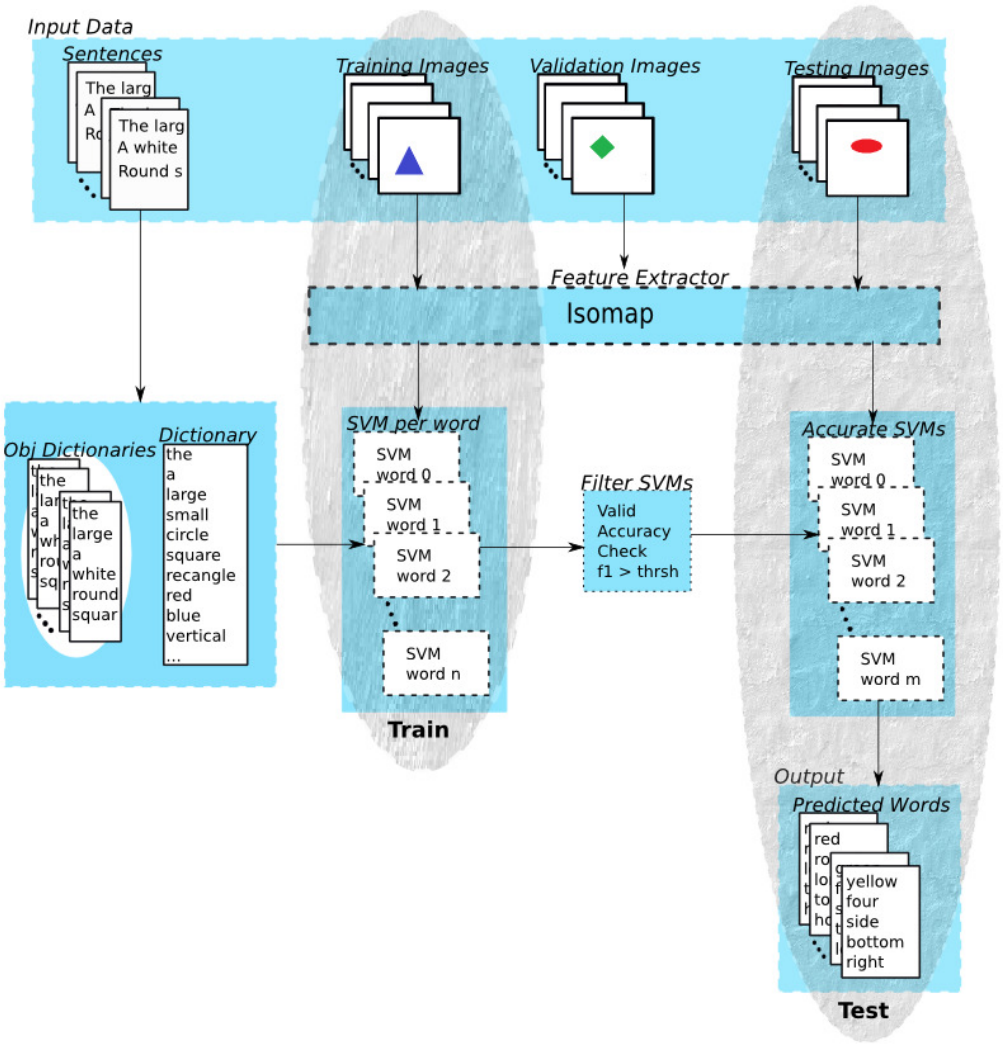


Figure 3-1: Training and Testing Process

the unlabeled data is negative, or not. In the next prototype, these examples are counted as negatives. A method to label these unknowns is briefly explored in this project, but was too inaccurate at this point to be used.

3.2 Implementation

For this implementation, Isomap [48] and Codebooks [7] were each used for feature extraction and compared. Features provided by both methods were then fed into a One-Class Support Vector Machine, one per class to perform classification. This com-

bination showed that certain attributes were well-learned, but also reveals a number of difficulties related to the complexity of the data.

The systems were trained on two sets of data, (generated and real,) each split into multiple training and validation groups, and performance is evaluated on the unseen validation data using precision, recall and F1 scores.

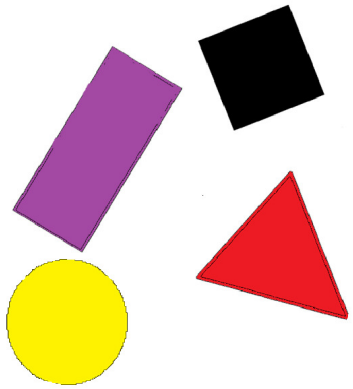


Figure 3-2: Example synthetic images.

The format of the datasets used for training was based on the eventual goal of learning from video and audio. Audio is not used, since this is a complex problem on its own, but there are resources available to transform audio of speech into text, or to use the split audio words as labels directly. The generated images are just shapes in primary colors in a variety of locations and positions on plain white backgrounds, which were all of 500 by 500 pixel resolution. This set is also used in greyscale to investigate the quality of non-color predictions when unnecessary information is removed. The other set consists of RGB images captured with a Kinect camera, some of which were collected within the lab where the system will be used. The rest are from the RGB-D dataset [29]. The Kinect has a resolution of 640 by 480. None of these had attribute labels originally, although there were object labels in the case of the RGBD dataset. A sample of the generated data can be seen in 3-2, while the Kinect data can be seen in Figure 3-3.

For this experiment the background was masked. The images were annotated in complete, positive, English sentences. These sentences are provided by humans and vary in complexity and length, although constraints on positive statements do limit their content. Generally these sentences include words describing the shape, position and colors of an object. The positive constraint was added to avoid the complexities of parsing sentences into positive and negative attributes, which is, as with audio, a separate and not fully solved problem. There are systems which specialize in this, and

the system could be extended with their use. For example, a sentence may be "The shape is red" but not "The shape isn't square." There are separate files containing negative labels, but a side question studied in this work is the influence of removing negative examples, or having limited negative examples.

As discussed earlier, labels can be unknown or contradictory. This is even worse when negative labels are not known. While it would be simple to assume that an object labeled "red" is not "blue," this assumption cannot be made. Even when providing strict rules for labeling, there will be mistakes made by human users, and disagreements between them. Unfortunately this is a part of the problem which is being solved: the ability to adapt to these variations and to be correct. While giving good, clean data to the system will improve performance, ensuring these features of the data is almost impossible without eliminating samples which may still aid in learning. It is assumed that these labels are not subjective, and any disagreements will occur so rarely as to have a minimal influence on learning.

Figure 3-3: Example masked Kinect images from the RGB-D database.

A data collection system was implemented that collected the data using the Point Cloud Library [5]. It uses a depth filter followed by a RANSAC-type planar segmenter to remove the table beneath the object. The removed values are set to white for color and a depth of zero, which is used by the Kinect to indicate unknowns or out of range values. In later prototypes the Kinect data is not masked.

3.2.1 Feature Extraction

This step uses Isomap and an image patch codebook to reduce the training data feature space from several hundred thousand raw pixel values to around a hundred features. The version of Isomap used is provided in Scikit Learn, a Numpy and Scipy based library for machine learning [39]. The codebook method is described in 2.2.1.

This technique uses K-Means clustering on the patches taken from throughout the training images to create template patches which become the codes for the feature vector. These template patches result in simple edge, pattern, and color templates. For complicated learning problems k has to be very large, this is supported by the results presented here. This method is generally faster than Isomap, but reduces dimensionality less, as discussed in section 3.3. Isomap, in this case was a better choice for the data, although it requires more memory and time, since the standard implementation's memory usage grows with the number of examples, and requires comparing every pair. Codebooks may be useful on larger datasets, where speed and space are more important than final dimensionality.

3.2.2 Learning Word-Feature Relations

The feature vectors from either method are used as input to the One Class Support Vector Machine for each word. The labels are created by stripping out punctuation from the sentences and transforming to lower case. Had more complicated parsing been used, then these features would add useful information, such as the difference between "Titanic" the proper noun and "titanic" the adjective, or "its" versus "it's." It is assumed that the vocabulary for this dataset is narrow enough that many of these problems are avoided, and that many of these will be removed as un-learnable labels or *stop words*. *Stop words* are those which contribute little or no information in sentences, such as *a, the, is,* and *and*. These are removed both by using a short list of such words, and by removing any words which occur in less then 5% of samples, or more than 95%. These labels are unlikely to be learned either because they apply to everything, such as *thing*, or to almost nothing, as in the case of *vertically*.

This approach of using each word as a potential independent label is a "bag of words" method. Many words change meaning in context, which suggests that pairs (bigrams) or triples (trigrams) of words would be more useful to find correlation of features. This would, again raise the complexity of learning and the size of the problem significantly while providing additional useful information. For now these

are not used.

For each word, such as *red*, every image associated with this label is assumed to contain *red*. These images were split into training and validation sets, with the majority of the data in training. The validation set is used to check the performance of the trained classifiers on data that was not used for training, to verify generalization. Multiple such divisions are created and shuffled to be in a random order. This is done because some training sets result in better or worse classifiers, and even the order in which examples are presented can influence performance. Thus, averaging across several possible splits provides more accurate measure of general performance. Each of the training images is used to train a *red* classifier. Since the labels were collected from "positive" statements, negatives were not assumed or provided. This is why a One Class SVM is used. As mentioned previously, these classifiers do not perform as well as the standard classifiers do, especially when provided with little training data. Providing negative labels is not as easy as taking the complement of the positive labels, or even as simple as asking the audience to label what an object isn't. This will be discussed further in Section 3.3.

The One-Class SVM allows for the parameters of kernel choice, training error bound, and degree of the kernel. These parameters were set according to a small grid search. The best across classifiers was of degree 3, and error of 0.3, with radial basis functions, although it would be better to set these on a per class basis.

3.3 Results

The system was tested on 128 generated images of *oval, circles, rectangles, triangles* and *squares* in *red, blue, green, yellow, purple, orange, brown,* and *black*. There were 134 real objects in the Kinect data, each of which had a dozen images from different angles. This data was, as mentioned before, split into positive training cases, and several positive and unknown validation cases. The validation sets contained objects and features that were not seen before, in order to show not just generalization from

Table 3.1: List of what techniques were used with what parameters varied.

Technique	Application	Parameters
Background Removal	simplifies data; preprocessing	
Scaling	Normalization preprocessing	axes to scale across
Isomap	Dimensionality Reduction; Feature Extraction	number of dimensions; number of neighbors
Codebooks	Dimensionality Reduction; Feature Extraction	size of patch; size of overlap; number of patches
One Class SVM	Classification; Outlier Detection;	kernel type; degree of kernel; bound on error

one image to another of the same object, but also generalization of a feature from one object to another.

For validation purposes, negative labels were used. These labels were provided for cases where it was clear that the label was negative, but could not be provided for all words, as many were ambiguous, such as *several* which may or may not describe a learnable feature as in the phrases *several shades*, *several sides*, *several corners*, and *several buttons*. This again, reveals the ambiguous and contradictory nature of the labels. To provide positive labels, sentences were collected. Each image was presented to a human user, and they were asked to describe the object in short, positive sentences. These sentences were required to describe what the object was, focusing on color, shape, texture, and position, without referencing objects which are not in the image, and without listing traits the object does not have. Thus "The red cube is on the left" is permitted, while "the cube is not blue like the ball" is not allowed. To provide negative labels, the list of positives was provided, and used to create the negatives for each image. This labeling proved difficult to make for labels such as *sharp*, which could refer to the resolution, the edges, the brightness, the flavor or any number of other qualities which may or may not be learned. As such, the final list of negatives included the words which were definite and inarguable. For example,

a white hat without any other color would be labeled as negative for every other color label, while rarer labels such as *side* or *edge* could not be added to either list since it could be used to describe the hat in certain situations, but not in general. The positive labels, tended to be obvious, and easy to agree on. It is possible to improve upon the negative labels provided in this dataset, but in the long-term the problem of conflicting labels will need to be dealt with as part of the learning procedure, rather than by forcing structure on the data. To make a truly robust system that is intended to interact with people in a dynamic setting, it should be able to handle the disagreements between people, and shades of meaning.

As a result of this ambiguity, partially labeled data was used to update the ground truth. That is, experiments were run where the system was allowed to update the labels of unknown data with labels if the known accuracy of the predictor was reasonably high. Unfortunately this did not result in a marked increase in performance, since the examples which were added to training in this way were those most similar to those already seen. If the threshold for accuracy was varied more, it may be that generalization would improve, but this was not studied further.

The dictionary resulting after filtering the sentences of "stop" words was 91 words for the generated data, and 195 words for the Kinect data. Of these words, 60% were nouns and adjectives, which were expected to be learned, while the rest may or may not result in useful classifiers. These other words were left in as a comparison for the others, and to identify if the system could automatically remove words which cannot be learned based on performance. The system was implemented in a memory limited version which avoid loading data until necessary, and clears it as soon as it is no longer needed, and a memory unlimited trainer which loads everything needed at once. Training takes roughly forty minutes without memory limits, and an hour and a half with memory limits.

Table 3.2: Generated Data Results: 128 total samples, permitted error of 0.2, degree 3, radial basis kernel.

Word	Samples	Precision	Recall	F1
yellow	22	1.0	0.45	0.62
four	26	1.0	0.5	0.66
black	37	1.0	0.62	0.76
circle	37	1.0	0.54	0.70
triangle	38	1.0	0.52	0.68
square	21	1.0	0.33	0.5
blue	18	1.0	0.5	0.66
red	17	1.0	0.64	0.78

Table 3.3: Generated Data Results with many unknowns: 128 total samples, permitted error of 0.2, degree 3, radial basis kernel.

Word	Samples	Precision	Recall	F1
three	24	1.0	0.66	0.8
top	19	1.0	0.63	0.77
oval	27	1.0	0.55	0.71
right	19	1.0	0.47	0.64
corners	46	0.96	0.56	0.71
round	34	1.0	0.55	0.71
upper	16	1.0	0.5	0.66
shape	77	1.0	0.74	0.85

3.3.1 Feature Extraction Results

After several values of k were tried, Isomap was given a limit of 100 dimensions to reduce to based both on performance and memory limits. The generated data was reduced from 750,000 features of raw color data to 75 features. This is essential, since the SVM performs well on high dimension data, but only when the number of samples exceeds the number of dimensions. Raising the number of features beyond this may better represent the Kinect data, but may also reduce performance of the SVM, unless further examples are collected as well.

3.3.2 Word-Feature Learning Results

The results for the synthetic data reveal that words referring to *color*, *location* and *shape* were well-learned in this simple system. Many other words were removed out-

right due to poor representation, and even those which remained had very few samples. Partial results are presented in Table 3.2 and Table 3.3, where the first table contains results for a subset of the data, while the second contains further examples which performed well in spite of having few positive samples. For example, many images were labeled as *corners* and these were predicted well for squares, rectangles and triangles, as well as reported negative on circles and ovals, but due to the number of unlabeled images, these results may not be generalizing as well as they seem to be, and so should be verified by further testing. Unlabeled data predictions are not used when calculating the precision, recall, and F1 scores, although these provided interesting minimum and maximum results when assumed to be either correct or incorrect.

Varying the allowed error improved performance of some labels and worsened performance of others as shown in Table 3.4, which reveals that further gains may be made by actively learning the parameters of the model. The error was increased to 0.3 from 0.2, which improved results for words such as *four* and *square* which are likely correlated, but resulted in worse performance on others such as *red*.

The Kinect data did not perform as well, most likely because it covers a broader, more complex space. This data contained a wider range of colors, had variations in coloring across objects and more complicated objects, as well as having many more labels, which were less well represented. Many of the labels only occurred in relation to a single object, which while providing fair prediction on that item, will not be generalized to others, and may be correlated with any of that single object's features, and so is associated with the object as a whole. For example, *shiny* might only be associated with a flashlight, and so will never be predicted for another item, and will be associated with the entire flashlight, rather than any color or lighting specific aspect. Colors continued to perform well, but other features are too sparsely represented to be trusted. Shape was no longer quite as prominent in the vocabulary, since the shapes were far more complex combinations of simpler shapes. Shape words were more general, as in *round* or *flat*. Location was also less present, since almost

Table 3.4: Generated Data Results: 128 total samples, permitted error of 0.2, degree 3, radial basis kernel.

Word	Samples	Precision	Recall	F1
yellow	22	1.0	0.36	0.53
four	26	1.0	0.57	0.73
black	37	1.0	0.56	0.72
circle	37	1.0	0.62	0.76
triangle	38	1.0	0.52	0.68
square	21	1.0	0.66	0.8
blue	18	1.0	0.33	0.5
red	17	1.0	0.41	0.58

Table 3.5: Real Data Results: 128 total samples, permitted error of 0.3, degree 3, radial basis kernel.

Word	Samples	Precision	Recall	F1
yellow	22	1.0	0.64	0.78
black	34	1.0	0.64	0.78
blue	35	1.0	0.74	0.85
purple	10	1.0	0.6	0.75
red	60	0.83	0.73	0.77
white	36	1.0	0.58	0.73

all of the objects appeared in the center in the RGB-D dataset, so *left*, *right*, *top*, and *bottom* did not appear.

A much larger dataset is necessary for real data, and it may be valuable to collect images for certain vocabularies in order to narrow the field of possibilities, since adding any data will also broaden the vocabulary. The color results for real data are shown in Table 3.5.

These results were published in ICINCO 2013 [50].

3.4 Discussion and Further Work

This project revealed the difficulties surrounding the labeling task, including ambiguous labels, unknowns, and negatives. It also revealed the complexities introduced

by even simple real-world scenes, and the scale of the problem. Given hundreds of thousands of training images, all at least partially labeled, these problems may disappear, in the sense that contradicting labels will be overwhelmed by a majority which agree, and lack of negatives is less of a problem when a class is well represented by the training data. However, collecting such large amounts of data accurately is an issue. Even using methods like bootstrapping or using images labeled on the Internet can introduce an undesirable amount of error, in the first case because your initial dataset must still have fair coverage of the data space, and in the second because many more of these labels are likely to be incorrect, and the images will vary in setting, resolution, and quality.

Small gains may be made with a broader parameter search for Isomap and the SVMs, and with parameters specific to each label in the case of SVMs. However, this will not solve the existing problems with these methods.

It is possible that Isomap poorly represents the features, or only represents certain features well. It has been shown to take up more time and space than other methods, and is not robust to noise. This analysis leads to the goal of an improved feature extraction procedure which adapts to the problem with training as well, and hopefully which is more memory efficient, which will improve features, as well as make it possible to work with more data.

The problem seen for most of the data is a high false negative rate. This means the bounds created by the one class SVM are too tight, cutting out examples that are too close to the boundary, usually because they were not well represented in training, or because the degree and error of the classifiers were too small. Generally negatives are well labeled since they are far from the space the positives lie in. It would be beneficial to use even a few negative examples, and a one class classifier not be used in order to learn a more reasonable boundary that relies on more data.

More data would certainly aid in learning, and more samples are added for the experiments in the following chapter. It may also be advisable to limit the vocabulary further and verify the completeness of labeling with respect to these select labels,

rather then allowing for variation and unknowns.

Memory use quickly became an issue, and in order to add more data, it is necessary to investigate online techniques, and other ways of loading data in smaller sets during training and validation.

3.5 Summary

This chapter presented an initial prototype for attribute prediction in images which relies on Isomap and a separate One Class SVM for each label. The data for the prototype was simplified by background removal, and the sentences contained only positive statements. It was shown that this system is capable of recognizing colors, shapes and location in generated data, and colors and shapes in real data taken from a Kinect camera. While these result are fair, attaining an F1 score from .7 to .9 for the best represented labels, improved results are needed, especially on the real data which performed poorly compared to the generated data. This protoype possesses the capability to predict certain attributes well, and indicates the potential of this system if results can be improved by use of additional data, and improved feature extraction and classification. The results from this chapter were in the proceedings of ICINCO 2013 [50].

Chapter 4

Model Parameters and Neural Networks

After results were collected from the Isomap and SVM system, it was determined that more data should be collected, and that improved feature extraction be pursued. It was also a goal to find methods that would allow a portion of our data to be loaded and trained one at a time, to avoid filling up memory. In order to handle limited or incompletely labeled data, it was also useful to find techniques that were partially unsupervised to make best use of all samples. Since there are constraints on both time and space, limiting the number of classifiers is a goal, which is why multiclass and multilabel systems are explored as well as single class, although in this prototype they are not used as such. More complete parameter searches were also made for each of the techniques. Negative examples were also added to training of the classifiers which permit them.

4.1 Approach

This phase broadens the Isomap and SVM approach and adds additional feature extractors and classifiers to the system. A framework that provided three formats for the data, a choice of feature extractors and classifiers, and a selection of parameters for

each of these was developed. Data may be scaled across different axis and values, for example across pixels, across a color, or within a given location. Additional samples may be bootstrapped either by small transforms, blurring or reflection, and may be presented to the extractors and classifiers as flat vectors, patches, or as the raw images. The feature extractors that were used include Isomap, Convolutional Neural Networks, and Denoising Autoencoders. Classifiers include One Class SVMs, and Nu SVC, which are variations on Support Vector Machines, as well as Convolutional Neural Networks and Denoising Autoencoders, which can also be used for feature extraction.

In the initial test, Isomap and one class SVMs were used as in the previous Chapter. A separate classifier is trained for each label, and a grid search of parameters is performed per classifier to find the best combination of settings for each label. This was done to show the improvements to be had by carefully setting the parameters of the system.

In the second test, Isomap was used, this time in combination with a Nu SVC. Nu SVC is a version of nonlinear support vector machines which uses the Nu parameter to put a limit on the number of support vectors. Positive and negative examples were used for training. This is done to see the value of using even a very limited number of negative examples to train a multiclass SVM rather than a one class SVM. Parameters were searched and set per classifier, with a classifier per label.

In the final tests, autoencoding and convolutional neural networks are used. Autoencoders are a type of neural network which provide an unsupervised way to use data which is partially labeled or unlabeled, which allows for use of data which before was simply dropped from training [16]. They are also able to handle more variation in the data. Networks have batch training, which makes them easier to train with limited memory, although the networks themselves may take up more space than other methods. Convolutional Autoencoders are also intended to work on images, and should perform better than flat networks. Both Convolutional Autoencoders and standard Convolutional Neural Networks were implemented and used as joint feature

extractors and classifiers.

4.2 Implementation

A framework was developed which allowed the selection of different feature extractors and classifiers, as well as different data pre-processing. Isomap, CodeBooks, Convolutional Neural Networks, and Autoencoding Convolutional Neural Networks were made available as feature extractors, as well as a number of others which were not used for these experiments. One class SVMs and the standard Support Vector Classifier with Nu parameter, as well as the two types of neural networks were available as classifiers. Combinations of these were tested, with narrow grid searches of their parameters on both the synthetic and Kinect data. Theano was used to implement the neural networks with the guidance of the Deep Learning Tutorials provided at deeplearning.net [3]. Theano provides a system for defining mathematical expressions, including those that define neural networks, which are then compiled into optimized code for either the CPU or GPU. It provides functions useful for deep learning, such as gradient calculation for parameters within an arbitrary function such as the function describing the neural network. Codebooks were implemented in python according to [7]. The other classifiers, feature extractors and scaling were provided by Scikit Learn modules [39].

4.3 Results

The one class SVM gained some in performance with a more complete parameter search. Shown below in Table 4.2 are the results for the word “black” with Isomap set to use the two nearest neighbors and reduce to 100 dimensions. As can be seen, the F1 measure and accuracy have both increased when the degree was decreased from the previous best. Similar improvements were found for other labels. Further examples follow. For further results see Appendix C.

Table 4.1: List of what techniques were used with what parameters varied.

Technique	Application	Parameters
Masking	simplifies data; removes background; preprocessing	
Scaling	Normalization preprocessing	axes to scale across
Isomap	Dimension Reduction; Feature Extraction	number of dimensions; number of neighbors
Codebooks	Dimension Reduction; Feature Extraction	size of patch; size of overlap; number of patches
One Class SVM	Classification; Outlier Detection;	kernel type; degree of kernel; bound on error
NuSVC	Classification;	kernel type; degree of kernel; bound on error
CNN	Classification; Feature Extraction	number of layers; number of filters; size of filters; size of overlap; momentum; learning rate; batch size; etc
Autoencoder	Classification; Feature Extraction	see CNN;

Table 4.3 and 4.4 show the initial results of using Isomap with a non-linear SVC per class. Nu is the NuSVC permitted training error. Table 4.3 shows results using radial basis functions and Table 4.4 polynomial kernels. The parameters of each SVM are varied on a per class basis, and this shows that, as determined in the earlier project, performance can be improved by changing these carefully per class. Unbalanced classes and lack of training data continue to be an issue. Labels which had very few examples or very many examples have been removed because the learned behavior was to predict all zero or all one. This effect can be handled by adding more examples, removing examples, or by weighting the examples with prior probabilities, but in this case it is used to eliminate labels which cannot be learned from the dataset.

These poorly learned labels are found by the behavior of the accuracy and F1 score in both training and validation. Over-fitting is shown as very high accuracy, and a very small F1 score, or good performance on training with poor performance on testing. Overall the results are better for more classes with the addition of the negative examples, extra hyper-parameter searches and some additional data, but several labels are better modeled with a One Class SVM than with a binary classifier which uses negative examples.

The lack of variation in the data, as well as the correspondence of certain labels can be seen in these values. The fact that the one degree SVM performs so well for most cases may mean that these labels are simple to identify but are more likely to be the result of a very narrow representation in the data. For example, *box* and *cereal* only occurred on *cereal boxes* and so their scores are exactly the same due to the narrow set of objects in the data. Had there been additional objects, such as *Kleenex box*, or *cereal bowl* the results would vary more, and it is likely that they would be worse. There are also words like *side* which was mostly used with square or box shapes, but also occasionally used to describe objects which varied in color, shape or pattern as in "The cap has a picture on one side" or which describe position such as "The box is lying on its side." This classifier successfully marks most boxes as *side* but does not generally identify the rarer uses. Select examples of a complete

grid search are shown in Appendix C.

The initial use of convolutional neural networks showed a tendency to overfit as well as respond poorly to unbalanced data. The network would initially do fairly well, and then quickly move to predicting the most common class only. This is at least partly due to the data which is unbalanced, and contains mainly negative and unknown examples. This may be one of the situations where a multiclass classifier would do better, since there is better representation across all classes then for each class. To clarify, if each class is represented by a fifth of the samples, but the classifier is only predicting a single class, then it appears the positive case occurs a fifth of the time, which can skew prediction. However, if the classifier is aware of other classes it is usually designed to better use this information. It is also likely that additional extensions to the network are needed. Straight implementation of a neural network without special additions produces fair results on the most simple data, such as MNIST, but when applied to more complicated data, additional regularization is needed. This means adding preprocessing, or the methods discussed in Section 2.2.6.

On a side note, there were certain combinations of parameters that caused feature extractors or classifiers to produce NaN or Inf values. This is not likely to be either an error in the data or the library, both of which are tested, but rather a set of parameters which cannot be used to represent the data, and cause the feature extraction to fail to represent the data well. This was not further investigated, but should be kept in mind during further experiments.

4.4 Discussion and Further Work

This extension to the initial project showed improved performance, and the variety of performance that could be attained with proper hyper-parameter search. Results improved across classes with Isomap and standard SVMs. However, space became a serious issue, and data had to be loaded in batches. It was also time consuming to train separate classifiers for each word, and to find the best classifiers in a larger

parameter space on a per word basis. Proper search of parameters involved, for example, setting two parameters for isomap, one can be set two ways, the other four ways, three for support vector machines, which can be set two, four and five ways respectively. Each combination is trained for each word, with seventy-five words for generated data, and perhaps ten different cross-validation groups. This means $2 * 4$ possible settings for Isomap and $4*2*5$ possible settings for the SVMs, or 320 possible combinations that should be tried for each word, ten times, for a total of $240,000$ separate training iterations. One of the largest contributions to this number is the per word training, which leads to the pursuit of multiclass and multilabel classifiers in the next chapters.

4.5 Summary

This chapter described extensions to the previous system, including a larger parameter search and the addition of several other feature extraction and classification methods. These included convolutional neural networks and standard multiclass support vector machines. Results from the previous chapter were improved by altering the parameters of both Isomap and the One Class SVM. Several labels showed improved performance when classified by the standard SVM, while others were better modeled by the One Class SVM.

The neural networks implemented here failed to model the data, though this is likely due to the very limited set of parameters that were tested and the lack of regularization. Such extensions are likely to greatly improve the performance of the networks.

Table 4.2: Kinect Data Results for One Class SVM Black, with Isomap set to 2 neighbors and 100 dimensions.

Kernel	Error	Degree	Accuracy	Precision	Recall	Specificity	F1
RBF	0.2	2	0.95	1.0	0.80	1.0	0.89
RBF	0.2	3	0.95	1.0	0.80	1.0	0.89
RBF	0.2	4	0.95	1.0	0.80	1.0	0.89
RBF	0.3	2	0.79	1.0	0.20	1.0	0.33
RBF	0.3	3	0.79	1.0	0.20	1.0	0.33
RBF	0.3	4	0.79	1.0	0.20	1.0	0.33
RBF	0.4	2	0.89	1.0	0.60	1.0	0.75
RBF	0.4	3	0.89	1.0	0.60	1.0	0.75
RBF	0.4	4	0.89	1.0	0.60	1.0	0.75
RBF	0.5	2	0.73	0.0	0.0	1.0	0.0
RBF	0.5	3	0.73	0.0	0.0	1.0	0.0
RBF	0.5	4	0.73	0.0	0.0	1.0	0.0
POLY	0.1	1	0.83	0.65	0.74	0.86	0.69
POLY	0.1	2	0.52	0.34	0.86	0.39	0.48
POLY	0.1	3	0.62	0.39	0.74	0.58	0.51

Table 4.3: Kinect Data Results NuSVC RBF Kernel

Word	Nu	Precision	Recall	F1
black	0.2/0.4	0.54	0.68	0.60
blue	0.1	0.57	0.71	0.63
bottom	0.4	1.0	0.77	0.87
box	0.3/0.5	1.0	0.7	0.82
cereal	0.3/0.5	1.0	0.7	0.82
green	0.3	0.9	0.82	0.86
grey	0.2	1.0	0.64	0.78
light	0.1	1.0	0.58	0.74
orange	0.2/0.3	0.59	0.67	0.63
red	0.3	0.73	0.78	0.75
round	0.4	1.0	0.64	0.78
side	0.1	1.0	0.65	0.79
silver	0.1/0.2/0.3	1.0	0.72	0.84
stripe	0.2	1.0	0.62	0.76
top	0.4	1.0	0.68	0.81
white	0.1	0.56	0.69	0.62
writing	0.2/0.4	1.0	0.67	0.8
yellow	0.1/0.5	0.39	0.64	0.48

Table 4.4: Kinect Data Results NuSVC Polynomial Kernel

Word	Degree	Nu	Precision	Recall	F1
black	1	0.3	0.95	0.68	0.79
blue	1/1	0.2/0.3	1.0	0.61	0.76
bottom	1/1/1	0.2/0.4/0.5	1.0	0.62	0.76
box	1/1	0.2/0.3	1.0	0.65	0.79
cereal	1/1	0.2/0.3	1.0	0.65	0.79
green	3	0.4	1.0	0.55	0.71
grey	1/1	0.2/0.3	1.0	0.6	0.75
light	1/1	0.2/0.3	1.0	0.63	0.77
orange	2	0.5	1.0	0.67	0.8
red	1	0.2	0.84	0.65	0.74
round	4	0.1	1.0	0.55	0.71
side	1	0.3	1.0	0.6	0.75
silver	1	0.3	1.0	0.61	0.76
stripe	3/1	0.1/0.5	1.0	0.62	0.76
top	1	0.2	1.0	0.68	0.81
white	1	0.2	0.88	0.72	0.79
writing	2	0.3	1.0	0.6	0.75
yellow	1	0.2	1.0	0.52	0.68

Chapter 5

Convolutional Networks with Video

The goal of the system described in this chapter is to provide classification of objects in video, rather than investigate attributes. The system also shows the viability and difficulties involved in streaming entirely unlabeled data from a camera into a neural network. This is an initial test of multiclass classifiers, before multilabel classifiers are used. In this setup, it was necessary to either assume that classes do not overlap, or to cut down the labels to only include non-overlapping classes. In this case object prediction was selected, since attribute classes overlap heavily. Other approaches would include using only the most apparent attribute of a type, for example the most present color. This was not considered, since it is a subjective decision. During both training and testing, the complete image was split into patches at several scales to locate known objects in the image. For this experiment a NONE class was added to account for patches where no object was visible.

5.1 Approach

A number of papers drew attention to Neural Networks, especially Convolutional Neural Networks, as noted in Section 2.2.3. Neural Networks may be used for fea-

ture extraction, or for both extraction and classification. They can be used with a combination of unsupervised and supervised learning. For example, many unlabeled examples may be used to initialize weights in the network, and then labeled examples are used to fine-tune the output. Parts of networks may also be combined to create networks. For example, a complete network trained on a large database, can have the final classification layers removed, and instead be retrained on a similar, possibly smaller, dataset, and result in better classification then a network trained only on the small dataset. Neural Networks can also be trained in batch mode, which means not all of the training data must reside in memory at once, and larger datasets can be used more easily. While neural networks are more difficult to use then other methods, they have been shown to have state-of-the-art performance.

Since this system is intended to run on camera data, locating different objects (and later attributes) was important. Generally neural network classifiers are run on an entire image where the assumption is that objects are large enough to be seen and take up most of the image. There have been networks recently which are capable of producing bounding boxes as well [45]. For this prototype, we handle localization by applying the network to patches of the image. That is, when given an image we create a sliding window at several scales and run it across the image, producing a set of overlapping patches that represent the entire image. We keep track of where these patches came from, and present them to the image. Upon their return, they are recombined into the total image, producing final votes for a single label for each pixel. This is only applied to object recognition, and changes would have to be made for attribute prediction, for example, instead of taking the maximum label, all labels would be kept since more than one can apply at any location.

For both training and testing, only Kinect data was used. Other then scaling, no preprocessing was performed. Unlike in the previous chapters, the background is not removed. Roughly a fifth of each of objects in the RGBD dataset is included in training and testing here, as well as a dozen of each of the objects collected separately for these experiments.

5.1.1 Use of Depth Data

Objects, for the most part are visible, but their attributes may be abstract concepts, invisible to the camera. While it may be possible to "see" the shape of an object based on color variation alone, to give the system an additional advantage when dealing with words like *round* and *bumpy*, and discriminate between the textures and color patterns that replicate them, the system uses depth data.

Depth data as collected by a Kinect or primesense camera, is a 2D array of distance values, one for each pixel. This data introduces an additional dimension to the input, and may require scaling or normalization depending on the learning method used. Generally, the depth is in meters or millimeters, and stored as 16 bit floats, ranging from 0 to 10,000 in the case of millimeters. An important limitation of these specific cameras is that they fail to capture transparent or mesh materials. They also have a limited range, and lights directed at the camera can cause depth readings of zero, which is the default "unknown" depth value.

Depth on its own is an interesting classification problem. Our goal was not to focus on depth, but to see if it contributed to our learning without special alterations. The only special processing that was applied to the depth data was scaling for networks where the size of the depth data were likely to overwhelm the color data. While the initial weights for these values are separately learned, deeper layers rely on both color and depth. It may be possible that weights are learned which do not respond poorly to the difference in scale, so both scaled and unscaled depth networks were trained.

5.2 Implementation

For this experiment, Cuda-Convnet was used [28]. Convnet is a convolutional neural network framework developed for a Kaggle competition which challenged users to achieve the best top-1 and top-5 scores on the ImageNet dataset [11]. ImageNet is a database of around 14,000,000 images collected from across the web, sorted into 1000 synsets (synonym sets). Convnet provides the layer formats that helped provide the

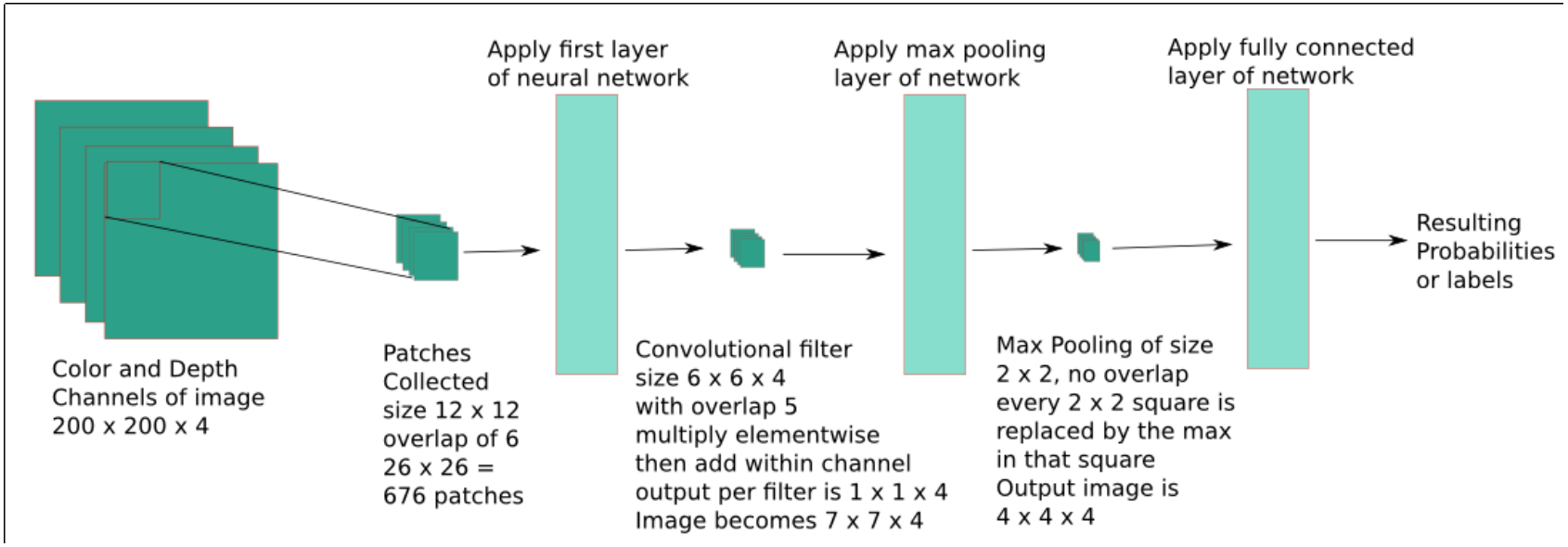


Figure 5-1: Example of movement through a convolutional network. For simplicity, only a single image and filter for each layer is shown, applied to only a single location. In practice each convolutional layer would contain around a hundred filters which would be applied across the image. The fully connected layer flattens the data into the resulting label predictions, and is often a logistic regression layer.

best performance on ImageNet, which are likely to provide similar benefits on other image datasets, with adjustments.

For the addition of camera use, a combination of OpenCV and OpenNI was used. OpenNI provides support for both Kinect and Primesense cameras, which were used in order to determine to usefulness of depth data. There are some issues at this time, with later versions of OpenNI not including the Kinect drivers. It is also now the case that OpenNI itself will no longer be available, so an alternative will be needed for later projects. The Primesense and Kinect cameras provide the same resolution of RGB and depth data, which can both be streamed and registered together. OpenNI provides a C++ interface and a very basic python interface as well. The cameras provide 30 frames per second.

As a starting point for network settings, CIFAR [27] and ImageNet [11] networks are used as provided by the Cuda-Convnet library. The settings are then varied slightly to find better performance. The CIFAR network is made of three convolutional layers followed by a fully connected layer. Each convolutional layer is followed by a max-pool layer, which down-samples the input by replacing a small region with the maximum value in that region. Each of the convolutional layers also use rectified linear units as activation functions. These "relu" neurons output the max of the input and zero, and improve training speed, as discussed in Krizhevsky's Imagenet paper [28]. The ImageNet networks contain five convolutional layers followed by two fully connected layers. The first three convolutional layers are followed by max-pooling. Each of these default networks are provided with Cuda-Convnet. Slight changes were made to the size of the filters to improve behavior on the Kinect resolution. This meant doubling the sizes of the first and second layer filters. The learning rate was also decreased, since a large learning rate led to NaN values as output.

To locate the objects within the image, a sliding window is used to produce patches at several resolutions. These patches are then scaled to a final resolution and presented to the network which can only accept one size of image. After prediction these patches are used to construct a final prediction for each pixel.

Table 5.1: List of what processes were varied.

Tests
multiclass object recognition
with depth vs without depth
raw depth vs scaled down vs color scaled up
small network vs large network
varied sizes of patches
varied scales of input to network
with mean subtraction vs without mean subtraction

5.3 Results

Cuda-Convnet is built to optimize overall accuracy, and does not allow easy access to more interesting values. In this case, accuracy values are collected across classes. Shown in Figure 5-2 is the error at each testing iteration as calculated on the test set. In a perfect world, this graph would be smooth. Much of this jitter is due to very small training batches which do not capture the scope of the data. Thus, each batch moves the model towards a better representation of those samples, but away from samples which belong to the next batch. Batch size was limited by the size of the images and memory available on the GPU, which had to contain the entire network as well as the training data.

Gradually, performance does improve, but with huge fluctuations. Towards the end the error rate increases, which is normal behavior as overfitting begins. In the initial setup, early stopping was not used, so training continued until the system was manually stopped. After this error curve was observed, generalization loss and progress quotient early stopping were implemented to automatically find a fair point to stop training, as described in Section 2.2.6.

Multiple scales of patches were used for training and testing, which were all rescaled to the same size for the network to run on them. Mean subtraction worked best when the mean was found across patches and subtracted from patches, though this may be merely an effect of the size of the dataset: less data means a better mean is found across patches then across images because the patches vary more then

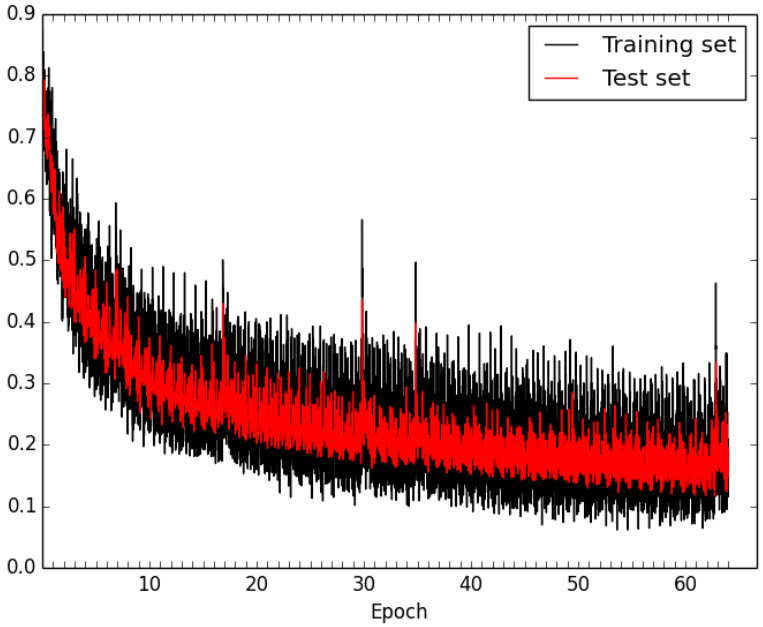


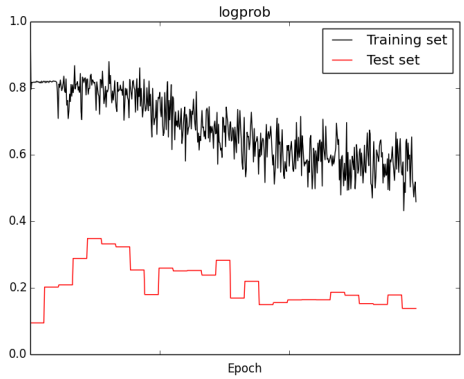
Figure 5-2: The error on training and testing sets. Training iteration along the bottom, percent error along the side.

a single region of the image. Basically, 10 images produces several hundred patches, which are in a sense, a larger dataset.

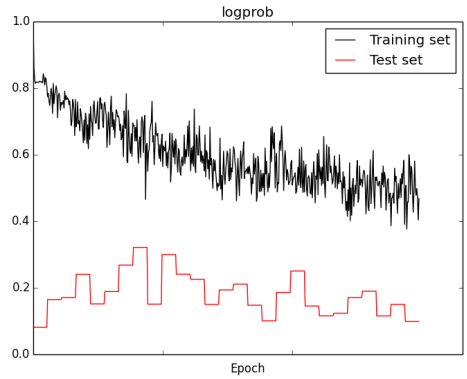
Patches worked best when reasonably large, although this may be influenced by the scale of the objects and their distance. In the case of this dataset, patch size of 96 pixels did well, even when scaled down to 48 pixels for the network. Smaller patches did not do well when scaled up, though they did reasonably well if left the same scale. The filters that resulted from this scaling appeared blurred. Other recent papers have indicated that scale does not matter, at least not when created artificially [12]. The largest resolution seems to be the only one that has influence. However this may not apply to collecting the data at different distances.

5.3.1 Use of Depth

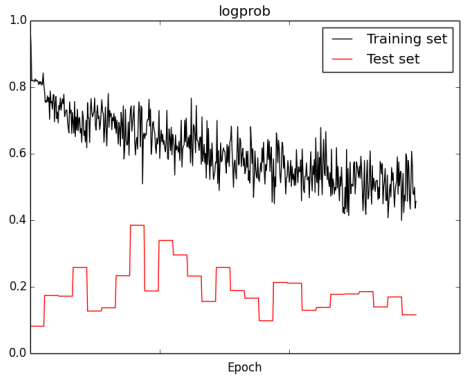
The same network was trained without depth, with depth, with depth scaled down to match the color range, and with color scaled up to match the depth range. The



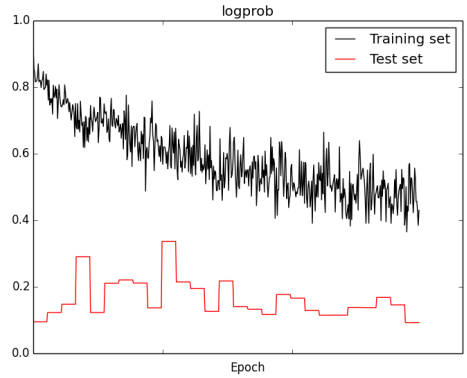
(a) Error rate with no depth.



(b) Error rate with unscaled depth.



(c) Depth scaled down to color range.

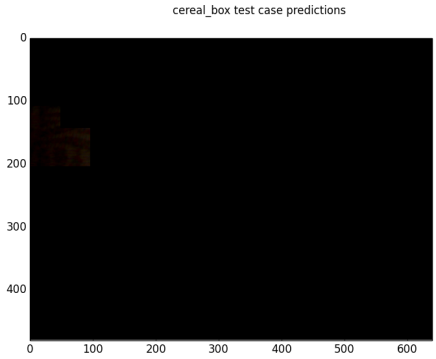


(d) Color scaled up to depth range.

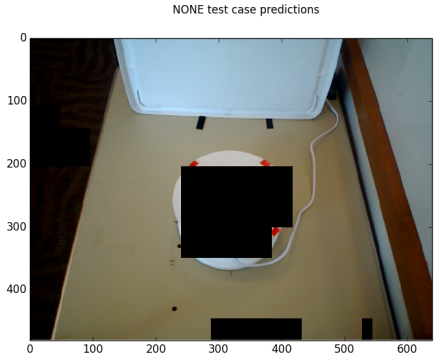
Figure 5-3: Comparison of error rates during learning using no depth data, depth data which is unscaled, depth scaled up, and color scaled down.

error rates of the first three epochs of training for these three are shown in Figure 5-3. Including the depth results in almost no performance increase at this point. There is a slight improvement of only a few percent. Towards the end of training this may become more significant. If attributes were separated, it may be clear whether depth contributes to certain labels and not others, such as *flat* and *round* but not *red*.

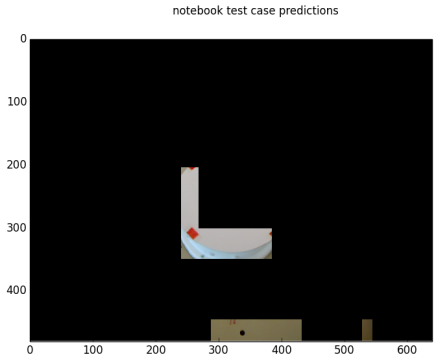
Seen in 5-7 is the error rate resulting from training a CIFAR based network on color and depth data where the color data was scaled up to match the depth data. The performance reaches roughly 80% accuracy on the training set, varying from over 90% to below 70%, with higher accuracy on the testing set due to presence of additional background patches which are easily classified as *NONE*. In training, these



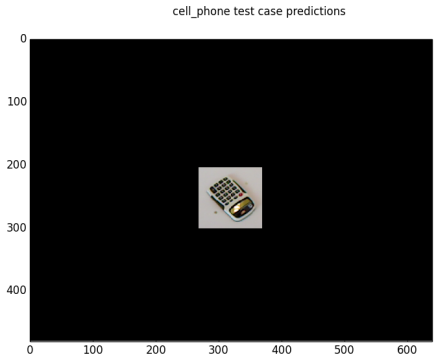
(a) Prediction for label *cereal box*.



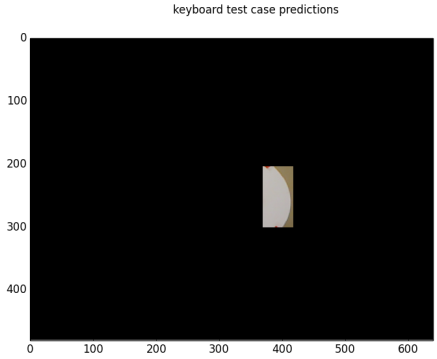
(b) Prediction for label *NONE*.



(c) Prediction for label *notebook*.

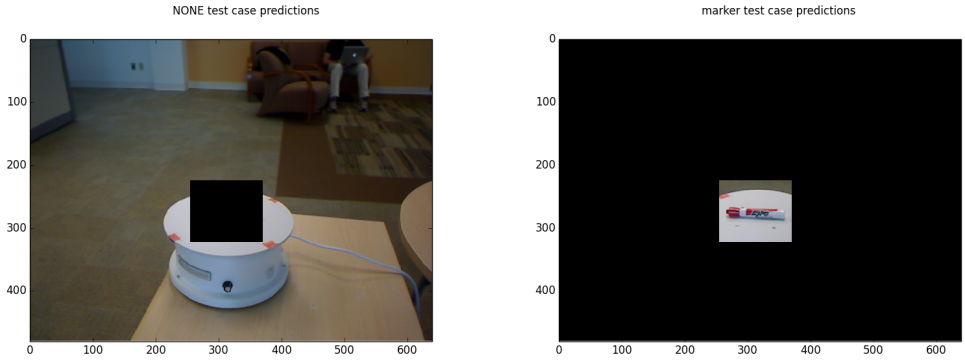


(d) Prediction for label *calculator*.



(e) Prediction for label *keyboard*.

Figure 5-4: An example of the system running on an image. Patches are taken from across the image and predicted separately, then recombined into the complete image again. Where patches overlap, the most common prediction is used as the final prediction. This case shows many false positives.



(a) Prediction for label *NONE*.

(b) Prediction for label *marker*.

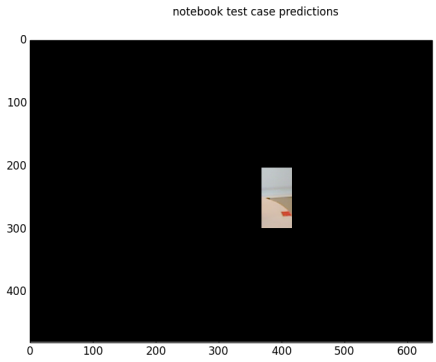
Figure 5-5: An example of the system running on an image. Patches are taken from across the image and predicted separately, then recombined into the complete image again. Where patches overlap, the most common prediction is used as the final prediction. This case has perfect prediction.

background patches are limited to avoid this misleading boost in performance, and the skew that additional background training causes. The fluctuations, as mentioned above, are likely due to the network shifting between representations which are better for some data, but not others. It could be that some of these labels cannot be well learned, whether this is because they are not visible or poorly represented. If this is not the case, it is likely that the network is underfitting, which can be improved by adding additional filters and nodes to the network so that it can learn additional features to represent the data.

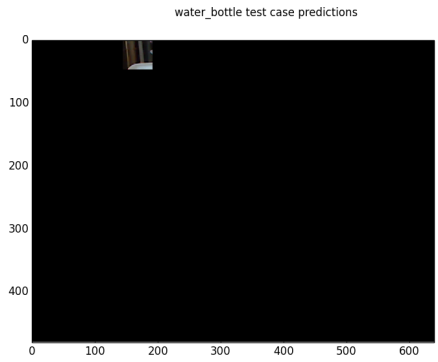
To test the underfitting effect, the Imagenet network was used. The larger network was trained on the same data. However, this resulted in worse performance. It is likely that a network slightly larger than the CIFAR network improves performance, while the ImageNet network, which was meant to classify 1000 classes, and trained on millions of images, is too large for this dataset.

5.3.2 Mean Subtraction

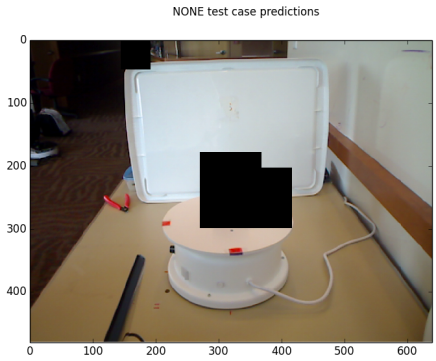
Several technical papers mentioned mean subtraction as an important preprocessing step, and a quick comparison showed the value of this step. Calculating the mean



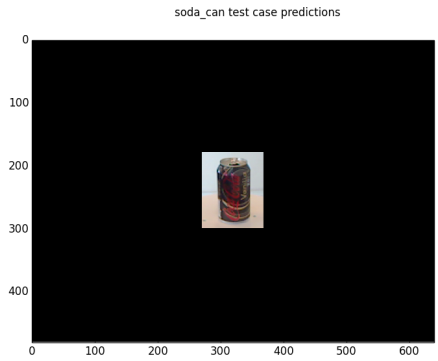
(a) Prediction for label *notebook*.



(b) Prediction for label *water*.



(c) Prediction for label *NONE*.



(d) Prediction for label *soda*.

Figure 5-6: An example of the system running on an image. Patches are taken from across the image and predicted separately, then recombined into the complete image again. Where patches overlap, the most common prediction is used as the final prediction. Notebooks may be often misclassified because of their flat shape: the cropped regions around the notebook include pieces of the turntable. Water bottles may also have poor classification because they are largely clear, though this is purely speculation.

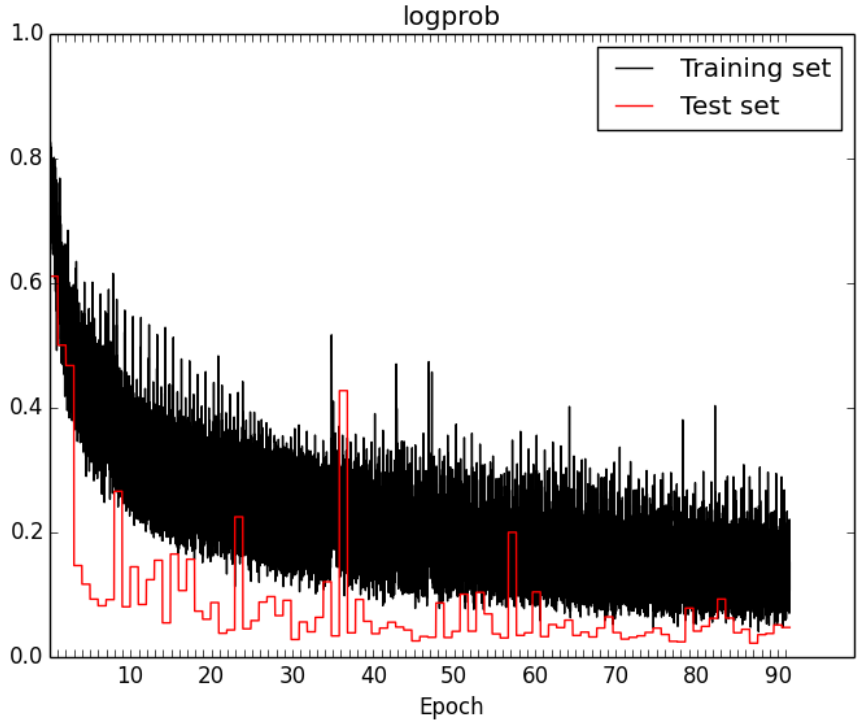
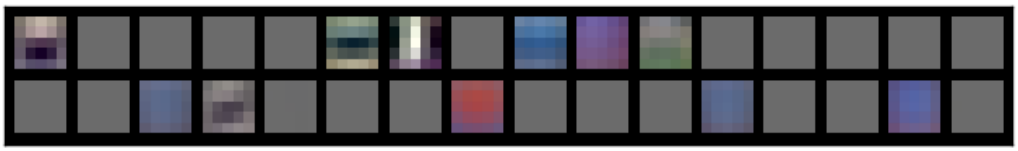


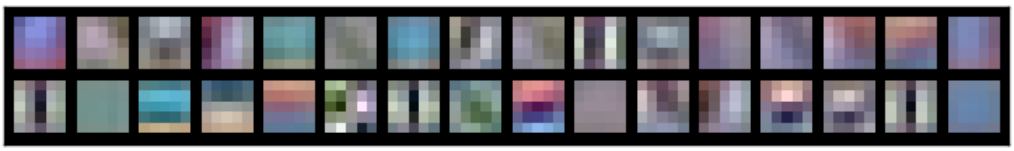
Figure 5-7: The error on training and testing sets. Training iteration along the bottom, percent error along the side. This network was trained with color scaled up to suit the depth data.

of the the data and subtracting this from each image prior to training, and testing resulted in much better performance. This is a simple type of normalization. Figure 5-8 shows a comparison between filters learned with and without mean subtraction on the CIFAR dataset. Since our system actually ran on patches of image data across the images, tests were run with either the mean as calculated across the image, or across patches.

It was found the subtraction of the mean per patch resulted in better performance, though this may be due to the size of the training set, and the repeated background in the RGBD dataset. For example, most of the images from the RGBD dataset are of an object in the center, on a turntable, with the same background. Thus the only place where values varied was in the center where the object changed. Stacking the patches from across the image together created a much more reasonable mean and normalization.



(a) No mean subtraction.



(b) Mean subtraction.

Figure 5-8: Comparison of filters learned without and with mean subtraction. Grey filters essentially contribute no features, and have learned nothing.

5.3.3 Speed

This implementation resulted in a framework which could classify patches at three frames per second. This is unfortunately not fast enough to be considered truly real-time. Analysis of the run-time of the experiment showed that processing of each patch took the most time. Were these processing steps moved to the GPU as well, then the process may be considerably faster.

The breakdown of times is shown in 5.2. In this particular run, the mean was subtracted on a per patch basis, which is why the second step takes almost no time. Since the patches were treated as separate inputs to the network, the mean was calculated across patches and subtracted from patches. This showed better performance than subtracting the mean of the full images from the full images. However, this also increases the amount of time taken. If mean subtraction and drop-out across the full image, both take less time, but performance decreases. Reshaping must be performed: this is the step that formats each image as a flattened C order vector, which is how the data must be moved onto the GPU, rather than as a 3D vector. Each of the Cuda kernels is designed to run on these flattened vectors. The more patches there are the longer this step takes.

Table 5.2: Time Taken to Process a Frame

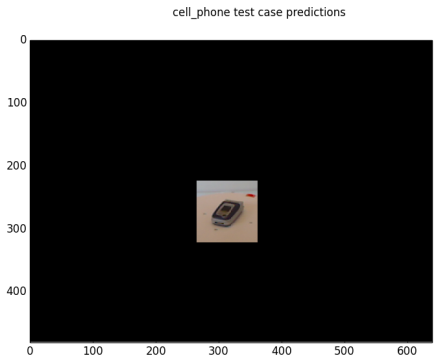
Step	Process	Time (s)
1	Scaling	9.798e-2
2	Removing Mean (full)	2.86e-6
3	Creating Patches	2.200e-1
4	Removing Mean (patches)	1.642e-1
5	Reshaping Data	3.277e-0

5.3.4 Object Detection

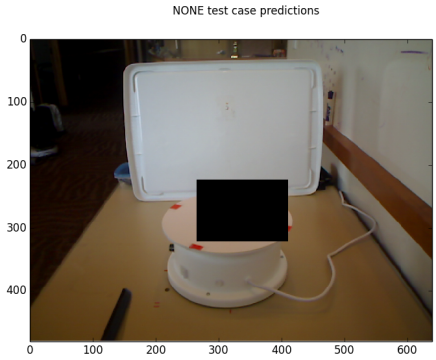
During detection of objects in the image, the system was able to quickly locate some items. However, the false positive rate remains high. While much of the background is identified as NONE, and certain objects are well recognized, such as *keyboard*, and *water bottle*, there are also false positives where portions of background are recognized, for example, portions of the table are often classified as *notebook* and *binder*. It is possible that the influence of the depth data is confusing flat objects, but this has not been investigated.

Accuracy across classes for the multiclass convolutional neural network is as good as the support vector machines, and only a single network was needed. However, per class accuracy varied. This was not immediately apparent, since the cuda-convnet framework was designed to present only the accuracy and negative log probability on a given batch. When the program was run, it was clear that again, well-represented classes performed better which is to be expected, but rare classes were being overfit. Also, while behavior on the held-out dataset was fair, when run on input form the camera it became clear that more examples taken from the environment where the system was run would be needed. This is seen as good performance on samples from the RGBD dataset, and worse performance on the additional samples and camera input.

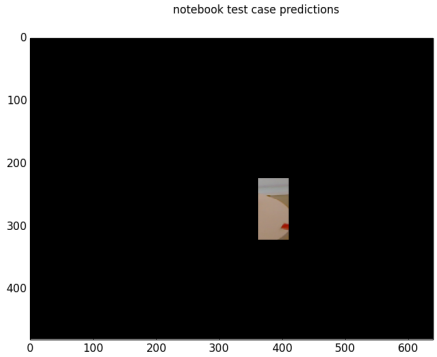
A portion of this is due to varied lighting. Krizhevsky [28] presents a possible addition to handle this issue. PCA is performed on pixel values, and additional training samples are created by adding multiples of the principal components, effectively simulating varied lighting. However, this method may only work correctly for object



(a) Portion of the image classified as *cell phone*.



(b) Prediction for label *NONE*.



(c) Prediction for label *notebook*.

Figure 5-9: An example of the system running on an image. Patches are taken from across the image and predicted separately, then recombined into the complete image again. Where patches overlap, the most common prediction is used as the final prediction. For further examples, see Appendix B.

Table 5.3: Summary of Results

Results
mutliclass prediction has good accuracy (80-95%)
depth makes little difference
scaling of depth and color make little difference
small networks perform better on this problem
larger patches performed better
scaling patches up decreases performance
mean subtraction improves performance
mean found across patches increases performance

recognition, where the object's identity is not entirely dependant on color and lighting. In the case of attributes, especially color and brightness, this could actually add poor examples to the dataset.

5.4 Discussion and Further Work

After observing the results of the Cuda-Convnet framework, it was decided that further tweaks were needed to handle unbalanced data and prevent overfitting. This would include finding alternative cost functions, regularization terms, adding features by increasing the number of filters in the network, and weighting the samples which occur rarely, or attempting to balance the data.

Depth data appears to have little effect on training. This may be because depth information is not useful to most labels, and so makes a small impact. The network may learn to ignore this information if it cannot quickly learn from it. It may help to have a separate network altogether for the depth data since it is so different from the color data. The effect of depth data on specific labels should also be further investigated.

In spite of the lower performance when compared to separate SVM classifiers, the improvements over the Theano implementation of neural networks were encouraging, and the advantage of training a single network rather than a set favors further development. It is still likely that the failures of the system are due to poor data; the lack of varied lighting, and the need for additional examples from the setting where the

system is to be used. To discover the source of the variance in performance, measures taken per class should be made using precision, recall and F1 score to better find the specific examples which fail, and possible fixes.

Speed was fair, but until performance is improved it should not be a major goal. If improved accuracy per class, and better F1 and false negative rates can be achieved, then additional steps may be taken to improve speed. For example, portions of the processing were done prior to transfer to the GPU, specifically the creation of patches of the image. This step is very similar to convolution itself, the different being that instead of applying a network to the entire image, it would be applied to each patch of the image. This would speed up all the steps beyond creation of patches in Table 5.2, although they would have to be adapted to either run correctly prior to patch creation, or be applied on the GPU again.

5.5 Summary

This chapter described a system which uses convolutional neural networks to locate objects in images streamed from a camera. This was done by applying the network on patches from across the image. The use of depth data made little difference in performance regardless of scaling, although this may only show that this dataset does not rely depth to differentiate labels. Mean subtraction worked best when done across the patches rather than the image, and was essential to learning. Results were improved by the use of early stopping, which ensured the model was stopped if error began to increase. Size of patches was best when the patches were larger, and worst if they were small and then scaled up. The best networks achieved 92% accuracy across classes.

Chapter 6

Multilabel Convolutional Neural Network

For the final system in this thesis, the attribute prediction problem was revisited. It was a goal to retain the single neural network, or attempt to, but now the system had to allow for samples to belong to multiple classes at a time. As discussed in Section 2.2.5, this is a more difficult problem. Work was also done to improve general performance and prevent overfitting as seen in the previous chapter. Most classifiers allow for multiclass prediction, rather than multilabel. It is also the case for neural networks that the best loss functions are not differentiable, and so sub-gradients must be used instead. As with multiclass prediction, it is more difficult to quantify performance, since performance across classes may not capture the fact that some classes are well learned. Added to this is the issue that a given sample belongs to some number of classes, and credit should be given in a performance measure for getting some of these correct. It becomes much more difficult to handle unbalanced data, since adding examples for any class adds to other classes as well, and if a sample is not labeled for all classes, it is a more complex unknown, and the choice of including it or not is more difficult since it is useful to some classes, but may contribute to poor learning of others. As learned from the previous Cuda-Convnet based system, special attention was paid to additional regularization, and improvements for net-

work behavior. A new library was used called Caffe, which added several types of neurons, and showed slightly improved performance over Convnet. Caffe also offers an entirely C++ codebase, which would hopefully be more efficient in both time and space. Unfortunately, Caffe has a much more strict format, which made changes and extensions difficult.

6.1 Approach

Convolutional neural networks of several shapes and depths are used again, based on Imagenet and CIFAR networks. To alter these structures for multilabel data, additional output was permitted and the final layer and cost functions had to be exchanged to those which produce values per class. As discussed in the related works chapter, there are several multilabel cost functions. Only the simplest, Softmax, was used, until sub-gradients can be implemented for the others.

The remaining samples from the RGBD dataset were added along with a few dozen images taken of other objects in the lab. These were labeled semi-automatically, with several being labeled manually and these labels being propagated across the objects where possible. This may be a source of some error, similar to the mislabeled crowdsource datasets discussed in 3.1.2. The addition of this data may in fact decrease performance, as discussed in the results below.

6.2 Implementation

As mentioned above, the Caffe system was used. Caffe uses Google's protocol buffers to represent data and networks, Google-glog for logging of output and errors, and Intel's MKL to improve CPU performance. Their default data is loaded from a levelDB, which is not a standard database, and only built for fast in-order access. This system is much more complicated then Cuda-Convnet, and more difficult to make changes to. This was not clear initially.

While their network was closer to accepting multilabel input then Convnet, there were still several functions which assumed a single label was true, especially in the final layers that calculate cost and accuracy. Changes were made to the initial and final layer definitions, and to the protocol buffer datum type to allow for multiple labels, and an extra channel of depth data, which forces the data value into float type, where the standard type is byte strings. Crop values were also added so that the system could take multiple sub-regions of each image without losing the object. No localization or detection was performed, only prediction of the center of each sample for all labels.

As in the previous chapter, the networks are set to CIFAR and ImageNet sizes, then varied. Also, early stopping, and measures for precision, recall and F1 score had to be added, including per class. Even these fail to completely capture performance, and additional measure were also investigated, including specificity, an average of precision, recall and specificity, and Matthew's Correlation Coefficient to find what failures each of these accounted for and what they failed to capture.

Currently a number of the features in Decaf, the Python version of this library, are not available in Caffe, the C++ version. While the network layers and activations are complete, and drop-out is included, only weight decay is available as regularization. It is likely that additional forms of regularization, multilabel data, and additional cost functions will be added to the main library eventually, but they are not currently included. There is an active community adding to the library, and it may continue to improve with time.

Softmax was used as the loss function, and it produces probabilities across all classes rather than values of 0 and 1. To find the ground truth probability from the labels, each label is divided by the total positive labels for that sample, so that all the positives share a maximum value, and the negatives share the minimum, which should always be zero. For further information see [15]. If the network trains well, then all the predictions should fall at nearly the maximum or the minimum. To allow for some error, a threshold can be used instead of probability match. When prediction

Table 6.1: List of what processes were varied.

Tests
with depth vs without depth
small network vs large network
multilabel prediction of attributes

is done on unlabeled samples, a threshold must be used.

This threshold will depend upon the number of true labels, which is unknown during prediction. It was decided that a threshold would be calculated at each prediction based on the predicted values. The maximum and minimum were found, and used to find a cutoff value which was subtracted from the maximum to create a threshold. Predictions that were close to the maximum are considered positive predictions while lower values are considered negative. This should force all of the "true" labels towards the maximum value, and all of the "false" labels towards the minimum, although it does not enforce a strict boundary. This evaluation produces fair results, although additional study should be done on how this effects learning.

$$cutoff = (\max Y - \min Y)/2.0 \tag{6.1}$$

where

if $Y_i >= \max(Y) - cutoff$: $C_i == 1.0$

else $C_i == 0.0$

Y is the set of predicted probabilities.

C_i is the class i prediction.

Y_i is the class i probability.

Table 6.2: Results For Imagenet size and CIFAR size Networks, with depth data

Size	Depth	Accuracy	Precision	Recall	Specificity	Max F1
ImageNet	no depth	.9329	.2659	.4061	.1781	.3214
ImageNet	depth	.9617	.4885	.9770	.05486	**.6513**
CIFAR	no depth	.9626	.4912	.9657	.3251	**.6512**
CIFAR	depth	.9631	.3239	.2383	.8398	.2485

Table 6.3: A sample run of training the ImageNet scale network of color and depth.

Iteration	Accuracy	Precision	Recall	Specificity	Max F1
2000	.9616	.4773	.9885	.0000	.6437
4000	.9617	.4885	.9770	.0549	**.6513**
6000	.9620	.5662	.2701	.8088	.3657
8000	.9619	.5354	.1590	.8725	.2452

6.3 Results

The multilabel results were expected to be worse then the separate classifiers, and they were. Similar to the results seen in [15], accuracy is fair, above 80%, and usually above 90%., while precision and recall are lower. These results are shown in Table 6.2. Since this prototype uses different data, no direct comparison can be made between the scores. Per class Precision and recall reached .96 and.48 respectively, contributing to an F1 score of .65. However, it should be noted that this network also has a specificity of .05, meaning the network predicted many labels well, but also produced many false positives and few true negatives. Thus labels were often predicted for samples which they did not apply to. The second best F1 score for the same run, shown in Table 6.3, has a much higher specificity of .80, but also a much lower F1 score of .36. The ImageNet size networks, in this case, performed better than the CIFAR size networks, which makes sense since these networks were expected to learn more complicated multilabel prediction on full-scale images rather than on patches.

Overfitting was observed in this training set as well. This is seen in high and low F1 scores, specifically when the system predicts all true, all false, or all correct for any label. When a common class is always predicted True it appears as a very high

recall, a reasonably high precision, and a reasonably high F1 score. The failure is only shown in the True Negative or Specificity measure. The reverse, an uncommon class which is always predicted as false, appears as an undefined Precision, with zero recall, and zero F1 measure. This appears more likely as training goes on, when either the positive or negative examples overwhelm the opposing examples. This is not shown in the averages, but was revealed by measures for each class.

Early stopping was added to the network and set to watch the average F1 score rather than the accuracy. This is not advisable, and was implemented here not to improve results but to cut training short. Since the loss function is not tied to the F1 score, the training does not directly improve F1 score, only accuracy. As training continues, the accuracy increases, but the F1 score in this case tended to increase then decrease as the accuracy forced the specificity up and precision and recall down. This happens because negative samples in the data out number the positive examples, leading to better prediction when negatives are predicted. With binary classifiers, this can be fixed by presenting the same number of positive and negative examples, but when each sample has multiple labels, a positive sample for one label may add several negatives and positives for other labels. Most likely weighting contribution of samples to each label will be more productive, adding greater weight to the value back-propagated through a rare label.

As well as testing multilabel performance, the reason for moving away from separate classifiers for each label was the time and memory necessary to train such systems. Training for both systems took several days, though part of this is due to parameter searches which may be done more efficiently for both separate and multilabel classifiers using better parameter search techniques, as discussed in Section 7.3. However, memory was much more efficient in the neural network in spite of the size and number of parameters.

While it can be seen in the table that the networks which used depth data performed better than those which did not, this should not be relied upon. The networks which did not use depth data produced NaN values more often, which cut training

Table 6.4: Summary of results

Results
depth data had more reliable behavior
large networks performed better
multilabel prediction had worse performance then separate classifiers

short. This is likely due to a poor learning rate, but was not changed since the networks were to be kept as similar as possible for comparison. Were this value changed, the network may have produced better results, but comparison would still be in doubt. It is interesting to note that without depth data, the smaller networks performed much better, but with depth data the larger networks performed better.

6.4 Discussion and Further Work

Caffe was difficult to work with. A number of assumptions made by the library made the addition of multilabel data and measures difficult. Correctly installing the MKL dependency, which requires a license, was difficult on most systems. However, the library is a work in progress, and MKL is being removed as a dependency and replaced with open-source alternatives. There are also continuing improvements and extensions, including several types of regularization which are applied per layer rather then across the entire network.

A serious issue with levelDBs, the initial data format provided for by Caffe, is that they are not very compact, and are not meant for random access. In order to correctly perform cross validation, the data should be shuffled and presented in different orders. This is not possible with the levelDB format, which is iterative. With large datasets, levelDB does not provide a compact format for storage. Both of these issues recommend for another data storage format. Another future addition to Caffe is other data layers which read other formats, although these only exist on their development branch and are incomplete at the moment.

An intriguing option for improving classification is grouping labels together into smaller multilabel classifiers. Learning features together across all images allows the

extractor to take advantage of all of the information in the dataset, but it may also mean much larger feature extractors, and sets of features. Separating the extraction techniques means learning just the features necessary for a single label, but this wastes space, since many labels rely on the same features, and certain features will only be found if they are needed to discriminate between classes. It is likely that clustering the labels in some way into medium sized multilabel classifiers will allow for a gain in space and accuracy between these two extremes. Perhaps a clustering based on which filters contribute to labels would be best.

Incorporating the F1, MCC or another measure of performance into the loss would likely improve results across classes. Also, since these measures are made necessary by unbalanced data which is not easily fixed in the multilabel case, weights may be useful when applied to each class, increasing the influence of rare labels. This was implemented but not fully tested.

6.5 Summary

The chapter presented a multilabel convolutional neural network system which predicts multiple attributes and objects for each sample. Performance was not as good as separate classifiers, and both systems took several days to train but the network took up less memory during training. While accuracy of over 96% across classes is achieved, and an F1 score of .65, these values do not reflect the specificity or true negative rate, which is much lower. It is suggested that an improved loss function which better represents true performance on all labels should be used.

Chapter 7

Discussion and Future Work

While results and possible changes were discussed at the end of each chapter, this chapter reviews these results and suggests improvements. Reasonable attribute prediction measured by both accuracy and F1 score was attained using multiple systems which included Isomap, SVMs, and neural networks. However, these result were found on a small dataset, and while work was done to ensure these measures were not overly optimistic, further evaluation should be done, especially with respect to the utility of depth information. Neural networks are being developed with better methods of training, techniques for selecting hyper-parameters, and faster implementations, which may be used in the future.

7.1 Data

Arguably, the training data will always be difficult for this problem owing to the complexity of human language, and the world. As more samples are added, the problem will be more difficult. To test systems, the language may be simplified, and images may be carefully prepared, but eventually the system should be able to handle any labels, and complex scenes. Recommendations for simplifying the data include using a smaller, more specific vocabulary, and ensuring full-labeling of samples with this vocabulary. The images could be simplified as mentioned in the first project, by

masking. More complicated data would include cluttered scenes and more complex language that may require parsing.

Collecting additional data and labeling it specifically for this problem at several different difficulties may be important to improving results. Labeling may be done more efficiently with Mechanical Turk [23]. The following data resources may also be useful.

7.1.1 RGBD Dataset

This dataset is, of course, already used. There is a chance that it will be broadened in the future, and there are other groups using it, who may contribute labels that would be useful. For example a very recent paper attempted to use this data for attributes, but their labels were not provided [46]. However, as shown in the previous chapter, this dataset is repetitive and may damage performance if used incorrectly. As a starting point it is useful, though it may be better suited to other problems.

7.1.2 NYU Depth Dataset

Other data that is provided with both color and depth is provided by NYU [37]. This dataset is intended for scene classification and segmentation. It is much more cluttered then the RGB-D dataset, however, it may be useful to pull out portions of images for training. For example, if the system uses patches of images, instead of the entire image, or if it can learn from multiple sizes of images, then the portion of the image that is labeled "chair,", "stapler," or "book" may be used for training. This still adds a layer of difficulty and complexity to the data, but in the long-term may be useful.

7.2 Use of Depth Data

Depth data does not have a significant impact on the attributes represented in the current dataset. This may be merely a property of the dataset: the labels learned

here may could be determined by color data alone, not because they refer to color, but because they are only used to refer to an object which can be labeled by color alone. For example a box may be a *blue cube* with *flat sides* and *straight edges*, and while we would expect the depth data to contribute to the labels *flat* and *cube*, if the cube is the only *blue* object, the system would learn to predict *cube* for anything *blue*. While this is an extreme case, it is quite possible in such a small dataset that depth is not important to differentiating between labels when those labels are not represented by many objects.

7.3 Convolutional Networks

While the first attempt at using convolutional neural networks showed poor performance due to overfitting, later attempts showed competitive results, producing better accuracy, precision and recall scores. Although networks are complex and require additional tuning in the form of both hyper-parameter setting and regularization, they offer more control and improved behavior, and may prove to be better able to handle the even larger datasets necessary to continue attribute learning.

Convolutional Neural Networks show promising power as both feature extractors and classifiers, although a continuing problem is choice of hyper-parameters and overfitting. These parameters are still often set by hand. Although applying grid search, random search, or Bayesian Optimization [44] would help select these, it would still be a time consuming process. The necessity of extensions like regularization, and early-stopping, which are often missing from the current libraries, imply that it is more effective to implement the networks in a system such as Theano then to rely on these existing frameworks. As an alternative to multilabel prediction it would be interesting to replace the classification layers of networks with SVMs, as done in several works mentioned in Section 2.2.4, which showed improved performance.

Several recent neural networks have split the network and reused portions of networks for different tasks. As mentioned earlier, a network was used purely for feature

extraction before being fed into standard classifiers, or into several other networks which were meant to perform significantly different predictions, such as object classes, and locations of objects [45, 13]. Such division of networks may help in this problem. For example, the same large network may be used for feature extraction where most of the time and memory are spent, and the final layers may be separated to improve final classification. Different compositions of classifiers, and divisions of either attributes, objects, or stages of the pipeline should be explored.

7.4 Multiple Label Prediction

Separate classifiers performed nearly as well as the neural networks, but the feature extraction techniques required large amounts of time and space to find fair representations of the data. While it is likely that more labels, and broader data would decrease performance, it is also likely that more can be done to improve multilabel prediction. Both types of classifiers produced poor results on the unbalanced cases, and balancing these classes by bootstrapping improved results to some extent in the case of the single label classifiers.

Use of the more complex multilabel Weighted Approximate Ranking Pairwise loss for multilabel neural networks may improve performance, although the comparison made between softmax and WARP showed only a very slight increase in performance. It would also be useful to monitor precision, recall, and specificity since these three capture both forms of biased overfitting, while accuracy does not, and it may be possible to incorporate these into a loss function.

A compromise between the extremes of one classifier and one for each label should be explored, by combining similar labels. The labels could be clustered into synonym groups as is done in the ImageNet dataset, or it may be possible to cluster them based on their learned representation in a network.

For example, if a classifier is a neural network, then the prediction is the result of the input image moving through a directed graph, and learning is performed by

moving errors backwards through this graph. The result is nodes within this structure develop to recognize specific features. For example, some specialize in edges, others in colors. Labels for color would learn features that react strongly to components of their color, and weakly, or not at all to components of other colors. Traveling backwards through this structure can determine which portions contribute most to a label and how they contribute. It is likely that labels that are visually similar would have similar activations in the network, such as *red* and *orange* sharing features. This could be used to cluster similar labels, and separate them into classifiers which could be made more compact and specialized.

Another method of multilabel prediction which is also used to set hyper-parameters is cokriging, a multi-output version of Gaussian Process prediction. A Gaussian Process is a collection of random variables, subsets of which, have a joint distribution. Kriging is Gaussian Process regression, and cokriging is the multiple output form. It predicts a variable Z auto-correlation and cross-correlation with other variables. This process is used in geostatistical analysis, but has been shown to be useful in setting hyper-parameters in neural networks, but can also be used to make multilabel predictions directly [1]. This method is computationally intensive, but may be worth looking at in more depth.

7.5 Improved Speed

To obtain the fastest possible classification performance, a hand written parallel, pipelined GPU implementation and specially designed hardware would both be needed. Without knowing the exact structure of a good classifier, general purpose code frameworks and hardware must be used to reach reasonable performance and explore possible systems. To improve the speed of any system used here, the best option is to implement it in CUDA for the specific GPU intended to be used. This of course means understanding GPU structure and the CUDA software development kit, and the algorithm which is being implemented.

There are many libraries that provide implementations of learning algorithms, but as seen here, they may be meant for a narrower problem, or missing features that are needed for a problem. As a starting point, libraries are a good way to get initial results and estimates of behavior, but they generally are not meant to be fast. Some do provide reasonable speed, but these will not be as fast as one created for a specific system or problem.

For each of the systems discussed here, only a single computer with a single GPU was used. As mentioned above, if the system is split into several classifiers it would be possible to distribute the system across several machines, placing a subset of classifiers on each machine, or a medium size network on each GPU, and combining the final results. This may speed up the entire system, although the communication between machines may also add time.

7.6 Automatic Structure Determination

Many of the issues seen relate to incorrectly assuming a much simpler structure and being unable to determine the true structure of relations between labels. Without manually selecting labels which we know to be independent, we cannot avoid this problem. To automatically recognize similarity between labels, or a hierarchy we need to be able to recognize those labels in the first place. A possible future goal is to use the learned classifiers to recognize the relations between labels which were not carefully chosen. This way a person might use whatever words they want to describe an object, and a robot could in time discover the synonyms or contributing attributes based on the visual relationships.

7.7 Libraries

Many different outside resources were used in this work, and often problems were discovered with each. This section recommends each and describes important limi-

tations of each of the systems which may or may not be remedied in later versions. While it may be necessary to implement new algorithms with no base code, it is not recommended.

7.7.1 Kinect, Primesense and OpenNI

The Kinect and the related Primesense camera are both small cheap cameras which produce both color and depth information. Neither works in sunlight. The Kinect, as provided by Microsoft, has associated tools in Windows, but is also available through PCL and OpenNI [38, 43]. More recently, both systems have dropped Kinect support due to licensing and driver issue, so only previous version support the Kinect. The Primesense camera is essentially the same, and while it costs more, still works with the current version of OpenNI. Neither OpenNI or PCL were difficult to work with, and in this simple system very few of their provided features were needed. OpenNI, unfortunately, has been purchased and removed from the web. While the system continues to be available in some repositories, the official version and support are no longer available. There may be additional open source systems released in the future, but at the moment the camera system used here will not be easy to reproduce.

7.7.2 Scikit Learn

Scikit Learn provides a huge variety of learning algorithms, as well as frameworks for creating validation sets, performing grid search, and evaluating performance [39]. They also provide useful tutorials, discussion of choice of techniques, and links to the papers that describe each technique. The one area they do not currently provide for is Neural Networks, though they have added Restricted Boltzmann Machines, and intend to add more. There is also little attention paid to speed.

7.7.3 Cuda-Convnet

This framework provides a reasonable CUDA implementation of network layers, as well as a simple way to alter network descriptions. They provide a reasonable collection of node types, layer types, and allow tuning of parameters [28]. The layer types include fully connected, softmax, convolutional, pooling, logistic regression and sum of squares. This system provides compressed checkpoint saving, which is extremely useful when training large nets which take many hours or days to train. They rely on the concept of a data provider, which must be written by the user, which makes it easy to plug in data from any file type, or from a camera. Change made to the layers and costs require change in the CUDA code, which can be intimidating. It can be used to build non-convolutional networks, but does not support autoencoders. Unfortunately, it does not provide any early stopping methods. Early stopping is used to prevent a neural network from spending too long in training, causing the network to overfit to the training data. For further details, see [41]. It also relies on python wrappers to prepare the data, which can be slow when tasked with flattening 50,000 images to be sent to the GPU.

7.7.4 Caffe and Decaf

Decaf was presented as a feature extractor, and relied on python wrapper and a network structure nearly identical to that of convnet's ImageNet Challenge [13]. Caffe, however, is being actively developed as an all C++ implementation, with the wrappers as optional extensions provided for python and Matlab [24]. It contains more types of neurons and layers, and is intended to run quickly. Caffe provides for construction of standard networks and convolutional networks. It is not currently intended for support of autoencoders, but they are being added soon. However, it is not as flexible as Convnet, and has a more complex structure. While it is possible to add types of data input, it is more difficult, and the provided data format makes use of multiple shuffles of the data difficult. It also does not provide early stopping, or other

performance measures besides accuracy.

7.7.5 Theano

Theano is the most flexible system, providing the user with the ability to quickly define layers, nodes and training for any type of neural network [3]. It provides for defining the entire network, and the data, without the restrictions that exist in the other libraries. It can be difficult to follow, but is as flexible as necessary to define networks, backpropagation, regularization and early-stopping, and also has good documentation online, as well as tutorials. It is written in python, but this may not be much of a constraint on speed if implemented well. Theano deserves further investigation and use.

7.8 Summary

This chapter discussed the results of each of the prototypes presented in this thesis. While the best results were obtained from single classifiers which were trained for each label, the memory usage of the multiclass and multilabel neural networks was much lower, and the results have potential for improvement. Depth data did not contribute greatly to learning on this dataset, but may prove useful in larger more complex datasets. Of the libraries used, Theano is recommended for further use, while the others need further development before they are considered again.

Multilabel classification may be improved with better cost functions, while general neural network performance may be improved with better data, further regularization, and better hyper-parameter setting. These changes should improve the results presented in previous chapters.

Chapter 8

Conclusion

This thesis has shown that attributes can be predicted reasonably well with several different techniques. While not all attributes are predicted well, those which fail are poorly represented in the data, or are not visible, and so impossible to learn from image data. Several different systems were created and used to explore possible data representations and classification techniques. The behavior varied, but each showed the ability on both generated and real data to predict labels drawn from descriptive sentences which include colors, shapes, and locations. The results led to recommendations for changes in the structure of the classifiers, presentation of the data and the use of certain types of feature extractors and classifiers.

This thesis shows the ability of several systems to predict attributes on a small RGBD dataset. Similar systems limit the learnable attributes to be disjoint, or from a much smaller space, while this system allows for multiple labels from the same classes, such as multiple colors or multiple shapes. While the performance is best with separate classifiers for each label, it is also shown that reasonable results can be attained with more compact multiclass and multilabel systems, which may then be improved upon.

The use of negative examples was shown to improve results in some cases, while others performed well when provided with only positive examples. Normalization of the input data proved to be a necessary step, either with standard scaling, mean

subtraction, or whitening. Depth data was somewhat useful in predicting certain labels, although it can also increase training time and complexity of the classifier, as in the case of neural networks which must be increased in size, both to accommodate the extra channel, and to learn additional features specific to the depth.

Speed of a simplified prediction system is shown to be near real time, and may be improved in several ways. More complicated systems, such as the multilabel prediction system, should not significantly decrease this speed. There is a limit on how fast the system may perform, in part due to the time required to transfer data to the GPU, though this can be partially masked with pipelining.

Additional work must be done to improve the performance, and it is quite likely that better data and altered loss functions are necessary, especially for multilabel prediction. To simplify the discovery of good structures for neural networks, automated search of hyperparameters should be investigated.

While it is clear that prediction of attributes is more difficult than object recognition, the techniques used here show good performance, and changes for further gains are possible. As the data presented becomes more complicated, in the sense of more attributes and more cluttered images, additional improvements and extensions to the system will be necessary. When these techniques are improved in predictive power, speed, and space, they become more feasible on mobile systems that function in complicated settings, bringing the system closer to achieving real autonomy.

Appendix A

Terms

Attribute A label describing an object rather then naming the object. These include colors, shapes and positions. This is not the same as feature.

Autoencoder A type of unsupervised learning, which can reduce dimensionality. A network where the inner layers are smaller then the input and output. They are trained to learn the identity, creating encoding and decoding layers in the network.

Classification Prediction of category labels.

Cokriging A specific subtopic in Gaussian Processes that handles multitask learning. See Gaussian Process citation.

Convolution A convolution is an integral that expresses the amount of overlap of one function g as it is shifted over another function f. It therefore "blends" one function with another. In the case of neural networks, this is a 2D function.

Feature A piece of data representing a sample, this could be an image, a pixel, a color value, or a learned value created by feature extraction. This is not the same as attribute.

Feature Extraction The process of reformatting data by dimensionality reduction

with a goal to maintain some measure of information. Techniques include, PCA, ICA, and Isomap.

Gaussian Process A collection of random variables, any finite number of which have a joint Gaussian distribution. Generally used to represent hyper-parameters, but has applications in multitask learning as well. May have limitations based on the number of outputs. See citation for better information: [42]

Ground Truth The known label(s) for a sample.

Isomap Isomap is a manifold learning technique for dimensionality reduction, which projects the data into a smaller dimensional space while preserving geodesic distance.

Kriging Gaussian Process prediction or regression, specifically in geostatistics.

Multiclass, multi-class This refers to classifiers which are meant to predict one of a number of classes (two or more). This may be used to describe multilabel problems as well in other literature, but in this work we use it for multiple classes, but only one expected true label.

Multilabel, multi-label This refers to classifiers which are meant to predict one or more of a number of classes. In other works, it may either mean multiclass or multitask, which is endlessly confusing. See multiclass and multitask for confusion.

Multitask, multi-task This term is used in literature to refer to systems that predict multiple targets. Again, this may refer to a problem that expects one positive prediction or several at a time. See multiclass and multilabel.

Neural Network (Artificial Neural Network) Any of the family of artificial networks a computer system modeled on the human brain and nervous system. Generally, shown as layered directed graphs with weighted edges and functions applied at nodes, which move from some input to output. These include ANN,

Perceptrons, Restricted Boltzmann Machines, Autoencoders, and Convolutional Networks.

Prediction Models continuous values, predicts unknown or missing values.

Support Vector Machine A maximum margin classifier that finds sets of hyperplanes in higher dimensional space to separate classes. The data is projected into higher dimensional inner product space, and calculations in this space are avoided by the kernel trick.

Appendix B

Figures

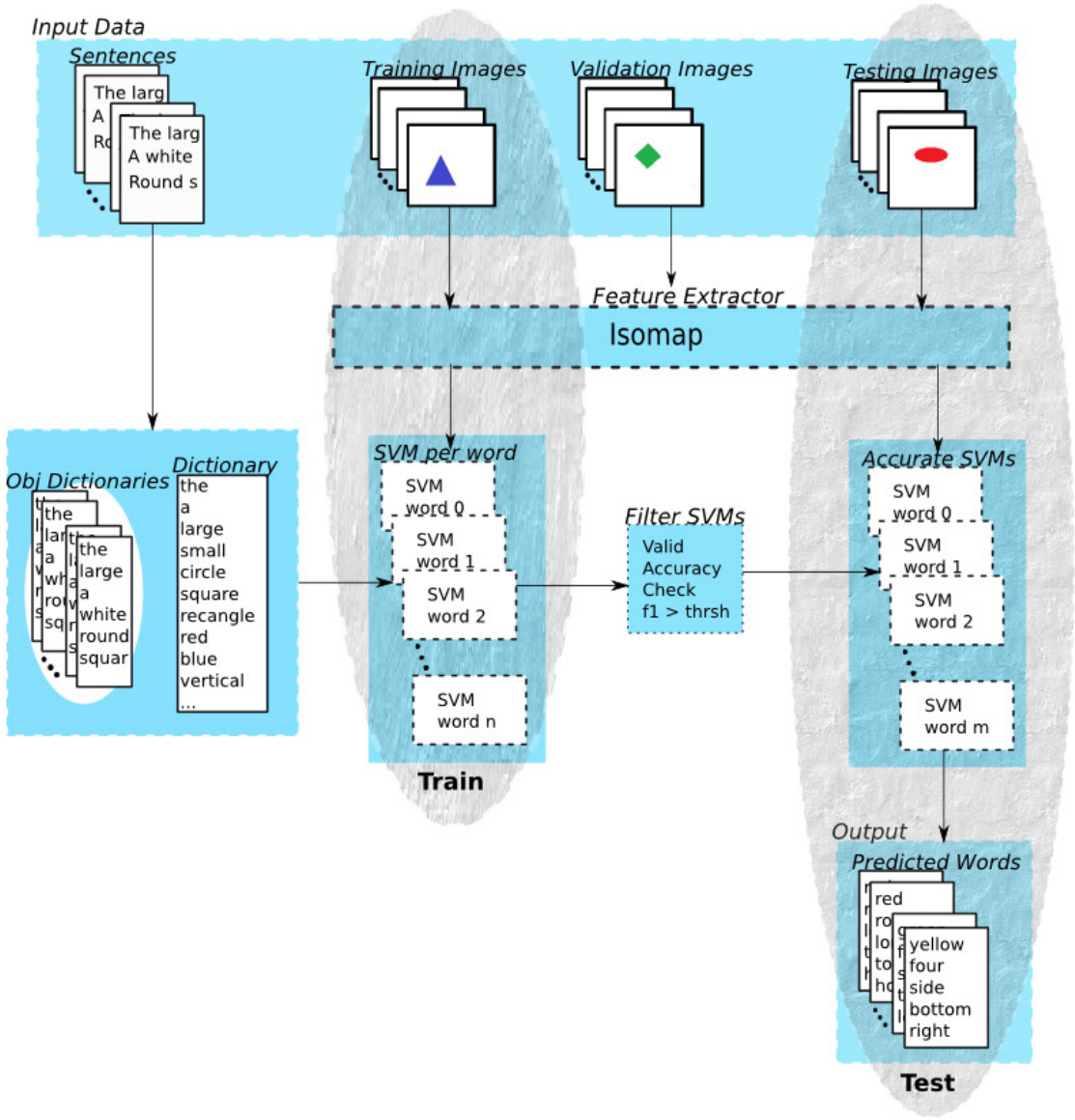


Figure B-1: Training and Testing Process

Appendix C

Tables

Table C.1: Generated Data Results: One Class SVM, Error Permitted 0.2, Degree 3, Kernel: radial

Word	Unknowns	Precision	Recall	F1
yellow	22	1.0	0.45	0.62
four	26	1.0	0.5	0.66
black	37	1.0	0.62	0.76
circle	37	1.0	0.54	0.70
triangle	38	1.0	0.52	0.68
square	21	1.0	0.33	0.5
blue	18	1.0	0.5	0.66
red	17	1.0	0.64	0.78

Table C.2: Generated Data Results with many unknowns: One Class SVM, Error Permitted 0.2, Degree 3, Kernel: radial

Word	Unknowns	Samples (128 Total)	Precision	Recall	F1
three	14	24	1.0	0.66	0.8
top	71	19	1.0	0.63	0.77
oval	67	27	1.0	0.55	0.71
right	66	19	1.0	0.47	0.64
corners	18	46	0.96	0.56	0.71
round	32	34	1.0	0.55	0.71
upper	77	16	1.0	0.5	0.66
shape	51	77	1.0	0.74	0.85

Table C.3: Generated Data Results: 128 total samples, One Class SVM, permitted error of 0.2, degree 3, radial basis kernel.

Word	Samples	Precision	Recall	F1
yellow	22	1.0	0.36	0.53
four	26	1.0	0.57	0.73
black	37	1.0	0.56	0.72
circle	37	1.0	0.62	0.76
triangle	38	1.0	0.52	0.68
square	21	1.0	0.66	0.8
blue	18	1.0	0.33	0.5
red	17	1.0	0.41	0.58

Table C.4: Real Data Results: 128 total samples, One Class SVM, permitted error of 0.3, degree 3, radial basis kernel.

Word	Samples	Precision	Recall	F1
yellow	22	1.0	0.64	0.78
black	34	1.0	0.64	0.78
blue	35	1.0	0.74	0.85
purple	10	1.0	0.6	0.75
red	60	0.83	0.73	0.77
white	36	1.0	0.58	0.73

Table C.5: Kinect Data Results NuSVC RBF Kernel for black. Nu is gradually increased.

Nu	Precision	Recall	F1
0.1	0.49	0.55	0.52
0.2	0.54	0.68	0.60
0.3	0.45	0.48	0.47
0.4	0.54	0.68	0.60
0.5	0.48	0.48	0.48

Table C.6: Kinect Data Results NuSVC Polynomial Kernel for "black". Nu and Degree are varied.

Degree	Nu	Precision	Recall	F1
2	0.1	1.0	0.55	0.71
3	0.1	1.0	0.48	0.65
4	0.1	1.0	0.45	0.62
1	0.2	0.95	0.65	0.77
2	0.2	1.0	0.45	0.62
3	0.2	1.0	0.45	0.62
4	0.2	1.0	0.45	0.62
1	0.3	0.95	0.68	0.79
2	0.3	1.0	0.52	0.68
3	0.3	1.0	0.48	0.65
4	0.3	1.0	0.39	0.56
1	0.4	0.95	0.58	0.72
2	0.4	1.0	0.52	0.68
3	0.4	1.0	0.35	0.52
4	0.4	1.0	0.48	0.65
1	0.5	1.0	0.52	0.68
2	0.5	1.0	0.35	0.52
3	0.5	1.0	0.39	0.56
4	0.5	1.0	0.42	0.59

Table C.7: Kinect Data Results One Class SVM for “blue”.

Kernel	Error	Degree	Accuracy	Precision	Recall	Specificty	F1
RBF	0.2	2	0.97	1.0	0.86	1.0	0.92
RBF	0.2	3	0.97	1.0	0.86	1.0	0.92
RBF	0.2	4	0.97	1.0	0.86	1.0	0.92
RBF	0.3	2	0.82	1.0	0.14	1.0	0.25
RBF	0.3	3	0.82	1.0	0.14	1.0	0.25
RBF	0.3	4	0.82	1.0	0.14	1.0	0.25
RBF	0.4	2	0.94	1.0	0.71	1.0	0.83
RBF	0.4	3	0.94	1.0	0.71	1.0	0.83
RBF	0.4	4	0.94	1.0	0.71	1.0	0.83
RBF	0.5	2	0.79	0.0	0.0	1.0	0.0
RBF	0.5	3	0.79	0.0	0.0	1.0	0.0
RBF	0.5	4	0.79	0.0	0.0	1.0	0.0
POLY	0.1	1	0.7	0.4	0.86	0.65	0.55
POLY	0.1	2	0.55	0.28	0.71	0.5	0.4
POLY	0.1	3	0.73	0.43	0.75	0.73	0.55

Table C.8: Kinect Data Results One Class SVM for “dark”.

Kernel	Error	Degree	Accuracy	Precision	Recall	Specificty	F1
RBF	0.2	2	0.8	1.0	0.8	0.0	0.89
RBF	0.2	3	0.8	1.0	0.8	0.0	0.89
RBF	0.2	4	0.8	1.0	0.8	0.0	0.89
RBF	0.3	2	0.2	1.0	0.2	0.0	0.33
RBF	0.3	3	0.2	1.0	0.2	0.0	0.33
RBF	0.3	4	0.2	1.0	0.2	0.0	0.33
RBF	0.4	2	0.6	1.0	0.6	0.0	0.75
RBF	0.4	3	0.6	1.0	0.6	0.0	0.75
RBF	0.4	4	0.6	1.0	0.6	0.0	0.75
RBF	0.5	2	0.12	1.0	0.12	0.0	0.21
RBF	0.5	3	0.12	1.0	0.12	0.0	0.21
RBF	0.5	4	0.12	1.0	0.12	0.0	0.21
PLY	0.1	1	0.92	1.0	0.92	0.0	0.96
PLY	0.1	2	0.92	1.0	0.92	0.0	0.96
PLY	0.1	3	0.88	1.0	0.88	0.0	0.94

Table C.9: Kinect Data Results One Class SVM for "green".

Kernel	Error	Degree	Accuracy	Precision	Recall	Specificty	F1
RBF	0.2	2	0.97	1.0	0.81	1.0	0.89
RBF	0.2	3	0.97	1.0	0.81	1.0	0.89
RBF	0.2	4	0.97	1.0	0.81	1.0	0.89
RBF	0.3	2	0.87	1.0	0.19	1.0	0.32
RBF	0.3	3	0.87	1.0	0.19	1.0	0.32
RBF	0.3	4	0.87	1.0	0.19	1.0	0.32
RBF	0.4	2	0.94	1.0	0.62	1.0	0.76
RBF	0.4	3	0.94	1.0	0.62	1.0	0.76
RBF	0.4	4	0.94	1.0	0.62	1.0	0.76
RBF	0.5	2	0.83	0.0	0.0	1.0	0.0
RBF	0.5	3	0.83	0.0	0.0	1.0	0.0
RBF	0.5	4	0.83	0.0	0.0	1.0	0.0
PLY	0.1	1	0.81	0.46	0.86	0.8	0.6
PLY	0.1	2	0.62	0.27	0.76	0.59	0.4

Table C.10: Best Kinect Data Results One Class SVM

Word	Kernel	Error	Degree	Acc.	Precision	Recall	Spec.	F1
black	RBF	0.2	2	0.95	1.0	0.8	1.0	0.89
blue	RBF	0.2	2	0.97	1.0	0.86	1.0	0.92
box	PLY	0.1	1	0.91	1.0	0.91	0.0	0.95
dark	PLY	0.1	1	0.92	1.0	0.92	0.0	0.96
green	RBF	0.2	2	0.97	1.0	0.81	1.0	0.89
grey	PLY	0.1	1	0.92	1.0	0.92	0.0	0.96
light	PLY	0.1	2	0.93	1.0	0.93	0.0	0.97
orange	RBF	0.2	4	0.98	1.0	0.81	1.0	0.90
red	RBF	0.2	2	0.9	1.0	0.8	1.0	0.89
round	PLY	0.1	2	0.83	1.0	0.83	0.0	0.90
silver	RBF	0.2	3	0.89	1.0	0.86	0.0	0.94
stripe	POLY	0.1	2	0.87	1.0	0.87	0.0	0.93
white	RBF	0.2	4	0.94	1.0	0.8	1.0	0.89
writing	PLY	0.1	3	0.94	1.0	0.83	1.0	0.91
yellow	RBF	0.2	3	0.96	1.0	0.85	1.0	0.92

Bibliography

[1] Bert Bakker and Tom Heskes. Model clustering for neural network ensembles. *Springer Lecture Notes in Computer Science*, LNCS 2415:p. 383 f, 2002.

[2] Yoshua Bengio and Yann Lecun. *Scaling Learning Algorithms towards AI*. MIT Press, 2007.

[3] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[4] Paul Bloom. *How Children Learn the Meaning of Words*. MIT Press, 2002.

[5] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *IEEE International Robotics and Automation (ICRA)*, May 2011.

[6] Evgeny Byvatov, Uli Fechner, Jens Sadowski, and Gisbert Schneider. Comparison of support vector machine and artificial neural network systems for drug/nondrug classification. *Journal of Chemical Information and Computer Sciences*, 43(6):1882–1889, 2003.

[7] Adam Coates, Honglak Lee, and Andrew Y. Ng. An analysis of single-layer networks in unsupervised feature learning. In *Advances in Neural Information Processing Systems*, 2010.

[8] Corinna Cortes and Vladimir Vapnik. Support-vector networks. In *Machine Learning*, pages 273–297, 1995.

[9] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In Cordelia Schmid, Stefano Soatto, and Carlo Tomasi, editors, *International Conference on Computer Vision & Pattern Recognition*, volume 2, pages 886–893, INRIA Rhône-Alpes, ZIRST-655, av. de l'Europe, Montbonnot-38334, June 2005.

[10] LISA lab Deep Learning: Copyright 2008-2010. Convolutional neural networks (lenet), 2010.

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

[12] Sander Dieleman. My solution for the galaxy zoo challenge. http://benanne.github.io/2014/04/05/galaxy-zoo.html.

[13] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *arXiv preprint arXiv:1310.1531*, 2013.

[14] Wei Gao and Zhi-Hua Zhou. On the consistency of multi-label learning. In Sham M. Kakade and Ulrike von Luxburg, editors, *COLT*, volume 19 of *JMLR Proceedings*, pages 341–358. JMLR.org, 2011.

[15] Yunchao Gong, Yangqing Jia, Thomas Leung, Alexander Toshev, and Sergey Ioffe. Deep convolutional ranking for multilabel image annotation. *CoRR*, abs/1312.4894, 2013.

[16] Ian J. Goodfellow. Technical report: Multidimensional, downsampled convolution for autoencoders. Technical report, Université de Montréal, 2010.

[17] Rafal Grodzicki, Jacek Mandziuk, and Lipo Wang. Improved multilabel classification with neural networks. In Gnter Rudolph, Thomas Jansen, Simon M. Lucas, Carlo Poloni, and Nicola Beume, editors, *PPSN*, volume 5199 of *Lecture Notes in Computer Science*, pages 409–416. Springer, 2008.

[18] hiit. Multilayer perceptrons, 2001.

[19] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[20] Jessica Horst, Lisa Oakes, and Kelly Madole. What does it look like and what can it do? category structure influences how infants categorize. *Child Development*, 76(3):614–631, May/June 2005.

[21] Fu Jie Huang and Y. LeCun. Large-scale learning with svm and convolutional for generic object categorization. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 284–291, June.

[22] A Hyvrinen and E Oja. Independent component analysis: algorithms and applications. *Neural Networks: The Official Journal of the International Neural Network Society*, 13(4-5):411–430, june 2000. PMID: 10946390.

[23] Panagiotis G. Ipeirotis. Analyzing the amazon mechanical turk marketplace. *XRDS*, 17(2):16–21, December 2010.

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

[25] I.T. Jolliffe. *Principal Component Analysis*. Springer Verlag, 1986.

[26] Rigas Kouskouridas, Efthimios Badekas, and Antonios Gasteratos. Simultaneous visual object recognition and position estimation using sift. In Ming Xie, Youlun Xiong, Caihua Xiong, Honghai Liu, and Zhencheng Hu, editors, *ICIRA*, volume 5928 of *Lecture Notes in Computer Science*, pages 866–875. Springer, 2009.

[27] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

[29] Kevin Lai, Liefeng Bo, Xaiofeng Ren, and Dieter Fox. Rgb-d (kinect) object database. http://www.cs.washington.edu/rgbd-dataset/index.html, 2012.

[30] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. A large-scale hierarchical multi-view rgb-d object dataset. In *ICRA*, pages 1817–1824. IEEE, 2011.

[31] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/, 2010.

[32] Shaoxin Li, Shiguang Shan, and Xilin Chen. Relative forest for attribute prediction. In *ACCV'12 Proceedings of the 11th Asian conference on Computer Vision - Volume Part I*, 2013.

[33] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[34] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International Conference on Artificial Neural Networks*, 2011.

[35] Warren S. McCulloch and Walter Pitts. Neurocomputing: Foundations of research. chapter A Logical Calculus of the Ideas Immanent in Nervous Activity, pages 15–27. MIT Press, Cambridge, MA, USA, 1988.

[36] S. Mukkamala, G. Janoski, and A. Sung. Intrusion detection using neural networks and support vector machines. In *Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on*, volume 2, pages 1702 –1707, 2002.

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

[38] OpenNI organization. *OpenNI User Guide*. OpenNI organization, November 2010.

[39] F. Pedregosa, G. Varoquaux, A. Granfort, V. Michel, B. Thirion, O. Grisel, Blondel M., P. Prettenhoffer, and et. al. Scikit-learn: machine learning in python. *Journal of Machine Learning Research*, 12:2825–283, 2011.

[40] Justus Piater and Roderic Grupen. Constructive feature learning and the development of visual expertise. In *Proc., Intl. Conf. on Machine Learning*, Stanford, CA, June 29 – July 2 2000. Morgan Kaufmann.

[41] Lutz Prechelt. Early stopping - but when? In *Neural Networks: Tricks of the Trade, volume 1524 of LNCS, chapter 2*, pages 55–69. Springer-Verlag, 1997.

[42] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[43] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *International Conference on Robotics and Automation*, Shanghai, China, 2011 2011.

[44] Louise Scott, Lucila Carvalho, D. Ross Jeffery, and John D'Ambra. An evaluation of the spearmint approach to software process modelling. In Vincenzo Ambriola, editor, *EWSPT*, volume 2077 of *Lecture Notes in Computer Science*, pages 77–89. Springer, 2001.

[45] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229, 2013.

[46] Yuyin Sun, Liefeng Bo, and Dieter Fox. Attribute based object identification. In *ICRA*, pages 2096–2103. IEEE, 2013.

[47] R. Tamburo. Rgb image color space transformations. 12 2010.

[48] J. B. Tenebaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–1323, 2000.

[49] D Thukaram, H P Khincha, and V H Pakka. Artificial neural network and support vector machine approach for locating faults in radial distribution systems. *IEEE Transactions on Power Delivery*, 20(2):710–721, April 2005.

[50] Liesl Wigand, Monica N. Nicolescu, and Mircea Nicolescu. A developmental approach to concept learning. In Jean-Louis Ferrier, Oleg Yu. Gusikhin, Kurosh Madani, and Jurek Z. Sasiadek, editors, *ICINCO (2)*, pages 337–344. SciTePress, 2013.

[51] Wikipedia. Colored_neural_network.svg — Wikipedia, the free encyclopedia, 2004. [Online; accessed 22-July-2004].

[52] Konrad Kording Wolfgang Einhauser, Christoph Kayser and Peter Konig. Learning multiple feature representations from natural image sequences. In *Proc., Intl. Conf. on Artificial Neural Networks*, 2002.