Evaluation of a Zero-Shot Cross-Modal Image Retrieval Technique using Ranking Support Vector Machines

By

EVAN NOVOTNY

Bachelor of Science in Electrical Engineering

Oklahoma State University

Stillwater, Oklahoma

2013

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
Master of Science
May, 2016

EVALUATION OF A ZERO-SHOT CROSS-MODAL IMAGE RETRIEVAL TECHNIQUE USING RANKING SUPPORT VECTOR MACHINES

Thesis approved:

Dr. George Scheets

Thesis Advisor

Dr. Keith Teague

Dr. Guoliang Fan

Acknowledgements

Firstly, I would like to express my gratitude to my thesis advisor, Dr. Scheets, for all of his support, wisdom, and guidance, throughout the entire thesis process.

To the members of my graduate committee: Dr. Scheets, Dr. Teague, and Dr. Fan - thank you for taking the time to be on my committee and providing so much knowledge in all of your courses. I also want to thank all of the professors at Oklahoma State University that I had the privilege to learn from.

A special thanks to Dr. Brady and Dr. Dagli at MIT Lincoln Laboratory who provided my the opportunity to join their team as an intern, introduced my to this research topic, and allowed me to continue working on the project after leaving. This paper could have never happened without your help. I especially want to thank Davis King for his pioneering work on this topic while at Lincoln Laboratory, which provided the opportunity for this research.

Lastly, I want to thank my friends and family for all of their support throughout my entire college career.

Name: Evan Novotny

Date of Degree: MAY, 2016

Title of Study: EVALUATION OF A ZERO-SHOT CROSS-MODAL IMAGE RETRIEVAL TECHNIQUE USING RANKING SUPPORT VECTOR MACHINES

Major Field: Electrical Engineering

Abstract:

The aim of this thesis is to evaluate the potential of a technique for cross-modal label based retrieval of images, from previously unseen classes, using a ranking support vector machine and investigate attributes of the system that effect performance.

Ranked retrieval performance of the method was measured using mean Average Precision and precision-recall. Monte Carlo simulations were used to compare against random chance retrieval and provide a baseline for performance. Commonly used and freely available datasets were used for training and performance evaluations. Several training attributes and the learned weights of the system were analyzed to better understand their effects on performance and characterize the method.

While comparison to other zero-shot methods are difficult to draw the results were comparable with the simulations and one similar technique evaluated on seen classes. The testing and simulations showed that the method is capable of better than random retrieval of unknown classes from a large database and has superior performance on retrieving seen classes than a recent comparable technique. Investigations into different attributes of the system, particularly training sizes, indicates that the method may benefit greatly from using a large amount of data for training.

Contents

List of Figures

List of Tables

List of Abbreviations

SVM:	**S**upport **V**ector **M**achine
RSVM:	**R**anking **S**upport **V**ector **M**achine
SSVM:	**S**tructural **S**upport **V**ector **M**achine
DNN:	**D**eep **N**eural **N**etwork
t-SNE:	**t**- Distributed **S**tochastic **N**eighbor **E**mbedding
MAP:	**M**ean **A**verage **P**recision
SIFT:	**S**cale - **I**nvariant **F**eature **T**ransform

Chapter I

Introduction

This paper describes the development and implementation of an image retrieval method using a Ranking Support Vector Machine (RSVM) to retrieve images from a database based solely on their similarity to a textual query. More specifically this paper evaluates the potential of this model for zero-shot retrieval, which is the retrieval of images belonging to a class that the model has not been trained on. Chapter I discusses the need for a system that can generalize to unknown classes and the method for doing so is introduced. Chapter II reviews some of the literature in the areas of image and zero-shot classification that lay the groundwork for this method. Chapter III details the development of the proposed method and the training of the RSVM used for retrieval. In Chapter IV the results are analyzed. Chapter V concludes the paper and discusses future work.

1.1 Background

In classic supervised machine learning a large labeled dataset with n classes is used to train a n-way classifier. While this can obtain excellent results in image classification by using a large number of visual classes in a neural network [12],[7],

having a discrete number of classes makes classifying a class that was neglected from training intractable. This problem cannot be solved by simply giving the classifier more classes because if a class is not present at training time then the classifier will have zero knowledge of the missing class to use in order to make a reasonable attempt at classification.

There are many situations where the ability to make a zero-shot attempt is useful. Sometimes it is difficult to obtain enough labeled training data for a specific class or it may not be possible to obtain any training data at all. There may also be the case where new classes are constantly being introduced, such as new products or new models of existing items; in these cases having zero-shot capabilities is very appealing.

In the case of a multi-media analyst, using a classifier with a rigid set of classes limits their flexibility when searching a database. The analyst would need to be aware of what classes could be searched for to make effective use of the classifier. However, having zero-shot capabilities would mean being able to have more classes available than the classifier had previously trained on and that new classes could be added to the classifier without having to retrain on new data.

1.2 Knowledge Bases

In order for a system to make a zero-shot attempt it needs to be able to draw knowledge from somewhere. Recent works have looked to using language as a source of knowledge to classify visual classes. Research in natural language processing has

found a method to create a vector space that embeds the semantic relationship between words in a language into the space by using unsupervised training methods on massive amounts of freely available text articles, such as Google news articles, from the web.

One such system, Word2Vec, implements both continuous bag-of-words (CBoW) [18] and skip-gram [17] architectures to learn vector representations of individual words in the English language. The CBoW learns to represent words by learning to predicting a word from the context of the words around it, so the system would take the words $w_{i-1}, w_{i-2}, w_{i+1}, w_{i+2}$ around w_i as input and then predict the word w_i. Skip-gram operates in the reverse, and tries to predict the context words $w_{i-1}, w_{i-2}, w_{i+1}, w_{i+2}$ around w_i. These embedded word spaces often contain millions of words that cluster in semantically similar groups while keeping dissimilar groups distant from each other in the space. For example words related to animals such as 'dog' and 'cat' would be found nearby to one another while words representing vehicles like 'truck' and 'car' would cluster close together but separately from the group of animal words. Because of this semantic embedding if one were given the coordinates of a single unknown word in this vector space it would be possible to discover semantic information about this mystery word just by observing the words nearest to it. Recent works [23], [4], [26], [20] have proposed mapping images and words into a common spaces so that pictures can be automatically annotated or identified. Figure 1.1 shows visualization of this process.

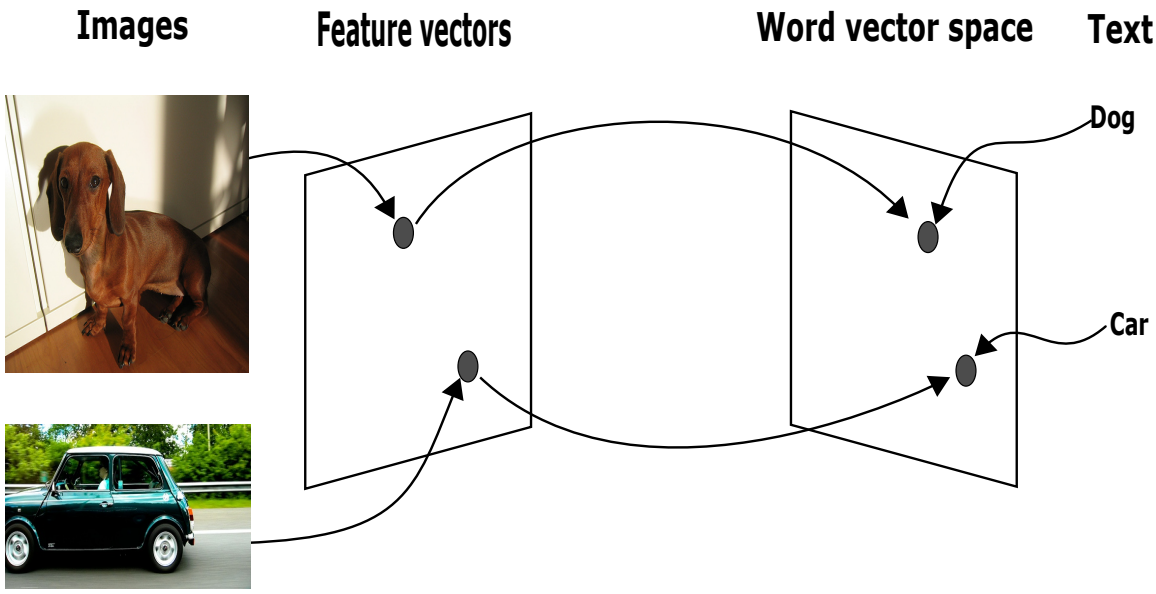


FIGURE 1.1: Images feature vectors being extracted and projected into a word-space

1.3 Initial Work

Previous work at Lincoln Laboratory by Davis King showed that it is possible to directly compare the similarity of images and words for image retrieval by mapping the image feature vectors into this rich semantic embedding space with a transform matrix learned from a multi-modal structural SVM. This transform matrix would map images close to corresponding words in the embedded space; for example dog images would map closely to the word 'dog' like in figure 1.1, allowing for images to be retrieved from free text queries based on the vector similarity of the transformed images and the search text. Interestingly he found that this method was capable of making reasonable selections from combinations of words, such as retrieving mostly buses and trains when given the 'public '+'transport' as

a query. This paper builds on the work of king and investigates this model's ability to retrieve zero-shot images by using the similarity of an image from an unknown class mapped into this embedded space with that of known words already present in the space.

Chapter II

Previous Works

Zero-shot learning is a topic that has gained a lot of attention in recent years; in particular zero-shot classification of images has become a popular topic of research in computer vision as image classification progresses. While deep neural networks are still the best in image recognition [7] there is a clear interest from researchers in developing models that are capable of generalizing to unknown classes. The ability to recognize a new class can be critical in situations where it is not possible to obtain training data or when there is an ever growing number of classes. In any other case it is a welcomed property that enhances the capabilities of a typical classifier. The following section will discuss previous work in the area of zero-shot learning and other works that were used in the method introduced by this thesis.

2.1 Zero-Shot learning

In [20] the authors describe as general approach to zero-shot learning using a semantic knowledge base to predict novel classes that were not present during the training of a classifier. Rather than trying to learn a function $f : X^d \rightarrow Y$ to assign raw input data X^d directly to an output label Y, the authors learn a mapping to

a p dimensional intermediate space constructed from a semantic knowledge base. This intermediate space encodes each of the p dimensions with the value of a different semantic property; there also exist a one-to-one mapping of each p dimensional vector in the space to a single output label. So the goal of the classifier is to map input data into the semantic space and then find the nearest output label in the space. That is if the function $f : X^d \rightarrow Y$ is separated into two functions S and L then:

$$S : X^d \rightarrow I^p$$

$$L : I^p \rightarrow Y$$

Where S maps the raw input X^d to a p dimensional vector I^p in the semantic space, then the L function finds the nearest class label Y to the semantic vector. For zero-shot learning this means that the classifier can first learn to map input data to a continuous semantic space, that way when data for a new class is input it can be mapped to a new position in the space. If this mapping is close to the true encoding of the class label then the L map in the second stage has a reasonable chance of recovering the correct class label. The authors note that for this method the knowledge base K must know the encodings of many more class labels in the semantic space than the number of classes used for training. So typically $M >> N$, where M is the number of semantic vectors to class label pairs known by the knowledge base and N is the number of classes present for the training of the classifier.

Recently [23] implemented a zero-shot learning method that builds off of [20] using a neural network to project image feature vectors into a semantic word space,

learned from the distribution of words in a text corpus [8], and use the known words in this space for a knowledge base for classification. The zero-shot method in [23] extends [20] by using a novelty detector to first differentiate between known and unknown classes and then applying different classifiers depending on the result. The authors experimented on the CIFAR-10 dataset, which contains images of ten different classes of animals and vehicles. Zero-shot performance was measured by withholding two classes at a time from training. As a result of different combinations of holdout sets it was discovered that in order to be able to make a reasonable attempt at classifying novel data the classifier needed to be trained with at least one class semantically similar to that of the unknown class. For example, holding out the 'car' and 'dog' class will lead to good zero-shot results, but if 'cat' and 'dog' are both held out zero-shot performance suffers because there is not a similar enough class for each of the two held out for the classifier to make a distinction between the two. However, the use of two different classifiers is cumbersome; preferably a zero-shot classifier would work with both known and unknown classes without having to apply a different model before classifying.

The choice of semantic space can be crucial to performance as it is where inferences are drawn for zero-shot classifications. A popular choice is to use attribute based label embedding [1], where each dimension of the semantic space represents a different attribute, such as *has a tail* or *has feathers*. Another choice is to use an automatically learned semantic space where the dimensions of the feature vectors do not necessarily have a human understandable meaning. One of these learned spaces, Word2Vec, uses a shallow neural network to learn the semantic relationships between words by training on a massive amount of text data, such a Internet news articles. The result is a vector space where each of the p dimensional vectors

in the space represents a single word. This can create a knowledge base of millions of labels, where words cluster in groups based on learned semantic relationships. Word2Vec can be implemented with two different architectures, the original continuous-bag-of-words version [18], and a slightly newer skip-gram version [17] which is more effective for large amounts of training data. The effectiveness of this knowledge base can be seen in both [4] and [19] where it is used to perform zero-shot automatic image annotation. [4] uses a similar method as [23] but at a much larger scale (1000 classes instead of 10) and with a knowledge base created by the previously mentioned Word2Vec. They also eliminate the need for two separate classifiers by using a neural network to directly predict the semantic embedding vectors for an input image. More recently [19] uses the same knowledge base for the same task but rather than having a neural network predict the embedding vectors they obtain the vectors for the class labels of an existing n-way classifier and then map new images into the semantic space using a convex combination of the embedding vectors for the class labels for that image. The convex combination in this case is just the vector addition of the scaled embedding vectors, such that the sum of the elements in the new vector is 1. More specifically, they employ a n-way neural network classifier that finds the probability of an image belonging to any of the n classes and then sums the embedding vectors for each class scaled by the probability of the image belonging to that class for the T classes with the highest probabilities. The image is then automatically annotated by collecting several of the closest words in the embedded space. This method is beautifully simple and makes effective use of existing models and tools, but it requires that the feature vector used for images has labels connected to each dimension of the vector and would be unsuitable for more general image feature vectors where the features

may not have a label with a corresponding embedding vector.

The method investigated in this thesis is most similar to [26] in which the author uses a structural SVM as described in [24] to learn a bilateral association between images and text. Text and images are represented using 'unimodal probability distributions of topics learned using latent Dirichlet allocation'. The structural SVM is trained on a joint text-image representation, $\Phi(x, y)$ where x is the image feature vector, y is the word embedding vector, and $\Phi = x \otimes y$, which is the tensor product of the image feature vector and the embedding vector. They also consider two loss functions, the Manhattan distance Δ_M and the Euclidean distance Δ_E. Once a weight vector w is learned images can be retrieved from a database by sorting the scores of $f(x, y|w) = w \cdot \Phi(x, y)$ or, in the case of text$f(y, x|w)$, where the higher the score the more relevant the image or text. Our approach differs from the previous in the following ways, firstly [26] does not consider the zero-shot application which is the primary concern for this thesis, secondly a freely available knowledge base trained on the Google news dataset using the method described in [17] is used, and finally because the final objective is to rank images or text based on relevance this thesis implements a ranking SVM, which is a variation of the structural SVM, as described in [9].

This section only covers a few of the key works on zero-shot learning, if the reader is interested they are encouraged to review the following papers: [22, 14, 13].

Chapter III

Methodology

The development, implementation, and testing setup used in this thesis are discussed in the next section. First the development of the original method created at Lincoln Laboratory that gave rise to this thesis is introduced.

3.1 Objective

While zero-shot retrieval is the attribute investigated in this paper, the primary goal of the original system was to project words and images into a space where they are directly comparable. This multi-modal approach allows for images and text to be used interchangeably to search directly for each other, e.g. an image could be used to find the words most relative to it, or a text query could be used to find an image most like the words in the query. The intuition behind using this method for a zero-shot attempt is that, if words can be directly used to retrieve images with relatively high accuracy then it should be possible to use a large collection of known words to describe a class that was not trained on and to retrieve an image belonging to this unknown class.

To reduce training time, increase ease of use, and ease of replication, freely available models were used for word representation [17] and image feature extraction [25]. A publicly available machine learning library [11] was also used for the RSVM implementation.

3.2 Development

Three versions of the system were developed; the first by Lincoln Lab personnel, specifically King, and the final two by the author. First, once while employed with Lincoln Lab and finally, here at Oklahoma State University. As each iteration helped lead to the final version used in this thesis, the first two iterations will be briefly discussed to show the full development path.

3.2.1 King's System

The zero-shot retrieval method examined in this thesis is an extension of the multi-modal image retrieval method originally created at Lincoln Laboratory, with the multi-modal aspect coming from the use of textual words to label images. Previous work by King at Lincoln Laboratory on using words as noisy labels for images lead to the development of a method to retrieve images based on their likeness to a text search.

If $\vec{w}$ is taken to be a vector representing a word or phrase and $\vec{i}$ is another vector that represents some image then the goal is to find a mapping M such that the image vector is mapped to a point in the word-space, which is better defined and easier to search within than the image-space, this can be written as, $M : \vec{i} \rightarrow \vec{w}_i$.

Where $\vec{w}_i$ is a new point in the word-space where the image $\vec{i}$ is mapped to. The search word or phrase $\vec{w}$ can then be compared with the mapped image-word $\vec{w}_i$, and if the two points are close enough then the image is considered to be similar to the search word or phrase, with a likeness based on the cosine similarity of the two points in the word-space.

The finding of the image-word vector $\vec{w}_i$ can be done with a simple matrix multiplication given by:

$$\vec{w}_i = \vec{i} \cdot M$$

Where the image vector $\vec{i}$ is projected into the word space by the mapping M. The matrix M can't be directly calculated, but a machine learning algorithm can be used to determine a suitable M from a set of images with tag words describing the image. Because images tagged by the same word can vary greatly, the words act as noisy labels to the images. Learning the mapping M can be seen as a supervised learning problem, where input images $\vec{i}$ are supplied with their noisy ground truth labels $\vec{w}$.

To solve for the mapping matrix M a Structural SVM (SSVM) was employed. A standard SVM is a supervised learning algorithm that learns to classify data by creating a set of hyperplanes that separates the data with the widest gap between classes as possible. However, a standard SVM can only give binary outputs, yes/no, as to whether a new data point belongs to a class, and the desired output space is a word embedding vector of real values, which is complex. A SSVM however is capable of predicting complex, or structured outputs [10], such as a real valued high dimensional vector.

3.2.2 First Recreation

Under the guidance of Dr. Dagli and Dr. Brady while working for Lincoln Laboratory, work done by the author of this thesis to recreate Kings' method revealed possible areas of improvement and further exploration. One such area was zero-shot applications. While King's method was effective at retrieving previously trained-on images using words or phrases its ability to retrieve an unknown class had not been investigated.

The first recreation of the system utilized a random projection hash, trained by King, with a dimension of 8192 for image features, and a 500 dimensional word vector representation trained on the Gigaword corpus [5], which is 'a comprehensive archive of newswire text data that has been acquired over several years by the Linguistic Data Consortium (LDC), at the University of Pennsylvania'. To reduce computational complexity the high dimensional random projection hash was replaced with a 1000 dimensional feature vector from the output layer of a deep neural network [25]. Each dimension of the output vector represented a probability of an image belonging to one of the 1000 classes. This provided a more meaningful feature vector and a large reduction in the size of data, which also meant a considerable reduction in training time. The SSVM was trained using the same code as the original iteration by King. At the time of this writing the SSVM training code and other software have not yet been disclosed from Lincoln Laboratory.

3.2.3 Final Recreation

With all software developed at Lincoln Lab confidential the multi-modal retrieval system had to once again be recreated. Because this iteration was starting from scratch it was desirable to make the retrieval system more accessible, and therefore easier to replicate. Only publicly available software and models were used.

The training for the retrieval system can be broken down into three major components: the image representation, the word representation, and the learning algorithm. All of the components used are freely available and easily changeable for different paths of investigation. Figure 3.1 shows a simplified flow-graph of the process. Images and text are converted into vector representations and then fed into the Ranking SVM to train the projection matrix S.

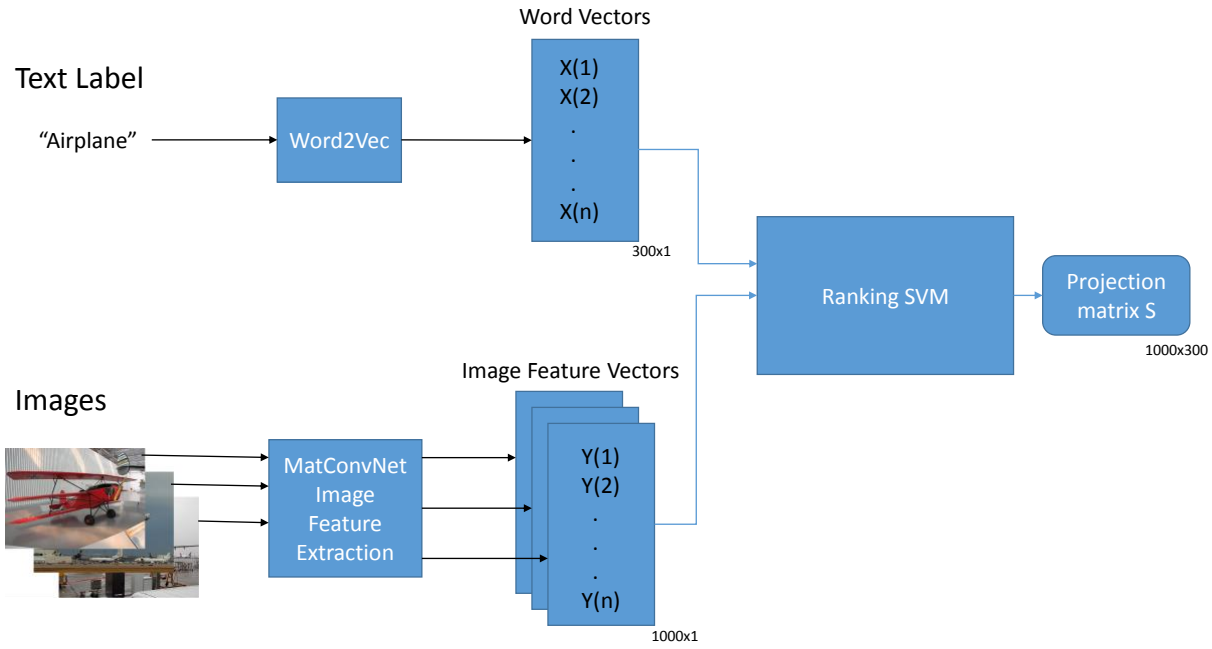


FIGURE 3.1: Flow graph of how images and text are fed into the ranking SVM

Image feature vectors were once again represented with a 1000 dimensional vector output from a deep neural network [25] with eight layers except with the final layer, the softmax layer, removed. The softmax function used for the final layer "squashes" the output to fit in the range of (0,1), using the following equation:

$$y_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ for } j = 1, \ldots, K$$

While the is a useful function for creating a probability distribution and selecting the T most probable outcomes, it leaves the majority of the values near zero. These zero values contribute no real information about the input image. The outputs from the final hidden layer are the features learned by the neural network that are normally fed into the softmax layer for classifying an image. These features are real valued and contain information that is lost from the softmax layer. This final part of the network is illustrated in figure 3.2.

Word vectors were represented with a freely available Word2Vec model from Google. The model learned by training a neural network to maximize the skip-gram model, which given a word attempts to predict the surrounding words, to learn a distributed representation of words. To explore this method further, see [17]. This model contains 3 million words, each represented by a 300 dimensional vector and was trained on the Google News dataset. Finally a publicly available machine learning library, dlib [11], was used to obtain the RSVM module.

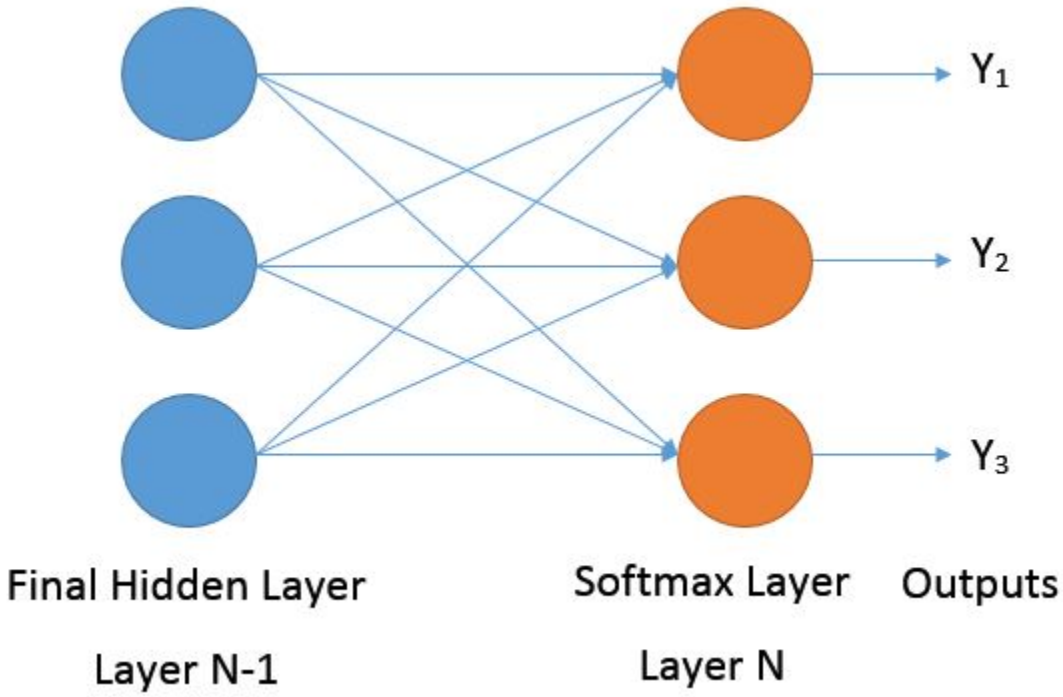


FIGURE 3.2: Final hidden layer passing inputs to a softmax layer. Outputs are the softmax results for each neuron

3.3 Implementation

For zero-shot evaluation the RSVM is trained on part of the pascal VOC 2012 dataset [2] and tested on the a-yahoo dataset [3]. The pascal dataset contains 20 classes of objects such as, 'person', 'car', and 'dog', and the a-yahoo set contains 10 new classes not present in the pascal set. Because the overwhelming majority of pictures in the pascal set contain the 'person' tag, images of other classes had to be filtered such that a person was not present in the majority of the images.

3.3.1 Image Representation

Each image is represented by a 1000 dimensional vector of real values taken from the output of the last hidden layer, the layer preceding the softmax output layer, of a deep neural network as seen in figure 3.2. Each image was passed through the MatConvNet toolbox [25], a convolutional neural network toolbox created for Matlab, which rescaled each image and used a pre-trained DNN to extract the feature vectors for each image. The DNN used, [12] ,was trained on 1000 different classes from ImageNet. The training classes were made up of more specific labels, such as 'tiger shark' or 'great white shark', than general purpose labels like simply 'shark'.

3.3.2 Text Representation

Text labels, words in this case, are represented by 300 dimensional vectors created by the skip-gram method described in [17]. Rather than training new word vectors a pre-trained model that contains 3 million word vectors was used. This model was trained on the Google News dataset allowing it to learn a word-space from real world documents. These vectors create a space in which the words they represent are distributed according to their semantic relationships.

Figure 3.3 uses t-distributed stochastic neighbor embedding (t-sne) to visualize the high dimensional word vectors in a three dimensional space, where color is the third dimension, to show how semantically similar words cluster together. T-sne attempts to preserve the structure of local groups in high dimensional space

together by converting Euclidean distance in this space into a conditional probability. This is the probability that a point x_i selects point x_j as a 'neighbor'. Neighbors are selected in proportion to a Student t-distribution at point x_i. A similar distribution is defined for points in the lower dimensional space, then the algorithm tries to minimize the Kullback-Leibler divergence between the two distributions. Using this dimensionality reduction technique it is easy to see how words of different semantic relationships cluster into distinct groups. For a more in-depth explanation of t-sne, see [16].

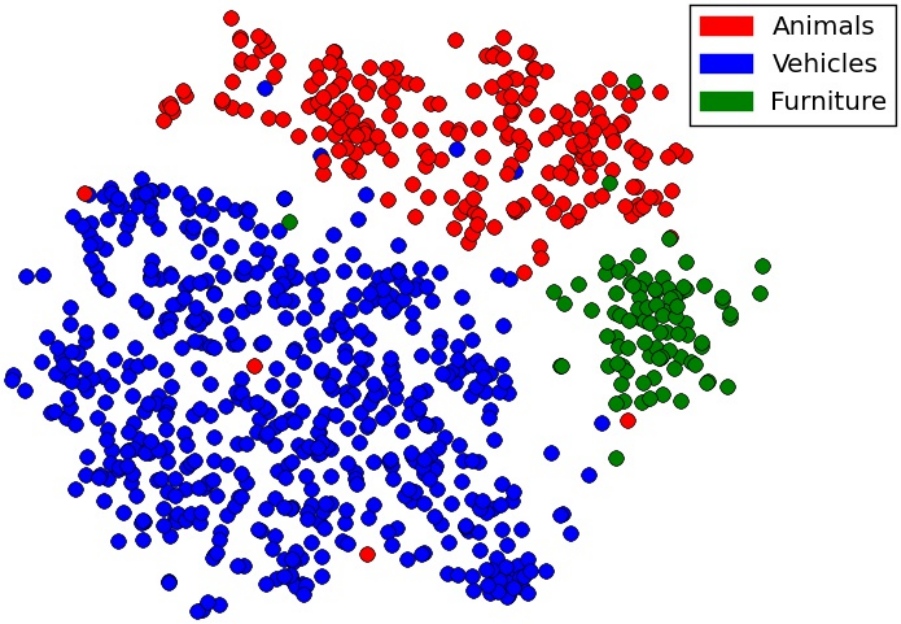


FIGURE 3.3: t-sne visualization of word clusterings for 3 categories: animals, vehicles, and furniture

Not only do these vectors group based on meaning, they capture connections between words, which can be seen through vector operations. A popular example

is addition and subtraction, such as, $\vec{king} - \vec{man} + \vec{woman}$ which creates a vector very close to the vector $\vec{queen}$.

The similarity between words can easily be measured in this space by finding the similarity between their embedding vectors using the cosine similarity between two vector A and B, which is defined as:

$$similarity(A, B) = cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} \tag{3.1}$$

Where the similarity is in the range $[-1, 1]$ with 1 being exactly the same. Using this method to compare all of the tags in the Pascal VOC dataset gives the confusion matrix shown in figure 3.4. In this plot darker red colors indicated more similarity and darker blue colors represent less similarity. The diagonal is the similarity of a word with itself and therefore is a perfect 1. Groupings of similarity can be seen from words in the same category, such as animals like 'dog', 'cat' and 'horse'.

3.3.3 Image Retrieval

With the representation for images and words introduced the framework for image retrieval is now presented. All vectors are assumed to be column vectors, with $\vec{vector}^T$ representing the transpose of a column vector into a row vector.

Let $D = (\vec{I_1}, \vec{T_1}), ..., (\vec{I_n}, \vec{T_n})$ be a set of images $\vec{I}$ with corresponding groundtruth text-labels $\vec{T}$. Each specific image $\vec{I_i}$ is represented by a p-dimensional vector of real values and each corresponding text-label $\vec{T_i}$ is represented by a k-dimensional

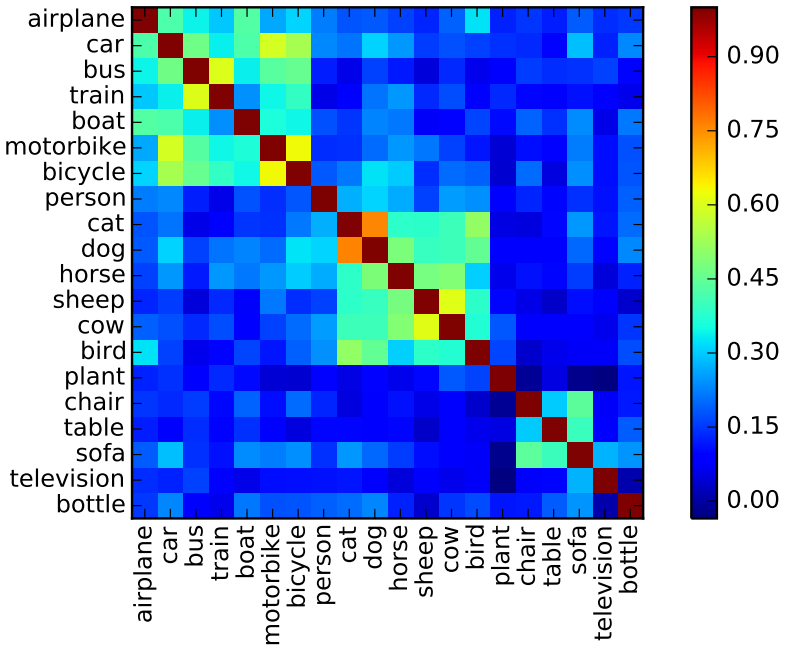


FIGURE 3.4: Confusion matrix of the similarity between Pascal VOC tags. Darker red colors denote higher similarity

real valued vector. Because images of the same class can vary so greatly we will attempt to return a ranked list of N most similar images for a query in the set rather than trying to directly classify the images. A projection is needed to make vectors of different spaces directly comparable; to project images into the same space as the text a projection matrix S is used. Once in the same space images and text can be compared using the cosine similarity shown in equation 3.1. This process is shown is equations 3.2, with $\vec{I_T}$ representing an image vector that has been projected into the word space.

$$\begin{aligned} \vec{I_T} &= \vec{I_i}^T \cdot S \\ score &= cosine\ similarity(\vec{I_T},\ \vec{T_i}) \end{aligned} \tag{3.2}$$

With a higher score indicating a better match between image and text. Once a satisfactory S matrix is found textual queries can be used to search directly for images in a database.

3.3.4 Ranking SVM

If image A is preferred to image B we will use the notation $A \succ B$. Because the ranking function returns a score for each item we wish to learn a ranking function such that, $F(\vec{I_i}, \vec{T_i}) > F(\vec{I_j}, \vec{T_i}) \iff \vec{w} \cdot \Phi(\vec{I_i}, \vec{T_i}) > \vec{w} \cdot \Phi(\vec{I_j}, \vec{T_i})$ where $\vec{I_i} \succ \vec{I_j}$ for the text-label $\vec{T_i}$ and $\vec{w}$ is a learned weight vector. $\Phi(\cdot)$ is a joint feature representation of the image and text features. The weight vector $\vec{w}$ can be found using a RSVM which solves the following optimization problem [9]:

$$
\begin{aligned}
&minimize : V(\vec{w}, \vec{\xi}) = \frac{1}{2}\vec{w} \cdot \vec{w} + C \sum \xi_{i,j} \\
&subject\ to : \forall(\vec{T_i}, \vec{T_j}) \in R : \vec{w}\Phi(\vec{I_1}, \vec{T_i}) - \vec{w}\Phi(\vec{I_1}, \vec{T_j}) \geq 1 - \xi_{i,j} \\
&\qquad \ldots \\
&\qquad \forall(\vec{T_i}, \vec{T_j}) \in R : \vec{w}\Phi(\vec{I_n}, \vec{T_i}) - \vec{w}\Phi(\vec{I_n}, \vec{T_j}) \geq 1 - \xi_{i,j} \\
&\qquad \forall i \forall j : \xi_{i,j} \geq 0
\end{aligned}
\tag{3.3}
$$

Here ξ is a non-negative slack variable that allows the solution to be approximated so that the problem is not NP-hard. C is a parameter that controls the trade-off between margin size and training error. Higher C values add more weight to training errors and cause the RSVM to fit to the training data, while lower C values will penalize errors less, allowing for a larger margin at the cost of a few errors. The

RSVM aims to minimize $\vec{w} \cdot \vec{w}$, which maximizes the margin $\frac{1}{\|\vec{w}\|}$, giving better generalization. The RSVM implemented is the svm_rank_trainer provided by dlib [11] solved using a cutting planes algorithm for efficiency.

3.3.5 Joint Feature Representation

The $\Phi(\cdot)$ function provides a joint representation of the input and output space. This joint input-output relation is what allows the SVM to learn to make structured predictions, for example predicting an output vector based on the input, rather than just simple yes/no classifications. Images and text are both represented by vectors and even though these vectors are found by different methods, they both describe semantically similar information. For example a picture of a dog and the word 'dog' have similar meaning, but it is presented in very different ways. The same is true for their respective feature vector representations. Because it is not obvious how the dimensions of the feature vectors represent similar semantic content it is reasonable to use every interaction between the two to create a joint representation. This can be done using the tensor product of $\vec{I}$ and $\vec{T}$. Therefore $\Phi(\vec{I}, \vec{T}) = \vec{I} \otimes \vec{T} \in R^r$ where $r = p \times k$. This leads to a $\Phi(\cdot)$ matrix with number of rows equal to p and the number of columns equal to k, which is a problem as the RSVM solves for a vector not a matrix. By using row major vectorization to reshape $\Phi(\cdot)$ into a $pk \times 1$ vector the weights $\vec{w}$ can be solved for normally.

For the actual task of image retrieval it is inefficient to calculate the tensor product.

$$F(\vec{I}, \vec{T}; \vec{w}) = \vec{w} \cdot vect(\vec{I} \otimes \vec{T}) \tag{3.4}$$

By reshaping the $pk \times 1$ vector $\vec{w}$ into a $p \times k$ matrix, the result is the projection matrix S from earlier; thus the ranking function $F(\vec{I}, \vec{T}; \vec{w})$ can be written in bilinear form:

$$F(\vec{I}, \vec{T}; \vec{w}) = \vec{I}^T \cdot S \cdot \vec{T} \tag{3.5}$$

Using this approach has the advantage of reducing the number of multiplications from $\Pi = (pk)^2$ to $\Pi = pk + k$. If $p = k = N$ then the reduction can be more cleanly seen as reducing $\Pi = N^4$ to $\Pi = N^2 + N$. Using the bilinear form in equation 3.5, the final multiplication can be replaced with the cosine similarity as shown in equations 3.2 with no loss of information since the cosine similarity function is simply the normalized dot product.

3.4 Experimental Setup

The RSVM retrieval system was fully implemented in the python programming language. While the core functions of the dlib rank svm training code are implemented in C++, a python API exists for the rank svm trainer, making it easy to set training parameters and pass data to the C++ functions. To test the performance of the RSVM at a retrieval task the average precision of each retrieved class is examined along with the mean average precision (MAP) of the database searched.

All experiments and code were run using a 64-bit home computer with an Intel i7-4790k processor, which has a clock speed of 4.2 GHz and 8 cores (4 physical, 4

virtual), and 16 GBs of RAM. Python version 2.7 was used with the 64-bit version of the Enthought Canopy IDE. Please note that the version of Python is important, Python 2.7 and the newer versions, Python 3.x, have some differences that can make code complied on one version incompatible with the other, such as how integer division is handled. So it is not guaranteed that any of the code used in this thesis will compile or work correctly on Python 3.x.

3.4.1 Creating Training Data

As previously mentioned the RSVM was trained on a subset of the Pascal VOC 2012 dataset. Matlab was used with the MatConvNet toolbox to create a training file of the full set of images. This file contained the name of the images, the tags associated with each image and the feature vector for each image. The Matlab script used for the Pascal set reads in every image in a target directory and uses the name of the images to extract the correct tags from the appropriate XML file provided in the Pascal dataset, then it extracts the image feature vectors as described previously in section 3.2.3 using the neural network from [12] and writes the data to a text file that can be used later for training.

This training file was read into a custom python data object, named Xdata whose code is found in appendix N, for easier data manipulation. Because many of the images in the pascal set contain a mixture of classes and the large majority of the images are of people, images were filtered into a working group of 238 images per class, this number was taken from the motorbike class which had the lowest number of images. The reason for this was so that a single class did not dominate the training set. For example over half of the images contain the 'person' class, so if a

classifier were trained on the full set it could achieve 50% accuracy by always predicting the 'person' class. The working group was further split in to three smaller sets, 60% for training (~142 images/class), 20% for cross-validation (~47 images/-class), and 20% for testing (~47 images/class). The cross-validation set is used to prevent over-fitting to the training data when tuning the learning parameters. After the data is trained it is tuned to increase the accuracy on the cross-validation, then tested on the testing set. This keeps the algorithm from fitting to the testing set and receiving good test results but poor general results.

Four of the tags in the Pascal set had to be replaced as either their spellings were incorrect or were not included in the used Word2Vec model, they are: 'aeroplane' -> 'airplane', 'pottedplant' -> 'potted_plant', 'tvmonitor' -> 'TV', and 'diningtable' -> 'table'.

The Xdata object also stores a Word2Vec model. The Gensim topic modeling tool box [21] for python is used to load a pretrained Word2Vec model into the Xdata object. The Gensim toolbox stores the whole Word2Vec model as a python dictionary, this way it is a simple lookup to find the vector for a given word. The Xdata object uses the Word2Vec model in a member function called *export_training_data*() to create the training file for the RSVM. The function writes data to a file arranged into groups with a target class word vector, a set of 'relevant' image vectors that fall into that class, and a set of 'non-relevant' examples, which are any images not of the target class. A random seed is used to randomly select non-relevant and relevant image vectors, up to the maximum number of available examples for a class, so that no duplicates are used. This is done for each training class and the data is placed into a single text file, which is readable by the dlib svm_rank_trainer. Appendix D contains a python script which details the exact steps to create training

data for the RSVM.

3.4.2 RSVM Training

Two functions are all that is needed to train the RSVM from a given training file. The first function in the pipeline *create_CDSVM_rank_trainer*() creates the dlib object that does the training with the specified hyper-parameters C, and epsilon. It also places the training data in a dlib object called 'queries' that is used for training. The second function *train_CDSVM_rank*() simply makes a call to the dlib smv_rank_trainer to train using the given data, and then outputs the resulting weight vector into a text file in numpy format, a commonly used scientific computing toolbox for python.

An advantage of using this two step process is that the queries do not need to be recreated each time one wishes to train the RSVM with different C and eps values. After the trainer is created all of its controlling parameters can be changed using 'dot' notation, e.g. trainer.c = new_value. Also when the queries object is created the tensor product between the image and word vectors is preformed, this way the RSVM trainer does not need to calculate $\Phi(\cdot)$ each time for training. This speeds up training but requires more memory to store the larger vectors. The output text file containing the learned weights can be reshaped into the S matrix used for transforming new image feature vectors into the word-space. A script to perform RSVM training is included in appendix E.

3.4.3 Testing Setup

To test zero-shot performance on the a-yahoo set image feature vectors were extracted using the same method as the Pascal dataset. All the images were loaded into a Xdata object along with the groundtruth labels. This allows for an easy lookup to check the class of a retrieved image.

Experiments were run in their own Ipython notebooks to make them easier to test and edit. The Ipython notebook code was converted to standard python scripts for readability and included in the appendices.

Chapter IV

Results

The results of experiments preformed and investigations made are detailed in the following section.

4.1 Measuring Precision

Evaluating the performance of an information retrieval system can be done several ways. A straight forward approach is to observe the precision for all of the retrieved items. If a system returns a list of items then the precision can be calculated as:

$$Precision = \frac{\{relevant\ items\} \cap \{retrieved\ items\}}{\{retrieved\ items\}} \tag{4.1}$$

If n items are returned then this represents the $precision\ at\ n$ or $P@n$. While this is an useful for measuring how many of the returned items are relevant to the search, it does not take into consideration the order in which items are presented to the searcher. For example, when searching for documents related to preparing

beef wellington it would be better if recipes appeared before other documents, that way a user could expect that the most relevant results would be returned first.

The average precision is often used in information retrieval to evaluate the results of an ordered list. This measure is not the arithmetic mean of the precision over some range, instead it is the average of the precision after each relevant item is returned. This weights the precision at each entry based on the ordering by scaling the precision at n by the inverse of the number of relevant items or total number of items, whichever is smaller if the entry at n is correct, and zero if it is wrong. This way results are not penalized by wrong guesses, but are penalized for not ranking relevant items first. Mathematically average precision is calculated as:

$$Average\ Precision = \frac{1}{M} \sum_{n=1}^{M} P@n \cdot I(n) \tag{4.2}$$

Where

$$I(n) = \begin{cases} 0 & \text{if the } n^{th} \text{ item is irrelevant} \\ 1 & \text{otherwise.} \end{cases}$$

In the case that there are less relevant items available than the number of items returned then equation 4.2 becomes

$$Average\ Precision = \frac{1}{\text{\# relevant items}} \sum_{n=1}^{M} P@n \cdot I(n) \tag{4.3}$$

Taking the mean of the average precision for multiple retrieval tasks gives a good,

single numerical value representation of the average performance of the system. The mean average precision at some number of retrieved items N, MAP@N, is simply:

$$MAP@N = \sum_{i=1}^{N} \frac{Average\ Precision_i}{N} \tag{4.4}$$

4.2 Zero-shot Performance

In the following section the results for the proposed zero-shot retrieval method are shown. As described in Chapter III, the system will attempt to return a ranked list of images, from classes not previously trained on, in order of relevancy to the search term. The model used for all of the following experiments was trained on 100 examples of each of the 20 training classes with an error trade-off parameter, C, of 0.1 and a convergence radius, epsilon, of 0.01, unless otherwise noted. The set of zero-shot images contains 12 classes that belong to the a-yahoo dataset [3] and share no classes with that of the training set from Pascal VOC 2012 [2]. The a-yahoo data totals 2237 images and does not have an even distribution of classes, for example there are 366 jetski images and only 48 centaur images. Table 4.1 shows the distribution of all 12 classes.

Class	# of images
building	213
donkey	128
monkey	156
mug	210
centaur	48
bag	279
carriage	147
wolf	171
zebra	179
statue	191
jetski	366
goat	149
total images	2237

TABLE 4.1: Table of a-yahoo class representation

To find a general representation of the systems performance as a whole the MAP@N as defined in equation 4.4 is used, which is the average of the Average Precisions for each class. Figure 4.1 shows the MAP@N score plotted against the number of items returned for each search.

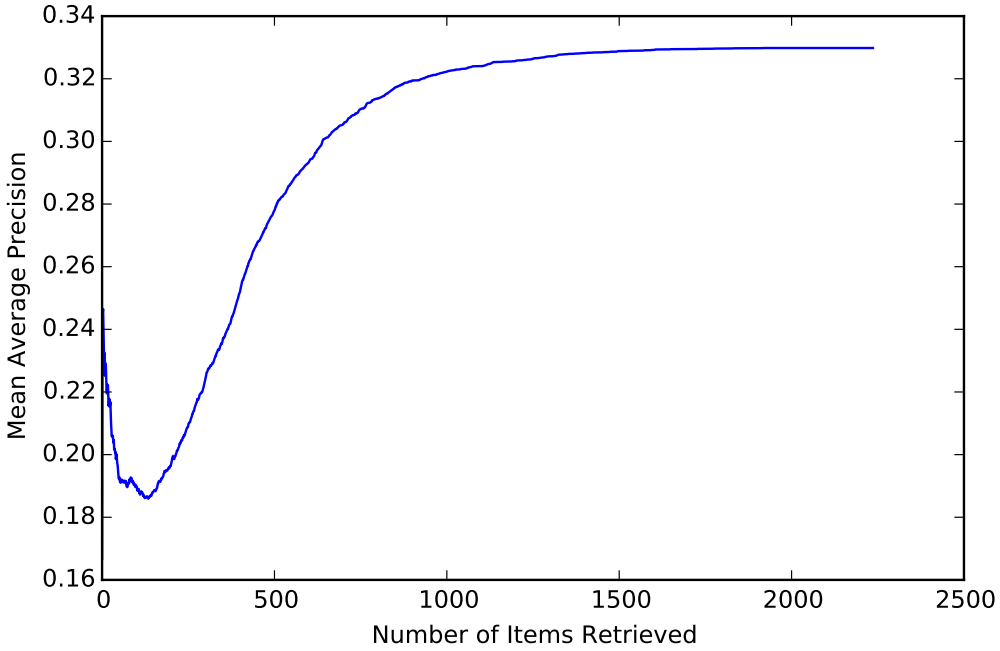


FIGURE 4.1: MAP scores verses number of items retrieved

# of items retrieved	MAP
1	0.166667
5	0.240556
10	0.229335
50	0.192191
100	0.189675
200	0.196376
300	0.223801
400	0.251852
500	0.277856
1000	0.322208
1500	0.328824
2000	0.329816
2238	0.329816

TABLE 4.2: Table of key zero-shot MAP@N scores

Table 4.2 summarizes the MAP scores with the results at some of the points of interest on the curve. These points were chosen as usual return values a user might

	Precision	AP	Recall
bag	0.26	0.100	0.465
mug	0.376	0.341	0.895
goat	0.23	0.259	0.771
donkey	0.204	0.200	0.796
zebra	0.126	0.029	0.351
wolf	0.312	0.253	0.912
monkey	0.216	0.163	0.692
centaur	0.068	0.131	0.708
jetski	0.644	0.830	0.879
carriage	0.25	0.176	0.850
statue	0.226	0.337	0.591
building	0.386	0.511	0.906
Mean	0.275	0.278	0.734

TABLE 4.3: Table of precision, average precision, and recall at 500 samples

be interested in when searching for pictures.

Generally, the MAP increases as the number of returned items increases. This is an expected result as the Average Precision function does not penalize the retrieval system for returning incorrect results, it only penalizes relevant items being ranked after non-relevant items. The downward trend in MAP scores from the initial point in the graph to a retrieval size of about 134 is interesting, as this is the point at which the retrieval size close to or slightly greater than the number of relevant images for all classes, except the jetski class. At this point the equation for Average Precision at N samples is close to equation 4.3 for the majority of classes. This indicates that for zero-shot classes the retrieval system returns poor ranking results when the number of retrieved images is much less than the total number of relevant images for a given class.

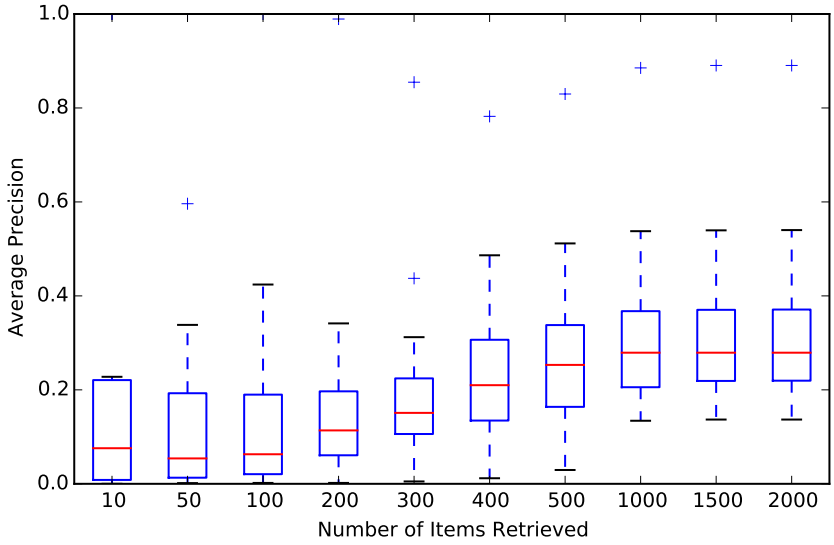


FIGURE 4.2: Average precision at different retrieval sizes

The MAP begins to increase exponentially after 134 samples, with the Mean Average Precision increasing by 8% from approximately 134 samples to 500 samples. Over the next 1000 samples there is only an increase of approximately 5%. This is perhaps the region in which the systems achieves the best performance as only about one fifth of the total number of images are retrieved to achieve a MAP of 27% and an average recall of 73%, shown in table 4.3.

Figure 4.2 contains a box plot of the average precision of all twelve classes at several return sizes. The box plot visualizes the distribution of the data by splitting it into three quartiles, with the outliers plotted as '+' symbols. In terms of a normal distribution, the box itself would represent the data falling within approximately $\pm 0.5\sigma$ and the whiskers indicate $\pm 2.5\sigma$ from the mean. The red dividing line represents the median value. The boxes were obtained by recording the Average Precision for each class at the return size indicated on the x-axis.

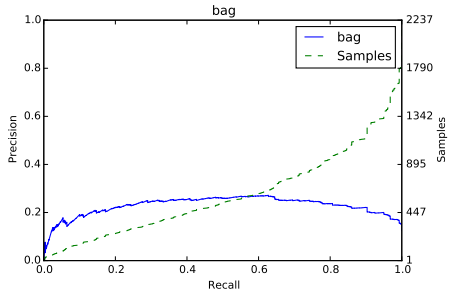
The same trend is seen here with a slight dip in median performance from 10 to 100

before increasing steadily to a maximum median average precision at return size of 1000. The box plot gives a better view of the underlying data used for calculating the MAP scores and shows that the Average Precision closely follows the same trend as the MAP. It can be seen that the distribution of the average precision is smallest at a return size of 300 and that there are noticeable outliers that have much higher average precision scores than the other classes. These outliers were found to belong to the 'jetski' class.

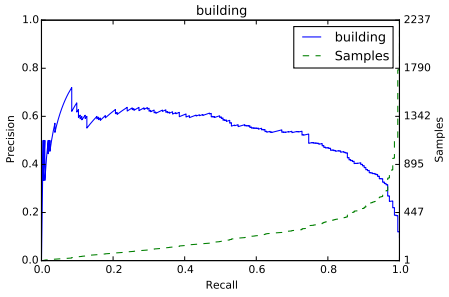
In figure 4.3 a precision-recall curve is plotted for each of the twelve zero-shot classes. Note that the precision-recall curve uses the standard definition of Precision at N from equation 4.1 and not the Average Precision.

The precision-recall curve is typically used to investigate the trade-off between precision and recall for different thresholds on a retrieval system, where, as the threshold for the retrieval system to accept or reject an item is lowered, more items are retrieved until the threshold reaches zero and all items are accepted. The proposed system however, does not use a threshold and simply returns a chosen amount of images with the highest scoring first, so as recall increases the number of items retrieved, relevant or non-relevant, increases. Only plotting the precision against the recall can be a bit misleading in this case because the recall value represents the percentage of relevant items that were returned and not the total number of items retrieved. Because it is not expected that the zero-shot items will have particularly high precision it is of interest how many total items were retrieved to achieve the plotted precision-recall values. To clarify this the number of samples is plotted against the recall as a dashed green line on the graphs to create a precision-recall-sample graph.

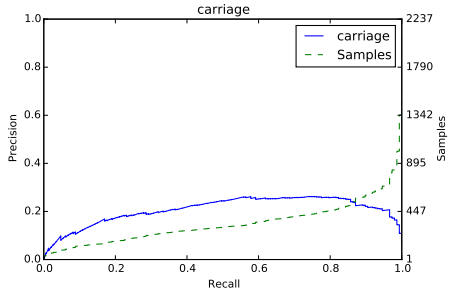
A perfect precision-recall curve with 100% precision from [0,1] recall would create a unit box, while a curve with peaks near the origin indicates high-precision at low-recall. Typically the curve slopes downward as the recall increases because there are more non-relevant items than relevant and therefore are more false-positives.



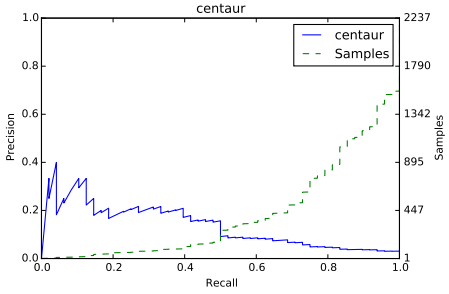
(A)



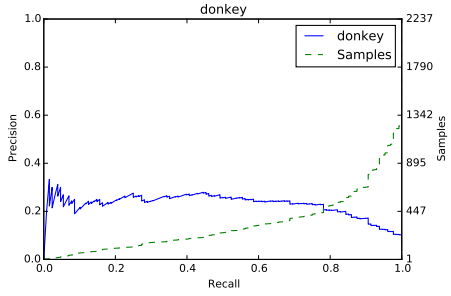
(B)



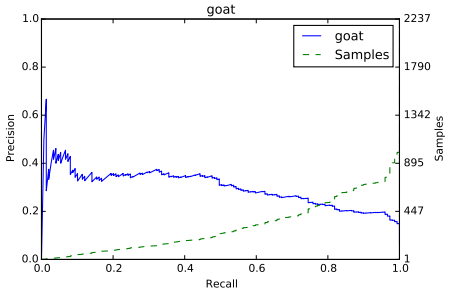
(C)



(D)



(E)



(F)

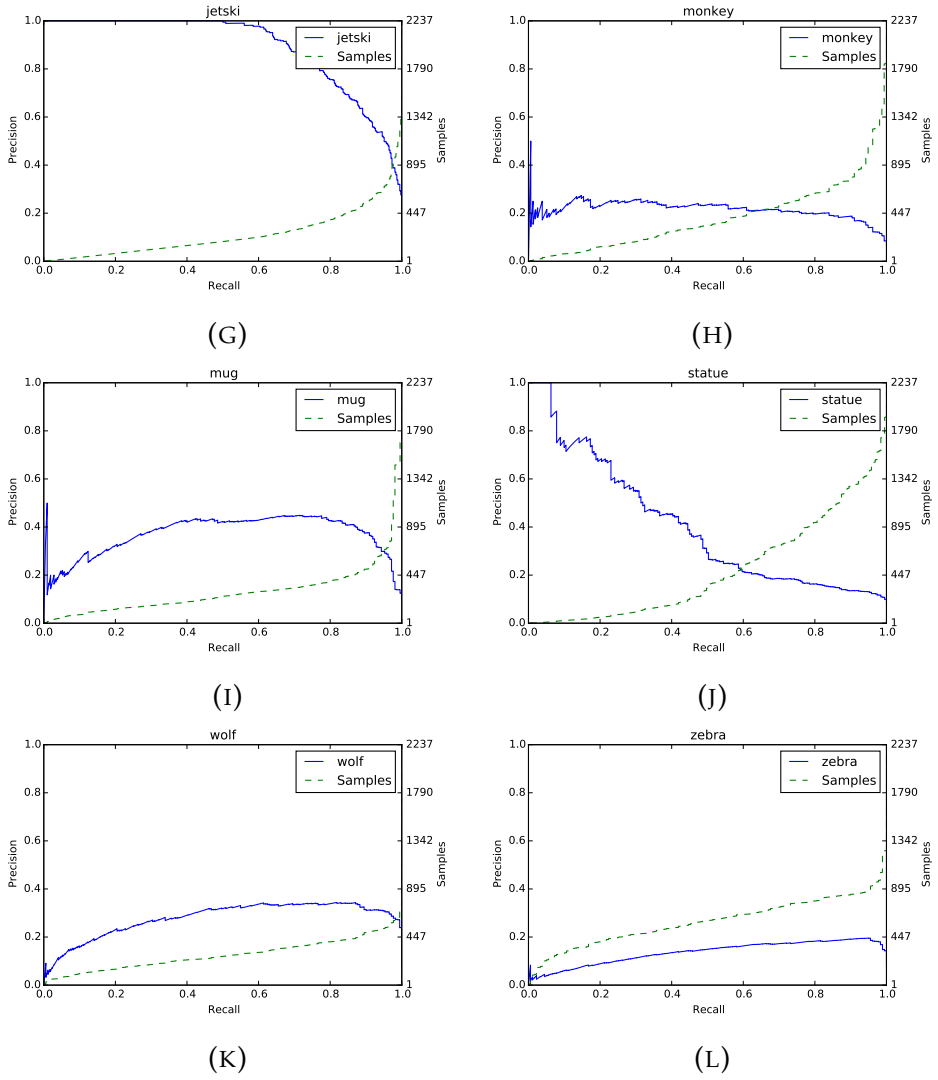


FIGURE 4.3: precision-recall-samples curves
A-L: bag, building, carriage, centaur, donkey, goat, jetski, monkey, mug, statue, wolf, zebra, respectively

These precision-recall curves give another view into the performance of each zero-shot class. While some of the classes, such as jetski, building, and statue have unexpectedly high performance the majority have precision around 20%-40%. The precision-recall-sample curve for the zebra class shows surprisingly low performance for all recall levels; given the understanding that a zebra is similar in many

ways to that of a horse, a training class, one would expect the zebra class to have a higher performance than a class with few similar training examples, such as 'bag'.

Table 4.3 shows the precision, Average Precision and recall results for each class at 500 retrieved images. As mentioned previously a return size of 500 is the point at which the majority of the classes have returned 70% or more of the total number relevant images.

4.2.1 Comparison to Random Retrieval

To validate that the results of the previous section as better than simply random guessing this section compares the results to that of random retrieval. As it is difficult to directly calculate the probability of retrieving a certain number of relevant images from a large database with a given selection size, Monte Carlo simulations are used to find the performance of random retrieval for each of the twelve zero-shot classes.

A Matlab script was written to perform the Monte Carlo simulations. To simulate a ranking of the 2237 images in the database a sampling without replacement of the integers ranging from [1,2237] was taken 2237 times, this in effect results in a randomly ranked list of 2237 unique objects. This was performed 10,000 times. As each class is considered separately the 10,000 lists can used for all twelve classes. For each class the first R integers are considered to be positive, where R is the number of images belonging to a class; for example, the 'wolf' class contains 179 images so integers [1,179] are considered to be positive results. For each class the randomly ranked lists are converted to binary row vectors, with 1's representing relevant objects and 0's representing non-relevant, and then the cumulative sum

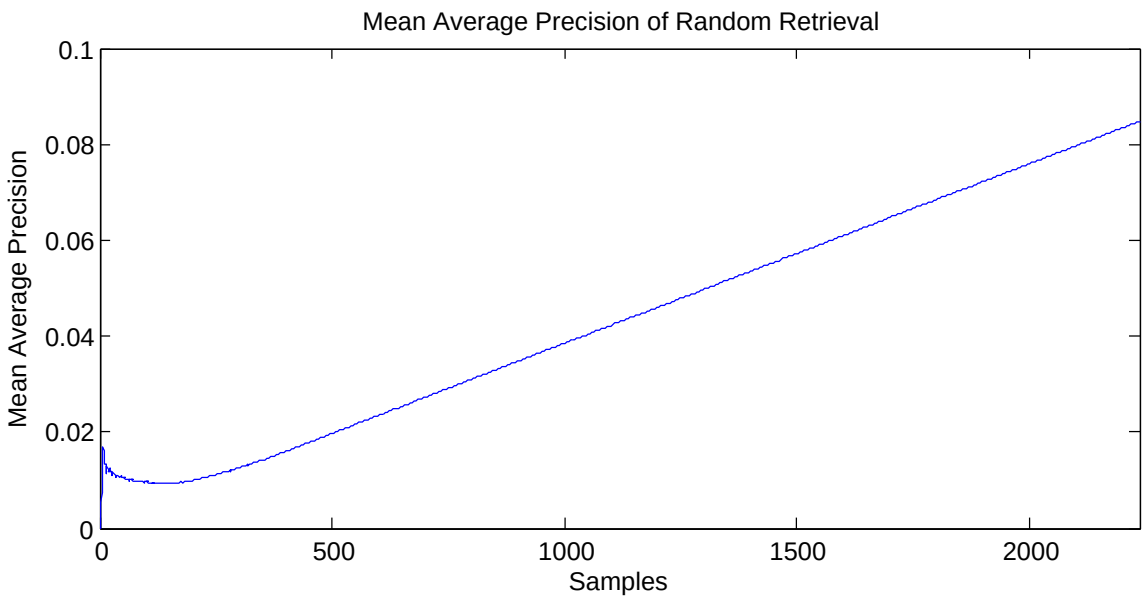


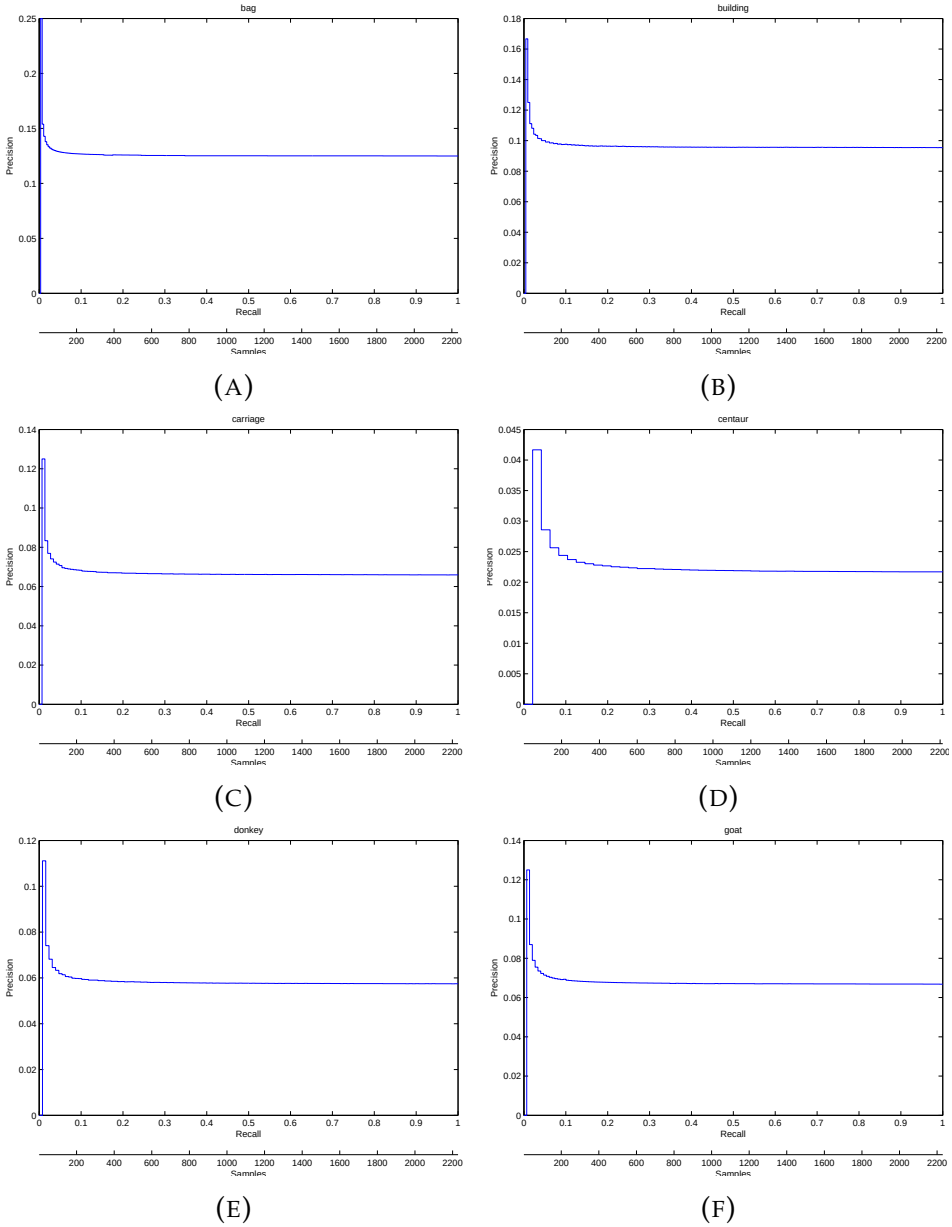
FIGURE 4.4: MAP scores verses number of samples retrieved for random retrieval

of each vector is taken so that each element represents the number of hits at that point in the list. The mean of each sample position, the column mean, is found and rounded to the nearest integer resulting in an average list for each class. From this list the precision, recall and MAP were calculated for each class at each return size.

Figure 4.4 shows the MAP@N for the Monte Carlo simulations of each class.

Reaching a final value of only about 8% the MAP@N for random image retrieval is four times lower than maximum the mean average precision of the purposed retrieval system, about 32%. At a return size of 500 the random MAP score is approximatively 2% where the proposed method achieves 27%.

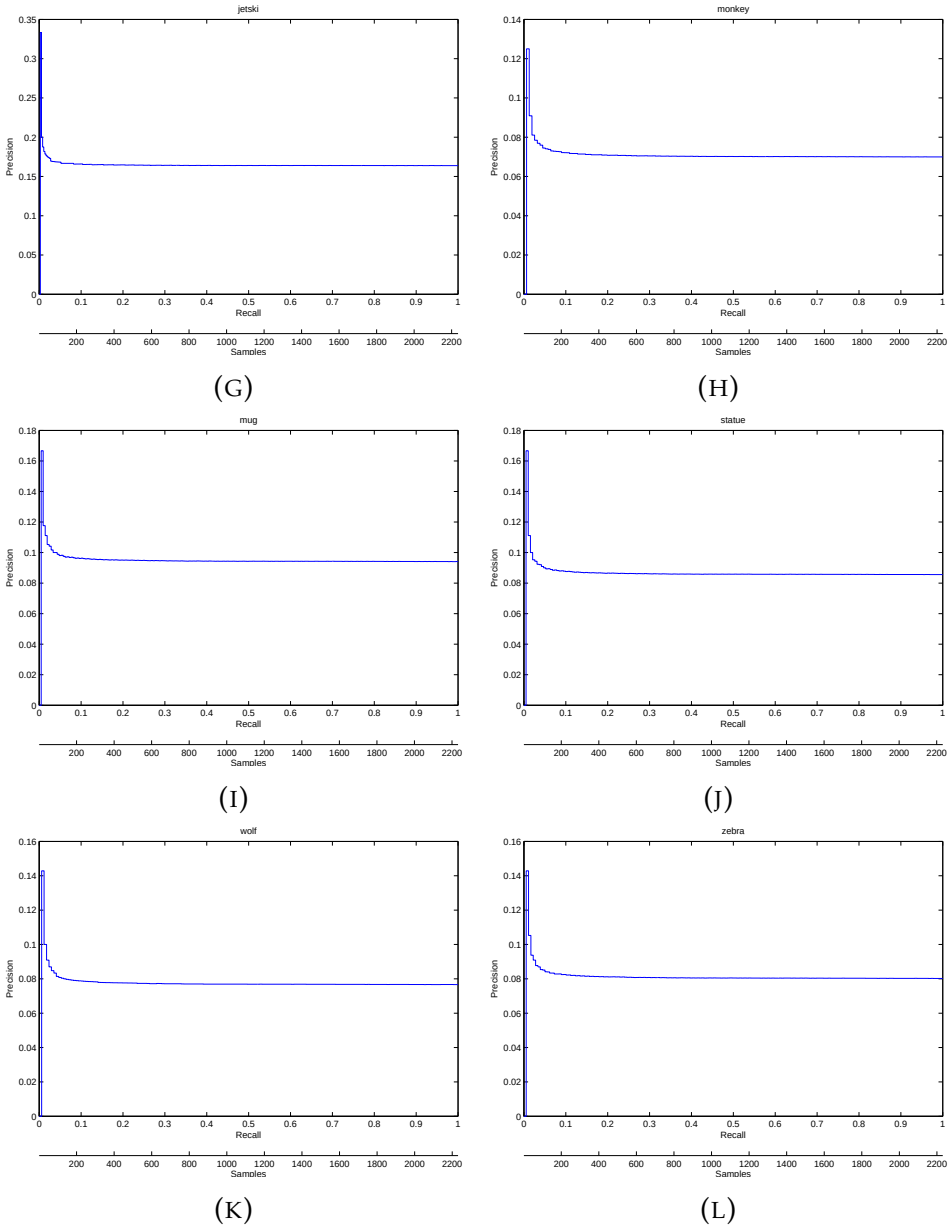
Figure 4.5 contains the twelve precision-recall curves for the Monte Carlo simulation of each zero-shot class. This time the number of samples had a nearly linear relationship with the recall, whereas in the previously it did not, allowing it to be easily plotted as a secondary x axis underneath the primary one to give a helpful visualization of the number of samples at each recall point.



(A) (B)

(C) (D)

(E) (F)



(G) (H)

(I) (J)

(K) (L)

FIGURE 4.5: precision-recall curves of Monte Carlo simulations
A-L: bag, building, carriage, centaur, donkey, goat, jetski, monkey, mug, statue, wolf, zebra, respectively

It can be seen that all of the precision-recall graphs for the random selection case approach a near constant precision for all recall levels. The initial spike is caused from the averaging and rounding of the 10,000 randomly ranked lists, where there

is typically a positive hit within the first ten samples, giving the graph a higher starting point. After enough results are retrieved each of the classes approaches a constant precision of $\frac{\text{class size}}{\text{total}}$.

Table 4.4 summarizes the precision of the Monte Carlo simulations for each class at several common points and table 4.5 shows the precision of the proposed method for comparison.

The shaded cells in table 4.5 represent precisions lower than that of random selection. The worst performance is seen at 10 and 50 items returned. At 100 or more all classes, excluding 'zebra', score higher than random and in most cases significantly better. With 500 returned items all classes score higher than the Monte Carlo simulations.

Table 4.6 shows the recall percentage for each class at the same return sizes as used in table 4.4. An interesting finding here is that the recall for each class at a return

	Precision of Random Retrieval						
	@10	@50	@100	@200	@300	@400	@500
building	0.100	0.100	0.100	0.095	0.097	0.095	0.096
donkey	0.100	0.060	0.060	0.055	0.057	0.058	0.058
monkey	0.100	0.060	0.070	0.070	0.070	0.070	0.070
mug	0.100	0.100	0.090	0.095	0.093	0.095	0.094
centaur	0.000	0.020	0.020	0.020	0.020	0.022	0.022
bag	0.100	0.120	0.120	0.125	0.123	0.125	0.124
carriage	0.100	0.060	0.070	0.065	0.067	0.065	0.066
wolf	0.100	0.080	0.080	0.075	0.077	0.077	0.076
zebra	0.100	0.080	0.080	0.080	0.080	0.080	0.080
statue	0.100	0.080	0.090	0.085	0.087	0.085	0.086
jetski	0.200	0.160	0.160	0.165	0.163	0.163	0.164
goat	0.100	0.060	0.070	0.065	0.067	0.068	0.066

TABLE 4.4: Table of precision scores for random retrieval of classes at seven values

	Precision of Proposed Method						
	@10	@50	@100	@200	@300	@400	@500
building	0.5	0.56	0.63	0.565	0.523	0.45	0.386
donkey	0.3	0.22	0.23	0.275	0.25	0.232	0.204
monkey	0.2	0.2	0.26	0.245	0.226	0.2175	0.216
mug	0.2	0.22	0.26	0.42	0.436	0.425	0.376
centaur	0.2	0.18	0.19	0.12	0.086	0.0775	0.068
bag	0.0	0.12	0.16	0.205	0.23	0.2525	0.26
carriage	0.0	0.04	0.1	0.185	0.243	0.2575	0.25
wolf	0.0	0.06	0.16	0.265	0.33	0.335	0.312
zebra	0.0	0.02	0.03	0.04	0.06	0.0875	0.126
statue	1.0	0.7	0.56	0.415	0.313	0.255	0.226
jetski	1.0	1.0	1.0	0.99	0.866	0.7425	0.644
goat	0.4	0.34	0.35	0.34	0.286	0.265	0.23

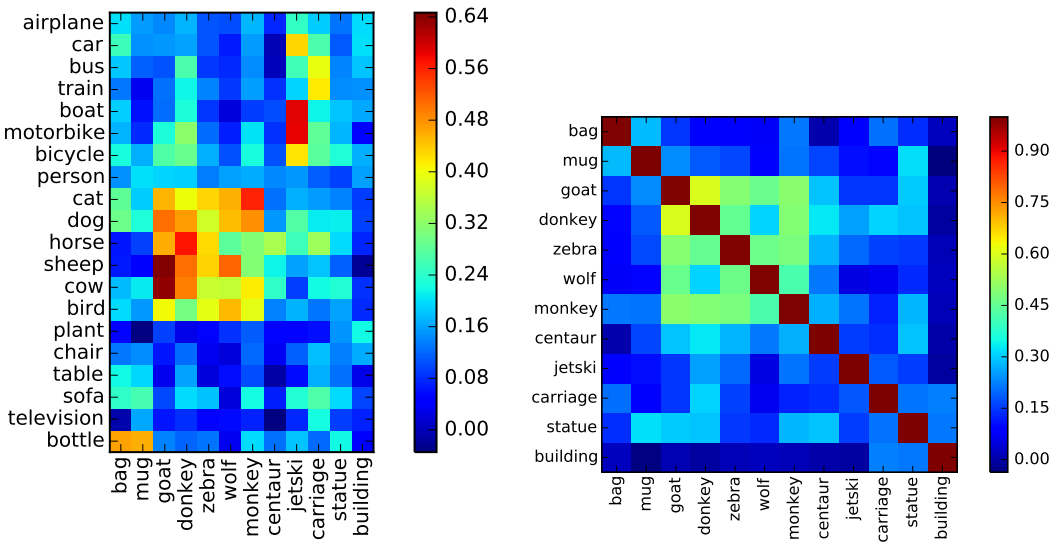
TABLE 4.5: Table of precision scores for proposed method at seven values, shaded cells indicate worse than random performance

size of 200 or more is approximately $\frac{\text{return size}}{total}$.

At a return size of 500 the purposed method's lowest recall is 35% for the zebra class and an average of 73%, significantly higher than that of the random retrieval

	Recall						
	@10	@50	@100	@200	@300	@400	@500
building	0.005	0.023	0.047	0.089	0.136	0.178	0.225
donkey	0.008	0.023	0.047	0.086	0.133	0.180	0.227
monkey	0.006	0.019	0.045	0.090	0.135	0.179	0.224
mug	0.005	0.024	0.043	0.090	0.133	0.181	0.224
centaur	0.000	0.021	0.042	0.083	0.125	0.188	0.229
bag	0.004	0.022	0.043	0.090	0.133	0.179	0.222
carriage	0.007	0.020	0.048	0.088	0.136	0.177	0.224
wolf	0.006	0.023	0.047	0.088	0.135	0.181	0.222
zebra	0.006	0.022	0.045	0.089	0.134	0.179	0.223
statue	0.005	0.021	0.047	0.089	0.136	0.178	0.225
jetski	0.005	0.022	0.044	0.090	0.134	0.178	0.224
goat	0.007	0.020	0.047	0.087	0.134	0.181	0.221

TABLE 4.6: Table of recall results for random retrieval of classes at seven values



(A) ayahoo-pascal (B) ayahoo-ayahoo

FIGURE 4.6: Word similarity confusion matrices of pascal and ayahoo tags

simulations.

4.2.2 Learning from Similarities

The information used to retrieve the zero-shot classes is learned by the RSVM through the relationship of words in the word-space. Figure 4.6a shows a confusion matrix of the cosine similarity between trained on class-labels and the zero-shot class-labels. The expectation is that the more training classes there are that are similar to a zero-shot class, the better the projection matrix will be able to project an image belonging to that zero-shot class to the appropriate point in the word-space.

The animal labels have the highest number of similar training class-labels compared to any of the other zero-shot class-labels. The word 'goat' has the highest single similarity score to a training class-label, 64% similar to the word sheep,

though this was not the class with the highest Average Precision. The highest performing class, 'jetski', has high similarity with only two classes, and is mostly dissimilar to all other training classes. It is likely that the relatively large number of animal type classes all being similar to each other act as noise, making it more difficult to distinguish one from another during retrieval. The words 'statue' and 'building' both have quite low similarity to any of the training class-labels.

Therefore, one would expect the building and statue classes to have the worse results than the animal classes, which all have several similar training classes. However this is found to not be the case, observing table 4.3 the building class has both a higher Average Precision and recall than any of the animal classes, the statue class has a higher Average Precision as well, though not a higher recall. Observing the similarity scores of the first 20 images returned for the building and statue class revealed that the highest scoring images were only 18-20% similar, while the top images for the animal classes were 28-30% similar. A likely explanation for the building and statue results is that, even though they have a low similarity to the appropriate word, all other classes are so dissimilar that the buildings and statues are the only images that have any sort of positive similarity. This is supported by figure 4.6b which shows the self-similarity between all of the class-labels in the ayahoo set. It is clear that all of the labels are dissimilar from each other, aside from the animal type labels, and the building label has particularly low similarity to all other labels, making it easier to distinguish images from this class as they only need to be slightly similar to the word 'building' to achieve the highest score.

4.2.3 Effect of Increasing the Number of Training Samples

For all of the previous experiments the RSVM was trained using 100 examples per training class. This number was selected because of hardware limitations, 100 and 20 classes consumed nearly all available memory during training. But does more training examples per class necessarily improve the performance of the retrieval system?

To test this the RSVM was trained multiple times with the number of training examples per class increasing from 1 to 100. Afterwards the Average Precision for each class was calculated at each number of training examples and the MAP for the entire system found for each point. Average Precision is used here rather than standard precision because it is indifferent to the number of items returned, so the entire database can be searched without penalizing the score and a single value calculated instead of several precision values at set points.

Figure 4.7 shows a scatter plot the results of the MAP percentage against the number of examples per class, increasing from 1 to 100, along with the line of best fit to the data in plotted in green with the slope of the line noted in the legend.

There is a definite trend overall of increasing MAP percentage with increasing numbers of training examples per class, but the slope of the line fitting the data is relatively small, equal to 0.092, increasing only about 8%, from 28% to 36%. If fact, observing the data falling between 60 and 100 examples per class, the MAP has little to no overall improvement with the increase of training examples, though the variation does appear to reduce a small amount.

The data in Figure 4.7 would seem to suggest that after approximately 60 examples

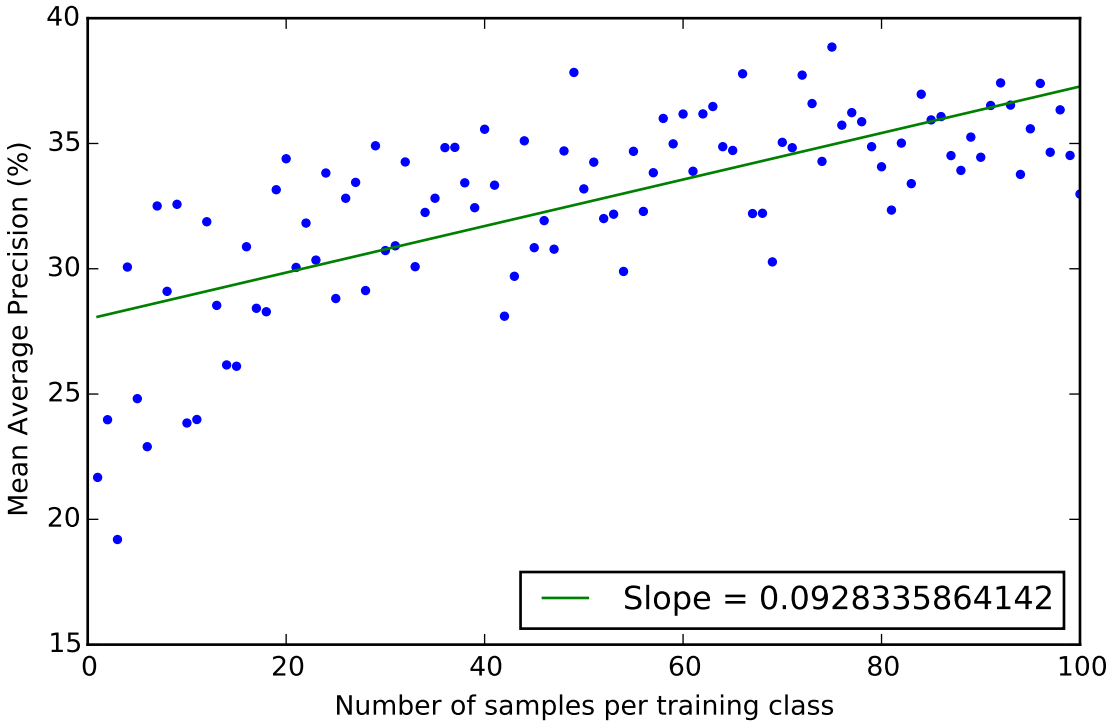


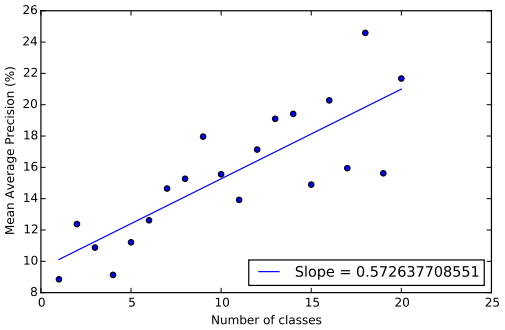
FIGURE 4.7: Mean Average Precision of zero-shot classes for increasing number of examples per training class

per class increasing the number of examples per class does not provide much addition information to the RSVM during training. This is likely an effect of using such a small number of examples, relative to other similar methods. For example [14] uses a subset of the Pascal VOC 2008 data, which consists of 12,695 images over the same 20 classes, giving approximately 600 images per class for training, six times the number of images per class used above. In [26] a similar SVM based method uses 17,658 total images with various labels for training a retrieval system and [23] uses 5,000 examples per class for zero-shot classification in one experiment and 500 per class for 106 classes in another. Compared to a web-scale method [7], which uses 1.28 million images of 1000 classes, the method in this thesis can be considered to be operating in a very low resource domain. Thus it is likely that a substantial

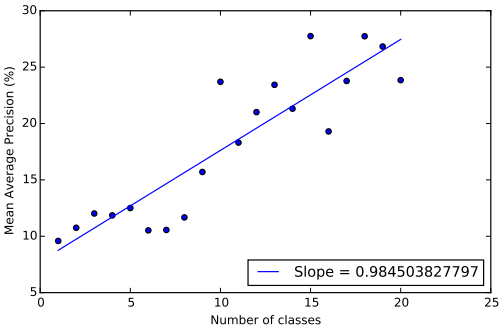
increase, hundreds or thousands, in the amount of examples per class would lead to better retrieval results.

4.2.4 Effects of Increasing Number of Training Classes

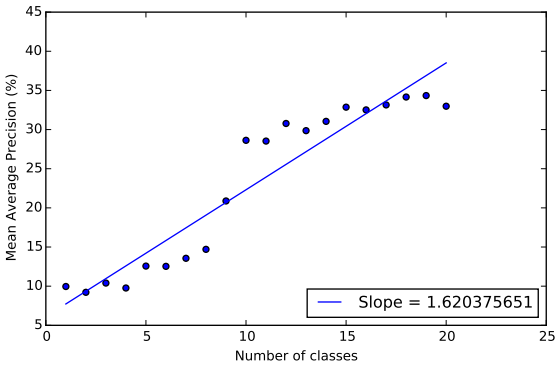
Acquiring more examples for each class is one likely way to improve the performance of the retrieval system, another would be obtaining more distinct classes to increase the known area in the word-space. Distinct in this context meaning classes of objects or concepts that were not previously known in any form rather than finer grain classes, such as 'corgi' when the the class 'dog' is already known. To investigate the effect of the number of classes on retrieval performance, three experiments were performed with the same setup and analysis method as the previous experiment, except this time the number of classes is increased from 1 to the maximum available, 20. This is done for three different magnitudes of examples per class 1, 10, and 100, shown in figure 4.8.



(A) 1 example per class



(B) 10 examples per class



(C) 100 examples per class

FIGURE 4.8: Mean Average Precision results for increasing number of training classes

In all three cases there are significant increases in Mean Average Precision as the number of classes increases. Expectedly the more classes, and the more examples for each class, the better the MAP; compared to the effects of increasing the amount of examples for each class, adding new classes has a much larger impact on the MAP with each addition. This is understandable, if the entire word-space is thought of as a plane and each trained on class-label vector is a known point on the plane, the projection matrix projects points onto the plane in relation to the already known points; so learning new points on the plane would impart a great deal of information about the space, especially when only a few points are already known. However it is probable that there is an upper limit to number of classes

that provide improved accuracy before the space becomes overly crowded causing an increase in false positives.

4.3 Effects of Changing Domain

The proposed image retrieval system projects images into a word-space to create better separation between images for retrieval or classification. Observing class similarity in different domains gives some insight on how changing a domain can improve class separation for easier identification. The expectation is that once images are projected into a word-space by the projection matrix S and compared to the Word2Vec representation of the class-label there will be a clear separation between images of the correct class and all other images; for example pictures of dogs compared to the word 'dog' will be clearly separable from other pictures other objects compared to the same word.

Figure 4.9 contains three images of different classes, 'dog', 'train', and 'sheep', which are clearly identifiable to a human observer. However, in the raw image feature space this is not so obvious.

(A) A dog (B) A train (C) A sheep

FIGURE 4.9: images used for domain comparison

4.3.1 Image Domain

Using 100 images from each of the three classes above, taken from the Pascal VOC dataset, the cosine similarity scores between the image feature vectors for each class and the vectors for the dog class are found. To be more specific, each of the 100 vectors chosen for the sheep and train classes are compared to each of the 100 vectors selected for the dog class, then to find the similarity of in-class images the cosine similarities of between each of the 100 dog vectors to each other are found, excluding the similarity of vectors to themselves which will always be one. The sheep class is chosen to compare images that are somewhat similar and the train class is used to display the image feature vector similarity of classes that are very different. The results are displayed in a set of histograms shown in figure 4.10. Each histogram contains 100 bins between [-1,1].

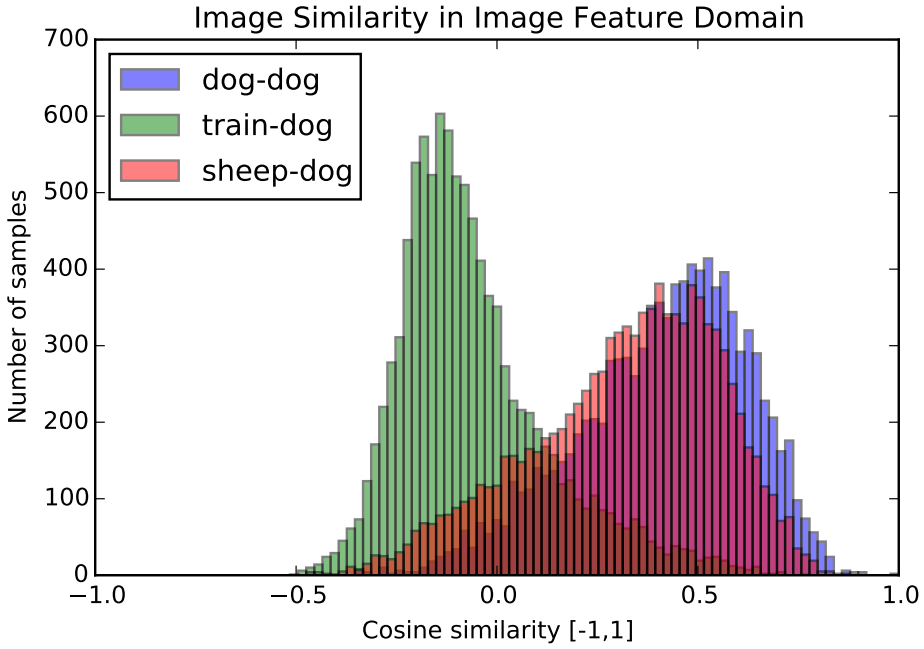


FIGURE 4.10: Image feature vector similarity of three different images

With dog-to-dog similarity plotted in blue, sheep-to-dog in green and train-to-dog in red the histogram shows a normal distribution for the similarity of the three classes to the dog class. The image feature similarity of dog images to other dog images is centered at approximately 0.5, about 50% similarity, with a long tail the reaches back to nearly -0.25. A negative cosine similarity indicated that the two vectors are opposite in direction, which can be interpreted as dissimilar for the domain of these vectors. With a typical similarity of about 50% for images within the same class, as indicated by figure 4.10, the retrieved image feature vectors from MatConvNet do not make a clear indication as to what is contained in the image. Even with a low similarity for in-class images, the train and dog classes are still separable as one would expect, however the when comparing the sheep class to the dog class the similarity distribution is nearly identical to the in-class dog-to-dog similarity. Making it difficult to differentiate between the two classes in the image feature domain.

4.3.2 Images Projected into a Word-Space

By using the dot product between an image feature vector and a projection matrix that has been learned by the RSVM as described in chapter III, image feature vectors can be mapped to points in a word-space. The point in the word-space that the images are mapped to should be close to the class-label associated with that image. The expectation is that this will help to separate images of different classes while grouping ones of the same class. Using the same three classes and the same method as was before the similarity with respect to the dog class was once again found. The resulting histograms are shown together in figure 4.11.

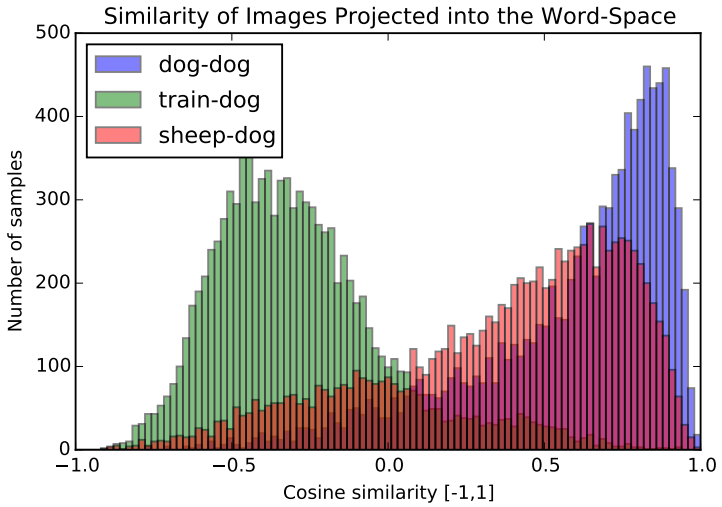


FIGURE 4.11: Cosine similarity of image vectors projected into the word-space

The immediate finding here is that the in-class similarity of projected 'dog' images is much higher; the train class is shifted about 0.25 units away from zero to -0.5. The projection of images into the word-space causes a major spreading of the sheep-to-dog similarity histogram, though there is still considerable amount of overlap, and a widening of the train-to-dog histogram, while the in-class results become shifted towards one.

4.3.3 Comparing Projected Images to Word Vectors

Finally each of the images are projected into a word-space and compared with the word vector representation of 'dog'. This way the desired class should show clean separation from other classes and less similar images should have a lower cosine similarity score. The same 100 images projected into the word-space are used for each of the three classes and their cosine similarities to the 'dog' word vector are reported in figure 4.12. Again the histograms contain 100 bins from [-1,1].

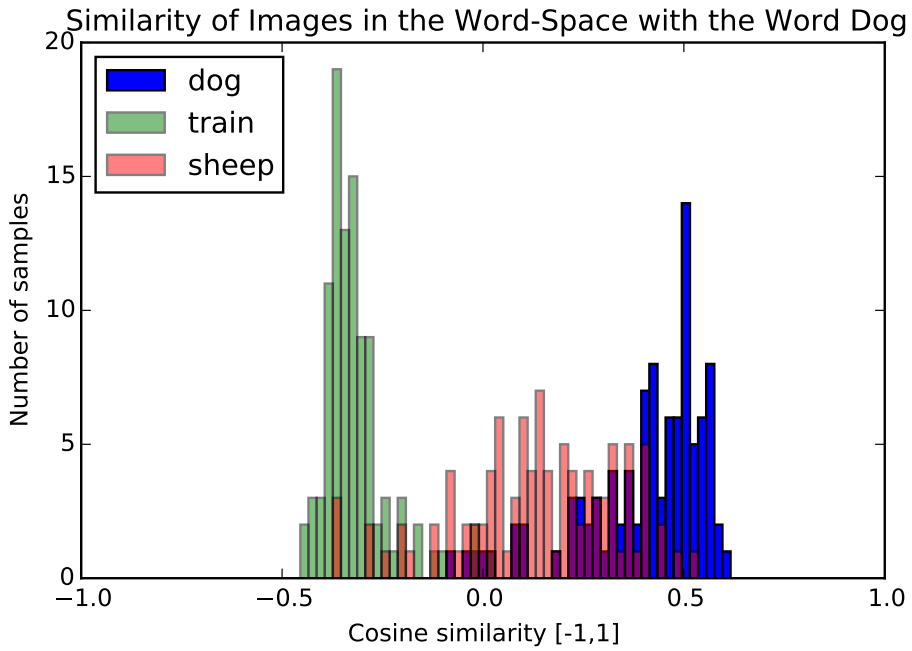


FIGURE 4.12: Cosine similarity of the word vector for 'dog' and three different image-classes projected into the word-space

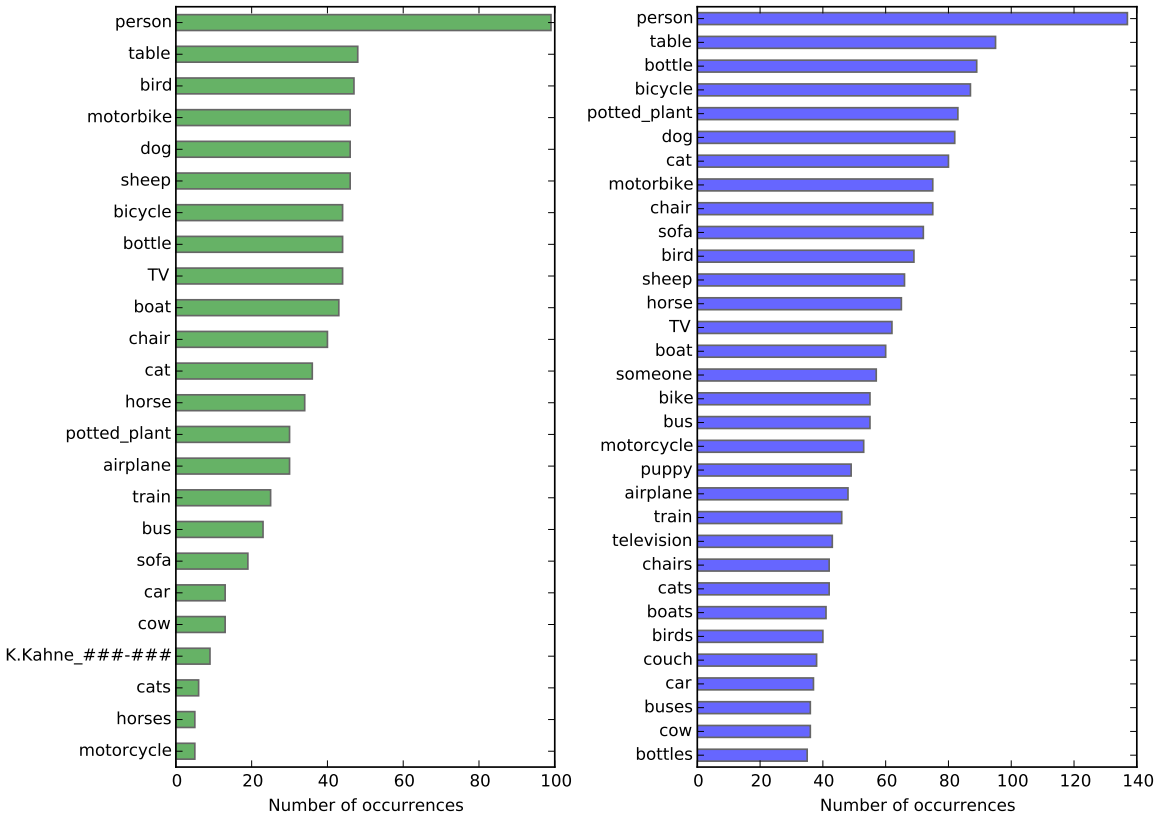
The three classes separate better when compared to the word vector for 'dog'. As expected the images of dogs have the highest similarity to the word vector while the majority of the sheep images are near zero and the pictures of trains have a similarity less than zero. The similarity score for the correct class, dog, has the majority of the images centered at approximatively 0.5. While this is enough to separate the correct class from the incorrect it is lower than expected. A likely cause is the size of the training dataset used; with a larger data set a better projection matrix can be learned that can mapped images closer to the correct point in the word-space.

4.4 What the Transformation Matrix Represents

The final output from the RSVM training is a vector of weights that, when reshaped, becomes the projection matrix used to map image feature vectors, from both known and unknown classes, into the word-space. Once reshaped it is unclear what, if anything, the rows and columns of the matrix represent.

The method in [19] projects images into a semantic space in a similar way to the method used in this thesis. As mentioned in chapter II, the probabilities of an image belonging to 1000 different classes with known semantic embedding vectors are found, then the top N probabilities are used to scale the appropriate embedding vectors before being summed together to create a new vector. If the 1000 known classes are thought of as a matrix, then each row is a word embedding vector and the new vector is found by a simple dot product between the matrix and the image feature vector, a list of probabilities in this case. One reasonable hypothesis then, is that the rows of the projection matrix learned by the RSVM are the embedding vectors of the necessary class-labels needed to project image feature vectors into the word-space.

Figure 4.13 shows the results of finding the most similar word, or words, in the known vocabulary of the Word2Vec model used to each of the rows in the projection matrix plotted as bar graphs, in descending order of number of occurrences for each word. Sub-figure 4.13a shows the results considering only the single most similar word for each row and plotting only those which were a top result five or more times. 4.13b show a similar experiment considering the top five terms most alike each row, this time only plotting those which appeared at least 30 times.



(A) Top 1 (B) Top 5

FIGURE 4.13: Bar plots of top 1(a) and top 5(b) most similar words to the rows of the projection matrix

The twenty highest occurring words in the bar plot (a) are in fact the class-labels from the Pascal dataset, with the 'person' label similar to twice as many of the rows in the matrix; of the remaining four words K.Kahne_###-### is a mostly meaningless token that was learned from the Google News articles used to train the Word2Vec model, and the other three are the plural forms, or synonyms, of the training class-labels. On average the words shown for 4.13a had a similarity of 51.8% to the rows they were most alike. Table 4.7 shows the average cosine similarity score for each word that was the most similar word to a row in the projection

	Average Similarity
chair	0.539803
K.Kahne_###-###	0.324566
sofa	0.520213
motorcycle	0.466810
bicycle	0.576961
boat	0.581447
bird	0.508973
cow	0.507324
table	0.497650
sheep	0.515342
horse	0.512639
motorbike	0.536742
cats	0.419419
potted_plant	0.549266
horses	0.455139
bus	0.569031
train	0.495481
car	0.583734
cat	0.522745
airplane	0.509863
person	0.574499
TV	0.503211
dog	0.628177
bottle	0.54876

TABLE 4.7: Table of average cosine similarity scores for the labels in figure 4.13a

matrix. For example, when 'chair' was the most similar word to a row of the projection matrix it, on average, had a cosine similarity of 53.9%. Looking five words deep for each row in 4.13b shows that, after the class-labels, synonyms or plural forms are the most similar terms, which is to be expected as the terms most nearest a word in the word-space are those with similar meanings, such as the pluralization.

It would seem then that the values learned by the RSVM are approximations of

the embedding vectors of the training labels. These 'anchor' words are used as reference points to map images into a word-space. A likely cause of repetition of some terms and the appearance of nonsense tokens from the word-space is the difference in the number of training class-labels and the size of projection matrix, leaving a large portion of it unused.

Whether this method could lead to greater performance than that in [19] remains to be seen though, as it is not only difficult to directly compare the two, the tasks evaluated are quite different, as [19] evaluates automatic image annotation whereas the method investigated here evaluates image retrieval, but because of the disparity in number of classes available to each model, they use 1000 where only 20 are used here. It is very likely that including more classes during training will lead to the RSVM learning more approximations of relevant terms in the word-space allowing better mappings between the image and word spaces. A potential advantage to learning approximate terms rather than using existing embedding vectors is the RSVM can estimate a shifted version of the words to create better separation between to like terms in the word-space. For example the terms 'cat' and 'dog' have a cosine similarity of 0.76, which can make it difficult to distinguish between images of the two, as an image of a 'cat' mapped slightly too far towards the word 'dog' may cause it to be classified as such. By placing the learned vectors for 'dog' and 'cat' farther away from each other than the actual embedding vectors a slight bias is created when projecting words that may improve separation. The potential disadvantage of this is of course that vectors may be shifted towards other terms causing a loss in similarity to the correct embedding vector.

4.5 Performance on Training Classes

While zero-shot retrieval is the main focus in this thesis performance on seen classes cannot be ignored. Not only is it valuable for the system to be able to identify training classes at a high accuracy, the performance on trained-on classes represents the best case performance of the system.

While the user knows what classes are present in the zero-shot dataset, when training the RSVM it is impossible to predict what classes the system will have to identify in real world use, so instead of tuning the C and epsilon hyper-parameters to the already known zero-shot classes, tuning was done as one would do for a standard classifier, to a cross-validation set with the same classes as the training set.

When training C parameter controls the trade-off between margin size, the distance between the two closest points, and error; epsilon controls the radius of convergence, which has a direct effect on the number of iterations performed. As described in chapter III, the RSVM was trained on a training set made up of 60% of the Pascal VOC data and then the performance was tuned with the training set and the cross-validation set, made up of half of the remaining data, about 20% of the total data. The reason for using both sets is to not over fit the RSVM on any one set of data. If parameter tuning was done only on the training set the RSVM would be able to achieve high scores quite easily by increasing the C parameter, which adds more penalties to miss-classifications causing a stronger fit to the training data, but when used on a new set of images the performance would be poor because of over-fitting. The same applies to using only the cross-validation set. By observing the MAP of both sets the hyper-parameters, C and epsilon, can be tuned

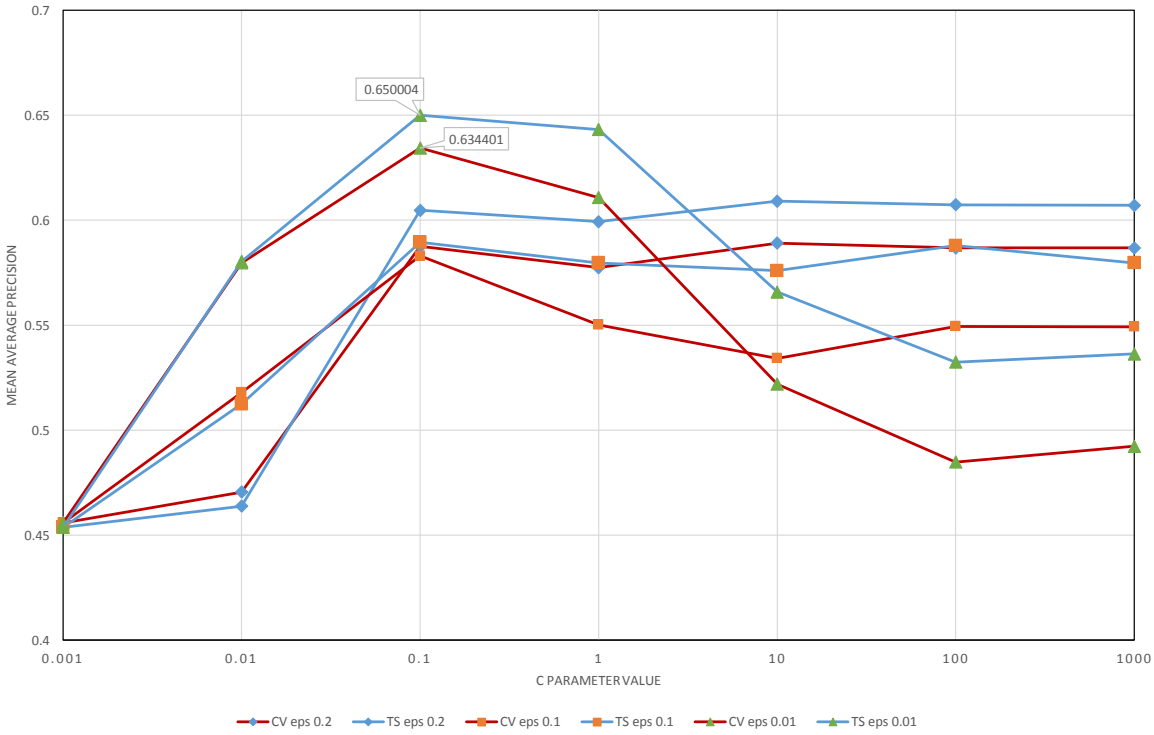


FIGURE 4.14: Plot of MAP scores for the Cross-validation, noted as 'CV' in the legend, and Training sets, noted as TS, at the 3 epsilon values [.2,.1,.01] over 6 decades of C values from 0.001 to 1000

without fitting to either of these datasets to make the learned model more general. The values that result in the highest performance between both sets are taken as the optimal values.

Figure 4.14 shows MAP of the system at different C and epsilon values on the cross-validation set and the training set. In the figure the cross-validation scores are plotted in red and the training set scores in blue, results sharing the same epsilon values are plotted with the same shaped marker; a maximum for both sets is seen at a C value of 0.1 and an epsilon value of 0.01. Call outs on the graph show the maximum score for each set, 65% for the training set and 63% for the cross-validation set.

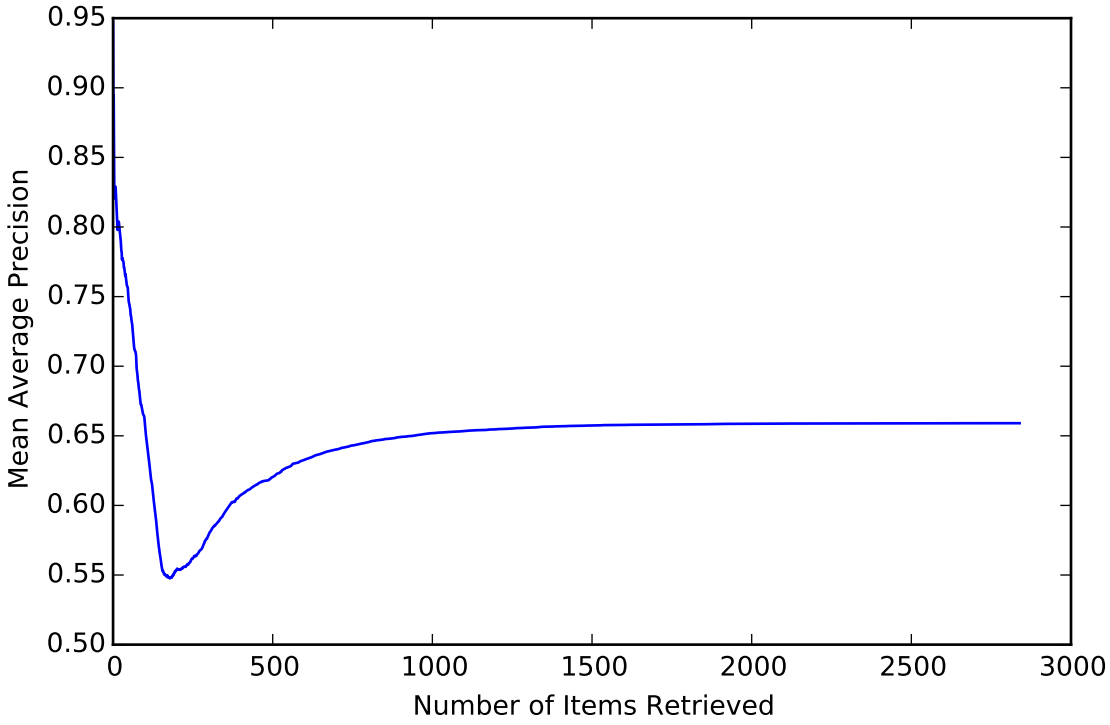


FIGURE 4.15: Graph Mean Average precision of the test set vs return size

Using the selected best C and epsilon combination, 0.1 and 0.01 respectively, performance is measured using the Pascal testing set, which is composed of the remaining 20% of the data. The same method is used as with the zero-shot classes, where the MAP@N is found and plotted for the entire dataset at every position in the list. Figure 4.15 displays the MAP at different selection sizes and table 4.8 summarizes the MAP scores at several points.

A similar downward trend is seen for the first 177 results as was seen with the zero-shot data, the difference here being that the minimum MAP score is only 53% and the performance for the first ten images is a great deal higher, 82%. After a return size of 177 the MAP quickly rises again to its final value of approximately 66% for the full size of the dataset, 2840.

return size	MAP
1	0.947368
5	0.820526
10	0.824378
50	0.746607
100	0.660107
200	0.554058
300	0.578800
400	0.607274
500	0.620319
1000	0.651903
1500	0.657473
2000	0.658678
2840	0.659056

TABLE 4.8: Table of MAP@N scores on seen classes at key points

Table 4.8 shows that the average precision for the first image returned by each class is 95% and 82% for the first ten images from seen classes. The MAP score places weight on ranking position, so a score of about 82% for 10 images either means that on 2 of the 10 images were non-relevant or the one non-relevant image was ranked higher than some of the relevant pictures. The results indicate that the projection matrix learned by the RSVM is able to retrieve images of trained classes on from a database at a high precision with a relatively small amount of training data.

4.6 Comparisons to Other Methods

The work done by the authors in [26] uses a similar SVM based method to the one described in this thesis and while the authors do not consider zero-shot potential the experiments performed on text-based image retrieval provide a useful comparison for the method purposed by this thesis. There are some differences between the two methods however, the authors in [26] use a Structural SVM and

learn their own image and text representations, which are both probability distributions over learned visual and textual topics respectively; training and testing are done on the IAPR TC-12 dataset [6], which is a collection of 20,000 natural images ranging from animals to 'action-shots' with the labels for each image taken as the nouns in the provided free-flowing text describing it. Whereas this thesis uses a Ranking SVM with deeply learned image features and word embedding vectors, and evaluates retrieval from the Pascal dataset, which has only 20 classes, though multiple classes may appear in a single image.

The method in [26] uses the inner product of the weights learned by the SSVM and the joint-feature vector of image and word vectors, $\Phi(\cdot)$, to determine relevance, where higher scores are more relevant, whereas in this paper the weights are reshaped into a matrix to first project image feature vectors into a word-space where the cosine similarity is used to determine relevance. In both methods the joint-feature vector $\Phi(\cdot)$ used for training is the tensor product of image and word vectors, that is $\Phi(\vec{I}, \vec{T}) = \vec{I} \otimes \vec{T}$.

In one experiment in particular they report the results of an image retrieval experiment where given a text label images are ranked according to relevance. Performance is reported with precision at 1 (P@1), precision at 5 (P@5) and MAP score, mean Average Precision; they do not report how many images were retrieved for the MAP score but it is likely that this is the MAP for the entire set of images. Table 4.9 compares these results with those of the purposed method.

	P@1	P@5	MAP
BITR	11%	8%	5%
Proposed	95%	85%	66%

TABLE 4.9: Comparison of [26]'s method BITR and the purposed method

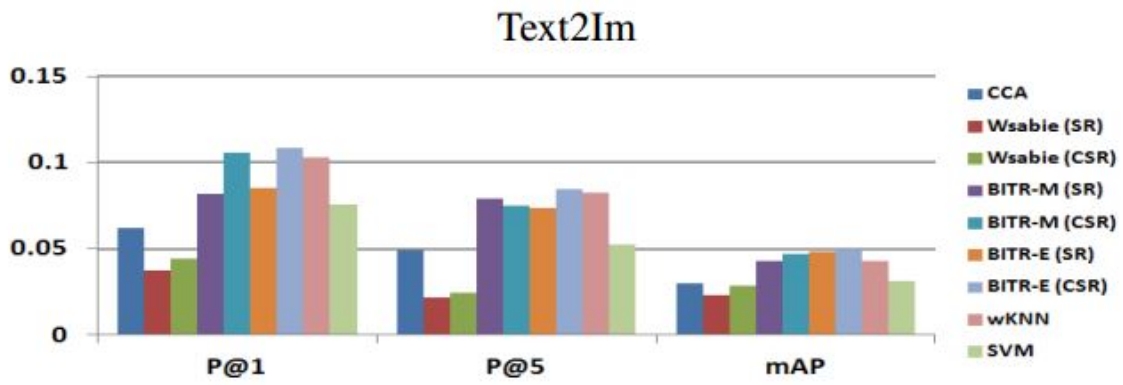


FIGURE 4.16: Bar graphs comparing several image retrieval results [26]

Figure 4.16 shows the results of several other image retrieval systems that used text labels to retrieve images, Text2Im, that were compared with the results from [26] in their paper. The purposed retrieval method out scores [26]'s methods, all of the BITR bars in figure 4.16, and the various other methods they compare to by significant amounts in every reported precision metric. A likely reason for the large difference in performance is the choice of datasets for training and evaluation. The IAPR TC-12 dataset consists of many different categories and uses free-flowing text to describe them, making the dataset much more complex, where the Pascal dataset only has 20 categories and uses only the labels to describe the images making the image labeling much more consistent. Another likely reason is the difference in image and word representations. The word-space used in this thesis was trained on the Google news dataset which included about 100 billion words to learn vector representations of words, [26] does not state the number of words used to learn their text representation for labels.

Comparing results to other zero-shot methods is difficult as most other methods

evaluate on classification results instead of retrieval. One aspect that is comparable across all zero-shot methods however is the size of the training data used. In comparison to other methods cited in this document, [14, 23, 19, 4], both in terms of the number of classes and examples per class, the method purposed here uses very little data. The closest method in comparison uses the same number of classes but approximately six times the number of pictures, and evaluates on classification results from the same zero-shot classes [14].

4.7 Viability

The image retrieval system purposed in this thesis shows high performance at retrieving images from a text label and has superior performance compared to several other similar methods in [26], though the difference in dataset complexity must be considered. The system also displays encouraging results at retrieving zero-shot images, those belonging to classes that the RSVM has never trained on or received any information about. The performance is exciting given the small amount of training data, especially that of the zero-shot which will likely improve with additional training data.

With acceptable performance on trained-on image classes adding the ability to make a reasonable zero-shot attempt adds considerable value to an image retrieval system. Having zero-shot capabilities makes the system much more robust, as in the case of text-based retrieval it allows the user to search with the most appropriate term rather than specific training labels. Being able to project images into the same space as text makes operations between the two much easier and allows for applications that extend beyond retrieval, such as automatic annotation.

The three main blocks that compose this method, the word-space model chosen, the type of image features used, and the SVM, are all separate from each other allowing any one to be exchanged for a different model or algorithm easily for further experimentation or different applications. The process for determining relevancy of an image is a simple matrix inner product and a normalized vector inner product, the cosine similarity, which requires no extra steps aside from sorting scores from highest to lowest. The training and evaluation require little in terms of computer resources and can be performed on a home computer system. Training takes only minutes using 20 classes with 100 images each and the mathematical operations for determining relevance are simple, making experimentation, implementation, and evaluation available to users without the use of high-end hardware or supercomputers.

Currently this is a supervised training method, where the groundtruth label must be provided with the training images. However, since the words used as label are considered noisy they need not be expertly labeled. This allows for the possibility of using large amounts of 'casually' labeled data, data that is not purposefully labeled for high accuracy, for example images from flikr where users of the app tag the pictures that they take. This would provide a large source of easy to obtain noisily labeled images from many different categories to train the RSVM with.

In this document the images from each dataset were transformed into their respective image feature vectors before performing retrieval. To make the system real world implementable additional code would be needed to extract the feature vector for images from an arbitrary source, in order to project the pictures into a word-space. This step would likely be application dependent and its implementation would be left up to the user.

Chapter V

Conclusion

5.1 Conclusion

An evaluation of a method for retrieving images which had no labeled data available for training from a database, based on the similarity between the images and the text search term in the same semantic space, was performed. The results show that method is able to achieve better than random results for most classes at small retrieval sizes and better than random for all classes at retrieval sizes greater than 400. While there is no other zero-shot method to directly compare with, the performance on unknown classes is exciting.

Several training factors were investigated to determine avenues of further experimentation that are likely to improve zero-shot results. The size of the training data used, both in terms of number of images per class and the amount of classes, seems to have a direct impact on the zero-shot performance and results indicate that increasing the amount of training data would be a simple way to improve overall performance, the caveat to this being that increased data will lead to an increase in training time and for big data, with thousands of classes and images, use of supercomputing may be necessary.

Experiments were performed on the projection matrix learned by the RSVM to determine what the RSVM is learning from the training data. Results indicated that the rows of the matrix are learned word vectors from the training classes, much like those in [19]; based on the number of rows that all learn the same word vector approximation the results support the idea that more training data will improve performance by allowing the RSVM to learn different word vector approximations for each row.

Finally the purposed method was compared to another label based image retrieval paradigm in terms of performance on seen training classes. The purposed method was found to have significantly greater performance, though the difference in datasets used must be considered. With a precision of 95% for the first image returned however, the results are impressive enough to encourage further development of the system.

5.2 Future Work

Evaluation of the zero-shot performance was done in a 'clean' space consisting of only the zero-shot images, further testing should be done using a 'dirty' space where both trained-on and zero-shot images are present. Results for this experiment will give a better indication of how the method will perform in a real world environment and how well the method can differentiate between trained-on classes and new zero-shot ones.

Zero-shot performance should be re-evaluated using large amounts of data as there is evidence that a substantial increase in the amount of training data can lead

to significant improvements in zero-shot accuracy. The results should be analyzed for varying amounts of data to determine at which point the increased amount of data begins to give only minor improvements for the increase in training time.

The method could also be modified to accept multiple words or phrases as the search term to allow for finer retrieval. Because of the nature of the semantic space multiple word vectors can be averaged together from a search phrase to obtain a new aggregate vector that can be compared to images projected into a word-space in the same manner as before to determine relevance. The effect of this on zero-shot accuracy should be analyzed as there may be other ways to combine word vectors, such as weighting those which are considered to be more important or informative, that lead to better results. The process could also be reversed to allow for automatic image annotation based on the closest word vectors to a given image projected into a word-space.

It would also be interesting to use different word-space models and image feature vectors to observe their effects on performance. One worthwhile experiment may be to use the popular scale-invariant feature transform, SIFT, features to represent images [15]; because the method purposed here makes no assumption on the image feature vectors used the user can train the RSVM using any image features. SIFT features may be a good choice because of their robustness, easy of computation and popular use in object recognition.

5.2.1 Structural SVM Implementation

The method described in this paper is implemented with a Ranking SVM because the final goal was to return a ranked list of results, but for other applications it may

be more beneficial to use a standard Structural SVM, SSVM. When implementing a SSVM the most violated constraint of all the training examples must be calculated and added to the current working set of constraints. The optimization problem the SSVM solves is as follows:

$$\min_{w,\xi \geq 0} \frac{1}{2}\|w\|^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i$$

$$\text{s.t. } w \cdot \Phi(x_i, y_i) - w \cdot \Phi(x_i, \bar{y}) \geq \Delta(y_i, \bar{y}) - \xi_i (\forall i, \bar{y} \neq y_i)$$

Where y_i is the correct label for input x_i, $\bar{y}$ is an incorrect label, ξ is a slack-variable, $\Delta(y_i, \bar{y})$ is the loss for predicting $\bar{y}$ when the correct label is y_i, and w is the weight vector learned by the SSVM. In this paper the joint-feature vector $\Phi(x, y)$ is the tensor product of the input x and output y. The SSVM calculates the joint-feature vector multiple times each iteration and the tensor product is expensive to calculate, however, as shown in equations 3.4 and 3.5 the product of the tensor product and the weight vector can be rewritten in a bilinear form that requires only two inner products, which is much easier to compute. As most SSVM code is written to work with weight vectors and not a matrix a change would have to be made to the SSVM code to implement this speed up. This change would need to be made by the user but for implementations or experiments done using a SSVM this small change would result in a considerable speed up in training time especially as the number of training examples used grows.

Bibliography

[1] Zeynep Akata et al. "Label-Embedding for Attribute-Based Classification". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2013.

[2] M. Everingham et al. *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*. URL: http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[3] Alireza Farhadi et al. "Describing objects by their attributes". In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pp. 1778–1785.

[4] Andrea Frome et al. "DeViSE: A Deep Visual-Semantic Embedding Model". In: *Advances in Neural Information Processing Systems 26*. Ed. by C.J.C. Burges et al. Curran Associates, Inc., 2013, pp. 2121–2129. URL: http://papers.nips.cc/paper/5204-devise-a-deep-visual-semantic-embedding-model.pdf.

[5] David Graff et al. "English gigaword". In: *Linguistic Data Consortium, Philadelphia* (2003).

[6] Michael Grubinger et al. "The iapr tc-12 benchmark: A new evaluation resource for visual information systems". In: *International Workshop OntoImage*. Vol. 5. 2006, p. 10.

[7] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *arXiv preprint arXiv:1512.03385* (2015).

[8] Eric H Huang et al. “Improving word representations via global context and multiple word prototypes”. In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*. Association for Computational Linguistics. 2012, pp. 873–882.

[9] Thorsten Joachims. “Optimizing search engines using clickthrough data”. In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2002, pp. 133–142.

[10] Thorsten Joachims et al. “Predicting structured objects with support vector machines”. In: *Communications of the ACM* 52.11 (2009), pp. 97–104.

[11] Davis E. King. “Dlib-ml: A Machine Learning Toolkit”. In: *Journal of Machine Learning Research* 10 (2009), pp. 1755–1758.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[13] Hugo Larochelle, Dumitru Erhan, and Yoshua Bengio. “Zero-data Learning of New Tasks.” In: *AAAI*. Vol. 1. 2. 2008, p. 3.

[14] Xin Li, Yuhong Guo, and Dale Schuurmans. “Semi-Supervised Zero-Shot Classification with Label Representation Learning”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 4211–4219.

[15] David G Lowe. “Object recognition from local scale-invariant features”. In: *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*. Vol. 2. Ieee. 1999, pp. 1150–1157.

[16] Laurens Van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of Machine Learning Research* 9.2579-2605 (2008), p. 85.

[17] Tomas Mikolov et al. "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[18] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[19] Mohammad Norouzi et al. "Zero-shot learning by convex combination of semantic embeddings". In: *arXiv preprint arXiv:1312.5650* (2013).

[20] Mark Palatucci et al. "Zero-shot learning with semantic output codes". In: *Advances in neural information processing systems*. 2009, pp. 1410–1418.

[21] Radim Řehůřek and Petr Sojka. "Software Framework for Topic Modelling with Large Corpora". English. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. http://is.muni.cz/publication/884893/en. Valletta, Malta: ELRA, May 2010, pp. 45–50.

[22] Bernardino Romera-Paredes and PHS Torr. "An embarrassingly simple approach to zero-shot learning". In: *Proceedings of The 32nd International Conference on Machine Learning*. 2015, pp. 2152–2161.

[23] Richard Socher et al. "Zero-shot learning through cross-modal transfer". In: *Advances in neural information processing systems*. 2013, pp. 935–943.

[24] Ioannis Tsochantaridis et al. "Support vector machine learning for interdependent and structured output spaces". In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 2004, p. 104.

[25] A. Vedaldi and K. Lenc. "MatConvNet – Convolutional Neural Networks for MATLAB". In: 2015.

[26] Yashaswi Verma and CV Jawahar. “Im2Text and Text2Im: Associating Images and Texts for Cross-Modal Retrieval”. In: *Proceedings of the British Machine Vision Conference. BMVA Press*. 2014.

Appendix A

GetPascalFeatsScript.m

```
%% Pascal VOC data extraction script
%
%
% Evan Novotny
% 9/25/2015
%
% This script creates a data file that has the following contents:
% filename: someimage.jpg
% tags: some extraced tags ... lasttag
% scores: # # # # # # # ... #
% ...
% filename is the name of the image file that has the following tags
    listed
% in tags: and vgg feature score listed in scores:
%
% The VGG featrues are taken from the layer preceeding the softmax layer
%
% what this file does:
%   This file extracts the relevant tags for a pascal VOC dataset image
%   from the provided annotation xml files and then uses matconvnet to
%   calculate the vgg features for the image. These are combined in the
%   output file contating data for each of the images in the dataset to
```

```
%   create a .txt file with the format above, which is readable by the
%   Xdata python object.
%   Specifically this script first extracts all of the tags for each
    image
%   and removes any duplicates and stores the results in a cell array,
    then
%   calculates the vgg scores for each image and writes them with the
%   appropriate tags to a .txt file.

%% set-up path and file variables
% tic %uncomment for timing

%change directories to the one containing the Pascal VOC dataset
cd D:\Thesis\training_data\Pascal\VOCdevkit\VOCcode
%path to the xml files
xmlPATH = 'D:\Thesis\training_data\Pascal\VOC2012\Annotations\';
%directory containing the xml files
xml_files = dir('D:\Thesis\training_data\Pascal\VOC2012\Annotations\*.
    xml');

% path to images
imPATH = 'D:\Thesis\training_data\Pascal\VOC2012\JPEGImages\';
im_list = dir('D:\Thesis\training_data\Pascal\VOC2012\JPEGImages\*.jpg')
    ;

%file to write to
output_file = 'D:\Thesis\training_data\pydata_no_sofmax.txt';
%% extract tags from xml files
img_cell = cell(length(xml_files),3); %container for all the tags

```

```
sprintf('Extracting annotations ...')
for i = 1:length(xml_files)
    xml_soup = VOCreadxml([xmlPATH,xml_files(i).name]);
    numtags = length(xml_soup.annotation.object); %get number of classes
     present
    tags = cell(1,numtags);

    imgname = xml_soup.annotation.filename;
    img_cell{i,1} = imgname;

    for j=1:numtags
        tags{j} = xml_soup.annotation.object(j).name;
    end
    img_cell{i,2} = unique(tags); %remove duplicates

end
sprintf('Done!\n')

%% get vgg scores

%change directories to the MatConvNet directory
cd 'D:\Libraries\Documents\GitHub\matconvnet'
sprintf('Running matconvnet setup...')
run matlab/vl_setupnn
sprintf('Done!\n')

%load our net
sprintf('Loading nueral net moodel...')
net = load('imagenet-caffe-alex.mat') ;
```

```
% Get feature layer before softmax (last layer)
net.layers = net.layers(1:end-1);
sprintf('Done!\n')

sprintf('Calculating vgg scores...')
N = numel(im_list); % get number of images in directory
for i=1:N
    imgname = im_list(i).name;
    im = imread([imPATH imgname]) ;
    im_ = single(im) ; % note: 255 range
    im_ = imresize(im_, net.normalization.imageSize(1:2)) ;
    im_ = im_ - net.normalization.averageImage ;

    % run the CNN
    res = vl_simplenn(net, im_) ;

    % get classification scores
    scores = squeeze(gather(res(end).x)) ;

    %can't be totally sure images were readin in the same order as
    %annotations, so find the cell index that contains the proper image
    %name and store the score in the scores column
    idx = find(strcmp(img_cell(:,1),imgname));
    img_cell{idx,3} = scores';

end
sprintf('Done!\n')

%% write to file
```

```
%name of file
fileID = fopen(output_file,'wt');
sprintf('writing to file...')
for i = 1:N
    fprintf(fileID,['filename: ',img_cell{i,1},'\n','tags: ']);
    %write all associated tags
    for j = 1:length(img_cell{i,2})
        fprintf(fileID, [img_cell{i,2}{j},' ']);
    end
        fprintf(fileID,['\n','scores: ',num2str(img_cell{i,3}),'\n']);
end
fclose(fileID);
sprintf('Done!\n')
% toc %uncomment for timming
```

Appendix B

GetaYahooFeats.m

```
%% aYahoo data extraction function

function [ fileID ] = GetaYahooFeats( Path, output_file )
%Function to extract image features of the aYahoo data set
% Variables:
% Path - a string path to the folder containing the aYahoo images
% ouput_file - the path the the file to output results to
%
%   Provide the path to a folder containing the images with the Path var
    .
%   VGG features will be extracted using matconvnet and placed output to
%   a text file with the path provided in output_file
%   The data is formatted for the Xdata python object with:
%   filename: someimage.jpg
%   tags: some extraced tags ... lasttag
%   scores: # # # # # # # ... #
%   ...
%   The VGG featrues are taken from the layer preceeding the softmax
    layer
%
%   For aYahoo the tag is extracted from the filename as the word before
%   the underscore. So bag_21 will take the word 'bag' as its tag. This
```

```
%   function would be appropriate for other image sets in this format.
22
%  Evan Novotny 1/20/2016
24
%% get vgg scores
26
cd 'D:\Libraries\Documents\GitHub\matconvnet'
28 sprintf('Running matconvnet setup...')
run matlab/vl_setupnn
30 sprintf('Done!\n')

32 %load our net
sprintf('Loading nueral net moodel...')
34 net = load('imagenet-caffe-alex.mat') ;
sprintf('Done!\n')
36
% obtain and preprocess an image
38 imPATH = Path;
im_list = dir([Path '*.jpg']);
40
sprintf('Calculating vgg scores...')
42 N = numel(im_list);

44 % create a 3 column cell to hold the filename, tags and vgg score for
    each
% image
46 img_cell = cell(N,3);


48
% Get feature layer before softmax (last layer)
```

```
net.layers = net.layers(1:end-1);

for i=1:N
    imgname = im_list(i).name;

    %get image name and tag
    img_cell{i,1} = imgname; %get image filename
    tags = strsplit(imgname,'_'); %split name at underscore
    img_cell{i,2} = tags{1};    %the first word before _ becomes the tag

    im = imread([imPATH imgname]) ;
    im_ = single(im) ; % note: 255 range
    im_ = imresize(im_, net.normalization.imageSize(1:2)) ;
    im_ = im_ - net.normalization.averageImage ;

    % run the CNN
    res = vl_simplenn(net, im_) ;

    % get classification scores
    scores = squeeze(gather(res(end).x)) ;

%     store the score
    img_cell{i,3} = scores';

end
sprintf('Done!\n')


%% write to file
fileID = fopen(output_file,'wt');
```

```
80 sprintf('writing to file...')
   for i = 1:N
82     fprintf(fileID,['filename: ',img_cell{i,1},...
           '\n','tags: ',img_cell{i,2},...
84         '\n','scores: ',num2str(img_cell{i,3}),'\n']);

86 end
   fclose(fileID);
88 sprintf('Done!\n')

90 end
```

Appendix C

monte_carlo_sims.m

```
% Monte carlo simulations for retrieving images from a database.
% Each class is reduced to a binary yes/no problem where there are
% X relevant images in the database which totals Y images
% therefore the probaility of choosing a relevant image is X/Y
% each time the database is sampled without replacement
% this is acomplished in matlab using the randsample function which
    samples
% a vector of values 1:n Z times without replacement
% this way the first X values are considered positive for a class and
% the remaining values are negative

% Evan Novotny
% 4/12/2016

%% initialize variables and allocate containers
%number of examples for each class
building = 213;
donkey =  128;
monkey =  156;
mug =  210;
centaur =  48;
bag =  279;
```

```
carriage =  147;
wolf =  171;
zebra =  179;
statue =  191;
jetski =  366;
goat =  149;

% the break points for positive/negative for each class
break_points = [building donkey monkey mug centaur bag carriage ...
    wolf zebra statue jetski goat];

size_database = sum(break_points);

%number of times to run random sampling
num_experiments = 10000;

%create array to hold monte carlo results
monte_carlo = zeros(num_experiments, size_database);

%create a structure to hold the results of monte carlo for each class
results = struct(   'building',zeros(num_experiments, size_database),...
                    'donkey',zeros(num_experiments, size_database),...
                    'monkey',zeros(num_experiments, size_database),...
                    'mug',zeros(num_experiments, size_database),...
                    'centaur',zeros(num_experiments, size_database),...
                    'bag',zeros(num_experiments, size_database),...
                    'carriage',zeros(num_experiments, size_database),...
                    'wolf',zeros(num_experiments, size_database),...
                    'zebra',zeros(num_experiments, size_database),...
                    'statue',zeros(num_experiments, size_database),...
```

```
                    'jetski',zeros(num_experiments, size_database),...
                    'goat',zeros(num_experiments, size_database)...
                );
% get the field names for the results
fns = fieldnames(results);

%% Monte Carlo
for i = 1:num_experiments

    %take a random sampling of the database. This can be done once for
    all
    %classes as only the number of positives change
    monte_carlo(i,:) = randsample(1:size_database,size_database);

end

%% Reduce results for each class to Precision@n
for j = 1:numel(fns)

% reduces the simulation result to a binary matrix for values less than
% or equal to the break point of the current class
binary = (monte_carlo <= break_points(j));

%index structure row 1, field j(a matrix of results) = cumulative
%sum of each row of the binary matrix
results(1).(fns{j}) = cumsum(binary,2);
end

%% Calculate precision and recall for each class
```

```
precision = struct(    'building',zeros(1, size_database),...
                       'donkey',zeros(1, size_database),...
                       'monkey',zeros(1, size_database),...
                       'mug',zeros(1, size_database),...
                       'centaur',zeros(1, size_database),...
                       'bag',zeros(1, size_database),...
                       'carriage',zeros(1, size_database),...
                       'wolf',zeros(1, size_database),...
                       'zebra',zeros(1, size_database),...
                       'statue',zeros(1, size_database),...
                       'jetski',zeros(1, size_database),...
                       'goat',zeros(1, size_database)...
                   );
recall = struct(    'building',zeros(1, size_database),...
                       'donkey',zeros(1, size_database),...
                       'monkey',zeros(1, size_database),...
                       'mug',zeros(1, size_database),...
                       'centaur',zeros(1, size_database),...
                       'bag',zeros(1, size_database),...
                       'carriage',zeros(1, size_database),...
                       'wolf',zeros(1, size_database),...
                       'zebra',zeros(1, size_database),...
                       'statue',zeros(1, size_database),...
                       'jetski',zeros(1, size_database),...
                       'goat',zeros(1, size_database)...
                   );

sample_array = 1:size_database; %an array of ideal rank  1 to end

%continers for simulation mean and standard deviation
```

```
111 sim_mean = zeros(1,size_database);
    sim_std = zeros(12,size_database);
113
    %average precision container
115 AP = zeros(12,size_database);

117 %for each class find the mean result of random ranking and calculate the
    %mean precision and recall at each sample point for the average randomly
119 %ranked list
    for j = 1:numel(fns)
121
        sim_mean = round(mean(results(1).(fns{j})));
123     sim_std(j,:) = std(results(1).(fns{j}));

125     precision(1).(fns{j}) = sim_mean./sample_array;
    %       precision(1).(fns{j}) = mean(bsxfun(@rdivide,results(1).(fns{j}),
        sample_array));
127     recall(1).(fns{j}) = sim_mean/break_points(j);
    %       recall(1).(fns{j}) = mean(results(1).(fns{j})/break_points(j));
129
    %  measure Average Precision, the recall increases monotonically, so
        when
131 %  there is an increase there is a hit. This increases the running sum
        rsum of
    %  precision. Finally divide the rsum for P@n by the minimum(# of
133 %  positive samples, # of samples returned which is the position in the
    %  ranked list)
135     prev_val = 0; %reinitialize for each class
        rsum=0;
137     for k  =1:length(sample_array)
```

```
        %when recall increases there's a hit
        if prev_val < recall(1).(fns{j})(k)
            rsum = rsum + precision(1).(fns{j})(k);
            prev_val = recall(1).(fns{j})(k);
        end
        AP(j,k) = rsum/min(k,break_points(j));

    end
end
 %calculate MAP score for Average Precisions
MAP = mean(AP);

%% Plot results

%plot precision vs position in ranked list (# of samples returned)
for i = 1:numel(fns)
    figure(i)

    %create secondary axes for sample values
    b=axes('Position',[.1 .1 .8 1e-12]);
    set(b,'Units','normalized');
    set(b,'Color','none');
    xlabel('Samples')

    %create primary axes
    a=axes('Position',[.1 .2 .8 .7]);
    set(a,'Units','normalized');

    %get unique values and indecies
    [U,ia,ic] = unique(recall.(fns{i}));
```

```
169     %stair plot precision vs recall
        stairs(recall.(fns{i})(ia),precision.(fns{i})(ia))
171     set(b,'xlim',[1 max(sample_array(ia))]);
        title(fns{i});
173     xlabel('Recall')
        ylabel('Precision')
175
        % save figure to location given
177 %      saveas(figure(i),strcat('D:\Thesis\GIT\latex\Figures\monte_carlo_
    ',fns{i}),'pdf')

179 end

181 %Plot MAP scores
    figure(i+1)
183 stairs(MAP)
    xlim([1 size_database])
185 title('Mean Average Precision of Random Retrieval')
    xlabel('Samples')
187 ylabel('Mean Average Precision')
    % save figure to location given
189 % saveas(figure(i+1),'D:\Thesis\GIT\latex\Figures\MAP_random',pdf')
```

Appendix D

training_data_filtering_script.py

```
# coding: utf-8

# This script filters the output of the Matlab file data_munge so that 'person'
# is not the dominate class and classes that are missing word labels are
# converted to an equivalent class. All picture containing only the person class
# are removed then training data is created using images that do not contain
# people for the person class, half contain only people and half contain people
# with other classes.
#
# Evan Novonty
# 4/12/2016

# script from Ipython notebook python_make_training_data.ipynb
import random, math
random.seed(2015)
import numpy
import Xdata
```

```
19 all_data = Xdata.Xdata(data_file='D:/Thesis/training_data/
       pydata_no_softmax.txt') #load in a set of all the images with tags,
       and image features

21 data_length = len(all_data.names)

23 tags = ['airplane','car','bus','train','boat','motorbike','bicycle','
       person','cat','dog','horse','sheep','cow','bird','potted_plant','
       chair','table','sofa','TV','bottle']

25 #replace tags that aren't in the word2vec model with ones that are
   for i in range(0,data_length):
27     all_data.tags[i] = ['airplane' if x == 'aeroplane'
                           else 'potted_plant' if x == 'pottedplant'
29                         else 'TV' if x == 'tvmonitor'
                           else 'table' if x == 'diningtable'
31                         else x for x in all_data.tags[i]]

33 noppl =[idx for idx,tag_set in enumerate(all_data.tags) if 'person' not
       in tag_set]

35 #get index sets of only people tags and mixture with people tags

37 onlyppl = [idx for idx,tag_set in enumerate(all_data.tags) if 'person'
       in tag_set and len(tag_set)==1]
   #print len(onlyppl)
39 withppl = [idx for idx,tag_set in enumerate(all_data.tags) if 'person'
       in tag_set and len(tag_set)>1]
   #print len(withppl)
41
```

```
#initialize some constants

#taken from the smallest represented class to give a even class
    representation
total_per_class = 238
# 60% for training ~142 each
# 20% for crossvalidating and testing ~47 each
num4training = int(math.floor(total_per_class*0.6))
num4crossvalidate = int(math.floor(total_per_class*0.2))
num4test = int(math.floor(total_per_class*0.2))

#create train test cv sets note: these are lists of indecies
training_idx = list()
CV_idx = list()
test_idx = list()

#for person tag get half with only ppl and half mixed with other classes
person_samples = int(math.floor(total_per_class/2.0))

#contruct training, cross validation and testing sets by taking random
    samples
#from each class
for tag in tags:
#     print tag
    class_total = list()

#for a nonperson class take a sample for the total number of images used
    ,
#then divide that into 60/20/20 for train cross validate and test sets
    if tag != 'person':
```

```
69          imgs_with_tag = [idx for idx in noppl if tag in all_data.tags[
    idx]]

71          #random sampling of total_per_class images with desired tag.
    returned
            #in sample order so doesn't need to be sampled again to keep
    randomness
73          class_total.extend(random.sample(imgs_with_tag,total_per_class))

75          training_idx.extend(class_total[0:num4training])
            CV_idx.extend(class_total[num4training:num4crossvalidate+
    num4training])
77          test_idx.extend(class_total[num4crossvalidate+num4training:])

79      #for the person tag just take half the samples from the list of
    images with
        #only people and half from the one that is a mixture same 60/20/20
    division
81      else:
            class_total.extend(random.sample(withppl,person_samples))
83          class_total.extend(random.sample(onlyppl,person_samples))
            random.shuffle(class_total) #shuffle the person class for
    slicing
85          training_idx.extend(class_total[0:num4training])
            CV_idx.extend(class_total[num4training:num4crossvalidate+
    num4training])
87          test_idx.extend(class_total[num4crossvalidate+num4training:])

89  f = open('D:/Thesis/training_data/seperated_pascal_training_nsm.txt','w'
    )
```

```
for i in training_idx:
    #numpy array of scores to string
    rel = numpy.array_str(all_data.scores[i])
    rel = rel.translate(None,"[]\n")

    #join elements seperated by a space
    labels = ' '.join(all_data.tags[i])

    #write filename,tags,score to file
    f.write('filename: '+all_data.names[i]+'\n'
            +'tags: '+labels+'\n'
            +'scores: '+rel+'\n')#no space for scores
f.close()

f = open('D:/Thesis/training_data/seperated_pascal_CV_nsm.txt','w')
for i in CV_idx:
    #numpy array of scores to string
    rel = numpy.array_str(all_data.scores[i])
    rel = rel.translate(None,"[]\n")

    #join elements seperated by a space
    labels = ' '.join(all_data.tags[i])

    #write filename,tags,score to file
    f.write('filename: '+all_data.names[i]+'\n'
            +'tags: '+labels+'\n'
            +'scores: '+rel+'\n')
f.close()

f = open('D:/Thesis/training_data/seperated_pascal_test_nsm.txt','w')
```

```
for i in test_idx:
    #numpy array of scores to string
    rel = numpy.array_str(all_data.scores[i])
    rel = rel.translate(None,"[]\n")

    #join elements seperated by a space
    labels = ' '.join(all_data.tags[i])

    #write filename,tags,score to file
    f.write('filename: '+all_data.names[i]+'\n'
            +'tags: '+labels+'\n'
            +'scores: '+rel+'\n')
f.close()
```

Appendix E

RSVM_training_script.py

```
# Evan Novotny
#4/12/2016

# coding: utf-8

# script for training the RSVM. The weights are output to the file
    outfile as
# a numy array.

import Xdata
import CD_rank_svm

#load the pascal training data into a Xdata object
train_data = Xdata.Xdata(data_file='D:/Thesis/training_data/
    seperated_pascal_training_nsm.txt')

train_data.load_word_space() #load a knowledge base

# output path for the training file being created
training_file = 'D:/Thesis/training_data/CDSVM_training/
    num_examples_compare/nsm100.txt'
```

```
# use the pascal training set to create a training file readable by the RSVM trainer
train_data.export_training_data(num_examples=100,out_file=training_file)

del train_data #free up memory

#create the trainer with desired C and epsilon values
#create the joint feature representation and save vectors into queries
[trainer,queries] = CD_rank_svm.create_CDSVM_rank_trainer(C=0.1,eps=0.01,
                                                max_iter=10000,infile=training_file)
# output file to save learned weights to
outfile = 'D:/Thesis/training_data/CDSVM_models/no_softmax_models/C0.1_eps0.01_samp100.txt'

#train the RSVM using the previously created trainer and queries
CD_rank_svm.train_CDSVM_rank(trainer,queries,outfile=outfile)
print 'model saved to '+outfile
```

Appendix F

parameter_tuning_script.py

```
# -*- coding: utf-8 -*-

# Evan Novotny
#4/12/2016

# coding: utf-8

# Script to find the MAP scores for the cross-validation and training
    sets at different
# C and epsilon values

import gensim
import Xdata
import CD_rank_svm

# load in a word-space model. If memory is an issue a reduced dictionary
     of needed word
# vectors can be used where the python dictionary key is the word and
    the value is the
# numpy vector from the word-space model
model_path='D:/Thesis/training_data/Google/GoogleNews-vectors-
    negative300.bin.gz'
```

```
isBinary=True
word_space = gensim.models.Word2Vec.load_word2vec_format(model_path,
    binary=isBinary)

#load in the cross-validation and training set to use for parameter
    tuning
CV = Xdata.Xdata('D:/Thesis/training_data/seperated_pascal_CV_nsm.txt')
train_data = Xdata.Xdata('D:/Thesis/training_data/
    seperated_pascal_training_nsm.txt')

# training file used to retrain the RSVM at different C and epsilon
    values
training_file = 'D:/Thesis/training_data/CDSVM_training/
    num_examples_compare/nsm100.txt'

# To reduce run time pre-calculate joint-feature representation which is
    the tensor
# product of the image and word vectors.
queries = CD_rank_svm.get_data(training_file)

# C and epsilon hyper-parameter range
C_values = [0.001,0.01,0.1,1,10,100,1000]
epsilon_values = [0.2,0.1,0.01]

# Sweep over the C and epsilon values using the cross-validation set and
    training set.
# CD_rank_svm.Tune_parameters returns dataframes that can be used to
# visualize the results
CV_tuning = CD_rank_svm.Tune_parameters(CV,C_values,epsilon_values,
    training_file,word_space,with_queries = queries)
```

```
42 training_tuning = CD_rank_svm.Tune_parameters(train_data,C_values,
    epsilon_values,training_file,word_space,with_queries = queries)
```

Appendix G

zero-shot_evaluation.py

```
# -*- coding: utf-8 -*-
# Evan Novotny
#4/12/2016

# Script for zeo-shot performance analysis.
# This script calculates and plots the MAP@n for each of the ayahoo zero
    -shot
# classes and plots the precision-recall-sample curves for each class.

import numpy
import matplotlib.pyplot as plt
import Xdata

#create list of all of the aYahoo tags for evaluation
yahoo_tags = ['bag','mug','goat','donkey','zebra','wolf','monkey','
    centaur','jetski','carriage','statue','building',]

#create an Xdata object for the aYahoo set
xyahoo = Xdata.Xdata(data_file='D:/Thesis/training_data/ayahoo_nsm.txt')

##load in the default word-space. Provide a path to a differnt word-
    space if desired
```

```
xyahoo.load_word_space()

#load in the RSVM weights
xyahoo.load_RSVM_weight_vector('D:/Thesis/training_data/CDSVM_models/
    no_softmax_models/C0.1_eps0.01_samp100.txt')

# Find the AP and MAP@n for every class at every point in the ranked
    list
# data_results is a 13 by 2238 pandas dataframe of the results
data_results = Xdata.AP_sweep(xyahoo,xyahoo.size,yahoo_tags)

#some points of interest and adjusted points for 0 indexing
points_of_interest = [10,50,100,200,300,400,500,1000,1500,2000]
points_adjusted = [9,49,99,199,299,399,499,999,1499,1999,2236]

#plot the MAP at each point
bx_list = list()
fig,ax = plt.subplots()
ax.plot(data_results.loc['MAP'])
ax.set_xlabel('Number of Items Retrieved')
ax.set_ylabel('Mean Average Precision')
fig.tight_layout()
fig.savefig('D:/Thesis/GIT/latex/Figures/MAPvSamples.pdf')
fig.show()

fig,ax = plt.subplots()
#creat a box plot of average precision at our points of interest
for p in points_of_interest:
    bx_list.append(data_results.iloc[:,p])
ax.boxplot(bx_list,widths=.5,labels=points_of_interest)
```

```
ax.set_xlabel('Number of Items Retrieved')
ax.set_ylabel('Average Precision')
fig.tight_layout()
fig.savefig('D:/Thesis/GIT/latex/Figures/AP_boxes.pdf')
fig.show()
#
##print some points of interest, remember in python indexs from 0
points_of_interest = [0,4,9,49,99,199,299,399,499,999,1499,1999,2236]
print data_results.loc['MAP',points_of_interest]

#pickle data to save it
data_results.to_pickle('D:/Thesis/pickle_jar/zero_shot_MAP_results.pkl')


recall_levels=[0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]
pr_list = list()
x = numpy.linspace(0,1,2237)

#create precision-recall-sample curves for each zero-shot class at the
    recall
#levels in the recall_levels variable
for t in yahoo_tags:
    ROC = Xdata.get_true_false_rate(xyahoo,t)
    pr_list.append(ROC)
    fig,ax1 = plt.subplots()
    ax1.plot(ROC.iloc[0,:],ROC.iloc[2,:],label=t)
    ax1.set_title(t)
    ax1.set_ylim([0,1.0])
    ax1.set_xlabel('Recall')
    ax1.set_ylabel('Precision')
```

```
78      ax2 = ax1.twinx() #clone axes for double y axis
        #plot on same x axis so legend works
80      ax1.plot(ROC.iloc[0,:],x,'g—',label='Samples')
        ax2.set_ylabel('Samples')
82      ax2.set_yticklabels([1,447,895,1342,1790,2237])
        ax1.legend(loc='upper right')
84      fig.tight_layout()
        fig.savefig('D:/Thesis/GIT/latex/Figures/PR_'+t+'.pdf')
86      fig.show()
```

Appendix H

seen_class_evaluation_script.py

```
# coding: utf-8
# Evan Novotny
# 4/12/2016
# This script calculates and plots the MAP@n for each of the Pascal
    dataset
# classes and plots box plots of the average precision at 10 points

import matplotlib.pyplot as plt
import Xdata

#list of tags to evaluate on
tags = ['airplane','car','bus','train','boat','motorbike','bicycle','
    person','cat','dog','horse','sheep','cow','bird','potted_plant','
    chair','table','sofa','TV','bottle']

#load in the testing set
testset = Xdata.Xdata(data_file='D:/Thesis/training_data/
    seperated_pascal_training_nsm.txt') #load cross-validation set

#load default word space
testset.load_word_space()
```

```
#load the RSVM weights
testset.load_RSVM_weight_vector('D:/Thesis/training_data/CDSVM_models/
    no_softmax_models/C0.1_eps0.01_samp100.txt')

# Find the AP and MAP@n for every class at every point in the ranked
    list
# seen_class_results is a pandas dataframe of the results
seen_class_results = Xdata.AP_sweep(testset,testset.size,tags) #get a
    pandas dataframe for the ap and map @ i)

#pickle data to save it
seen_class_results.to_pickle('D:/Thesis/pickle_jar/seen_class_results.
    pkl')

points_of_interest = [10,50,100,200,300,400,500,1000,1500,2000]
points_adjusted =    [9,49,99,199,299,399,499,999,1499,1999]
bx_list = list()

#plot and save MAP@n
fig1,ax1 = plt.subplots()
ax1.plot(seen_class_results.loc['MAP'])
ax1.set_xlabel('Number of Items Retrieved')
ax1.set_ylabel('Mean Average Precision')
fig1.tight_layout()
fig1.savefig('D:/Thesis/GIT/latex/Figures/seenMAPvSamples.pdf')
fig1.show()

#list of list of values for each point in box plot
for p in points_adjusted:
```

```
    bx_list.append(seen_class_results.iloc[:,p])

#plot and save boxplots for AP at different points
fig2,ax2 = plt.subplots()
ax2.boxplot(bx_list,widths=.5,labels=points_of_interest)
ax2.set_xlabel('Number of Items Retrieved')
ax2.set_ylabel('Average Precision')
fig2.tight_layout()
fig2.savefig('D:/Thesis/GIT/latex/Figures/AP_boxes.pdf')
fig2.show()

#print out some points that may be of interest
points_of_interest = [0,4,9,49,99,199,299,399,499,999,1499,1999,2839]
print seen_class_results.loc['MAP',points_of_interest]
```

Appendix I

tag_similarity_confusion_ matrix_script.py

```
# -*- coding: utf-8 -*-
#Evan Novotny
# 4/13/2016

#This script plots confusion matrices of the similarities between word vectors
#from the used word-space. More specifically, it plots confusion matrices for
#the pascal dataset class tags similarity to each other and to the ayhaoo class
#tags as well as the ayahoo class tag similarity to each other

import gensim
import numpy
import matplotlib.pyplot as plt

#tags used
ayahoo_tags = ['bag','mug','goat','donkey','zebra','wolf','monkey','centaur','jetski','carriage','statue','building',]
```

```
16 pascal_tags = ['airplane','car','bus','train','boat','motorbike','
       bicycle','person','cat','dog','horse','sheep','cow','bird','plant','
       chair','table','sofa','television','bottle']

18 #load in a word-space model to use
   word_space = gensim.models.Word2Vec.load_word2vec_format('D:/Thesis/
       training_data/Google/GoogleNews-vectors-negative300.bin.gz',binary=
       True)
20
   #create an array to hold confusion matrix values
22 pascal_cm = numpy.zeros((20,20))

24 #calculate similarity between each word in pascal_tags
   for i in range(0,20):
26     for j in range(0,20):
           pascal_cm[i,j] = word_space.similarity(pascal_tags[i],
       pascal_tags[j])
28
   #plot confusion matrix with imshow() with diagonal as self similarity
       and darker
30 # red colors representing higher similarity
   fig,ax = plt.subplots()
32 cax = ax.imshow(pascal_cm,interpolation='none')
   ax.set_xticks(numpy.arange(0,20))
34 ax.set_xticklabels(pascal_tags,rotation=90)
   ax.set_yticks(numpy.arange(0,20))
36 ax.set_yticklabels(pascal_tags)
   fig.colorbar(cax)
38 fig.tight_layout()
   fig.savefig('D:/Thesis/GIT/latex/Figures/tag_similarity_matrix.pdf')
```

```
fig.show()

#get confusion matrix of yahoo to pascal tag similarity
yahoo_pascal_cm = numpy.zeros((20,12))
for i in range(0,20):
    for j in range(0,12):
        yahoo_pascal_cm[i,j] = word_space.similarity(pascal_tags[i],
    ayahoo_tags[j])


#plot confusion matrix with imshow() with darker red colors representing
# higher similarity
fig,ax = plt.subplots()
cax = ax.imshow(yahoo_pascal_cm,interpolation='none')
ax.set_xticks(numpy.arange(0,12))
ax.set_xticklabels(ayahoo_tags,rotation = 90)
ax.set_yticks(numpy.arange(0,20))
ax.set_yticklabels(pascal_tags)
fig.colorbar(cax)
fig.tight_layout()
fig.savefig('D:/Thesis/GIT/latex/Figures/
    ayahoo_pascal_tag_similarity_matrix.pdf')
fig.show()

#get confusion matrix of yahoo tag self similarity
yahoo_cm = numpy.zeros((12,12))
for i in range(0,12):
    for j in range(0,12):
        yahoo_cm[i,j] = word_space.similarity(ayahoo_tags[i],ayahoo_tags
    [j])
```

```
68 #plot confusion matrix with imshow() with diagonal as self similarity
       and darker
   # red colors representing higher similarity
70 fig,ax = plt.subplots()
   cax = ax.imshow(yahoo_cm,interpolation='none')
72 ax.set_xticks(numpy.arange(0,12))
   ax.set_xticklabels(ayahoo_tags,rotation = 90)
74 ax.set_yticks(numpy.arange(0,12))
   ax.set_yticklabels(ayahoo_tags)
76 fig.colorbar(cax)
   fig.tight_layout()
78 fig.savefig('D:/Thesis/GIT/latex/Figures/ayahoo_tag_similarity_matrix.
       pdf')
   fig.show()
```

Appendix J

domain_change_script.py

```
# -*- coding: utf-8 -*-
# Evan Novotny
# 4/13/2016
# coding: utf-8

#image similarity comparision script. Script plots the image similarity
# in different domains, used in ch4 image domain comparision.

import matplotlib.pyplot as plt
import numpy as np
from scipy import spatial
import Xdata

# load dataset of images as Xdata object
dataset = Xdata.Xdata(data_file='D:/Thesis/training_data/
    seperated_pascal_training_nsm.txt')

# get the set of images with dogs in them
list_of_dogs = dataset.with_tags(['dog'])
#get the vectors for images 0-100
list_of_dog_vectors = [dataset.scores[ind] for ind in list_of_dogs
    [0:100]]
```

```
21
# get a list of cosine similarity between dog images
23 inclass_similarity = list()
for vec in list_of_dog_vectors:
25     for im in list_of_dog_vectors:
           if np.not_equal(im,vec).all():
27             inclass_similarity.append(1-spatial.distance.cosine(vec,im))


29
# get the set of images with trains in them
31 list_of_trains = dataset.with_tags(['train'])
#get the vectors for images 0-100
33 list_of_train_vectors = [dataset.scores[ind] for ind in list_of_trains
    [0:100]]

35 # get a list of cosine similarity between dog and train images
train_dog_sim = list()
37 for train_im in list_of_train_vectors:
       for dog_im in list_of_dog_vectors:
39         train_dog_sim.append(1-spatial.distance.cosine(train_im,dog_im))


41
# get the set of images with sheep in them
43 list_of_sheep = dataset.with_tags(['sheep'])
#get the vectors for images 0-100
45 list_of_sheep_vectors = [dataset.scores[ind] for ind in list_of_sheep
    [0:100]]
# get a list of cosine similarity between dog and sheep images
47 sheep_dog_sim = list()
for sheep_im in list_of_sheep_vectors:
```

```
49      for dog_im in list_of_dog_vectors:
            sheep_dog_sim.append(1-spatial.distance.cosine(sheep_im,dog_im))
51
53  # Plot histograms of image space similarity
    fig1,ax1 = plt.subplots()
55  bins = np.linspace(-1,1,100) #set bin count -1 to 1 in 100 bins
    #note alpha controls opacity
57  ax1.hist(inclass_similarity,bins,alpha = 0.5,label='dog-dog')
    ax1.hist(train_dog_sim,bins,alpha = 0.5,label='train-dog')
59  ax1.hist(sheep_dog_sim,bins,alpha = 0.5,label='sheep-dog')

61  #add legend title labels and such
    ax1.legend(loc='upper left')
63  ax1.set_xlabel('Cosine similarity [-1,1]')
    ax1.set_ylabel('Number of samples')
65  ax1.set_title('Image Similarity in Image Feature Domain')
    fig1.savefig('D:/Thesis/GIT/latex/Figures/
        Similarity_of_images_in_image_domain.pdf')
67  fig1.show()

69  # load a RSVM weight vector and reshape into projection matrix
    S = np.loadtxt('D:/Thesis/training_data/CDSVM_models/no_softmax_models/
        C0.1_eps0.01_samp100.txt')
71  S = np.reshape(S,[1000,300])

73  #transform dog images into word space
    wordimage_dogs = [np.dot(im,S) for im in list_of_dog_vectors]
75  # get a list of cosine similarity between dog images in the word-space
    wordimage_dog_sim = list()
```

```
77 for vec in wordimage_dogs:
       for im in wordimage_dogs:
79         if np.not_equal(im,vec).all():
               wordimage_dog_sim.append(1-spatial.distance.cosine(vec,im))
81
   #transform train images into word space
83 wordimage_trains = [np.dot(im,S) for im in list_of_train_vectors]
   # get a list of cosine similarity between dog and train images in the
       word-space
85 wordimage_train_dog_sim = list()
   for vec in wordimage_trains:
87     for im in wordimage_dogs:
           wordimage_train_dog_sim.append(1-spatial.distance.cosine(vec,im)
       )
89
91 #transform sheep images into word space
   wordimage_sheep = [np.dot(im,S) for im in list_of_sheep_vectors]
93 # get a list of cosine similarity between dog and sheep images in the
       word-space
   wordimage_sheep_dog_sim = list()
95 for vec in wordimage_sheep:
       for im in wordimage_dogs:
97         wordimage_sheep_dog_sim.append(1-spatial.distance.cosine(vec,im)
       )
99 # plot histogram of similarity between images projected into a word-
       space
   fig2,ax2 = plt.subplots()
101 bins = np.linspace(-1,1,100)
```

```
#note alpha controls opacity
ax2.hist(wordimage_dog_sim,bins,alpha = 0.5,label='dog-dog')
ax2.hist(wordimage_train_dog_sim,bins,alpha = 0.5,label='train-dog')
ax2.hist(wordimage_sheep_dog_sim,bins,alpha = 0.5,label='sheep-dog')

#add legend title labels and such
ax2.legend(loc='upper left')
ax2.set_xlabel('Cosine similarity [-1,1]')
ax2.set_ylabel('Number of samples')
ax2.set_title('Similarity of Images Projected into the Word-Space')
fig2.savefig('D:/Thesis/GIT/latex/Figures/
    Similarity_of_images_in_wordspace.pdf')
fig2.show()

#load a word-space model
dataset.load_word_space()

# get vector representation of words
dog = dataset.Word2Vec['dog']
train = dataset.Word2Vec['train']
cow = dataset.Word2Vec['cow']
wolf = dataset.Word2Vec['wolf']

# get a list of cosine similarity between dog images in the word-space
dog_dogword_sim = list()
train_dogword_sim = list()
sheep_dogword_sim = list()

for vec in wordimage_dogs:
    dog_dogword_sim.append(1-spatial.distance.cosine(vec,dog))
```

```
131 for vec in wordimage_trains:
        train_dogword_sim.append(1-spatial.distance.cosine(vec,dog))
133 for vec in wordimage_sheep:
        sheep_dogword_sim.append(1-spatial.distance.cosine(vec,dog))
135
137 # plot histogram of projected word similarity to word vector dog
    fig3,ax3 = plt.subplots()
139 bins = np.linspace(-1,1,100)
    #note alpha controls opacity
141 ax3.hist(dog_dogword_sim,bins,label='dog')
    ax3.hist(train_dogword_sim,bins,alpha = 0.5,label='train')
143 ax3.hist(sheep_dogword_sim,bins,alpha = 0.5,label='sheep')

145 #add legend title labels and such
    ax3.legend(loc='upper left')
147 ax3.set_xlabel('Cosine similarity [-1,1]')
    ax3.set_ylabel('Number of samples')
149 ax3.set_title('Similarity of Images in the Word-Space with the Word Dog'
        )
    fig3.savefig('D:/Thesis/GIT/latex/Figures/
        Similarity_of_images_to_word_dog.pdf')
151 fig3.show()
```

Appendix K

varying_number_ of_samples_script.py

```
# -*- coding: utf-8 -*-
# Evan Novotny
# 4/13/2016

# Script to measure MAP at different number of training examples.

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import Xdata
import CD_rank_svm
import dlib

#load training data as Xdata object
train_data = Xdata.Xdata(data_file='D:/Thesis/training_data/
    seperated_pascal_training_nsm.txt') #get set of training data
train_data.load_word_space() #load a knowledge base

```

```
#create training files for all the models for each number of examples
    per class
#1-100. this way we can retrain the RSVM at different # of examples any
    time
for i in range(1,101): #python counts from 0 so this only reacheas max-1
    descriptor = 'nsm'+str(i)+'.txt'
    training_file = 'D:/Thesis/training_data/CDSVM_training/
    num_examples_compare/'+descriptor
    train_data.export_training_data(exclusion_list=[],num_examples=i,
    out_file=training_file)

del train_data #free space


#create a dlib trainer object
trainer = dlib.svm_rank_trainer()
#from CV tuning results
trainer.c = .1   #fit trade off, from cd training results
trainer.epsilon = 0.01   #convergence size
trainer.max_iterations = 10000   #max # of iterations

#create all the models from each of the training file made previously.
    Models
#are saved to .txt files so they can be resued
# C  = .1 epsilon = 0.01
for i in range(1,101): #python counts from 0 so this only reacheas max-1
    descriptor = 'nsm'+str(i)+'.txt'
    training_file = 'D:/Thesis/training_data/CDSVM_training/
    num_examples_compare/'+descriptor
```

```
41      model_out = 'D:/Thesis/training_data/CDSVM_models/
        num_examples_compare_C.1/'+descriptor
        queries = CD_rank_svm.get_data(training_file)
43      CD_rank_svm.train_CDSVM_rank(trainer,queries,outfile=model_out)
        del queries #free memory
45
#load zero-shot test set as Xdata object
47 xyahoo = Xdata.Xdata(data_file='D:/Thesis/training_data/ayahoo_nsm.txt')
xyahoo.load_word_space() #load knowledge base
49
#tags to search for
51 tags = ['bag','mug','goat','donkey','zebra','wolf','monkey','centaur','
    jetski','carriage','statue','building',]

53 #  Get the AP and MAP scores for each number of examples for the
    completel list
# this may take some time
55 series_list = list()
for i in range(1,101):
57      descriptor = 'nsm'+str(i)+'.txt'
        model_file = 'D:/Thesis/training_data/CDSVM_models/
        num_examples_compare_C.1/'+descriptor
59      xyahoo.load_CD_model(model_file)
        #get pandas series of ayahoo average precision results
61      new_series = Xdata.create_series(xyahoo,xyahoo.size,tags)
        series_list.append(new_series)
63 #concatenate series into a dataframe
num_example_results = pd.concat(series_list,axis=1)
65

#uncomment to print a table of results
```

```
#print num_example_results

# pickle save for later
num_example_results.to_pickle('D:/Thesis/num_samples_C.1.pkl')

#range is set to only plot row 12, the MAP scores, change to range(0,13)
    to plot
#all of the classes
for ind in range(12,13):
    fig,ax = plt.subplots()
    #multiply score by 100 for percentage. get degree 1 polyfit
    [m,b]=np.polyfit(range(1,101),100*num_example_results.iloc[ind,:],1)

    #scatter plot MAP@n
    ax.plot(range(1,101),100*num_example_results.iloc[ind,:],'.')
    x = np.asarray(range(1,101)) #x axis
    #plot line of best fit
    ax.plot(range(1,101),m*x+b, label='Slope = '+str(m))

    #plot settings
    ax.set_xlabel('Number of samples per training class')
    ax.set_ylabel('Mean Average Precision (%)')
    ax.legend(loc='lower right')
    fig.tight_layout()
    fig.savefig('D:/Thesis/GIT/latex/Figures/num_examplesC01.pdf')
    fig.show()
```

Appendix L

varying_number_of_classes_script.py

```
# -*- coding: utf-8 -*-
# Evan Novotny
# 4/13/2016

# This script evaluates the zero-shot classes in the ayahoo dataset at different
# numbers of classes used for training. The number of examples used for each
# class is set to 1, 10, and 100.

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import dlib
import Xdata
import CD_rank_svm

#load in training data as Xdata object
train_data = Xdata.Xdata(data_file='D:/Thesis/training_data/seperated_pascal_training_nsm.txt') #get set of training data
train_data.load_word_space() #load a knowledge base

```

```
#tags to include/exclude
tags = ['airplane','car','bus','train','boat','motorbike','bicycle','
    person','cat','dog','horse','sheep','cow','bird','potted_plant','
    chair','table','sofa','TV','bottle']

#create training files with only the first i tags
for num_examples in [1,10,100]: #create for 3 set of examples per class
    for i in range(1,20): #index of tags
        #create path for saving output
        descriptor = 'nsm_'+str(i)+'_tags_'+str(num_examples)+'_examples
    .txt'
        training_file = 'D:/Thesis/training_data/CDSVM_training/
    incremental_classes/'+descriptor

        #excluding tags in list after the ith entry
        train_data.export_training_data(exclusion_list=tags[i:],
    num_examples=num_examples,out_file=training_file)

    #range stops at 19 so create one more model with all the tags
    included
    descriptor = 'nsm_all_tags_'+str(num_examples)+'_examples.txt'
    training_file = 'D:/Thesis/training_data/CDSVM_training/
    incremental_classes/'+descriptor
    train_data.export_training_data(exclusion_list=[],num_examples=
    num_examples,out_file=training_file)

del train_data #free up memory

#create a dlib traininer object
trainer = dlib.svm_rank_trainer()
```

```
trainer.c = .1    #fit trade off,
trainer.epsilon = 0.01    #convergence size
trainer.max_iterations = 10000    #max # of iterations

#create all the models for each number of classes at each set of examples
for num_examples in [1,10,100]:
    for i in range(1,20): #python counts from 0 so this only reacheas max-1
        #create path variables
        descriptor = 'nsm_'+str(i)+'_tags_'+str(num_examples)+'_examples.txt'
        training_file = 'D:/Thesis/training_data/CDSVM_training/incremental_classes/'+descriptor
        model_out = 'D:/Thesis/training_data/CDSVM_models/incremental_classes_C.1/ex'+str(num_examples)+'/C.1_eps.01_'+descriptor

        #calculate joint-feature vectors
        queries = CD_rank_svm.get_data(training_file)
        #train RSVM with current setting as save model to outfile
        CD_rank_svm.train_CDSVM_rank(trainer,queries,outfile=model_out)
        del queries #free up memory

    #repeat with all tags included
    descriptor = 'nsm_all_tags_'+str(num_examples)+'_examples.txt'
    training_file = 'D:/Thesis/training_data/CDSVM_training/incremental_classes/'+descriptor
```

```
63      model_out = 'D:/Thesis/training_data/CDSVM_models/
        incremental_classes_C.1/ex'+str(num_examples)+'/C.1_eps.01_'+
        descriptor
        queries = CD_rank_svm.get_data(training_file)
65      CD_rank_svm.train_CDSVM_rank(trainer,queries,outfile=model_out)
        del queries #free up memory
67
# load zero-shot data set as Xdata object
69 xyahoo = Xdata.Xdata(data_file='D:/Thesis/training_data/ayahoo_nsm.txt')
xyahoo.load_word_space() #load knowledge base
71 #list of zero-shot tags
yahoo_tags = ['bag','mug','goat','donkey','zebra','wolf','monkey','
    centaur','jetski','carriage','statue','building',]
73
#evaluate change in number of tags for 1 example per training class
75 num_examples = 1
series_list = list()
77 for i in range(1,20):
        descriptor = 'nsm_'+str(i)+'_tags_'+str(num_examples)+'_examples.txt
        '
79      model_file = 'D:/Thesis/training_data/CDSVM_models/
        incremental_classes_C.1/ex'+str(num_examples)+'/C.1_eps.01_'+
        descriptor
        xyahoo.load_CD_model(model_file)
81      #get pandas series of ayahoo average precision results
        new_series = Xdata.create_series(xyahoo,xyahoo.size,yahoo_tags)
83      series_list.append(new_series)
#evaluate for all tags
85 descriptor = 'nsm_all_tags_'+str(num_examples)+'_examples.txt'
```

```
model_file = 'D:/Thesis/training_data/CDSVM_models/incremental_classes_C
    .1/ex'+str(num_examples)+'/C.1_eps.01_'+descriptor
xyahoo.load_CD_model(model_file)
new_series = Xdata.create_series(xyahoo,xyahoo.size,yahoo_tags)
series_list.append(new_series)
#concatenate series_list into dataframe
df1ex_C01 = pd.concat(series_list,axis=1)


#evaluate change in number of tags for 10 examples per training class
num_examples = 10
series_list = list()
for i in range(1,20):
    descriptor = 'nsm_'+str(i)+'_tags_'+str(num_examples)+'_examples.txt
    '
    model_file = 'D:/Thesis/training_data/CDSVM_models/
    incremental_classes_C.1/ex'+str(num_examples)+'/C.1_eps.01_'+
    descriptor
    xyahoo.load_CD_model(model_file)
    #get pandas series of ayahoo average precision results
    new_series = Xdata.create_series(xyahoo,xyahoo.size,yahoo_tags)
    series_list.append(new_series)
#evaluate for all tags
descriptor = 'nsm_all_tags_'+str(num_examples)+'_examples.txt'
model_file = 'D:/Thesis/training_data/CDSVM_models/incremental_classes_C
    .1/ex'+str(num_examples)+'/C.1_eps.01_'+descriptor
xyahoo.load_CD_model(model_file)
new_series = Xdata.create_series(xyahoo,xyahoo.size,yahoo_tags)
series_list.append(new_series)
#concatenate series_list into dataframe
```

```
111 df10ex_C01 = pd.concat(series_list,axis=1)


113

#evaluate change in number of tags for 100 examples per training class
115 num_examples = 100
series_list = list()
117 for i in range(1,20):
    descriptor = 'nsm_'+str(i)+'_tags_'+str(num_examples)+'_examples.txt
    '
119     model_file = 'D:/Thesis/training_data/CDSVM_models/
    incremental_classes_C.1/ex'+str(num_examples)+'/C.1_eps.01_'+
    descriptor
    xyahoo.load_CD_model(model_file)
121     #get pandas series of ayahoo average precision results
    new_series = Xdata.create_series(xyahoo,xyahoo.size,yahoo_tags)
123     series_list.append(new_series)
#evaluate for all tags
125 descriptor = 'nsm_all_tags_'+str(num_examples)+'_examples.txt'
model_file = 'D:/Thesis/training_data/CDSVM_models/incremental_classes_C
    .1/ex'+str(num_examples)+'/C.1_eps.01_'+descriptor
127 xyahoo.load_CD_model(model_file)
new_series = Xdata.create_series(xyahoo,xyahoo.size,yahoo_tags)
129 series_list.append(new_series)
#concatenate series_list into dataframe
131 df100ex_C01 = pd.concat(series_list,axis=1)

133 #save to pickle files
df1ex_C01.to_pickle('D:/Thesis/pickle_jar/incemental_classes_1ex_C.1.pkl
    ')
```

```
df10ex_C01.to_pickle('D:/Thesis/pickle_jar/incemental_classes_10ex_C.1.pkl')
df100ex_C01.to_pickle('D:/Thesis/pickle_jar/incemental_classes_100ex_C.1.pkl')


#scatter plot results with a line of best fit

x = np.asarray(range(1,21)) #x axis

#plot results for 1 example per class
for i in range(12,13):
    fig,ax = plt.subplots()
    ax.set_title(df1ex_C01.index[i]+' 1 example')
    [m,b] = np.polyfit(x,100*df1ex_C01.iloc[i,:],1)
    ax.scatter(x,100*df1ex_C01.iloc[i,:])
    ax.plot(x,m*x+b,label='Slope = '+str(m))
    ax.set_xlabel('Number of samples per training class')
    ax.set_ylabel('Mean Average Precision (%)')
    ax.legend(loc='lower right')
    fig.tight_layout()
    fig.savefig('D:/Thesis/GIT/latex/Figures/num_classes1exC01.pdf')
    fig.show()

#plot results for 10 examples per class
for i in range(12,13):
    fig2,ax2 = plt.subplots()
    ax2.set_title(df10ex_C01.index[i]+' 10 examples')
    [m,b] = np.polyfit(x,100*df10ex_C01.iloc[i,:],1)
    ax2.scatter(x,100*df10ex_C01.iloc[i,:])
```

```
    ax2.plot(x,m*x+b,label='Slope = '+str(m))
    ax2.set_xlabel('Number of samples per training class')
    ax2.set_ylabel('Mean Average Precision (%)')
    ax2.legend(loc='lower right')
    fig2.tight_layout()
    fig2.savefig('D:/Thesis/GIT/latex/Figures/num_classes10exC01.pdf')
    fig2.show()

#plot results for 100 examples per class
for i in range(12,13):
    fig3,ax3 = plt.subplots()
    ax3.set_title(df100ex_C01.index[i]+' 100 examples')
    [m,b] = np.polyfit(x,100*df100ex_C01.iloc[i,:],1)
    ax3.scatter(x,100*df100ex_C01.iloc[i,:])
    ax3.plot(x,m*x+b,label='Slope = '+str(m))
    ax3.set_xlabel('Number of samples per training class')
    ax3.set_ylabel('Mean Average Precision (%)')
    ax3.legend(loc='lower right')
    fig3.tight_layout()
    fig3.savefig('D:/Thesis/GIT/latex/Figures/num_classes100exC01.pdf')
    fig3.show()
```

Appendix M

projection_matrix_ word_similarity_script.py

```
# -*- coding: utf-8 -*-
# Evan Novonty
# 4/13/2016
# coding: utf-8

# This script finds the 5 most similar word vectors to each of the rows in a
# projection matrix then plots bar graphs of the number of occurences for each
# word given that it appear more than a miniumum amount of time to filter noise.
# The same is done for the single most similar word to each row.

import gensim
import numpy
import matplotlib.pyplot as plt
import pandas as pd

#load in vocab of all known words in the word-space
```

```
vocab = gensim.models.Word2Vec.load_word2vec_format('D:/Thesis/
    training_data/Google/GoogleNews-vectors-negative300.bin.gz',binary=
    True)

#load RSVM weights and reshape into a projection matrix
weights = numpy.loadtxt('D:/Thesis/training_data/CDSVM_models/
    no_softmax_models/C0.1_eps0.01_samp100.txt')
S = numpy.reshape(weights,[1000,300])

# use gensim built in most_similar function to find the top 5 most
    similar words
# to each of the rows of the projection matix
list_of_unicode_words = list() #empty list to hold all of the topn
    unicode words
list_of_similarity_scores = list()
count_limit = 30
print 'Finding most similar words...'
topn = 5

for index in range(0,1000):
    #most_sim is a tuple of words and their score in descending order
    most_sim = vocab.most_similar(positive=[S[index,:]],topn=topn)

    print '\r'+str(index),

    for word_n in range(0,len(most_sim)):
        #get the nth word, the word is always in the zeroth place in the
    tuple
        list_of_unicode_words.append(most_sim[word_n][0])
        list_of_similarity_scores.append(most_sim[word_n][1])
```

```
41
# get a set that contains each unique word that occures for counting
43 unique_words = list(set(list_of_unicode_words))
#keeping it a list to ensure it is ordered
45 print 'done'

47 print 'Counting number of words...'
word_counts = list()
49 pd_index = list()
loop_counter = 0
51 avg_sim = list()

53 #count each word that occurs
for word in unique_words:
55     loop_counter += 1
    print '\r'+str(loop_counter),
57     count = list_of_unicode_words.count(word)
    # only keep words that occure 30 or more times
59     if count >= count_limit:
        word_counts.append(count)
61         pd_index.append(word)
        #get the similarity scores for this word
63         this_word_similarities = [list_of_similarity_scores[ind] for ind
    ,val in enumerate(list_of_unicode_words) if word is val]
        #find the average similarity to this word each time it was in
    the topn
65         avg_sim.append(sum(this_word_similarities)/float(count))

67 top5_words = pd.DataFrame([word_counts,avg_sim],index=['words','avg
    similarity'],columns=pd_index).T
```

```
print 'done'
#pickle to save data for later
top5_words.to_pickle('D:/Thesis/pickle_jar/S_mat_top5_most_similar.pkl')

#plot horizontal bar plots of number of word occurences
fig = plt.figure(figsize=(5,7))
ax  = fig.add_subplot(111)
t5_series = pd.Series(top5_words['words'])
t5_series.sort(ascending=True)
t5_series.plot(kind='barh',ax=ax,alpha = 0.6)
ax.set_xlabel('Number of occurrences')
ax.set_title('Top 5 words per row')
fig.tight_layout()
fig.savefig('D:/Thesis/GIT/latex/Figures/S_mat_top5_most_similar.pdf')
fig.show()

list_of_unicode_words = list() #empty list to hold all of the topn
    unicode words
list_of_similarity_scores = list()
count_limit = 5
print 'Finding most similar words...'
topn = 1
for index in range(0,1000):
    #most_sim is a tuple of words and their score in descending order
    most_sim = vocab.most_similar(positive=[S[index,:]],topn=topn)
    print '\r'+str(index),
    for word_n in range(0,len(most_sim)):
        #get the nth word, the word is always in the zeroth place in the
    tuple
        list_of_unicode_words.append(most_sim[word_n][0])
```

```
            list_of_similarity_scores.append(most_sim[word_n][1])
97
# get a set that contains each unique word that occures for counting
99  unique_words = list(set(list_of_unicode_words))
#keeping it a list to ensure it is ordered
101 print 'done'

103 print 'Counting number of words...'
word_counts = list()
105 pd_index = list()
loop_counter = 0
107 avg_sim = list()

109 #count each word that occurs
for word in unique_words:
111     loop_counter += 1
        print '\r'+str(loop_counter),
113     count = list_of_unicode_words.count(word)
        # only keep words that occure 5 or more times
115     if count >= count_limit:
            word_counts.append(count)
117         pd_index.append(word)
            #get the similarity scores for this word
119         this_word_similarities = [list_of_similarity_scores[ind] for ind
    ,val in enumerate(list_of_unicode_words) if word is val]
            #find the average similarity to this word each time it was in
    the topn
121         avg_sim.append(sum(this_word_similarities)/float(count))
```

```
top1_words = pd.DataFrame([word_counts,avg_sim],index=['words','avg
    similarity'],columns=pd_index).T
print 'done'

top1_words['avg similarity']

#pickle to save data for later
top1_words.to_pickle('D:/Thesis/pickle_jar/S_mat_top1_most_similar.pkl')

#plot horizontal bar plots of number of word occurences
fig2 = plt.figure(figsize=(5,7))
ax2  = fig2.add_subplot(111)
t1_series = pd.Series(top1_words['words'])
t1_series.sort(ascending=True)
t1_series.plot(kind='barh',ax=ax2,alpha = 0.6,color='g')
ax2.set_xlabel('Number of occurrences')
ax2.set_title('Top words for each row')
fig2.tight_layout()
fig2.savefig('D:/Thesis/GIT/latex/Figures/S_mat_top1_most_similar.pdf')
fig2.show()
```

Appendix N

Xdata.py

```
# Evan Novotny
#4/12/2016

# This file contains the Xdata object class definitions and functions
    used
# by the object

import numpy
import gensim
from scipy import spatial
import random
import math
from PIL import Image
from PIL import ImageTk
import Tkinter as tk
import pandas as pd

class Xdata(object):
    """
    init with a path to a data set with filename: XXXXX, tags: XXXXX,
    scores: XXXX
```

```
    Order matters! filenames, tags and scores are stored in lists such
    that
    the index of filename corrosponds to the index of its tags and
    scores
    """
    def __init__(self,data_file='D:/Thesis/GIT/pydata.txt'):
        #list containers for the names tags and scores of each image in
    the dataset
        self.names = list()
        self.tags = list()
        self.scores = list()

        self.Word2Vec = gensim.models.Word2Vec() #instantiate a word2vec
     model
        #a list of indecies probably used for creating the training data
     file
        self.index_list = list()

        f = open(data_file,'r')

        #extract data from the data file
        for line in f:
            if 'filename' in line:
                self.names.append(line.rstrip().split('filename: ')[1])#
    need to remove \n

            elif 'tags' in line:
                self.tags.append(line.rstrip().split()[1:]) #this gets
    ALL the tags with [0] = 'tags:' so remove the zeroth element
```

```
43              elif 'scores' in line:
                    self.scores.append(numpy.array(map(numpy.float32,line.
rstrip().split()[1:])))
45
        self.size = len(self.names) #size of data
47


49      def load_word_space(self,model_path='D:/Thesis/training_data/Google/
GoogleNews-vectors-negative300.bin.gz',isBinary=True):
        """
51      Enter the path to a word embedding space model to load with
gensim.
        isBinary = true by default
53      """
        self.Word2Vec = gensim.models.Word2Vec.load_word2vec_format(
model_path,binary=isBinary)
55


57      def from_existing_model(self,model):
        """
59      Load an existing Word2Vec model from the workspace into the
object
        """
61      self.Word2Vec = model

63      def view_images(self,Image_index_list,path='D:/Thesis/training_data/
Pascal/VOC2012/JPEGImages/'):
        """
65      View images listed in the image_index_list
        function attempts to open images in the provided path
```

```
        """
        #path = 'D:/Thesis/training_data/Pascal/VOC2012/JPEGImages/'
        #path = 'D:/Thesis/training_data/ayahoo_test_images/'

        win = tk.Toplevel()

        #estimate number of column for grid
        C = int(math.ceil(math.sqrt(len(Image_index_list))))

        image_count = 0

        for ind in Image_index_list:
            #image resizing and gridding
            image_count += 1
            r,c = divmod(image_count-1,C)
            img = Image.open(path+self.names[ind])
            resized = img.resize((80,80),Image.ANTIALIAS)
            tkimage = ImageTk.PhotoImage(resized)
            myvar = tk.Label(win,image=tkimage)
            myvar.image = tkimage
            myvar.grid(row=r,column=c)

        win.mainloop() #open windows with images


    def with_tags(self,search_tags=[None],view=False):
        """Xdata.with_tags(list_of_tags) -> list -- returns a list of
    indecies
            get a list of indecies of images with tags in the list given
        """
```

```
        #num_tags = len(tag_list)
97      self.index_list = list() #clear the current list

99     #for tag in tag_list:
           #for each tag that we want the index to get a set of tags,
101        #check that the tag we want is present,
           # if the index is not already in the index_list add it to
    the list
103    all_index_list =[idx for tag in search_tags
                        for idx,tag_set in enumerate(self.tags)
105                     if tag in tag_set]

107    self.index_list = list(set(all_index_list)) #remove duplicates

109    if view:
           self.view_images(self.index_list)
111
       return self.index_list
113
 def export_training_data(self,exclusion_list=[],inclusion_list=[],
    num_examples=20,
115                          out_file='D:/Thesis/training_data/
    CDSVM_training/CDSVM_train.txt',myseed=2015):


117
       export_index_list = list()
119
       #get the indecies of the tags to be included/removed, if
    inclusion_list is empty
121    #or doesn't contain any valid tags with_tags will return [] and
```

```
        #export_list will remain empty
123     #note: idx_remove beats idx_include
        idx_include = self.with_tags(inclusion_list) #indecies of tags
    to include
125     idx_remove = self.with_tags(exclusion_list) #indecies of tags to
     remove


127

        #check that the inclusion list contained valid tags and returned
     indecies for export
129     #if it failed or wasn't used just include everything
        if  idx_include:
131         export_index_list = [i for i in idx_include if i not in
    idx_remove]
        else:
133         export_index_list = [i for i in range(0,len(self.names))
                                    if i not in idx_remove]
135

        #get a set of unique tags in the group to export
137     export_tags = list()
        for i in export_index_list:
139         export_tags.extend(self.tags[i])
        export_tags = set(export_tags)
141

        #create a dictionary where the key is a tag and values are
    indecies of images
143     #with that tag
        tag2ind = dict()
145     training_set_count = 0
        for key in export_tags:
```

```
147             #the dict contains every index for images with a given tag
                clean_idx_list= [i for i in self.with_tags([key]) if i not
    in idx_remove]
149             tag2ind[key] = clean_idx_list

151         f=open(out_file,'w')

153         #remove duplicates
            set_all = set(export_index_list) #set of all indecies being
    exported
155
            random.seed(myseed)#seed RNG
157
            for key in export_tags:
159
                try:
161                 #get embedding vector and make it a string for writing
                    target = numpy.array_str(self.Word2Vec[key])
163                 target = target.translate(None,"[]\n")

165                 print '\r writing key '+key,

167                 f.write('target: '+target+'\n')

169                 #get the maximum number of relevant vectors for a tag
                    num_rel = len(tag2ind[key])
171


173
                    #if theres more than 20 examples of an image only use 20
```

```
175                 if num_rel > num_examples:
                        num_rel = num_examples
177                     rel_ind = random.sample(tag2ind[key],num_rel)
                    else: #else just use however many there are
179                     rel_ind = tag2ind[key]

181                 #write all of the relevant vectors
                    rel_counter = 0
183
                    #write set of relevant image vectors for current class
185                 for i in rel_ind:
                        rel_counter+=1
187                     rel = numpy.array_str(self.scores[i])
                        rel = rel.translate(None,"[]\n")
189                     f.write('relevant: '+rel+'\n')
                    print '\r done, ',rel_counter,' relevant samples',
191
                    #get a set of relevant indecies and subtract from all to
     get a
193                 #set of nonrelevant indecies
                    set_exclude = set(rel_ind)
195                 set_nonrel = set_all.difference(set_exclude)
                    for i in range(0,num_rel):
197                     #print 'writing nonrelevant ',i
                        nonrel = numpy.array_str(self.scores[random.sample(
    set_nonrel,1)[0]])
199                     nonrel = nonrel.translate(None,"\n[]")
                        f.write('nonrelevant: '+nonrel+'\n')
201                 f.write('\n')
                    print '\r done, ',i,' nonrelvant samples',
```

```
203                 training_set_count+=1

205             except KeyError:
                    print 'KeyError for key: '+key
207
        print 'done\n', training_set_count, ' training sets written'
209     f.close()

211 def load_RSVM_weight_vector(self,model_file='D:/Thesis/training_data
    /CDSVM_models/CD_model.txt'):
        """weights = numpy.loadtxt(model_file)-> self.weights
213     Load the weights learned by the RSVM
        """
215     self.weights = numpy.loadtxt(model_file)

217 def word2im(obj,query,topn=25, view=False, Path=False):
    """
219 Take a word or phrase as an input query and find the most similar
    image
    in the database
221 """

223 #convert query to a vector
    wordvec = obj.Word2Vec[query]
225
    # here we append all of the scores for each image to a new results
    list
227 # order is important here as the index of the score aligns with the
    stored
    #score/tag/name information
```

```
229     results = []
        S = obj.weights.reshape(1000,300)
231
        for img_score in obj.scores:
233         results.append(image_word_similarity(img_score,wordvec,S))

235     #convert to a numpy array and use argpartition to ge the INDEX of
        the top
        #20 or so images
237     results = numpy.array(results)

239     #obj.top = numpy.argpartition(results,-topn)[-topn:] #gets indeces
        of top
        #N scores, returned scores are unordered here sort all of results
        smallest
241     #to largest, get the top n biggest andreverse the order so they are
        largest
        #to smallest
243
        #numpy argsort returns the index of the results ordered smallest to
        largest
245     #then the list is fliped. obj.top is a list of indecies
        obj.top = numpy.fliplr([numpy.argsort(results)[-topn:]])[0]
247
        top_tags=list()
249     similarity_scores = list()
        for i in obj.top:
251         top_tags.append(obj.tags[i])
            similarity_scores.append(results[i])
253
```

```
    #get a list of the classes found and a list of all tags returned
255 unique_tags = list(set(y for x in obj.tags for y in x))
    all_list = list(y for x in top_tags for y in x)
257
    #print the number of each class in the results
259 print "class representation"
    for i in range(0,len(unique_tags)):
261     print unique_tags[i] , ": ", all_list.count(unique_tags[i]),'\n'

263 #print the number of times the searched for class is in the top n
    placements_list = list()
265 print "search term in top "
    for i in range(0,len(top_tags)):
267     placements_list.extend(top_tags[i])
        if i == 0:
269         print '1: ',placements_list.count(query),'\n'
        elif i == 4:
271         print '5: ',placements_list.count(query),'\n'
        elif i == 9:
273         print '10: ',placements_list.count(query),'\n'
        elif i ==24:
275         print '25: ',placements_list.count(query),'\n'
    print (1+i),": ",placements_list.count(query)
277
    #view pictures
279 if view:
        if Path:
281         obj.view_images(obj.top,Path)
        else:
283         obj.view_images(obj.top)
```

```
285     #return list of image tags and similarity scores in order of
        rankings
        return top_tags,similarity_scores
287
    def detect_obj(obj,query,threshold, view=False,Path=False):
289     """
        Find images in a database that score >= the given threshold
291     """

293     #convert query to a vector
        wordvec = obj.Word2Vec[query]
295
        # here we append all of the scores for each image to a new results
        list
297     # order is important here as the index of the score aligns with the
        stored score/tag/name information
        results = []
299     results_index = []
        S = obj.weights.reshape(1000,300)
301
        for idx, img_score in enumerate(obj.scores):
303
            similarity = image_word_similarity(img_score,wordvec,S)
305         if similarity >= threshold:
                results.append(similarity)
307             results_index.append(idx)

309     #convert to a numpy array and use argpartition to ge the INDEX of
        the top 20 or so images
```

```
    results = numpy.array(results)
311
    #obj.top = numpy.argpartition(results,-topn)[-topn:] #gets indeces
    of top N scores, returned scores are unordered
313 #here sort all of results smallest to largest, get the top n biggest
    and
    #reverse the order so they are largest to smallest
315 obj.top = numpy.fliplr([numpy.argsort(results)])[0]

317 top_tags=list()
    similarity_scores = list()
319 for i in obj.top:
        top_tags.append(obj.tags[results_index[i]])
321     similarity_scores.append(results[i])

323 unique_tags = list(set(y for x in obj.tags for y in x))
    all_list = list(y for x in top_tags for y in x)
325
    print "class representation"
327 for i in range(0,len(unique_tags)):
        print unique_tags[i] , ": ", all_list.count(unique_tags[i]),'\n'
329
    #print the number of times the searched for class is in the top n
331 placements_list = list()
    print "search term in top "
333 for i in range(0,len(top_tags)):
        placements_list.extend(top_tags[i])
335     if i == 0:
            print '1: ',placements_list.count(query),'\n'
337     elif i == 4:
```

```
            print '5: ',placements_list.count(query),'\n'
        elif i == 9:
            print '10: ',placements_list.count(query),'\n'
        elif i ==24:
            print '25: ',placements_list.count(query),'\n'
    print (1+i),": ",placements_list.count(query)

    #view pictures
    if view:
        if Path:
            obj.view_images(results_index,Path)
        else:
            obj.view_images(results_index)

    return top_tags,similarity_scores


def stats(obj):
    """print the class representation for the object"""
    #list of unique tags
    unique_tags = list(set(y for x in obj.tags for y in x))

    #list of every tag with duplicates
    all_list = list(y for x in obj.tags for y in x)

    print "class representation"
    for i in range(0,len(unique_tags)):
        print unique_tags[i] , ": ", all_list.count(unique_tags[i]),'\n'

def retrieve_all(obj,topn=50):
```

```
    """
    return a pandas dataframe with the P@1 P@5 P@n AP@n and MAP@n for
each class
    in the Xdata object.
    """

    #get a list of unique classes in the object
    unique_tags = list(set(y for x in obj.tags for y in x))

    #list of every tag with duplicates
    all_list = list(y for x in obj.tags for y in x)

    #create list of all tags to search by finding each tag that occures
    query = list()
    query.extend(unique_tags)

    S = obj.weights.reshape(len(obj.scores[0]),len(obj.Word2Vec[query
[0]]))

    #Create output data frame with rows labels = to query tags
    query.append('Mean') # add MAP as a column
    dataframe = pd.DataFrame(index=query,columns=['P@1','P@5','P@n','AP'
])
    query.remove('Mean') #remove it from tags list

    #get the number of class-labels to search
    num_words = len(query)
    #container for AP results
    np_ap = numpy.zeros((num_words,1),dtype = numpy.float64)
```

```
395     for j in range(0,num_words):
            wordvec = (obj.Word2Vec[query[j]])
397         results = list()

399         for img_score in obj.scores:
                #Get image word similarity
401             results.append(image_word_similarity(img_score,wordvec,S))

403         #convert to a numpy array
            results = numpy.array(results)
405
            #get the indeces for the topn largest scores
407         obj.top = numpy.fliplr([numpy.argsort(results)[-topn:]])[0]

409         top_tags=list()
            tot_prec = 0.0
411         num = 0.0
            ind = 0.0
413
            #chane in recall = min(# of relevant items, # of items returned)
415         num_relevant = float(all_list.count(query[j]))
            if num_relevant > topn:
417             #if there are more relevant items than there are returned
                #use number returend as recall
419             change_in_recall = topn
            else: #if more items are returned than there are relevant use #
    relevant
421             change_in_recall = num_relevant

423         #obj.top is a ranked list of the indecies of the top tags
```

```
        for ind,index in enumerate(obj.top,start=1): #start counitng at
    1
425             top_tags.extend(obj.tags[index]) #add tags to a current list
     of tags

427             if query[j] in obj.tags[index]: #if the query is a tag for
    image i
                num += 1.0
429                 tot_prec += num/float(ind) #sum of precision up to this
    point

431         if num == 0:
                np_ap[j,0] = 0 #set value to 0 if 0 items a correct
433         else:
                np_ap[j,0] = tot_prec/change_in_recall #divide sum by change
     in recall
435
            #count the number of images with query as a tag in the topn
437         tp = float(top_tags.count(query[j]))

439         #find precision at 1 5 and n
            pa1 = float(obj.tags[obj.top[0]].count(query[j]))
441         if topn >= 5:
                pa5=0
443             for ind in range(0,5):
                    if query[j] in obj.tags[obj.top[ind]]: pa5 += 1
445             pa5 = pa5/5.0
            else:
447             pa5 = 0
```

```
449             pan = float(tp)/topn #precision @ topn or # relevant available
                recall = tp/change_in_recall
451
                dataframe['P@1'][query[j]] = pa1
453             dataframe['P@5'][query[j]] = pa5
                dataframe['P@n'][query[j]] = pan
455             dataframe['AP'][query[j]] = np_ap[j,0]

457             #print recall@n and AP@n
                print query[j],' recall@',topn,': ',recall,' average precision:
            ',np_ap[j,0]
459
            mean_ap = numpy.mean(np_ap,dtype=numpy.float64)
461         dataframe['P@1']['Mean'] = dataframe['P@1'].mean()
            dataframe['P@5']['Mean'] = dataframe['P@5'].mean()
463         dataframe['P@n']['Mean'] = dataframe['P@n'].mean()
            dataframe['AP']['Mean'] = mean_ap
465
            print 'Mean average precision: ',mean_ap
467
            return dataframe
469
    def get_true_false_rate(obj,query):
471         """
            find the true positive, false positive, precision and AP for an
            Xdata
473         object at each point over the full size of the set of images.
            """
475         topn = obj.size
            S = obj.weights.reshape(len(obj.scores[0]),len(obj.Word2Vec[query]))
```

```
477
    wordvec = (obj.Word2Vec[query])
479
    results = list()
481
    for img_score in obj.scores:
483     #Get image word similarity
        results.append(image_word_similarity(img_score,wordvec,S))
485
    #convert to a numpy array and use argpartition to ge the INDEX of
    the top 20 or so images
487 results = numpy.array(results)

489 #get the indeces for the largest scores
    obj.top = numpy.fliplr([numpy.argsort(results)[-topn:]])[0]
491
    top_tags=list()
493 tot_prec = 0.0
    num = 0.0
495 ind = 0.0

497 num_relevant = float(len(obj.with_tags([query])))
    if num_relevant > topn:
499     #if there are more relevant items than there are returned
        #use number returend as recall
501     change_in_recall = topn
    else: #if more items are returned than there are relevant use #
    relevant
503     change_in_recall = num_relevant
```

```
505     tpr = 0 #true positive rate/recall
        fpr = 0 #false positive rate
507     count = 0.0

509     tpr_list = list()
        fpr_list = list()
511     precision_list=list()

513     # get the true and false positive rate and the precision@n
        #obj.top is a ranked list of the indecies of the top tags
515     for index in obj.top:
            count = count+1.0
517         if query in obj.tags[index]: #if the query is a tag for image i
                tpr = (tpr+1)
519             tpr_list.append(tpr/float(num_relevant))
                fpr_list.append(fpr/float(topn-num_relevant))
521         else:
                fpr = (fpr+1)
523             #fpr = count - tpr
                fpr_list.append(fpr/float(topn-num_relevant))
525             #fpr_list.append(fpr/float(count))
                tpr_list.append(tpr/float(num_relevant))
527         precision_list.append(tpr/float(count))

529     # Calculate average precision
        avg_precision = list()
531     num=0
        ind = 0
533     tot_prec=0
```

```
535     if num_relevant > topn:
            #if there are more relevant items than there are returned
537         #use number returend as recall
            change_in_recall = topn
539     else: #if more items are returned than there are relevant use #
    relevant
            change_in_recall = num_relevant
541
        #obj.top is a ranked list of the indecies of the top tags
543     for index in obj.top: #start counitng at 1 instead of 0
            top_tags.extend(obj.tags[index]) #add tags to a current list of
    tags
545         ind +=1.0
            if query in obj.tags[index]: #if the query is a tag for image i
547             num += 1.0
                tot_prec += num/float(ind)
549         avg_precision.append(tot_prec/change_in_recall)

551     #create pandas dataframe with true positive rate, false positive
    rate, and ap
        dataframe = pd.DataFrame([tpr_list,fpr_list,precision_list,
    avg_precision],index=['True positive rate','False positive rate','
    Precision','Average Precision'])
553
        return dataframe
555
    def image_word_similarity(image,word,S):
557     """
        return a similarity score for the given word and image using a given
559     tranformation matrix S
```

```
    """
561     sim = (1-spatial.distance.cosine( numpy.dot(image,S) ,word))
    return sim
563 
565 
def create_series(obj,topn=100,query=[]):
567     """
    create a pandas series containing the AP@n for each point from 1 to
    topn
569     for the given query
    """
571 
    #list of every tag with duplicates
573     all_list = list(y for x in obj.tags for y in x)

575     S = obj.weights.reshape(len(obj.scores[0]),len(obj.Word2Vec[query
    [0]]))

577     num_words = len(query)

579     np_ap = numpy.zeros((num_words+1,1),dtype = numpy.float64)
    for j in range(0,num_words):
581         wordvec = (obj.Word2Vec[query[j]])
        #wordvec = ((wordvec+0.5))**2
583 
        results = []
585 
        #get the similarity scores for each image to the current query
587         for img_vec in obj.scores:
```

```
            results.append(1-spatial.distance.cosine( numpy.dot(img_vec,
    S) ,wordvec))
589
        results = numpy.array(results)
591     #get the indeces for the largest scores
        obj.top = numpy.fliplr([numpy.argsort(results)[-topn:]])[0]
593
        top_tags=list()
595     tot_prec = 0.0
        num = 0.0
597
        num_relevant = float(all_list.count(query[j]))
599     if num_relevant > topn:
            #if there are more relevant items than there are returned
601         #use number returend as recall
            change_in_recall = topn
603     else: #if more items are returned than there are relevant use #
    relevant
            change_in_recall = num_relevant
605
        #obj.top is a ranked list of the indecies of the top tags
607     for counter,index in enumerate(obj.top,start = 1): #start
    counitng at 1 instead of 0
            top_tags.extend(obj.tags[index]) #add tags to a current list
     of tags
609         #ind +=1.0
            if query[j] in obj.tags[index]: #if the query is a tag for
    image i
611             num += 1
                tot_prec += num/float(counter)
```

```
613
            if num == 0:
615             np_ap[j,0] = 0
            else:
617             np_ap[j,0] = tot_prec/change_in_recall

619     np_ap[j+1] = numpy.mean(np_ap[:num_words])

621     row_names = []
        row_names.extend(query)
623     row_names.append('MAP')
        #create a pandas series of the average precision(ap) with the query
        tags
625     #as the row labels(index)
        ap_series = pd.Series(np_ap[:,0],index=row_names)
627
        #return this series so it can be placed in a dataframe
629     return ap_series


631
def AP_sweep(obj,topn=100,query=[]):
633     """
          Calculate the AP@n from 1 to topn for each of the tags in query
        and return
635       a pandas dataframe containing the data.
        """
637     #list of every tag with duplicates
        all_list = list(y for x in obj.tags for y in x)
639
        #reshape weights into the projection matrix S
```

```
641     S = obj.weights.reshape(len(obj.scores[0]),len(obj.Word2Vec[query
        [0]]))

643     num_words = len(query)

645     #create a matrix of average precision results at each sample point
        np_ap = numpy.zeros((num_words+1,topn),dtype = numpy.float32)
647
        #iterate through the word in query
649     for j in range(0,num_words):
            wordvec = (obj.Word2Vec[query[j]])
651
            results = list()
653
            #get the similarity scores for each image to the current query
655         for img_vec in obj.scores:
                results.append(1-spatial.distance.cosine( numpy.dot(img_vec,
        S) ,wordvec))
657
            results = numpy.array(results)
659         #get the indeces for the largest scores
            obj.top = numpy.fliplr([numpy.argsort(results)[-topn:]])[0]
661
            top_tags=list()
663         tot_prec = 0.0
            num = 0.0
665
            num_relevant = float(all_list.count(query[j]))
667
```

```
    #calculate AP for the current class at every point in the ranked
 list
669     for counter,tag_index in enumerate(obj.top,start = 1):

671         if num_relevant > counter:
                #if there are more relevant items than there are
returned
673             #use number returend as recall
                change_in_recall = float(counter)
675         else: #if more items are returned than there are relevant
use # relevant
                change_in_recall = float(num_relevant)
677
            top_tags.extend(obj.tags[tag_index]) #add tags to a current
list of tags
679         #ind +=1.0
            if query[j] in obj.tags[tag_index]: #if the query is a tag
for image i
681             num += 1
                tot_prec += num/float(counter)
683
            #j is the tag index, counter-1 is the numpy average
precision index
685         if num == 0:
                np_ap[j,counter-1] = 0 #-1 to adjust for indexing
687         else:
                np_ap[j,counter-1] = tot_prec/change_in_recall
689
 np_ap[j+1,:] = numpy.mean(np_ap[:j,:],axis=0,dtype=numpy.float32)
691
```

```
    row_names = []
693 row_names.extend(query)
    row_names.append('MAP')
695 #create a pandas datframe of the average precision(ap) with the
    query tags
    # as the row labels(index)
697 precision_dataframe = pd.DataFrame(np_ap,index=row_names)

699 #return the dataframe
    return precision_dataframe
```

Appendix O

CD_rank_svm.py

```
# -*- coding: utf-8 -*-
# Evan Novotny
#4/12/2016

# Module containing RSVM creation, training and hyper-parameter tuning functions

# COMPILING THE DLIB PYTHON INTERFACE
#   Dlib comes with a compiled python interface for python 2.7 on MS Windows. If
#   you are using another python version or operating system then you need to
#   compile the dlib python interface before you can use this file.  To do this,
#   run compile_dlib_python_module.bat.  This should work on any operating
#   system so long as you have CMake and boost-python installed.
#   On Ubuntu, this can be done easily by running the command:
#       sudo apt-get install libboost-python-dev cmake

import dlib
import numpy as np
```

```
18 import pandas as pd
   import Xdata
20
   def psi_function(y,x):
22     """
       returns the joint-feature vector for x and y, which is the tensor(
      outer)
24     product flattend into a vector and converted to a list
       """
26     result = np.outer(x,y) #resulst is dim(x) by dim(y)
       result = result.flatten()
28
       result = result.tolist()
30     return result

32 def get_data(in_file):
       """gets data for RSVM training"""
34
       f = open(in_file,'r')
36
       #these are dlib objects used by the rank svm algorithm
38     queries = dlib.ranking_pairs()
       data = dlib.ranking_pair()
40
       first_line = True
42     for line in f:
           if 'target' in line:
44
               #each new target marks the end of a ranking pair so we need
      to
```

```
46              #handle the first time differently from everyother time
                if first_line:
48                  target = np.array(map(np.float32,line.rstrip().split()
    [1:]))
                    first_line = False
50              else:
                    queries.append(data)
52                  #we don't need to hold onto the previous data values
    because
                    #they are stored in the dlib queries variable
54                  del data
                    data = dlib.ranking_pair()
56                  #drop the [0] term because it is just the work 'target:'
     and
                    #map the rest as float 32 to save on memory(default is
    float64)
58                  target = np.array(map(np.float32,line.rstrip().split()
    [1:]))

60          #nonrelevant needs to be first so the 'relevant' substring isn't
     found
        elif 'nonrelevant' in line:
62              nonrelevant = np.array(map(np.float32,line.rstrip().split()
    [1:]))
                #the psi fuction is the joint data representation
64              vec = psi_function(target,nonrelevant)
                data.nonrelevant.append(dlib.vector(vec))
66
        elif 'relevant' in line:
```

```
68              relevant = np.array(map(np.float32,line.rstrip().split()
    [1:]))
                vec = psi_function(target,relevant)
70              data.relevant.append(dlib.vector(vec))


72

    f.close()
74  queries.append(data)
    return queries
76



78
def create_CDSVM_rank_trainer(C=1000,eps=0.001,max_iter=10000,infile='D
    :/Thesis/training_data/CDSVM_training/CDSVM_train.txt'):
80  """
    create a dlib rank svm trainer object with the input parameters C,
    eps, max_iter
82  and create a ranking_pairs object named queries to train on
    """
84  #create dlib rank svm trainer object
    trainer = dlib.svm_rank_trainer()
86

    trainer.c = C   #fit trade off,
88  trainer.epsilon = eps   #convergence size
    trainer.max_iterations = max_iter   #max # of iterations
90  #trainer.be_verbose = 1

92  #create a dlib ranking_pairs object and load it with relevane/
    nonrelevant
    #from our input data file
```

```
94      queries = get_data(infile)

96      return trainer, queries

98  def train_CDSVM_rank(trainer, queries, outfile = 'D:/Thesis/
        training_data/CDSVM_models/CD_model.txt'):
        """
100     Perform RSVM training and output to the file outputfile
        """
102     # Do rank svm training.
        rank = trainer.train(queries)
104
        S = np.asarray((rank.weights))
106
        #save the calculated rank svm weights as a numpy vector in a .txt
        file
108     np.savetxt(outfile,S)


110
    def cross_validate_CDSVM_rank(trainer, queries, folds=4):
112     """
        preform n fold cross-validation of dlib SVM rank trainer using the
        given
114     ranking_pairs queries
        """
116
        print 'C: ', trainer.c, ' Epsilon: ',trainer.epsilon, 'folds: ',
        folds
118     print("Cross validation results: {}".format(
        dlib.cross_validate_ranking_trainer(trainer, queries, folds)))
```

```
120     print '\n'


122
def Tune_parameters(obj,c_range,eps_range,training_file,label_dict,
    with_queries=0):
124     """
        Preform Cross-validation tuning
126


128     Preform Cross-validation hyperperameter training on a set of cross-
        validation
        data. To free up memory a dictionary label_dict must be passed with
        the word
130     vectors for training labels. A training object is created along with
        the queries
        used for training.
132
        -----inputs-----
134     obj - a Xdata object
        c_range - a list of C values to evaluate at
136     eps_range - a list of epsilon values to evaluate at
        training_file - a training file to retrain the RSVM with
138     label_dict - a dictionary or word-space containing word vectors
        with_queries - precomputed queries for the RSVM, these are the joint
        feature
140                    vectors used for training.
        -----outputs-----
142     results_frame: a pandas dataframe of results
        """
144
```

```
    image_length = len(obj.scores[0])
146 word_length = 300
    column_names = ['C','eps','MAP','P@1','P@5','P@100']
148 results_frame = pd.DataFrame(columns=column_names)

150 #seach full set of data
    topn  = obj.size
152
    unique_tags = list(set(y for x in obj.tags for y in x))
154
    #list of every tag with duplicates
156 all_list = list(y for x in obj.tags for y in x)

158 #create list of all tags to search by finding each tag that occures
    query = list()
160 query.extend(unique_tags)


162
    if with_queries == 0:
164     # create a rank svm trainer and queries for training C and eps
values don't matter here
        [trainer,queries] = create_CDSVM_rank_trainer(C=10,eps=0.1,
max_iter=10000,infile=training_file)
166 else: #if queries were given use them for training
        queries = with_queries
168     trainer = dlib.svm_rank_trainer()

170 #cycle through eps range
    for eps_val in eps_range:
172
```

```
        trainer.epsilon = eps_val
174
        #cycle through C value range
176     for c_val in c_range:
            trainer.c = c_val
178
            #print '@ C=',str(c_val),' eps=',str(eps_val)
180
            # retrain with new values
182         rank = trainer.train(queries)

184         #get new S matrix by reshaping weights as a numpy array
            S = np.asarray((rank.weights)).reshape(image_length,
word_length)
186
            #save the S models for reuse
188         save_weights = 'D:/Thesis/training_data/CDSVM_models/
no_softmax_models/C'+str(c_val)+'_eps'+str(eps_val)+'_samp100.txt'
            np.savetxt(save_weights, np.asarray((rank.weights)))
190
            #preallocate arrays for holding values
192         num_words = len(query)
            np_ap = np.zeros((num_words,1)) #average precision numpy
array
194         precision1_array = np.zeros(num_words)
            precision5_array = np.zeros(num_words)
196         precision100_array = np.zeros(num_words)

198         #evaluate Cross-validation results for each C epsilon value
pair
```

```
            for j in range(0,num_words):
200             #wordvec = (obj.Word2Vec[query[j]])
                wordvec = label_dict[query[j]]
202
                results = []
204
                for img_score in obj.scores:
206                 #Get image word similarity
                    results.append(Xdata.image_word_similarity(img_score
    ,wordvec,S))
208
                #convert to a numpy array
210             results = np.array(results)

212             #get the indeces for the largest scores
                obj.top = np.fliplr([np.argsort(results)[-topn:]])[0]
214
                top_tags=list()
216             tot_prec = 0.0
                num = 0.0
218             ind = 0.0

220             # calculate AP, P@1, P@5 and
                num_relevant = float(all_list.count(query[j]))
222             if num_relevant > topn:
                    #if there are more relevant items than there are
    returned
224                 #use number returend as recall
                    change_in_recall = topn
```

```
226                 else: #if more items are returned than there are
    relevant use # relevant
                        change_in_recall = num_relevant
228
                    #obj.top is a ranked list of the indecies of the top
    tags
230                 for index in obj.top: #start counitng at 1 instead of 0
                        top_tags.extend(obj.tags[index]) #add tags to a
    current list of tags
232                     ind +=1.0
                        if query[j] in obj.tags[index]: #if the query is a
    tag for image i
234                         num += 1.0
                            tot_prec += num/float(ind)
236
                    #ger average precision for query #j
238                 if num == 0:
                        np_ap[j,0] = 0
240                 else:
                        np_ap[j,0] = tot_prec/change_in_recall
242
                    pa1 = float(top_tags[0].count(query[j]))
244                 if topn >= 5:
                        pa5=0
246                     for ind in range(0,5):
                            if query[j] in obj.tags[obj.top[ind]]: pa5 += 1
248                     pa5 = pa5/5.0
                    else:
250                     pa5 = 0
```

```
252                 if topn >=100:
                        pa100=0
254                     for ind in range(0,100):
                            if query[j] in obj.tags[obj.top[ind]]: pa100 +=
    1
256                     pa5 = pa100/5.0
                    else:
258                     pa100 = 0

260                 #save for averaging later
                    precision1_array[j] = pa1
262                 precision5_array[j] = pa5
                    precision100_array[j] = pa100
264             #get means for the current c eps pair
                mean_ap = np.mean(np_ap)
266             meanat1 = np.mean(precision1_array)
                meanat5 = np.mean(precision5_array)
268             meanat100 = np.mean(precision100_array)

270             series_Ceps = pd.Series([c_val,eps_val,mean_ap,meanat1,
    meanat5,meanat100],index=column_names)

272             #append results to the results frame
                results_frame = results_frame.append(series_Ceps,
    ignore_index=True)
274
     return results_frame
```

VITA

Evan Novotny

Candidate for the Degree of

Master of Science

Thesis: EVALUATION OF A ZERO-SHOT CROSS-MODAL IMAGE RETRIEVAL TECHNIQUE USING RANKING SUPPORT VECTOR MACHINES

Major Field: Electrical Engineering

Biographical:

Education:

Completed the requirements for the Master of Science in Electrical Engineeringat Oklahoma State University, Stillwater, Oklahoma in May 2016.

Completed the requirements for the Bachelor of Science in Electrical Engineeringat Oklahoma State University, Stillwater, Oklahoma in December 2013.

Experience:

Teaching Assistant for Dr. Dr. George Scheets, Oklahoma State University, Stillwater, OK, January 2016 - May 2016.