



Norwegian University of  
Science and Technology

# Investigating Zero-Shot Learning techniques in multi-label scenarios

**Bjørnar Moe Remmen**

**Thomas Sve**

Master of Science in Informatics

Submission date: June 2017

Supervisor: Helge Langseth, IDI

Co-supervisor: Axel Tidemann, Telenor Research  
Cyril Banino, Telenor Research

Norwegian University of Science and Technology  
Department of Computer Science



# Abstract

Visual recognition systems are often limited to the object categories previously trained on and thus suffer in their ability to scale. This is in part due to the difficulty of acquiring sufficient labeled images as the number of object categories grows. To solve this, earlier research have presented models that uses other sources, such as text data, to help classify object categories unseen during training. However, most of these models are limited on images with a single label and most images can contain more than one object category, and therefore more than one label. This master's thesis implements a model capable of classifying unseen categories for both single- and multi-labeled images.

The architecture consist of several modules: A pre-trained neural network that generates image features for each image, a model trained on text that represents words as vectors, and a neural network that projects the image features to the dimension native to the vector representation of words. On this architecture, we compared two approaches to generate word vectors using GloVe and Word2vec, with different vector dimensions and on spaces containing different numbers of word vectors. The model was adapted to multi-label predictions comparing three approaches for image box generation: YOLOv2, Faster R-CNN and randomly generated boxes. Here each box represents a section of the image cut out and this approach was chosen to fit each label to a one of these boxes.

The results showed that increasing the word vector dimension increased the accuracy, with Word2vec outperforming GloVe, and when adding more words to the word vector space the accuracy dropped. In the single-label scenario the model achieves similar results to existing models with similar architecture. While in the multi-label scenario, the model trained on boxes generated by Faster R-CNN and predicted on random generated boxes had highest accuracy, but was not able to outperform comparative alternatives. The architecture gives promising results, but more investigation is needed to answer if the results can be improved further. The code for this thesis is publicly available at Github: [https://github.com/thomasSve/Msc\\_Multi\\_label\\_ZeroShot](https://github.com/thomasSve/Msc_Multi_label_ZeroShot).

---

# Sammendrag

Systemer for bildegjenkjenning er ofte begrenset til å kun gjenkjenne objektene systemet tidligere har trent på og sliter når de blir presentert nye usette objekter. Dette er tildels fordi det er vanskelig å få tak i tilstrekkelig treningsdata i form av bilder, etterhvert som antall objektkategorier øker. Tidligere løsninger har vist at maskinlæringsmodeller kan bruke andre kilder, for eksempel tekstdata, for å hjelpe til å klassifisere nye usette objekter. Tidligere modeller har stort sett hatt begrensingen at de bare kan behandle bilder hvor hvert bilde har en klasse tilknyttet til den, for eksempel kun katt eller hund. Siden mange bilder kan ha flere objekter per bilde, for eksempel både katt og hund, presenterer vi i denne masteroppgaven et system som kan klassifisere bilder med både én og flere klasser.

Systemets arkitektur består av flere moduler: et ferdigtrent nevralt nettverk som genererer egenskapsvektorer av bildene, en tekstmodell som representerer ord som vektorer, og et nevralt nettverk som reduserer bildets egenskapsvektor til en vektor i lik dimensjon som ordvektoren. Vi har i denne oppgaven testet to ulike systemer for å generere ordvektorer, med navnet GloVe og Word2vec. Systemet ble tilpasset til å kunne klassifisere bilder med flere objekter, der vi sammenligner tre ulike fremgangsmåter for å generere bokser av bilder: YOLOv2, Faster R-CNN og tilfeldig plasserte bokser. Her representerer hver av boksene et utsnitt av et bilde hvor hver boks er tilknyttet et objekt.

Resultatene vi fikk, viser det at å øke dimensjonen på ordvektorene ga bedre resultater, at Word2vec presterte bedre enn GloVe og ved å øke antall ordvektorer ble resultatene dårligere. I eksperimentet hvor hvert bilde er tilknyttet kun en klasse, fikk vi likt resultat som eksisterende forskning. I det andre eksperimentet, hvor bildene hadde flere klasser, så vi at modellen trent med Faster-R-CNN og klassifisert med tilfeldige bokser hadde best resultat. Dette systemet presterte ikke bedre enn eksisterende systemer. Arkitekturen gir lovende resultater, men mer forskning må til for å kunne si om resultatene kan forbedres. Koden ligger åpent på Github: [https://github.com/thomasSve/Msc\\_Multi\\_label\\_ZeroShot](https://github.com/thomasSve/Msc_Multi_label_ZeroShot).

---

---

# Preface

This thesis is written at the Department of Computer Science (IDI) at Norwegian University of Science and Technology (NTNU) between the period August 2016 to June 2017. The project is built in collaboration between Telenor Research and NTNU.

We would like to thank our supervisor, Helge Langseth for allowing us to take on this thesis in collaboration with Telenor, and his guidance and helpful insights. Without his positive attitude and interest in our thesis, this would not have been possible. We are grateful for the collaboration with the people at Telenor Research, where we had Axel Tidemann and Cyril Banino as co-supervisors. We want to thank Axel and Cyril for the help during our thesis with both feedback on the master thesis and the valuable discussions. We would like to thank all three supervisors for the direct feedback on throughout our master thesis, which has helped us a lot. Lastly we would also like to thank our brothers Bjarne Drotninghaug and Christoffer Astad Sve for their general feedback and spell checking.

Bjørnar Moe Remmen

Student, Bjørnar Moe Remmen

Thomas Astad Sve

Student, Thomas Astad Sve

Trondheim, 8th of June 2017

---



# Table of Contents

<b>Abstract</b>	<b>i</b>
<b>Sammendrag [Norwegian]</b>	<b>iii</b>
<b>Preface</b>	<b>v</b>
<b>List of Tables</b>	<b>xi</b>
<b>List of Figures</b>	<b>xiii</b>
<b>Acronyms &amp; Abbreviations</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Task Description . . . . .	2
1.2 Goal and Research Questions . . . . .	3
1.3 Thesis Outline . . . . .	3
<b>2 Background</b>	<b>5</b>
2.1 A brief introduction to Artificial Intelligence . . . . .	5
2.2 The Concept of Zero-Shot Learning . . . . .	6
2.2.1 One-Shot Learning . . . . .	7
2.3 Artificial Neural Network . . . . .	7
2.4 Convolutional Neural Networks . . . . .	9
2.5 Object Detection Framework . . . . .	10
2.5.1 Non-Maximum Suppression . . . . .	11
2.6 Word embedding . . . . .	12
2.6.1 Word2vec - Word to Vector . . . . .	12
2.6.2 GloVe - Global Vectors for Word Representation . . . . .	13
2.7 Evaluation metrics . . . . .	14
2.7.1 Flat Hit Precision . . . . .	14

---

2.7.2	Mean Average Precision . . . . .	14
2.7.3	Average Cosine Similarity . . . . .	15
<b>3</b>	<b>Related Work</b>	<b>17</b>
3.1	Zero-Shot Learning . . . . .	17
3.2	Multi-label Zero-Shot Learning . . . . .	18
3.3	Word Embedding Space . . . . .	18
3.4	Deep Convolutional Neural Networks . . . . .	19
<b>4</b>	<b>Design &amp; Implementation</b>	<b>21</b>
4.1	The general model architecture . . . . .	21
4.2	Single-label scenarios . . . . .	23
4.3	Multi-label Scenarios - Generate image boxes . . . . .	24
4.3.1	Multi-label training . . . . .	25
4.3.2	Multi-label testing . . . . .	25
4.3.3	Random image boxes . . . . .	26
4.3.4	Using Object Detection Framework (ODF) . . . . .	27
4.4	Language Model . . . . .	28
4.4.1	Training language models . . . . .	28
4.4.2	Predicting word vectors . . . . .	29
4.5	Technical information . . . . .	31
4.5.1	Deep learning frameworks - Keras and Tensorflow . . . . .	31
4.5.2	Hardware specification . . . . .	31
<b>5</b>	<b>Experiments &amp; Results</b>	<b>33</b>
5.1	Datasets overview . . . . .	33
5.2	Normalization Layer test . . . . .	34
5.3	Selecting parameters . . . . .	35
5.3.1	Optimization and learning rate . . . . .	35
5.3.2	Batch Size . . . . .	35
5.3.3	Loss function . . . . .	35
5.3.4	Iterations and Stopping criterion . . . . .	36
5.4	Experiment 1 - Single-label Zero-Shot Learning . . . . .	37
5.4.1	Description . . . . .	37
5.4.2	Goal of the experiment . . . . .	39
5.4.3	Results . . . . .	39
5.4.4	Was the goal met? . . . . .	42
5.5	Experiment 2 - Multi-label Zero-Shot Learning . . . . .	42
5.5.1	Experiment 2.1 - Tune model on multi-label dataset . . . . .	43

## TABLE OF CONTENTS

---

5.5.2	Experiment 2.2 - Test single-label model directly on multi-label data . . . . .	43
5.5.3	Goal of the experiment . . . . .	43
5.5.4	Results . . . . .	44
5.5.5	Was the goal met? . . . . .	47
<b>6</b>	<b>Discussion</b>	<b>49</b>
6.1	Language models & prediction spaces . . . . .	49
6.2	Multi-label scenarios . . . . .	55
6.3	Limitations . . . . .	60
<b>7</b>	<b>Conclusion</b>	<b>63</b>
7.1	Contributions . . . . .	64
7.2	Future Work . . . . .	65
	<b>Bibliography</b>	<b>67</b>



# List of Tables

5.1	Class and size of images of popular <i>Zero-Shot Learning</i> datasets . . . . .	33
5.2	Test without normalization-layer . . . . .	34
5.3	Test with normalization-layer . . . . .	34
5.4	ImageNet 1k - Loss function results . . . . .	36
5.5	Experiment 1 - Results single-label Zero-Shot Learning . . . . .	40
5.6	Experiment 1 - Flat hit@5, comparative results . . . . .	42
5.7	Experiment 2 - MAP@gt (%), comparative results . . . . .	45
5.8	Experiment 2 - Results multi-label Zero-Shot Learning . . . . .	46
6.1	Experiment 1 - Top-5 results - 5 labels . . . . .	51
6.2	Highest scoring labels - flat hit@5 . . . . .	53
6.3	Five labels - flat hit@5 = 0 . . . . .	54
6.4	5 random results for multi-label on NUS-WIDE . . . . .	58



# List of Figures

2.1	Domain <i>Zero-Shot Learning</i> example . . . . .	6
2.2	Fully connected neural network . . . . .	8
2.3	Gradient descent - 3D . . . . .	8
2.4	CNN filter illustration . . . . .	10
2.5	YOLOv2 - Bounding Box illustration . . . . .	11
2.6	Non-Maximum Supression illustration . . . . .	11
2.7	<i>Continuous bag-of-words</i> (CBOW) . . . . .	13
4.1	The suggested <i>Zero-Shot Learning</i> solution . . . . .	22
4.2	Suggested solution to train single-label <i>Zero-Shot Learning</i> . . . . .	24
4.3	Suggested solution to test single-label <i>Zero-Shot Learning</i> (ZSL) . . . . .	24
4.4	Suggested solution to train multi-label <i>Zero-Shot Learning</i> using by generating boxes . . . . .	25
4.5	Suggested solution to test multi-label <i>Zero-Shot Learning</i> using by generating boxes . . . . .	26
4.6	Demonstration of randomly generated boxes for multi-label . . . . .	27
4.7	t-SNE normalized illustration - euclidean vs cosine . . . . .	30
5.1	Word Space for ImageNet 1k unseen set . . . . .	38
5.2	Word Space for ImageNet 1k containing both seen and unseen classes . . . . .	38
5.3	Flat hit@5 accuracy distributed among all classes using Word2vec in 300-D vector space as language model. . . . .	41
6.6	Demonstrating YOLOv2, Faster R-CNN and random boxes . . . . .	56
6.11	Demonstrate how the average precision changes with different numbers of ground truth labels . . . . .	59





## Acronyms & Abbreviations

<b>AI</b>	<i>Artificial Intelligence</i> : The term used when talking about machine showing intelligent behavior and decisions [10].
<b>ANN</b>	<i>Artificial Neural Network</i> : Computational model used in machine learning, which is based on a large collection of connected neurons mimicing the human brain.
<b>CBOW</b>	<i>Continuous bag-of-words</i> : One of the methods in Word2vec to learn word vectors.
<b>CNN</b>	<i>Convolutional Neural Network</i> : Specialized neural network that is often used in the field of computer vision.
<b>CV</b>	<i>Computer Vision</i> : Sub-field in Artificial Intelligence focusing on vision.
<b>GloVe</b>	<i>Global Vectors for Word Representation</i> : A method by Stanford University that trains word vectors from a text corpus.
<b>MAP</b>	<i>Mean Average Precision</i> : Metric to measure quality of test-result for single-label and multi-label problems.
<b>NLP</b>	<i>Natural Language Processing</i> : Sub-field in Artificial Intelligence that focuses on languages and human-computer interaction.
<b>NMS</b>	<i>Non-Maximum Supression</i> : An algorithm that removes bounding boxes that overlap with a certain amount.
<b>ODF</b>	<i>Object Detection Framework</i> : A framework designed to detect objects in images. Example of such frameworks are Faster-RCNN [1] and YOLOv2 [2].
<b>OSL</b>	<i>One-Shot Learning</i> : A field in machine learning where a model learns new classes with only a few or one samples.
<b>Word2vec</b>	<i>Word to vector</i> : A method that trains word vectors from a text corpus.
<b>ZSL</b>	<i>Zero-Shot Learning</i> : Extreme type of transfer learning where new classes are introduced at test time.



# Chapter 1

## Introduction

As the amount of data increases, so does the motivation to use this data in an intelligent way in order to extract knowledge from it. Much progress has been made by developing machine learning models that can represent and solve complex tasks. The reasons for this are two-fold: we have a lot more computing power, but we also have more labeled data. One of the issues with these models, is that they are limited on the classes they are trained on and for each class a machine learning model requires a large set of well-labeled samples to train from. As a result, when building recognition systems for categories one cannot expect to be able to build systems that can recognize all natural categories.

It has been estimated that humans can categorize 30,000 object categories [3], and also many more sub-categories, such as breeds of dogs [4]. As each category needs a large amount of samples to get a good result, training a model to categorize so many visual categories would require millions or billions well-labeled training images. Therefore, numerous models for reducing necessary numbers of training images have been developed, so-called one-shot learning [5] or few-shot learning, as well as models for classifying classes with no training samples, known as *Zero-Shot Learning (ZSL)* [6, 7, 8].

Motivated from the humans ability to recognize new objects only from having a description, ZSL tries to train a classifier that can classify unseen objects. For example, a child can simply recognize a zebra without ever have seen one before if she has seen a horse and told a zebra is a horse but with black and white stripes. ZSL uses a similar approach and consists in recognizing new categories, by providing a high level description of the unseen category and relate this description to previously learned categories.

ZSL is especially important in domains where there are many categories and the cost of labeling each category with enough samples is too high. In addition a major challenge when scaling in object recognition systems is the lack of annotated images

for real-world categories. With ZSL the need for annotated images is reduced, as some object categories can be left unannotated and still be recognizable for the model. ZSL have been tested for various problems such as image-tagging [6, 7, 9] and brain activity measurements [8].

Most of the research in ZSL has focused on single-label problems where each image only has one category connected to it, for example an image only containing a dog or a cat. However, most images can be labeled with more than one label, known as a multi-label problem, where the image can contain for instance both a cat and a dog. When working with multi-label problems in ZSL a classifier can use the knowledge from known objects to classify unknown objects. For example, if an image consists of two known objects and one unknown, this can help the model to classify the unknown object by using information about the image it already has. For instance, zebras live in certain habitats and if the classifier already can label the habitat shown in the image, it can be easier to label the unseen zebra using the already known information.

This thesis will investigate how ZSL can be solved for both single-label and multi-label problems. In the multi-label problem, in our proposed solution we have made the assumption that each of the labels in an image can be connected to different areas on that image. Therefore this image is split into a set of boxes, where these boxes are labeled with one label each. For example, with an image containing both a dog and a cat we propose to split this image into boxes where some of them show the cat, while others show the dog. This will reduce the problem from a multi-label to a single-label and will follow similar techniques as with single-label ZSL to make predictions on unknown objects.

## 1.1 Task Description

The task of this thesis is to train an image-recognition classifier to be able to classify unseen classes. The ultimate task is to train a classifier that can take large amounts of images and classify tens of thousands new labels without having to annotate them first. This is however a very complex task, so the focus will be on a smaller, yet large, task where the experiments are run on a smaller set of unseen labels. The research project will also investigate how knowledge about words learned from large text corpus can help the classifier predict new object-classes on images, and if a classifier is able to successfully predict multi-label *Zero-Shot Learning*.

## 1.2 Goal and Research Questions

**Goal** *Investigate Zero-Shot Learning techniques in multi-label scenarios.*

**Research Question 1:** *How can the knowledge from text be used to help an image-classifier predict unseen classes?*

We will investigate if and how well knowledge from text can be used to help an image-classifier predict new unseen object-classes.

**Research Question 2:** *How does our Zero-Shot learning model scale when increasing the number of available classes?*

Many of the existing ZSL models focuses on a small set of data, and on a limited amount of unseen labels. We want to investigate how the model scales when increasing the number of unseen labels, and also when increasing the number of labels possible to predict on.

**Research Question 3:** *How can Zero-Shot learning be applied to multi-label scenarios?*

As most real-world images can consist of more than one object, this research question investigates if a ZSL can predict unseen labels for multi-label problems.

## 1.3 Thesis Outline

Readers who are interested in the implementation can jump directly to chapter 4, Design & Implementation, and continue from there. Chapter 2, Background, is written for readers who would like to gain knowledge with the technologies used in the thesis. The complete outline of the thesis is:

- Ch. 2 Background, introduces the different approaches used to solve Zero-Shot Learning. The chapter gets the reader up to speed on important subjects that builds up the thesis.
- Ch. 3 Related Work, describes related work and looks at papers that have tried to solve similar problems earlier, and the approached used.
- Ch. 4 Design & Implementation, explains the implementations used and gives a thorough explanation of the designed architecture.

- Ch. 5 Experiments & Results, experiments with the implementations to investigate the research goal and questions. The chapter contains multiple experiments, testing different scenarios, data and language models before reporting the results.
- Ch. 6 Discussion, discusses the results reported in Chapter 5.
- Ch. 7 Conclusion, concludes on the results produced and the discussions of these. It reviews the research goal and questions and suggest future work.

# Chapter 2

## Background

*This chapter serves as an introduction to Zero-Shot Learning and the techniques used to approach this. It is important to have some familiarity with these subjects to follow the thesis, as they build up the domain of the thesis.*

### 2.1 A brief introduction to Artificial Intelligence

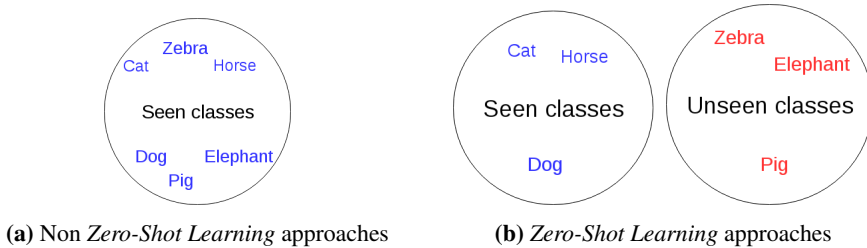
*Artificial Intelligence (AI)* has been referred to by Shapiro [10] as a sub-field in computer science that focuses on constructing machines or systems that can display intelligent behavior that is commonly called intelligent behavior. However, defining the term intelligence can be hard and debated by researchers. The overall research goal of AI is to create technology that enable computers to perform a range of different tasks. Examples are perception tasks, learning tasks, language processing tasks, planning, knowledge and many more including general intelligence. General Intelligence is a term used when an AI-machine combines various sources and methods to create a machine intelligence that exceeds human ability in most cases. Currently AI has the classified capabilities to understand human speech, compete in high strategic game (such as Chess and Go) and control vehicles (self-driving cars). AI is a large field that can be related to many problems and tasks and our focus lies within the subfields of Computer Vision and Natural Language Processing.

The field of *Computer Vision (CV)* deals with how computers can get a high-level understanding of digital images or videos. In other words, give computers the ability to see and understand in the same way as the human visual system [11]. It is a sub-problem within machine perception that involves around tasks where you use the input from sensors (such as cameras and microphones) to understand the world. While CV focuses on vision, *Natural Language Processing (NLP)* focuses on language and the interaction

between computer and human languages. The field of NLP works with speech and written text and this thesis will focus on a set of techniques that maps words into high-dimensional vectors, known as word embedding (section 2.6).

## 2.2 The Concept of Zero-Shot Learning

In *Zero-Shot Learning* (ZSL) a model is trying to predict classes unseen to a classifier. Figure 2.1 shows how ZSL works in practice. During training time the classifier is trained on one set of classes, and during test time new classes are introduced.



**Figure (2.1):** Here you can see a venn diagram visualization the ZSL approach. In most cases illustrated in figure *a*, the model is trained on all sets of classes, so-called seen classes, while in ZSL some classes are unseen to the model during training and first introduced during testing.

ZSL is an extreme case of transfer learning. **Transfer learning**, also known as learning to learn or inductive transfer, shares with ZSL the aim of extracting knowledge from a set of sources that can be applied in future tasks. It is often used when very good knowledge of one task is available, but little information of the second task. For example, if you want to build an accurate heart disease classifier, but lack enough data. Then, if you have a huge amount of data for other related diseases, you might want to use this data to help build a heart disease classifier. When output of one type of algorithm helps to either increase accuracy, or reduce training time, of another algorithm you have a successful transfer learning. For further reading, we recommend reading the extensive survey in the work of Pan and Yang [12].

Transfer learning is often associated with **domain adaptation**. Both domain adaptation and transfer learning refers to a situation where what has been learned in one setting is exploited to improve generalization in another. The difference however is that with domain adaptation, the task remains the same between each setting, while the input distribution is different [13]. One example here is when analyzing user reviews, a predictor could be trained to predict positive or negative review for a media content page, and is later used to analyze comments about for example electronics. The domain here is



slightly different, and some style or vocabulary is different from the two, but the end goal or task is the same.

### 2.2.1 One-Shot Learning

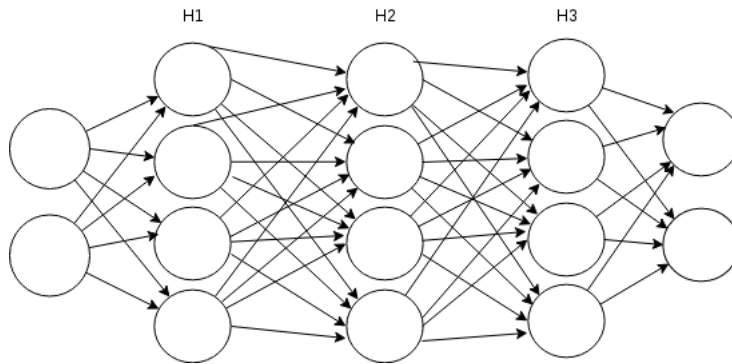
A concept that is very similar to ZSL is *One-Shot Learning* (OSL). While ZSL aims to predict unseen classes, OSL tries to learn new classes using as few examples as possible and aims to learn these classes with only one example. For example, a child is shown a zebra once and told that is a zebra, and is able to recognize it later. Typically similar approaches is used in both OSL and ZSL, using various versions of transfer learning. [5, 14]

## 2.3 Artificial Neural Network

*Artificial Neural Networks* (ANNs) mimics the neural networks in the brain, and was first introduced by McCulloch and Pitts [15] in 1943. ANNs are built up by neurons and synapses, where neuron activation is based on the incoming synapses. Weights are stored in the synapses which changes value when introduced to new examples during training, but unchanged during test time. A neuron in ANN can have multiple input and output synapses and the activation of a neuron is determined by an activation function, which can be for example the sigmoid function as demonstrated in figure 2.1. The sigmoid function can be used to calculate the probability of class  $y$  in datapoint  $x$  [16]. This function is typically used in binary class problems to see if a class is there or not. In multi-class problems it is common to use softmax, which can be thought of as a extension of the sigmoid function. This function makes it so that all probabilities sums up to one for the multi-class problem and can therefore be threatred as true probabilities [17].

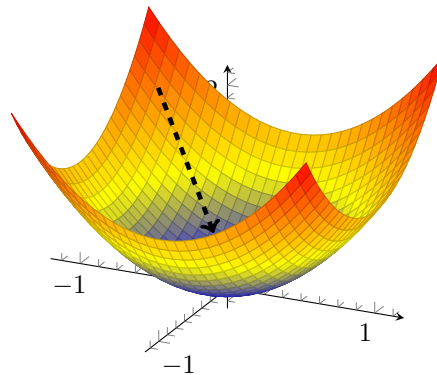
$$y = \sigma(W^T x) = \frac{1}{1 + \exp(W^T x)} \quad (2.1)$$

Looking at figure 2.1, we can see how to calculate the output of the network, where the incoming data  $X$  and the weights  $W$  calculates the probability of the output,  $\sigma(W^T X)$ , called forward pass [16]. While linear classifiers can be used in some simple contexts, it can not be used for more complex problems, because most problems are too complex to be solved linearly. The solution to this is to build a neural network consisting of multiple nodes (neurons), and in newer networks millions of nodes. This means that a simple neural network can ANN looks something like in figure 2.2.



**Figure (2.2):** Fully connected neural network

Figure 2.2 shows one of the most common neural networks, called the fully connected neural network. ANN is built up of many layers where the middle layers are called hidden layers. Each node in the layer contributes to getting the answer. When the network guesses the wrong answer under training, the weights in the network are changed and iterates backwards from the output to check the error contribution of each weight,  $\frac{\delta E_{total}}{\delta w_{ij}}$  [16]. This is done using the algorithm called backward propagation which is more effective than performing many forward passes to measure the error. In 2.3 the weights are visualized in a 3D space where error is the the y-axis (vertical) and the weights the other axis. This is to illustrate that model gradually learns by calculating the slope to reduce the error, where the lowest error here is the bottom of the slope.



**Figure (2.3):** Gradient descent in a 3D weight space.

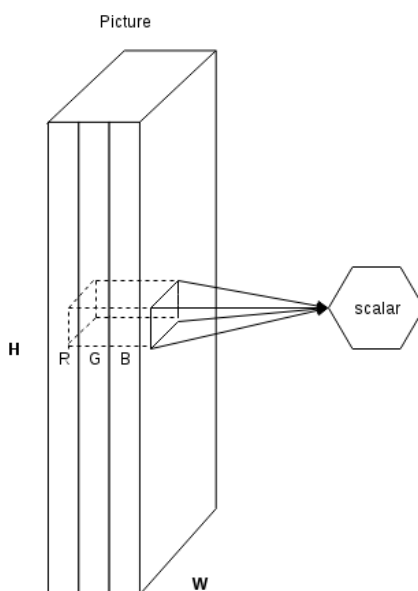
## 2.4 Convolutional Neural Networks

*Convolutional Neural Networks* (CNNs) are similar to simple *Artificial Neural Networks* (ANNs), as they are based on many of the key concepts in ANNs. This includes, but is not limited to concepts like forward- and backward-propagation we introduced earlier in section 2.3. Introduced by LeCun et al. in 1999, CNNs were first used in image recognition, but have also been applied to other applications such as sentence classifications. A CNN consists of a sequence of layers, mainly Convolution Layer, Pooling Layer and a Fully-Connected Layer all stacked together.

The Convolution Layer is the core building block in CNN and is comprised of a set of independent filters. Using images as an example, a filter slides over an image and takes the dot product between the filter and a portion of the input image, as shown in figure 2.4. For each dot product taken, the result is a scalar that together forms a new convoluted picture smaller than the original, called feature map. Each of the filters corresponds to the weights in an ANN and over time these filters will change during training, and different filters learn to detect different parts of an image, such as edges of an image. CNN uses the two concepts parameter sharing and local connectivity. In parameter sharing, the weights are shared between the neurons in a feature map, which makes it possible for the network to detect i.e. edges over the entire image. Local connectivity is the concept where each neuron is only connected to a subset of the input image, which makes sense because close pixels often correlate. Going deeper with more convolution layers, the filters are doing dot products to the input of the previous convolution layers.

Another building block in CNN is the pooling layer, that reduces the size of the representation to reduce the amount of parameters in the network. A pooling layer operates on each feature map independently. In the end of the architecture, a CNN usually has a fully-connected layer that works the same way as in a simple ANN explained in section 2.3. This can be used together with a softmax function to get the probability for which object is inside the picture.

For a more comprehensive and visualization of the layers in a CNN we recommend reading the paper by Zeiler and Fergus [19].

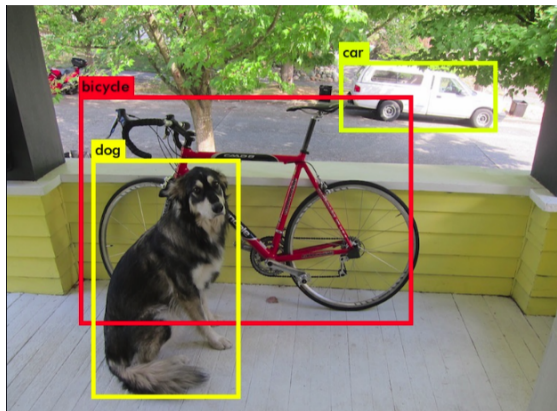


**Figure (2.4):** Showing an illustration for a filter that slides over the input data.

## 2.5 Object Detection Framework

A *Object Detection Framework* (ODF) tries to detect all instances of objects such as people, cars and cats inside an image. Typically only a small number of objects are presented in an image, but with a large number of possible locations. A ODF tries to detect the locations of these objects reported in terms of bounding boxes as shown in figure 2.5. Two ODFs are **Faster R-CNN** [1] and **You Only Look Once (YOLO)** [2], which are considered to be the current state-of-the-art.

Presented by Ren et al. [1] in 2015, **Faster R-CNN** is an object detection framework based on deep CNNs, which includes a Region Proposal Network (RPN) and an Object Detection Network. The RPN is trained end-to-end to generate region proposals, which are used by the object detection network for detection. Further the two networks are merged into one single network by sharing the convolutional features with the RPN telling the detection network where to look, and the detection network predicting the classes.

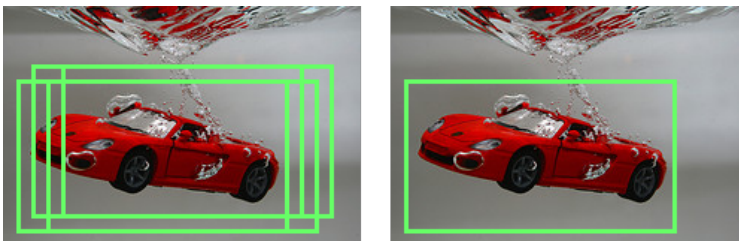


**Figure (2.5):** Object detection example using YOLOv2 [2].

While Faster R-CNN combines a region proposal network with a object detection network, **YOLO** uses a single neural network to do both region proposal and detection. By looking at the entire image, it manages to increase the speed and also helps understand the full context of the picture. The original YOLO reported to be much quicker than Faster R-CNN, but suffered a little in terms of accuracy. However, recently in 2016, Redmon and Farhadi released an updated version of YOLO that claims to outperform Faster R-CNN on both speed and accuracy. This new model, YOLOv2 [2] outperforms Faster R-CNN while still making predictions for 67 images a second, in contrary to the 5-15 images a second in Faster R-CNN.

### 2.5.1 Non-Maximum Suppression

When working with object detection frameworks, the ODF can often report boxes that overlaps. A way to calculate the overlapping boxes and remove boxes that overlap too much is to use an edge thinning technique called Non-Maximum Suppression [20]. Non-Maximum Suppression is demonstrated in figure 2.6, where boxes that overlap too much are removed.



**Figure (2.6):** Illustration of Non-Maximum Suppression in ODFs removing the overlapping boxes.

## 2.6 Word embedding

Word Embedding is the name for a set of language modelling and feature learning techniques in *Natural Language Processing* (NLP) where words or phrases are mapped to high-dimensional vectors (from 50 to 1000 dimensions), also known as word vectors. Using a large corpus of text, such as Wikipedia, a good word embedding can successfully place words with similar meaning close together in the vector space.

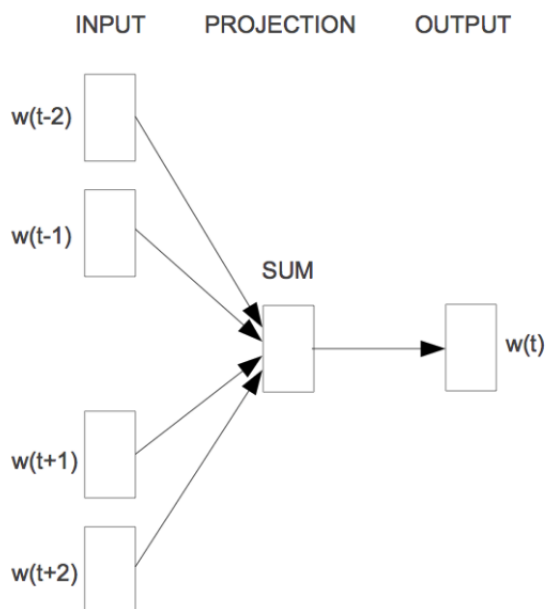
### 2.6.1 Word2vec - Word to Vector

One of the most common word embedding models is Word2vec [21]. It was introduced by Mikolov et al. in 2013 and is a collection of two algorithms: CBOW and Skip-Gram. CBOW and Skip-Gram uses both a single hidden layer, fully connected neural network. Which means they are not in the definition of deep learning due to the simplicity of the networks. The algorithms focuses on efficiency, so it can be able to train on large amount on data. The algorithms in Word2vec are built up from three layers:

1. Embedding layer: Multiplies the index vector with a word embedding matrix to generate word embedding.
2. Intermediate layer: One fully-connected layer that produce an intermediate representation of the input.
3. Softmax layer: Final layer that produces the probability distribution.

### Continuous Bag-Of-Words - CBOW

In *Continuous bag-of-words* (CBOW) a sliding window goes over the text, and uses the central word in focus as target and the  $n$  words before and after as context. The context words form the input layer. Each word is encoded in one-hot form, so if the vocabulary size is  $V$ , then each one of these words will have  $V$ -dimensions with one element set to one and the rest to zero. These context words are fed in the fully-connected neural network, with a single hidden layer.



**Figure (2.7):** Figure illustrating how CBOw learns. [22]

Figure 2.7 shows how the CBOw model receives a window with the surrounding words as input with the center word as target. The objective with the training is to maximize the conditional probability of observing the target word given the input words. As an example, consider the sentence: "the quick brown fox jumped over the lazy dog". Using  $n=2$  and target word "fox", the input words becomes ["quick", "brown", "jumped", "over"]. The model now wants to maximize the probability of getting "fox" as the output. Doing this over a large corpus, creates a vector space where the words that have similar meaning are close together.

The other algorithm in Word2vec, Skip-Gram, flips the CBOw algorithm around and uses the centre word to predict the surrounding words around. This algorithm is explained in depth in the paper by Goldberg and Levy [23] that uses the Skip-Gram with negative sampling.

## 2.6.2 GloVe - Global Vectors for Word Representation

Released in 2014 by Pennington et al., GloVe [24] is a neural word embedding model that is based on counting. First, GloVe creates a large matrix of co-occurrence information for each word, meaning it counts how often a word (row) occurs in context of another word (column). Next, the GloVe performs dimension reduction on the matrix where it tries to find the lower-dimensional representation that can explain most of the

variance in the higher-dimension. This leads to GloVe taking a large amount of memory while computing, but in return it trains quicker than Word2Vec and can also re-use the co-occurrence matrix to quickly factorize the word vectors into any chosen dimensions, whereas Word2vec needs to retrain the network from scratch after changing the embedding dimension.

## 2.7 Evaluation metrics

After a model has made predictions on the test set, there are many different ways to measure the quality. These metrics variate if measured for a multi-label or single-label scenario. In a multi-label scenario, each sample can have any number of true labels associated with it (e.g. both cat and dog). While in single label, each sample have only one true label (only dog or cat).

### 2.7.1 Flat Hit Precision

To measure the performance for single-label experiments where there is top-k predicted labels for each image, flat hit @k is one of the metrics used. Here the metric gives full score as long as the correct label is in the top-k predictions. Given an example where the correct label (ground truth) is 1 and the predicted values are [4, 2, 3, 4, 6, 1]. Here when measuring top-3, it will return incorrect as 1 is not among the first three values. However, if measuring top-6 it will return correct and full score as the label is in top-k. Meaning the orders of the predicted values do not matter when using Flat hit @ k.

### 2.7.2 Mean Average Precision

To measure the results in single-label and multi-label problems where the order of the top-k predicted values matter, *Mean Average Precision* is commonly used. To measure *Mean Average Precision* (MAP) the average precision for each prediction is calculated before measuring the mean. In other words, the mean for Average Precision (AP) is measured, hence the name Mean Average Precision. AP is a way to calculate the precision of for the k predictions, and is defined by the equation:

$$ap@k = \sum_{j=1}^n \frac{P(k)}{\min(m, n)} \quad (2.2)$$

where  $P(k)$  is the precision for the k-th item in the list and equals 0 when the k-th prediction is an incorrect one; m is the number of ground truth labels and n is the number of predictions. Take for example a test where the correct labels are [1, 2, 3, 4,



5] and the predicted values are [6, 1, 7, 4, 2]. Here 1, 4 and 2 are correct with some incorrect predictions in between. With AP@2 only the two first predictions matter: 6 and 1. First is wrong, so precision@1 is 0. Second is correct, so precision @2 is 0.5, which gives  $AP@2 = (0 + 1/2)/2 = 0.25$ . If the two predicted values changed order, then  $AP@2 = (1/1 + 0)/2 = 0.5$ . This means, the orders the predicted labels do matter on the results, note however if all the predictions are correct the order does not matter. After calculating the Average Precision for all samples, the Mean Average Precision is measured:

$$MAP@k = \sum_{i=1}^N \frac{ap@n_i}{N} \quad (2.3)$$

Which in the end gives a way to measure the quality of the predictions for both single-label and multi-label problems, where the order of the predictions matter.

### 2.7.3 Average Cosine Similarity

To be able to interpret how far away or close on average the model is from the ground truth, we would like to introduce average cosine similarity, that has the range  $[-1,+1]$ . This gives an intuitive understanding of how good or bad the model is in a range that is easily interpretable. The cosine similarity is:

$$d(P, GT) = \frac{\sum_{i=1}^n P_i GT_i}{\sqrt{\sum_{i=1}^n P_i^2} \sqrt{\sum_{i=1}^n GT_i^2}} \quad (2.4)$$

Where P is the vector of the predicted label, and GT is the vector of the ground truth label for the image. After calculating the cosine similarity, average cosine similarity is:

$$avg\_cosine\_similarity = \frac{\sum_0^N d(P, GT)}{N} \quad (2.5)$$

N here represents the number of calculated cosine similarities. For example doing this for all missed predictions means that N is the number of missed predictions in total.



# Chapter 3

## Related Work

*This chapter will go through previous work related to Zero-Shot Learning (ZSL). It will look at papers that have tried to solve similar problems earlier, and the approaches used.*

### 3.1 Zero-Shot Learning

The problem of *Zero-Shot Learning* has received an increasing interest as it tries to solve the need for expensive annotating of training data for large scale recognition problems. Researchers have reported some promising results, but have mostly been tested on a small set of unseen labels [25, 26, 27, 28, 29, 30, 31].

As one of the first to work with ZSL on a larger set of unseen labels, Frome et al. [6] presented in 2013 a model to do ZSL prediction with a set of 1000 seen classes and up to 20000 unseen classes. Their joint model, known as DeVISE [6], is initialized from two pre-trained neural network models, one language embedding model and one *Convolutional Neural Network* (CNN) as visual model. The last softmax layer from the pre-trained CNN is removed and outputs a 4096-D representation of each image, an added projection layer now maps this representation into the 500- or 1000-D representation native to the language model. Next a loss function is added to compare the two vectors, the word vector from the language model and the vector representation of the image.

Inspired by DeVISE [6], Norouzi et al. [9] presented ConSE. ConSE kept the last softmax layer in the visual model and considers the top  $T$  of the model. Then, the convex combination of the semantic vectors of these predictions is computed, which corresponds to the projection layer used by DeVISE. Also, while the DeVISE [6] model fine-tune the pre-trained CNN model, ConSE [9] kept the pre-trained model intact without training it any further.

## 3.2 Multi-label Zero-Shot Learning

All of the papers introduced in the previous section are limited to single-labeled problems. As most images can be labeled with more than one label, a few researchers have tried solving *Zero-Shot Learning* for multi-label problems.

One of the first to present a solution to the multi-label ZSL was Fu et al. [33]. With the lack of a dataset containing examples to learn co-occurrence the conventional way, they used transductive learning to exploit a label correlation at test time from the Skip-Gram model. They implemented two versions where one used nearest neighbor search for the matching label and the other used the knowledge from the known labels to find closest match. They successfully applied their strategy to previous mentioned models such as DeVISE, which made these models able to solve multi-label problems.

Ren et al. [34] introduced in 2015 a model with architecture similar to DeVise and ConSE, but for multi-label problems. They used the object detection framework Fast R-CNN [35] to generate boxes for each objects in the images and thus able to change a multi-label problem into a single-label problem. Inspired by the work of Frome et al. [6], each of the generated boxes are mapped to a semantic vector space using a single projecting layer. This produced promising results for multi-label ZSL, but with room for improvements.

Zhang et al. [32] presented in 2016 two different models inspired by the single-label solutions DeVISE and ConSE. One used a linear classification and the other used a neural network with four layers that maps the image representation from a pre-trained CNN into a 300-D representation native to the language model. In the neural network solution they use a loss function to train with both negative and positive feedback. This means that they train with both the correct ground truth labels as well as incorrect labels, to estimate the word vector in the language model. They implemented and tested various models, including ConSE [9]. They reported getting higher accuracy than the multi-label version ConSE with both the linear and neural network version solution.

Although there exist some papers on multi-label ZSL, it is not a problem that has been extensively studied.

## 3.3 Word Embedding Space

Most existing papers use a pre-trained language model that embeds words to word vector spaces. These models are trained on large amount of text, typically using Wikipedia articles trained with Word2vec [21] (see: [6, 9, 33, 7]) or GloVe [24] (see: [34]).

Akata et al. [27] did a comprehensive comparison of Word2vec and GloVe learned from a bird specific corpus, Wikipedia and also with or without a hierarchical embed-

ding. The results shows that combining output embeddings from text with a hierarchies such as WordNet [36] to build a semantic space is significantly better than only using a single embedding.

Also, a systematic comparison of Word2vec and GloVe was done in the work of Baroni et al. [37]. This paper compared the two models on textual score such as semantic relatedness and synonym detection, and concluded that the predictive model Word2vec scores higher than the counting model GloVe.

One issue with using Wikipedia to train the word vector space is that the space will be trained for textual data and the relationship between the words will mainly be based on textual relationship. For example, in images cup and table often appears together, which may not always be the case in articles. Giving cup and table a different relation in visual context compared to textual. A solution to this may be hard as there is significantly less resources describing the visual world, compared to the textual.

The different ZSL models uses different loss functions to train their models. These varies between hinge-loss [6, 34], cosine similarity [9] and euclidean distance [38, 27].

## 3.4 Deep Convolutional Neural Networks

The big breakthrough with deep *Convolutional Neural Network* (CNN) came in 2012 when Krizhevsky et al. presented AlexNet [39]. They won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [40] that same year and revolutionized the way computer vision challenges are solved. Every year after AlexNet, a CNN architecture have won ILSVRC with a top-1 accuracy going from 54% with AlexNet in 2012 to 80% using Inception-V4 [41] in 2016. Inception-V4 [41] is the forth version of GoogLeNet [42], the ILSVRC 2014 winner from Szegedy et al. from Google.

Many of the existing *Zero-Shot Learning* (ZSL) models uses a deep CNN model as visual model. Either the last softmax layer is removed and the feature from the last layer is used to project the representation of the image to the dimension of the semantic space [6, 32, 34], or the softmax layer is kept intact as in ConSE [9].

A pre-trained AlexNet was the chosen model by both DeVISE [6] and ConSE [9], and with the advances in CNN, Ren et al. [34] replaced AlexNet with GoogLeNet [42] in 2015. Today, both the AlexNet and GoogLeNet architectures are out of date, and upgrading to a current state-of-the-art CNN architecture would be a logical step to take to improve vision tasks in general.



# Chapter 4

## Design & Implementation

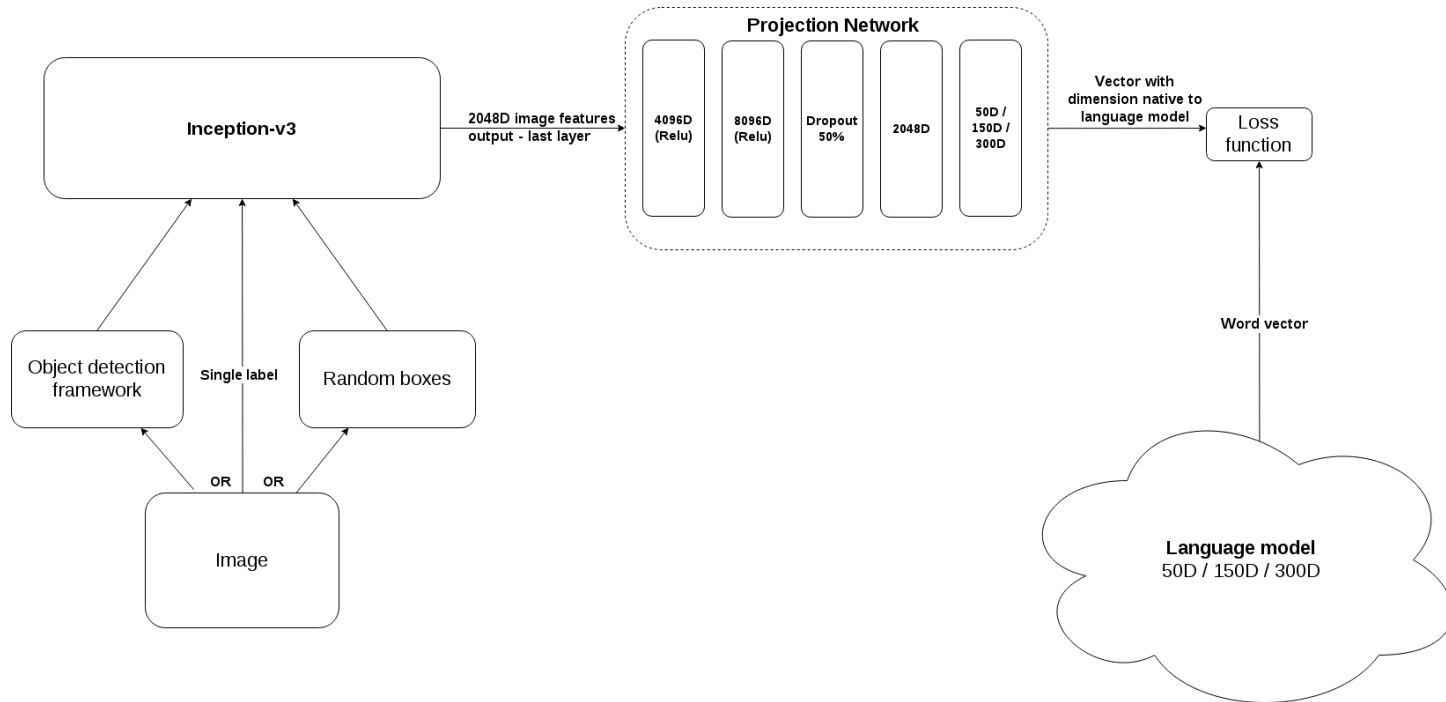
*This chapter introduces the architecture designed used to investigate the research questions. The chapter starts giving a general overview over the architecture, and explains throughout the rest of the chapter the choices for the implementation and how the architecture varies for different scenarios.*

### 4.1 The general model architecture

Figure 4.1 presents the designed solution. The design is inspired by different ideas that we have combined together to investigate if it can outperform state-of-the art in the field of ZSL.

The model works differently for a single-label problem compared to a multi-label problem. When working with multi-label data sets the model will split the image into a set of boxes, where each box runs separately through the rest of the model. After predicting for each box, the model counts and returns the labels that had highest number of predictions. This approach is inspired by the work of Ren et al. [34] that uses the object detection framework Fast R-CNN to generate the boxes. Our model have implemented box generating with both the option of using a object detection framework, but also have the option of generating boxes randomly. These two different approaches will be compared against eachother in the experiments. The reason to split the data into a set of boxes is because when working with multi-label, it is uncertain which parts of the picture belongs to which label. For example having a picture of both a cat and a dog gives the labels cat and dog, but will not say where the different objects are located, which is something we try to solve by using a model trained on single-label to help select label for box.

In both the multi-label and single-label case the image or sub-pictures are fed into



**Figure (4.1):** The general architecture showing suggestions to solve ZSL for both single-label and multi-label scenarios.



a CNN model to extract image features. Most of the CNN models mentioned in related work uses either AlexNet or GoogLeNet architecture to generate the image features. With image recognition significantly improved with newer models, we have chosen to use the state-of-the-art CNN architecture **Inception-v3** implemented in Keras [43]. We believe by utilizing a state of the art CNN, this can help us improve the performance of our ZSL model, because we get better features extracted. The model has been trained on the ImageNet Large Visual Recognition Challenge - ILSVRC [40] using the data from 2012. Inception-v3 is an upgraded version of the GoogLeNet architecture, and have improved the top-5 error from 6.67% to 3.46% on ILSVRC. Similar to the architecture of Ren et al. [34] the top layers are removed, making the model return 2048-D features, which is the result after the average pooling. The inception-v3 layers are frozen during training, meaning they will have the same weights before training as afterwards.

Inspired by the work of Zhang et al. [32] the design contains a four layered fully connected network that projects the 2048-D features to the dimension native to the language model, which is either 50-D, 150-D and 300-D. This network have the exact same layers Zhang et al. [32] and dimensions, but with the exception of the dropout layer having 50% dropout instead of 30%. Setting the dropout higher was chosen to prevent the model from over fitting.

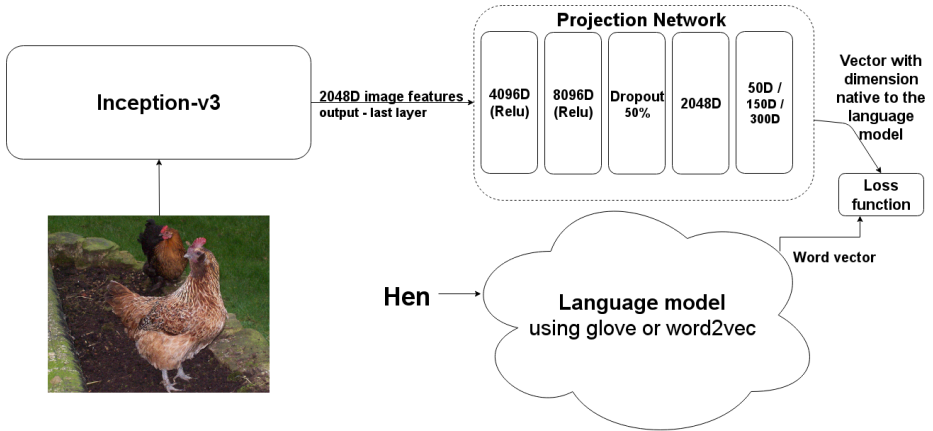
In a sense, we have mixed together ideas from different papers that we hope can enable us to make predictions for both single-label and multi-label scenarios. The architecture has variations for each of these scenarios which are described in the next sections.

## 4.2 Single-label scenarios

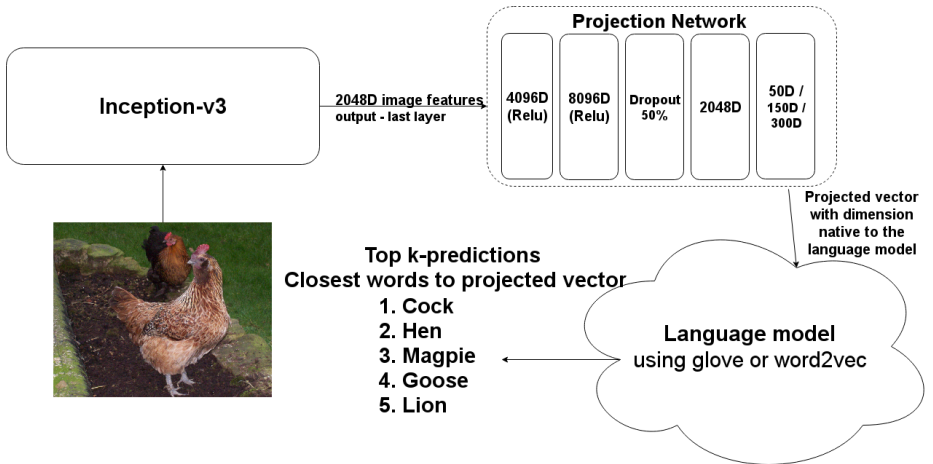
Figure 4.2 shows the architecture when training the model on single-label data that is strongly inspired by the work of Zhang et al. [32] and Frome et al. [6]. A projecting network is trained to map image-features extracted from a pre-trained inception-v3 to the native dimension of the language model. This is done by retrieving a word vector of the label using a pre-trained language model, and compare this word vector to the projected one from the projection network. The loss function then reports a loss that the projection network tries to minimize. This means that during training only the projection network is trained, while both the pre-trained inception-v3 and language model stays the same.

When testing on a single-label scenario the model returns the closest labels to the predicted word vector as shown in figure 4.3. The model first extracts image-features from inception-v3 and these features are passed through the projection network which outputs a predicted word vector. The words closest to this predicted vector is returned

as the top- $k$  result ranked after closest word first.



**Figure (4.2):** Showing the behavior when training the model on a single-label dataset.

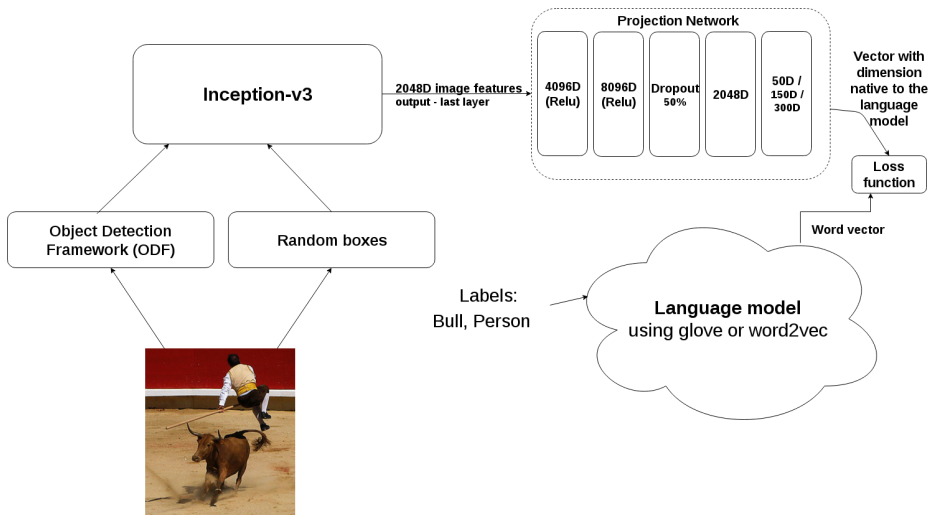


**Figure (4.3):** Showing the behavior when testing the model on a single-label problem. Here  $k = 5$ , meaning the model returns the closest 5 labels to the predicted vector.

### 4.3 Multi-label Scenarios - Generate image boxes

Working with multi-label data set, the model can be trained in two different approaches. One that randomly generates a set of boxes for each image and one using a *Object Detection Framework (ODF)*.

### 4.3.1 Multi-label training



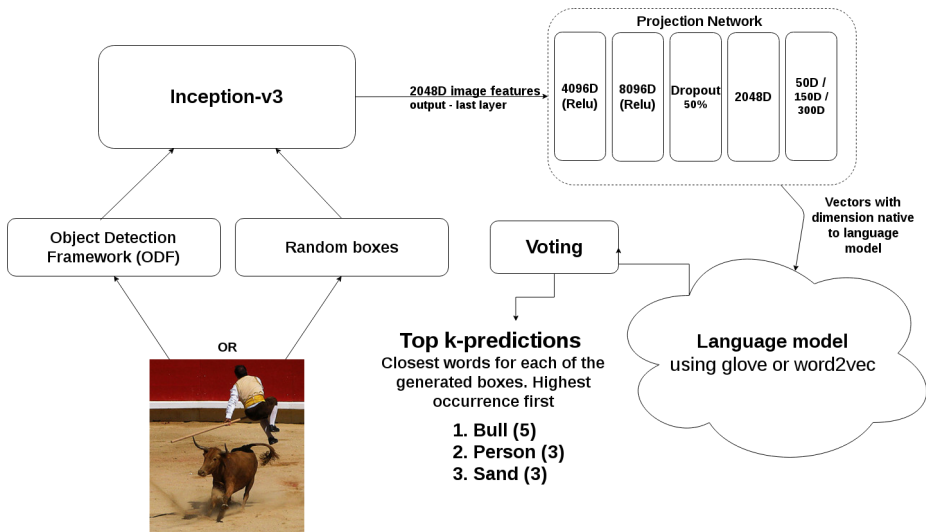
**Figure (4.4):** Suggested solution to train multi-label *Zero-Shot Learning* using by generating boxes

When training the model for a multi-label problem, the model first loads a network trained on a single-label data set. Then the model generates a set of boxes for each image, as shown in figure 4.4. Each of these boxes are separately sent through the network that generates projected vectors. Now the network will have a set of labels and a set of vectors each representing a box. The number of labels for each sample varies, so the model compares each label to the vector representations, and chooses the closest vector for each label. To enable the model to select the correct boxes, it depends on using a model pre-trained on the single-label data using the implementations in section 4.2. The only difference between the two multi-label implementations is how the image boxes is generated, but all other aspects of the network are similar.

Training on a multi-label scenario the model also only trains the projected network, meaning the model will work the same way as the single-label scenario and can be used to make predictions for both single-label and multi-label problems.

### 4.3.2 Multi-label testing

When making multi-label predictions the model loads the pre-trained weights, either just single-label weights or with weights that is tuned on multi-label data. The model makes a prediction for each of the boxes generated, and returns the labels in descending order with the highest occurring labels first.



**Figure (4.5):** Suggested solution to test multi-label *Zero-Shot Learning* using by generating boxes

For example, if the model generated 30 boxes where 20 predicted *Dog*, 5 predicted *Cat* and 5 predicted *Horse* the top-3 predicted list would return *Dog*, *Cat* and *Horse*. Looking at figure 4.5 we can see an example of this, where it ends up with predicting *Bull*, *Person*, *Sand* in this order because *Bull* had the most predictions.

### 4.3.3 Random image boxes

One of the approaches to generate the boxes for multi-label ZSL is to randomly generate boxes for each image as demonstrated in figure 4.6a and figure 4.6b. These boxes are generated using three different sizes of the image with and height: 1/2, 1/3, and 1/4. For each of the sizes 10 boxes are generated, which means the algorithm always returns 30 generated boxes in total.



(a) Original picture for illustrative purposes      (b) Same picture with the boxes - 30 boxes

**Figure (4.6):** Showing an example of how the picture could be cut. W and H corresponds to width and height.

### 4.3.4 Using Object Detection Framework (ODF)

Inspired by Ren et al. [34] the second approach for solving multi-label ZSL is to use a *Object Detection Framework* (ODF). The model has implemented to use boxes generated by both Faster R-CNN [1] and YOLOv2 [2]. Using two different ODFs we want to compare which one performs the best, and with two different frameworks there is naturally a higher chance for at least one of them giving a high score. The YOLOv2 model is already trained on a total of over 9000 objects, while Faster R-CNN we have trained ourselves on the ILSVRC [40] 2015 detection data set, which has 200 object classes.

The Faster R-CNN generates in total 2000 boxes for each image, and gives a score to each box on objectiveness. The model selects the top 100 boxes with highest score and uses Non-Maximum Suppression<sup>1</sup> [20] (section 2.5.1) to remove boxes that overlap too much.

Using YOLOv2, we have reduced the threshold for generating boxes to 0.01 to make sure YOLOv2 generates boxes for all images. The same approach was used where the top 100 boxes with highest score are selected with Non-Maximum Suppression, with the threshold set to 0.5, to remove overlapping boxes. Now the model will be left with an unequal number of boxes ranging from 5 around 30, and for each of the boxes the projection network predicts a word vector and the closest label is returned.

---

<sup>1</sup>Non-Maximum Suppression code used: [https://github.com/rbgirshick/py-faster-rcnn/blob/master/lib/nms/py\\_cpu\\_nms.py](https://github.com/rbgirshick/py-faster-rcnn/blob/master/lib/nms/py_cpu_nms.py). Downloaded: 10th of May 2017

## 4.4 Language Model

A *Zero-Shot Learning* (ZSL) model relies on the existence of a set of labeled seen classes and a knowledge about how these classes are semantically related to the unseen classes. The seen and unseen classes together generate together a higher dimensional vector space, called semantic space. By doing so, a classifier is able to predict an unseen class using the semantic space and the knowledge from the seen classes.

There are different ways of creating such a semantic space, one common way is to use high level attributes. For example a zebra could have the attributes tail, stripes, animal, etc. Collecting these attributes is however a very costly approach as they are manually collected by humans, therefore we have chosen to train a language model using the word embedding (section 2.6) techniques GloVe [24] and Word2vec [21]. The benefit with word embedding is that it can take a large corpus of unannotated text and train a model that puts similar words close together in a word space.

### 4.4.1 Training language models

The language models are trained using a large text corpus from the English Wikipedia. To train the language models the original GloVe code from Stanford [44] and Word2vec with Gensim [45] was used. A large xml file containing the whole English Wikipedia was downloaded<sup>2</sup>, where we used the tools presented by Mudge [46] to transform the xml files to one large text document containing only the relevant information. Doing this extracted the relevant parts of the XML and then generated multiple files containing the relevant parts. Next a python script was used to combine these files into one to transform this into one large text document where each article is on one line. Before training, the text corpus had to be processed to make sure to include compound words like "police station" and "garage door". This was a crucial step that made sure that the text corpus had words that only makes sense together, such as the ones mentioned. To connect these compound words, we connected words with underscore (\_) when there was two or more consecutive nouns after each other<sup>3</sup>. This text document was used to both train GloVe and Word2vec, and these models contain exactly the same words.

*GloVe and Word2vec was trained with window\_size 15 and vocab\_min\_count 5. The iterations over the dataset was left at standard on glove and Word2vec, which was 15 for GloVe and 5 for Word2vec.*

---

<sup>2</sup>The wikipedia dump can be located at <https://dumps.wikimedia.org/enwiki/> where we have used *enwiki-20160820-pages-articles.xml*. Downloaded: 15th September 2016

<sup>3</sup>The code for connecting compound words can be located here: <https://github.com/RaRe-Technologies/gensim/issues/881> Downloaded: 13th of March 2017

## 4.4.2 Predicting word vectors

The projected network will generate a vector with dimension native to the language model. To find the word vectors closest to this vector, we use Approximate Nearest Neighbor Oh Yeah (ANNOY) [47], developed by Spotify. ANNOY uses euclidean distance to search for closest vectors, and it is also common to use cosine similarity. Since we are only interested in ranking the vectors by closest word and not the actual calculated distance/similarity both the euclidean distance and cosine similarity will return the same ranking order when the vectors are  $l_2$ -normalized. This is because euclidean distance acts as a decreasing version of cosine distance, which means the most similar in cosine similarity is also the closest in euclidean distance.

In figure 4.7 we can see that euclidean distance and cosine similarity yields the same space, in the normalized case. If these two were different, we would expect two different t-SNE representations of their space.





## 4.5 Technical information

### 4.5.1 Deep learning frameworks - Keras and Tensorflow

With the growing interest in deep learning, many new frameworks have been developed for developing neural networks. From these we have chosen to use Keras [43] as our main framework, with Tensorflow [49] as backend. Keras gives us a level of abstraction that makes it easier to develop prototypes, as well as a big community around it. With Tensorflow as backend, the library uses CUDA technologies to boost the performance significantly by using GPU during training of the neural network.

Keras also has a Inception-v3 model pre-trained on the ImageNet data that we have used in the implementation.

### 4.5.2 Hardware specification

When working with large data and deep neural networks, a big issue is computing power. We used two personal computers, each running with Titan X Pascal GPU. The architecture is large, and takes a large amount of space on the GPU and RAM. Having the Titan X Pascal GPU and a memory card with a total of 32GB RAM has significantly reduced compute time, and thus helped making this thesis possible.

The datasets presented in section 5.1 are stored locally on each of the computers, requiring about 1 TB of space divided over multiple hard disks and SSDs. This also includes the language models which is around 30GB in one file and the text file after preprocessing of around 10GB.



# Chapter 5

## Experiments & Results

*This chapter has experiments on the implementation to investigate the research goal and questions, before reporting the result. The chapter contains multiple experiments, for different scenarios, data and language models. To evaluate the proposed model, we conduct experiments on two main tasks: One single-label Zero-Shot Learning (ZSL) and different approaches to multi-label ZSL. Each experiment is described and presented with a goal. The goals are connected to one or several hypotheses each, which we will conclude if met or not.*

### 5.1 Datasets overview

As the architecture is designed for both single-label and multi-label problems, there is a need to use datasets to fit these scenarios. We have downloaded datasets available publicly for research purposes, chosen on background of sizes and which datasets are used by related work.

Table 5.1 gives an overview of the datasets we have chosen to use. The selection of datasets are chosen from what similar papers have used.

**Table 5.1:** Class and size of images of popular *Zero-Shot Learning* datasets

<b>Datasets</b>	<b># of images</b>	<b># of classes</b>	<b>Multilabel problem</b>
ImageNet ILSVRC15 DET [40]	456,567	200	No
ImageNet ILSVRC15 CLS-LOC [40]	1,281,167	1000	No
Nus-Wide [50]	269,648	5018	Yes

The ImageNet datasets are collected and organized according to the WordNet [36] hierarchy. ImageNet has two different datasets which are provided by Stanford Vision Lab from Stanford University. ImageNet DET is a dataset where bounding boxes are provided for each image and has bounding boxes for 200 different classes. ImageNET CLS-LOC is a dataset with one class per image and it has 1000 different classes.

The NUS-WIDE dataset is a collection of images and their associated tags from Flickr, with a total of 5018 tags. It was collected by the National University of Singapore and in our multi-label experiment we have selected to use only the 1000 most popular tags and a selected 81 concepts list.

## 5.2 Normalization Layer test

The goal for this test was to determine whether or not normalization should be used in our last layer of the projection part of the network. Zhang et al. [32] used normalization to compare the word vectors with the projected image to vector space. Therefore we wanted to conduct an experiment to see if the normalization was important for our results.

**Table 5.2:** Results for ImageNet1k single label prediction  $k = 10$ , without l2-normalization after last layer.

Loss	MAP@10 (%)	Flat hit@10 (%)
Cosine similarity	<b>5.10</b>	<b>17.50</b>
Euclidean distance	1.70	4.70
Hinge	3.60	13.50
Squared hinge	1.50	4.10

**Table 5.3:** Results for ImageNet1k single label prediction  $k = 10$ , included l2-normalization after last layer.

Loss	MAP@10 (%)	Flat hit@10 (%)
Cosine similarity	<b>5.30</b>	17.10
Euclidean distance	5.10	17.90
Hinge	5.00	17.00
Squared hinge	<b>5.30</b>	<b>18.10</b>

Table 5.3 and table 5.2 shows that the predicting results are significantly better when adding normalization before the loss function. The reason cosine similarity reports sim-

ilar result for with and without normalization is because Keras automatically normalizes the data when using cosine similarity. After adding normalization the loss functions predict more similar results with squared hinge being slightly better. The experiments after this will use l2-normalization.

*Note: This experiment was done before we had made our language model include compound words. This is the reason the MAP and flat hit is not the same to the single label experiment later.*

## 5.3 Selecting parameters

The parameters were selected early in the experimentation. With the experiments being large and each one taking days to complete, there was not enough time to adjust them to test with many different parameters. Only a small experiment was run between loss functions to decide which one was the most optimal one, where the best was chosen for the main experiments.

### 5.3.1 Optimization and learning rate

The selected optimizer used by the experiments is the Adaptive Moment Estimation (Adam) [51]. Adam optimizer is known to be a stable optimizer as it prevents over fitting and leads to convergence faster. The learning rate,  $\eta$ , was set to  $\eta = 0.002$ , which is the default in Keras [43] when using Adam.

### 5.3.2 Batch Size

The batch size defines the number of samples that will be propagated through the network each iteration. The batch size effects how quickly a network trains, and how accurate the estimation of the gradient is. With a larger batch size the network typically trains quicker, but requires more memory. With smaller batch sizes the models typically gives less accurate estimations of the gradient, while using less memory than larger batch sizes. Since our model is quite large and requires a lot of memory, the batch size was set to 32. Batch size of 32 is a common number in many applications and is also Keras' default value.

### 5.3.3 Loss function

Selecting the loss function in any machine learning problem can be a crucial part and effect the result greatly. As the model proposed is trying to map features from an image

to a dimension native to the language model, a loss function that calculates the distance or similarity between two continuous vectors is needed. All of the experiments described are time consuming, and can not be trained for all different loss functions. Therefore, four different loss functions were tested on a small experiment to decide which loss function will be used. These four were: *cosine similarity*, *euclidean distance*, *hinge* and *squared hinge*.

The small experiment uses the language model trained on GloVe in 300-dimensions, and uses the same dataset and split as described in section 5.4.1. It is tested on a space containing only unseen classes (the space showed in figure 5.1).

**Table 5.4:** Results for four different loss functions. Trained on ImageNet 1k, and tested on a space with only unseen labels with different values of  $k$ .

Loss	MAP@ $k$ (%)				Flat hit @ $k$ (%)			
	1	2	5	10	1	2	5	10
Cosine similarity	12.0	14.5	17.0	18.3	12.0	17.0	26.0	35.8
Euclidean distance	12.3	14.9	17.6	18.9	12.3	17.5	27.3	36.7
Hinge	11.3	13.9	16.5	17.8	11.3	16.5	26.0	35.8
Squared hinge	<b>13.3</b>	<b>16.0</b>	<b>18.4</b>	<b>19.6</b>	<b>13.3</b>	<b>18.6</b>	<b>27.4</b>	<b>37.0</b>

Table 5.4 shows that squared hinge outperforms the other tested loss functions by a small margin. Based on this, squared hinge is chosen as the loss function for all later experiments in the thesis.

### 5.3.4 Iterations and Stopping criterion

Traditionally one iteration, or one epoch, is having the model train on all samples in the training set one time. However, when working with a large number of samples the experiments uses a fixed number of batches for each iterations. This number is set to 5000, meaning the models train on the total of  $5000 * 32$  images per iteration. At the start of each iteration the training set is shuffled, making the order of the image-list random.

The number of iterations can either be set to a fixed number or decided from a stopping criterion. As it can be hard to know when a model has reached its full potential, it can be hard to select a fixed number of iteration. Therefore a stopping criterion has been chosen. After each iteration, the model is tested against a validation set containing 20,000 images. Testing on the validation set reports a validation loss, when the validation loss has not changed over three iterations, the model stops training. Also, the model is saved each time the validation loss is improved, in case the training fails to complete.

## 5.4 Experiment 1 - Single-label Zero-Shot Learning

### 5.4.1 Description

This experiment will use the ImageNet ILSVRC15 CLS-LOC [40] dataset, which as described in section 5.1, contains 1.2 million images with a total of a 1000 classes. Following the same instructions as Mensink et al. [52] the experiment will split the data set into a seen set of 800 classes and an unseen set of 200 classes, which are randomly selected. Before starting the experiment, the classes that do not exist in the language model needs to be removed. As all the language models are trained on the same corpus of text, they contain the same words but it is not a guarantee that they contain all the class names in ImageNet 1k.

The experiment seeks to investigate how increasing the dimensions of the language models will effect the result. The language models are trained using both GloVe and Word2vec, and these will be trained on three different dimensions: one 50-dimension, 150-dimension and a 300-dimension space. As a result, six different models will be trained. Each of which will make predictions for two spaces, one containing only unseen classes and another containing both unseen and seen classes.

Figure 5.1 and figure 5.2 shows a t-SNE [48] representation of the two different prediction spaces using the 300-dimension GloVe. As the spaces are originally in 300-dimensions it can be hard to visualize as a 2-D figure, but using the dimension reduction technique t-SNE [48] the figures demonstrates how the unseen space contains less words and therefore contains less noise.





## 5.4.2 Goal of the experiment

This experiment will test if our proposed architecture can get good results when predicting single-label Zero-Shot Learning. A similar experiment was run by Frome et al. [6] in their model DeVISE, where they had a Flat hit@5 accuracy of 31.8% when predicting on the 200 unseen classes and 9.0% when the model could predict on both unseen and seen classes. This experiment aims to get similar results as theirs, and by this it seeks to demonstrate that the architecture returns promising results before applying it to the multi-label problem.

The experiment will investigate how the architecture scale both with language model dimensions and number of classes the model can predict on. Testing different language model algorithms the experiment seeks to find out if GloVe or Word2vec is best suitable for our ZSL architecture.

**Hypothesis 1:** We hypothesize that our implementation will be able to predict unseen labels for the single-label problem, with similar results as Frome et al. [6].

**Hypothesis 2:** The experiment will test how different dimensions of the language model algorithms will effect the result. Having higher dimensions in language models enables them to store more information, and will return better result.

**Hypothesis 3:** Increasing the word space with more words will generate more noise for the model and the model will suffer to predict the correct unseen label.

## 5.4.3 Results

Table 5.5 shows the result for single-label Zero-Shot Learning. A total of 185 classes did not have a word vector in the language models and was removed. This left a total of 815 classes in the dataset, 651 classes in the seen set and 164 in the unseen. The amount of images was reduced from 1.2 million to 1,051,219 images, where 839,930 images in the training set and tested on 211,289 images with unseen classes.

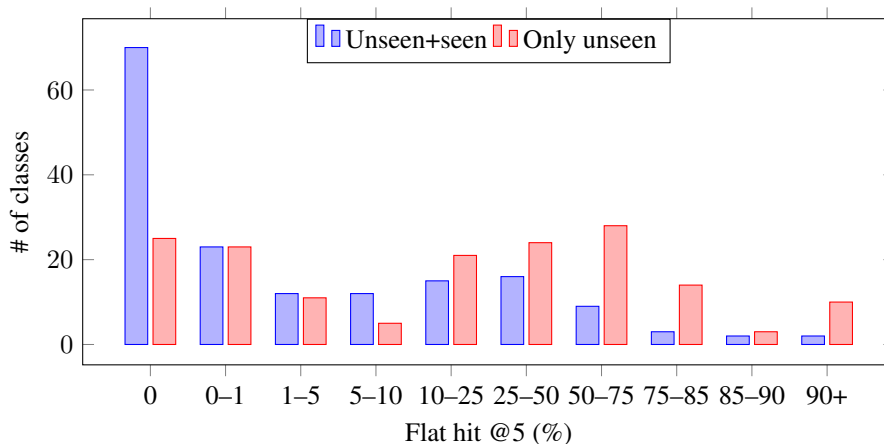
In table 5.5 the predict space column shows which words were inside each space. In the language model column, w2v is short for Word2vec. The results are measured in MAP and Flat Hit, described in section 2.7. With  $k = 5$  the model returns the 5 closest labels to the predicted vector, and calculates the score. Increasing  $k$  will make the model return more proposals, and the chance for the correct one to be among these is naturally higher.

The table 5.5 shows that the Word2vec algorithm is best suitable to use as language model. With Word2vec outperforming GloVe in all the scenarios. Also, increasing the dimension of the language model clearly improved accuracy for both Word2vec and GloVe.

From the two spaces, as expected the unseen space had significantly better result

**Table 5.5:** Results for ImageNet1k single label prediction measured with different values of  $k$  and on two predict spaces (which words are available to predict on) using different language models in different dimensions. In the language model column, the names first describes the language model method before the dimension. For example, GloVe-50D is trained with GloVe and represented in a space with 50-dimensions.

Predict Space (# Candidate labels)	Language model	MAP@ $k$ (%)				Flat hit @ $k$ (%)			
		1	2	5	10	1	2	5	10
Seen+Unseen (651+164)	GloVe-50D	0.26	0.78	1.42	2.02	0.26	1.30	3.89	8.42
	GloVe-150D	0.12	1.46	3.22	3.89	0.12	2.79	9.25	14.41
	GloVe-300D	0.03	2.11	3.84	4.71	0.03	4.19	10.86	17.47
	Word2vec-50D	<b>0.45</b>	1.75	3.1	3.77	<b>0.45</b>	3.06	8.14	13.21
	Word2vec-150D	0.13	1.82	3.78	4.73	0.13	3.52	10.8	17.99
	Word2vec-300D	0.04	<b>2.31</b>	<b>4.40</b>	<b>5.44</b>	0.04	<b>4.58</b>	<b>12.30</b>	<b>20.18</b>
Unseen (164)	GloVe-50D	3.48	5.45	7.80	8.98	3.48	7.43	16.26	25.37
	GloVe-150D	10.41	12.50	14.70	15.89	10.41	14.59	22.71	31.81
	GloVe-300D	12.01	15.04	17.80	19.00	12.01	18.07	28.51	37.17
	Word2vec-50D	9.01	11.27	13.83	15.24	9.01	13.54	23.05	33.8
	Word2vec-150D	14.34	17.34	20.19	21.72	14.34	20.33	30.95	42.37
	Word2vec-300D	<b>15.42</b>	<b>18.9</b>	<b>21.86</b>	<b>23.13</b>	<b>15.42</b>	<b>22.38</b>	<b>33.0</b>	<b>42.66</b>



**Figure (5.3):** Flat hit@5 accuracy distributed among all classes using Word2vec in 300-D vector space as language model.

than the space containing both the unseen classes and seen classes. More words create more noise and might have been one of the main reasons the results dropped so much when adding seen classes to the space. However, this would mean a model needs to be told if the object is unseen first which is unrealistic in most real world applications.

Figure 5.3 sorts the score distributed for each class. With flat hit @5 and on predicted space including seen and unseen labels, a total of 70 classes reported to miss on every sample, reporting a accuracy of 0%. While on the unseen space, this is reduced to 25. However on both of the spaces there is a large difference between the classes with some classes missing on every sample, and some classes reporting a hit accuracy higher than 90%.

### Comparative results

The results are compared against 4 alternatives in table 5.6, where flat hit@5 is used. As we removed some classes from both the seen and unseen set the results can not be compared directly, but gives us a good overview of where the results should be.

Also as we learned from figure 5.3 some classes got very high score, while some missed on every sample. With the unseen and seen split being random, being lucky with the split can also effect the results.

**Table 5.6:** Comparative results measured in flat hit@5

Model	Unseen	Unseen+seen
ConSE [9] <sup>1</sup>	28.5%	-
DeViSE [6] <sup>1,2</sup>	31.8%	9.0%
Mensink et al. [52] <sup>1,2</sup>	35.7%	1.9%
Zhang et al. [53] <sup>1</sup>	60.7%	-
Ours (table 5.5)	33.0%	12.3%

<sup>1</sup> Results not reproduced by us, but reported by Zhang et al. [53].

<sup>2</sup> Results also reported by Frome et al. [6].

#### 5.4.4 Was the goal met?

The goal (see section 5.4.2) for this experiment was to show that our ZSL architecture managed to predict single-label ZSL with expected results. In terms of the hypotheses the results show:

**Hypothesis 1:** The results show that the architecture is able to predict unseen labels for a single-label problem with similar results as DeViSE [6]. The best model manages a Flat hit score of 42.66 % on the unseen space when using  $k = 10$ , on Word2vec with 300 dimensional vectors.

**Hypothesis 2:** Increasing the dimension of the language models clearly improved the result as expected. With Word2vec in 150-dimensions reporting an Flat hit@5 accuracy of 30.95 % and 300-dimensions 33.0 %.

**Hypothesis 3:** Having a predicting space only containing the unseen labels did significantly improve the results, which is not surprising. Introducing more labels introduces more noise and do generally lead to a worse result and was not surprising.

In summary, the goal was met with expected results which provide a good basis for the multi-label experiment.

## 5.5 Experiment 2 - Multi-label Zero-Shot Learning

For the multi-label scenario we have split the experiment into two parts. One experiment where the trained model on the single-label data is used directly to predict for multi-label and one experiment that tunes the pre-trained model from experiment 1 on multi-label data before testing.

The experiments loads the model from experiment 1 that reported highest score, which table 5.5 shows is trained using Word2vec with 300 dimensions. The weights

from this model are loaded to the projection network for all models in this experiment.

Following the same configuration as Zhang et al. [32], the 81 concept tags in NUS-WIDE [50] will be used as unseen set with the most frequent 1000 Flickr tags forms the seen set. A total of 75 tags are shared with the 81 concept and 1000 tags, resulting in a seen set with the remaining 925 tags.

Same as in experiment 1 in section 5.4, the experiments will also predict for two different spaces, having one with only unseen classes and one with both unseen and seen classes. As this is a multi-label scenario, an image can contain multiple objects resulting in objects containing both unseen and seen labels. For example, if an image contains both cat and dog where cat is an unseen label and dog is seen, the vector space with only unseen labels will only contain cat and cat will be the actual label. For the vector space containing both unseen and seen, cat and dog will be kept as labels, which means that if the model is able to predict the seen label but fails to predict the unseen, the model will still get points for the seen label.

### **5.5.1 Experiment 2.1 - Tune model on multi-label dataset**

This experiment tunes the pre-trained model from experiment 1 on multi-label data. We will run one experiment following the random boxes generation implementation from section 4.3.3 and two experiment using a ODF from section 4.3.4. The ODF experiments will test the two ODFs Faster R-CNN [1] and YOLOv2 [2]. These two are trained on different data, with YOLOv2 model being pre-trained on 9000 objects and Faster R-CNN trained ourselves on the ILSVRC [40] 2015 detection data set with 200 object classes.

### **5.5.2 Experiment 2.2 - Test single-label model directly on multi-label data**

This experiment will test if a model trained on a single-label data set can be used to predict for multi-label ZSL. The experiment uses the model trained in experiment 1 that followed the description in section 5.4.1.

The experiment seeks to investigate if the model can be trained on a single-label data and later make predictions for multi-label problems using the different box generators only for testing.

### **5.5.3 Goal of the experiment**

The purpose of this experiment is to demonstrate that by splitting an image into a set of boxes, the architecture can also predict for multi-label ZSL, both predicting multi-label

after training for a single-label data.

**Hypothesis 1:** The model tuned on multi-label data (section 5.5.1) will outperform the one only trained on single-label images in section 5.5.2.

**Hypothesis 2:** Using a *Object Detection Framework* (ODF) will outperform the randomly generated boxes. This because the boxes will have more success in finding the objects in each image. Among the two different ODFs YOLOv2 [2] will outperform Faster R-CNN [1].

### 5.5.4 Results

Table 5.8 shows the result for multi-label ZSL. Same as in experiment 1, classes that did not have a word vector in the language models was removed. In NUS-WIDE, only 5 out of the 925 and none of the 81 concept classes missed a word vector. A total of 78,530 images was tested for both unseen and seen+unseen prediction space. The results are measured in MAP (section 2.7.2) with different k-values and one value, gt, where the k value is equal to the number of possible predicted words.

In addition to measuring the results in MAP we added missed average cosine. Missed average cosine calculates cosine similarity between every missed labels and its ground truth. With this we want to measure not only hit-miss ratio, but also how far away each missed prediction was the ground truth.

The table shows that tuning the model on a ODF using the details from section 4.3.4 and making predictions using the random box generator in section 4.3.3 have highest accuracy. When training the model with random boxes, the results are even worse compared to not tuned at all.

#### Comparative results

The results from table 5.8 are compared against three alternatives in table 5.7, that are all reported by Zhang et al. [32]. Our best model did not perform better than Fast0Tag by Zhang et al. [32], but had similar results to the multi-label ConSE reported by Zhang et al. [32].

**Table 5.7:** Comparative results measured in MAP@gt, our results compared to the ones listed by Zhang et al. [32]. gt is the number of possible words predicted.

<b>Model</b>	<b>Unseen</b>	<b>Unseen+seen</b>
ConSE [9]	32.4%	12.5%
Fast0Tag - linear [32]	40.1%	18.8%
Fast0Tag - ANN [32]	42.4%	19.1%
Ours (table 5.8)	33.0%	11.1%

**Table 5.8:** Results for multi-label ZSL tested on the NUS-WIDE [50] dataset. We have made predictions for both unseen and seen space, and with a model trained on single-label with the description from section 5.5.2 and one following section 5.5.1 where the single-label model is tuned on multi-label data. The multi-label model is both trained and tested with a box generator, while the single-label model is only tested using the designated box generator. With k value equal to gt it sets k value equal to the number of candidate labels in the prediction space, 1001 for seen+unseen and 81 for unseen.

Predict Space (# Candidate labels)	Tuned on multi-label	Box generator	MAP@ <i>k</i> (%)					missed Avg. Cosine <sup>3</sup>
			1	2	5	10	gt	
Seen + unseen (920 + 81)	No	Random generator	8.70	5.46	3.00	2.50	3.24	0.26355
	No	Faster R-CNN	0.76	0.60	0.43	0.39	0.39	0.23806
	No	YOLOv2	0.62	0.45	0.21	0.15	0.14	0.24769
	Yes	Random generator	1.47	1.93	1.60	1.35	1.27	0.22845
	Yes	Faster R-CNN <sup>1</sup>	4.53	3.36	2.07	1.51	1.34	0.25725
	Yes	Faster R-CNN <sup>2</sup>	<b>30.29</b>	<b>28.71</b>	<b>31.23</b>	<b>32.79</b>	<b>11.13</b>	0.32489
	Yes	YOLOv2 <sup>1</sup>	3.37	3.26	2.30	1.59	1.39	0.29646
	Yes	YOLOv2 <sup>2</sup>	26.48	17.72	10.18	8.17	7.73	<b>0.33408</b>
Unseen (81)	No	Random generator	11.86	10.23	11.04	11.82	10.48	0.19299
	No	Faster R-CNN	2.46	2.72	3.67	4.56	4.77	0.17813
	No	YOLOv2	1.65	1.72	1.71	1.70	1.70	0.17166
	Yes	Random generator	2.31	3.22	4.48	5.18	5.21	0.16872
	Yes	Faster R-CNN <sup>1</sup>	9.97	7.97	7.30	7.33	7.33	0.17489
	Yes	Faster R-CNN <sup>2</sup>	<b>31.45</b>	22.96	14.39	11.76	<b>32.95</b>	0.22316
	Yes	YOLOv2 <sup>1</sup>	12.06	9.01	8.13	8.16	8.16	0.22409
	Yes	YOLOv2 <sup>2</sup>	25.85	<b>23.42</b>	<b>25.03</b>	<b>26.16</b>	26.23	<b>0.23475</b>

<sup>1</sup> Trained and tested with ODFs.

<sup>2</sup> Trained with ODFs, but tested with random generator.

<sup>3</sup> Range [-1,+1], where 1 is most similar. Calculated closest ground truth label for each missed prediction.



### 5.5.5 Was the goal met?

The goal (see section 5.5.3) for this experiment was to train and test a model in multi-label ZSL.

In terms of the hypotheses the results show:

**Hypothesis 1:** Tuning the data on multi-label as described in section 4.3.1, did not always improve the results. Using the random boxes approach reported a worse result compared to the weights from single-label. However, training using the ODF approach did improve the model.

**Hypothesis 2:** The ODF outperformed the randomly generated boxes when training on the ODF boxes. However, generating random boxes outperformed ODF when making predictions. Comparing YOLOv2 and Faster R-CNN, the results showed that Faster R-CNN performed better during training, while YOLOv2 performed better during prediction.

These are surprising results, and the gap between the models that reported high accuracy and the ones that missed was large.



# Chapter 6

## Discussion

*In this chapter we discuss and analyze the results from chapter 5. The chapter tries to explain what the results might mean, how valuable they are and why. We discuss these results in terms of the research questions introduced in section 1.2.*

### 6.1 Language models & prediction spaces

The first experiment tested different techniques for single-label ZSL and compared these to find the best suitable model to train in our multi-label scenario.

Using Word2vec with 300-dimensions has the highest accuracy, with a 3 % higher flat hit@5 accuracy compared to the 150-dimension model. This indicates that increasing the dimension improves the accuracy and it would be interesting to train a 500- and 1000 dimensional language model, as Frome et al. [6] did in their model, to see if the accuracy continued to improve. Frome et al. [6] also used the Skip-Gram algorithm while we used CBOW, and since Mikolov et al. [22] reported that Skip-Gram got slightly better results on the semantic relatedness, we have suggested to test Skip-Gram in future work (section 7.2).

In our experiments, Word2vec outperformed GloVe and with GloVe considered a count based method, our results supports the work of Baroni et al. [37], where they concluded that Word2vec was better than count based methods in terms of semantic relatedness. In our experiment, even the Word2vec model trained on 150 dimensions outperformed the best GloVe model of 300 dimensions.

The language models used are all trained on the same text corpus, the English Wikipedia. When making predictions for images, it is natural to assume that having a language model based on image descriptions would outperform the English Wikipedia corpus. For example, cup and table may occur often together in an image, but may not

be the case in a textual context and is therefore not considered close in the language model. Not able to locate a large enough image description database, we have not been able to train a language model on image descriptions or tune the Wikipedia model with image descriptions.

The results from chapter 5 display quantitative results on the experiments. However, a more in depth analysis is needed to see what the models predict to understand how well they work. When working with a large number of sample images, it is impossible to look at every single prediction to analyze them. Therefore we have in table 6.1 chosen to take a few random samples from experiment 1 and used the best scored model to make predictions, and try to make sense of these.





Table 6.1 shows the input image with the five closest words to the predicted vector for the unseen and seen+unseen spaces. When all the seen labels are removed from the predicted space, the unseen labels are pushed up. So when showing a picture of a hen, the model predicts the seen label cock as top-1 and with hen as the closest word to cock, this gets pushed up as the closest word to the predicted vector when removing seen labels. The same happens when making a prediction on the input image of a leaf hopper, the correct label is not in top-5 when including the seen labels, but jumps up as the third closest label when removing all seen labels.

Removing the seen labels from the prediction space is however an unrealistic scenario. This is because in most cases a model do not know if the object is unseen or seen before making a prediction, to enable it to predict on only unseen words it would need to be able to tell that this is a new object. For example, when a child sees a zebra for the first time, it may be able to tell that it has never seen one before and that even if it may look like a horse it is not. Having a similar approach where the model could tell if it had seen the object before could enable the possibility to only predict on unseen words.

This means that even if our model have a high accuracy for predicting on only unseen labels, it may still be a poorly generalized model as we are letting the model cheat by removing noise from all the seen labels. Also the noise depends on the number of unseen labels we have chosen to include. In our experiments, we have chosen to let the unseen words be a set of chosen words we want to make predictions on. One could choose to include all the words available in the language model that would create way more noise and perhaps even make the flat-hit score go down to zero when calculating top-10.

For all the five examples included in table 6.1 the first predicted word in the unseen+seen space is a seen label. Looking at table 5.5 the model has a flat hit @ 1 accuracy of only 0.45 % for the seen+unseen space. This may mean that the model almost never predicts an unseen word, but only make a guess to what seen label it looks most similar to, which in return may mean that the results will always be limited on the language model and relies on visual similar objects to be placed together in the language

**Table 6.1:** Top-5 results for five randomly selected images from the ImageNet competition. The correctly annotated labels have a checkmark ✓ beside it. Best viewed in colors.

Images	Seen (blue) + unseen (red)	Only unseen
 <p><b>Figure (6.1):</b> Hen</p>	<ol style="list-style-type: none"> <li>1. Cock</li> <li>2. Hen ✓</li> <li>3. Magpie</li> <li>4. Goose</li> <li>5. Lion</li> </ol>	<ol style="list-style-type: none"> <li>1. Hen ✓</li> <li>2. Lion</li> <li>3. Ox</li> <li>4. Kelpie</li> <li>5. Baboon</li> </ol>
 <p><b>Figure (6.2):</b> Ocarina</p>	<ol style="list-style-type: none"> <li>1. Pencil sharpener</li> <li>2. Safety pin</li> <li>3. Screwdriver</li> <li>4. Plunger</li> <li>5. Sock</li> </ol>	<ol style="list-style-type: none"> <li>1. Plunger</li> <li>2. Washer</li> <li>3. Power drill</li> <li>4. Wine bottle</li> <li>5. Pickup</li> </ol>
 <p><b>Figure (6.3):</b> Ox</p>	<ol style="list-style-type: none"> <li>1. Plow</li> <li>2. Tractor</li> <li>3. Ox ✓</li> <li>4. Shovel</li> <li>5. Water buffalo</li> </ol>	<ol style="list-style-type: none"> <li>1. Ox ✓</li> <li>2. Kelpie</li> <li>3. Tripod</li> <li>4. Upright</li> <li>5. Samoyed</li> </ol>
 <p><b>Figure (6.4):</b> Leaf hopper</p>	<ol style="list-style-type: none"> <li>1. Leaf beetle</li> <li>2. Damselfly</li> <li>3. Dragonfly</li> <li>4. Lacewing</li> <li>5. Vulture</li> </ol>	<ol style="list-style-type: none"> <li>1. Dragonfly</li> <li>2. Wolf spider</li> <li>3. Leaf hopper ✓</li> <li>4. Weevil</li> <li>5. Marmoset</li> </ol>
 <p><b>Figure (6.5):</b> Police van</p>	<ol style="list-style-type: none"> <li>1. Ambulance</li> <li>2. Stretcher</li> <li>3. Oxygen mask</li> <li>4. Fire engine</li> <li>5. Police van ✓</li> </ol>	<ol style="list-style-type: none"> <li>1. Oxygen mask</li> <li>2. Police van ✓</li> <li>3. Crash helmet</li> <li>4. Forklift</li> <li>5. Pickup</li> </ol>

model.

With the results for experiment 1, we saw in figure 5.3 that classes had a large difference in accuracy. With some classes having high accuracy, some average and some very low. Table 6.2 shows the five classes with highest score on this experiment with their closest seen labels. These labels might give answers to why some classes reported very good accuracy. The best scored label is 'space bar' that has a flat hit score of 95.69. 'Space bar' refers to the space bar on the computer keyboard which also is the closest seen label. It is natural to assume that having a seen label that is both visually similar and also the closest word vector to the unseen will create an accuracy that is high, for example the model predicts computer keyboard as first prediction and then space bar will be picked up already when estimating top-2 words. Looking at the actual predicted labels, the model predicts typewriter keyboard and computer keyboard in 787 and 208 samples which also are the closest labels to space bar in the language model.

This is also true for the four other labels in the table 6.2, for example a dragonfly looks very similar to a damselfly, and 1157 times the model predicted damselfly as top-1 when showing an image of a dragonfly which will naturally give a precise top-5 with dragonfly being very close in the language model. Another example of this is when making prediction for the dog breed entlebucher. Here the model predicts appenzeller in most cases, which is visually a very similar dog breed to entlebucher, and even humans might struggle to spot the difference.

The table also list sunglasses with sunglass as most predicted. However, sunglass is not listed as the closest label to sunglasses, but when switching the roles and listing the closest labels to sunglass the word sunglasses did come up as closest label. Meaning, when sunglass is closest word to predicted vector, sunglasses will appear as top-1. This indicates that even if word A have word B as closest neighbor, word B does not necessary list word A as closest neighbor. Meaning small changes in a word vector might have a huge impact on the predicted word.

Looking from the other end of the scale, table 6.3 have randomly picked five unseen labels that have reported zero in accuracy. In the first example peacock is placed close together with other birds in the language model, but when making predictions the model fails to predict a word that has peacock as top-5 closest. This can be as simple as even if the birds closest to peacock are semantically similar, they may not be visually similar enough for the model.

In another example, both parking meter and a odometer have similar purpose and meanings. However, having never seen a parking meter before, the model fails to recognize this for every sample in the test set. For 727 images of a parking meter, the model predicts that it is seeing an abaya, a cloak worn by some women in the Muslim community. An abaya and a parking meter does not have remotely similar meanings to humans,

**Table 6.2:** The five unseen labels with highest flat-hit score on top-5 prediction, with the vector space containing both unseen and seen labels. Top-1 actual predicted column shows what was predicted as top-1 and the number of times in parentheses.

Unseen label	Score	Closest seen labels in Language model	Top-1 Actual predicted (# predicted)
Space bar	95.69	<ol style="list-style-type: none"> <li>1. Computer keyboard</li> <li>2. Joystick</li> <li>3. Typewriter keyboard</li> <li>4. Menu</li> <li>5. CD player</li> </ol>	<ol style="list-style-type: none"> <li>1. Typewriter keyboard (787)</li> <li>2. Computer keyboard (208)</li> <li>3. Desktop computer (10)</li> <li>4. Screen (6)</li> <li>5. Monitor (2)</li> </ol>
Dragonfly	93.53	<ol style="list-style-type: none"> <li>1. Damselfly</li> <li>2. Lacewing</li> <li>3. Hummingbird</li> <li>4. Ladybug</li> <li>5. Amphibian</li> </ol>	<ol style="list-style-type: none"> <li>1. Damselfly (1157)</li> <li>2. Fly (61)</li> <li>3. Ant (12)</li> <li>4. Lacewing (8)</li> <li>5. Harvestman (6)</li> </ol>
Entlebucher	89.84	<ol style="list-style-type: none"> <li>1. Appenzeller</li> <li>2. Affenpinscher</li> <li>3. Miniature pinscher</li> <li>4. Malinois</li> <li>5. Weimaraner</li> </ol>	<ol style="list-style-type: none"> <li>1. Appenzeller (1147)</li> <li>2. Bluetick (38)</li> <li>3. Rottweiler (34)</li> <li>4. Microphone (14)</li> <li>5. Beagle (7)</li> </ol>
Assault rifle	85.91	<ol style="list-style-type: none"> <li>1. Rifle</li> <li>2. Revolver</li> <li>3. Holster</li> <li>4. Half track</li> <li>5. Barrel</li> </ol>	<ol style="list-style-type: none"> <li>1. Rifle (1091)</li> <li>2. Military uniform (97)</li> <li>3. Revolver (26)</li> <li>4. Holster (9)</li> <li>5. Scabbard (4)</li> </ol>
Sunglasses	82.99	<ol style="list-style-type: none"> <li>1. Trench coat</li> <li>2. Cowboy hat</li> <li>3. Wig</li> <li>4. Sweatshirt</li> <li>5. Bow tie</li> </ol>	<ol style="list-style-type: none"> <li>1. Sunglass (1007)</li> <li>2. Seat belt (36)</li> <li>3. Bow tie (14)</li> <li>4. Safety pin (13)</li> <li>5. Trench coat (9)</li> </ol>

**Table 6.3:** Five unseen labels with a flat-hit@5 accuracy of 0.0, with the vector space containing both unseen and seen labels. Top-1 actual predicted column shows what was predicted as top-1 and the number of times in parentheses.

Unseen label	Score	Closest seen labels in Language model	Top-1 Actual predicted (# predicted)
Peacock	0.0	<ol style="list-style-type: none"> <li>1. Magpie</li> <li>2. Toucan</li> <li>3. Ostrich</li> <li>4. Partridge</li> <li>5. Cock</li> </ol>	<ol style="list-style-type: none"> <li>1. Damselfly (216)</li> <li>2. Indigo bunting (155)</li> <li>3. Cock (140)</li> <li>4. Leaf beetle (125)</li> <li>5. Flatworm (84)</li> </ol>
Parking meter	0.0	<ol style="list-style-type: none"> <li>1. Odometer</li> <li>2. Stopwatch</li> <li>3. Tow truck</li> <li>4. Vending machine</li> <li>5. Cash machine</li> </ol>	<ol style="list-style-type: none"> <li>1. Abaya (727)</li> <li>2. Safety pin (59)</li> <li>3. Ashcan (53)</li> <li>4. Binoculars (33)</li> <li>5. Sweatshirt (29)</li> </ol>
Coffeepot	0.0	<ol style="list-style-type: none"> <li>1. Ladle</li> <li>2. Spatula</li> <li>3. Stove</li> <li>4. Strainer</li> <li>5. Tub</li> </ol>	<ol style="list-style-type: none"> <li>1. Water jug (495)</li> <li>2. Espresso maker (424)</li> <li>3. Stove (62)</li> <li>4. Pitcher (50)</li> <li>5. Cocktail shaker (27)</li> </ol>
Waffle iron	0.0	<ol style="list-style-type: none"> <li>1. Dough</li> <li>2. Pretzel</li> <li>3. Rotisserie</li> <li>4. Wok</li> <li>5. Spatula</li> </ol>	<ol style="list-style-type: none"> <li>1. Chest (202)</li> <li>2. Spatula (117)</li> <li>3. Safety pin (83)</li> <li>4. Dutch oven (45)</li> <li>5. CD player (42)</li> </ol>
Horse cart	0.0	<ol style="list-style-type: none"> <li>1. Minibus</li> <li>2. Minivan</li> <li>3. Limousine</li> <li>4. Jeep</li> <li>5. Garbage truck</li> </ol>	<ol style="list-style-type: none"> <li>1. Jinrikisha (360)</li> <li>2. Plow (238)</li> <li>3. Barrel (144)</li> <li>4. Tractor (100)</li> <li>5. Shovel (47)</li> </ol>



and this shows how the model can be way off when it comes to some predictions.

While the model sometimes make predictions that are way off, it can also make wrong predictions that still can be considered good guesses. For example, with the coffeepot images the model has predicted water jug and espresso maker often as the top-1 predicted. Both of these have similar purpose as a coffeepot, and are also visually similar. However, the language model does not have coffee pot in the top-5 closest word for either of these. The same happens with the horse cart images, where the model predicts jinrikisha as top-1 in 360 samples. A Jinrikisha and a horse cart looks similar, and have also similar purposes where one is dragged by a human and the other by horses. One might suggest that this indicates a poorly trained language model that does not have the vector representation of these two words closer together.

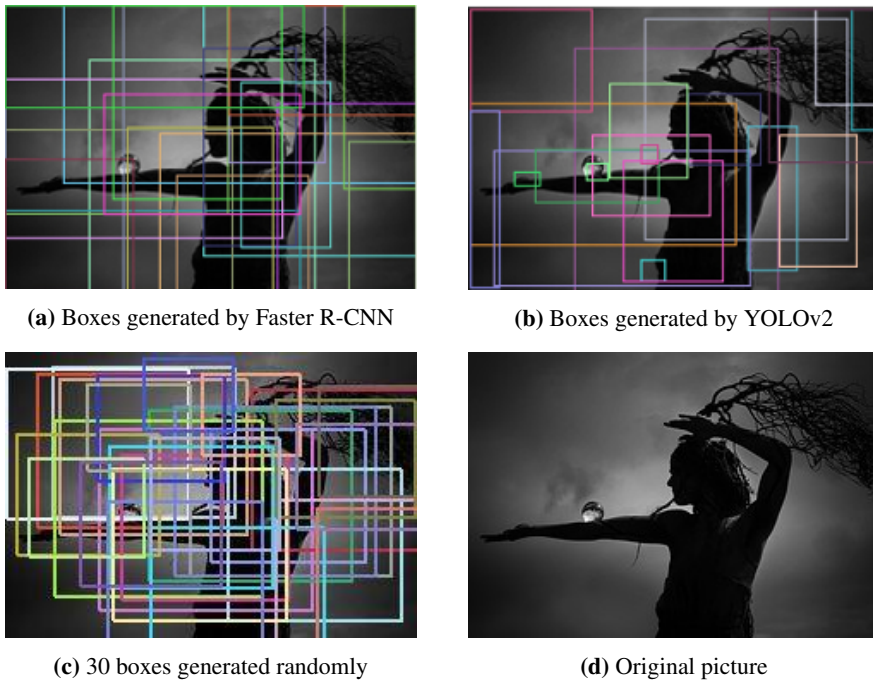
Table 6.2 and table 6.3 indicates together that being lucky with the unseen and seen split will have a high impact on the results, and that unseen labels relies on the model having trained on labels that are visually similar and also close together in the language model space.

The architecture performs similar results to the alternatives in table 5.6. Transforming an image into a vector that fits the language model proves to be possible and giving sensible results. However, it may be hard to tell if these results can be further improved using the same architecture proposed here or if the results always will be limited on the limitations mentioned in this section. With many aspects and networks merged together, the architecture might always be as weak as its weakest model or network.

## 6.2 Multi-label scenarios

We argued that training the model on a single-label experiment first would enable us to train on multi-label data by generating boxes for each image to connect these to labels. The expectation was that using a ODF to generate these boxes would outperform the randomly generated boxes, however the results shows that the randomly generated boxes for prediction clearly outperforms the ODFs. During training the models trained on ODF generated boxes was significantly better than the random generated. Using random generation during training actually made the model perform worse than using the model trained on single-label directly.

In an attempt to answer why the random generated boxes performed so well during predictions we have in figure 6.6 gathered the generated boxes for random, YOLOv2 and Faster R-CNN for one image in the NUS-WIDE dataset. In this figure, YOLOv2 has a few very small boxes that tries to detect small objects, but also larger boxes trying to detect larger objects. Comparing Faster R-CNN and the randomly generated boxes it is hard to tell them apart as they both seem to have large boxes that together cover the full



**Figure (6.6):** A comparison of YOLOv2, Faster R-CNN and the randomly generated boxes for an image in the NUS-WIDE dataset.

image. By looking closely at these two one might notice that the randomly generated boxes looks more squared than the Faster R-CNN boxes. Still, even if these are slightly different this should not be an answer in itself to the big difference in the accuracy.

We also noticed that some of the boxes generated by Faster R-CNN covers a large part of the image while the random boxes overlap more, and are also more evenly distributed over the entire image. This might be one of the reasons random bounding boxes performed better during prediction. From figure 6.6 we notice that YOLOv2 and Faster R-CNN fails to make boxes that successfully cover the objects in the image. A well-trained object detection framework should be able to recognize and detect objects, and especially the person.

Even if we can find some differences between the approaches, this is not enough to conclude why the accuracy is so different. A more thorough investigation is needed to fully answer this, and even if the examples in figure 6.6 may indicate a different pattern between the approaches, it is impossible to conclude the behavior based on a single image, especially considering there are 78,530 images in the the test set and 156,585 images in the training set.

The ODFs was also tested without using *Non-Maximum Supression* (NMS) (de-

scribed in section 2.5.1) to see if keeping more boxes would increase the accuracy. When evaluating these results we got slightly worse results as when using NMS, and concluded that this could not give an answer to why random boxes predicted better than ODFs.

While the random boxes performed well during prediction, the model fails to train on them. One of the reasons might be that only the boxes that fit best to a label are used which may fit better for the ODF boxes where a few might match the labels. This is, however, guesses and as with the prediction scenario, it is hard to find an answer to these behaviors without more investigation.





Our solution is based on the principle that each image contained objects connected to a region of the image where each of these were labeled, but in the NUS-WIDE data we noticed that some of the labels are not object focused. For example, an image could be labeled city, London and England which are all three different labels in NUS-WIDE, but not necessarily three different objects. Having an image of London could return all those three labels, but they are not connected to certain regions of the image, but rather the context of the full image in itself. This is demonstrated in table 6.4, where we have collected some random images and their predictions for both unseen and seen+unseen spaces using the model trained on Faster R-CNN and tested with random boxes.

The building in figure 6.9 is labeled with historic, historical and history. These labels are not connected to three different regions of the image, but may refer to the building having a historical value. This is three different labels with similar meanings and having the model predicting one of them correctly may indicate a well-trained model, but will not give full score as two are missing. One way of getting all the three tags could be to have co-occurrence statistics also when making predictions, this way the model could spot that if it predicted one of them, the two others often follows. This co-occurrence statistics could be calculated using the labels from the training set, which might improve the model and have added it for future work (section 7.2).

Animal is listed as the only ground truth label in figure 6.7 and the model receives full score when predicting only unseen, with animal listed as the first prediction. When making prediction for unseen+seen the model has a score of zero, although some of the predictions are good guesses. For example, the model predicts animals, rocks, wildlife and goat, but none of these are listed as ground truth.

With animals in the predicted list, and animal in the ground truth this also shows another challenge when validating the predictions. Both plural and singular forms of the same words are listed as two different labels, and when predicting one form with the other being ground-truth, it will be counted as a wrong prediction. This is seen many places in table 6.4, with words such as cloud/clouds, building/buildings, tree/trees and animal/animals. This may or may not be considered a limitation, depending how strict we want to be when validating. One may want to keep the difference between the plural

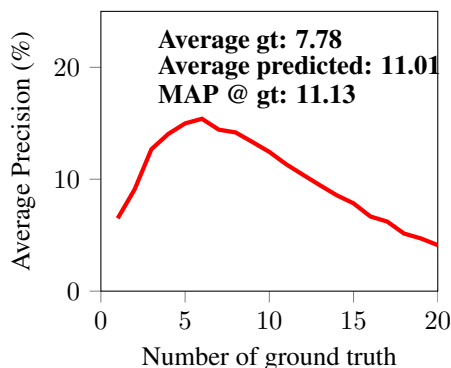
**Table 6.4:** Multi-label predictions for five randomly selected images from the NUS-WIDE test set, predicted using the model with highest MAP@*gt* accuracy in table 5.8 (trained on Faster R-CNN, predicted with random boxes). The correctly annotated labels have a checkmark ✓ beside it. The seen labels are marked in blue and unseen in red. Ground truth refers to the actual label-tags in the NUS-WIDE data, labeled by humans. When predicting on the unseen space, only the red unseen ground truth labels are valid.

Images	Seen + unseen	Only unseen	Ground truth
 <p>Figure (6.7)</p>	horses, snow, dog, rocks, deer, nature, europe, animals, wildlife, africa, mexico, desert, eyes, rabbit, goat, landscape	animal✓, elk, reflection, dog, cow, mountain, tree, horses, bear, snow	animal, stone, iran, interestingness, closeup, river, singing
 <p>Figure (6.8)</p>	clouds, sky✓, blue, boat, baseball, scenery, oil, winter, landscape, germany	clouds, sky✓, soccer✓, surf, reflection, sunset✓	rainbow, soccer, sky, sunset, football, explore, stadium, argentina
 <p>Figure (6.9)</p>	buildings✓, clouds✓, tree, germany, architecture✓, europe, palace, paris, hawaii	flags, buildings✓, cityscape✓, tower, mountain, train	cityscape, clouds, buildings, sky, building, brown, storm, hiking, art, sky, scenery, illinois, historic, architecture, historical, history, march, brick, walk, public, structure, red
 <p>Figure (6.10)</p>	mountain✓, clouds, sky, tree, lake✓, blue, volcano, animals, winter, nature, trees, underwater, landscape	mountain✓, sky, clouds, snow, tree, lake✓	mountain, glacier, fox, lake, sunset, pink, cloud

and singular, and reward the model only when it predicts the correct form, which makes our measurements correct.

In an attempt to answer how close the misses are to the ground truth, missed average cosine similarity is calculated in table 5.8. These results displays how the models with higher MAP accuracy also have higher average cosine similarity score, which indicates how close the missed predictions are to the ground truth. The models reports a lower average cosine for the missed labels when only including the unseen labels compared to including seen also, meaning a more precise vector is needed to make correct prediction for unseen and seen. This is, as mentioned in section 6.1, due to noise when including more words. The predicted words can be very similar to the actual word, but with all the noise it fails to have the correct word in the top-k predicted. This could also mean that when the model predicts on only the unseen labels it makes larger mistakes and therefore also naturally get a lower average cosine similarity.

From table 6.4 we can also see that the number of ground truth labels can variate a lot, with figure 6.9 having a total of 22 labels and 7 labels in figure 6.10 when including both unseen and seen. We calculated that the average number of labels connected to each image was at 7.78 and the average number of predicted labels was 11.01. This shows that the model predicts more labels than ground truth, but with the added voting the model will return the most occurred labels first.



**Figure (6.11):** Demonstrates how the average precision variate depending on the number of ground truth labels (gt), for space including both seen+unseen. Using the model trained on Faster R-CNN with predictions on random boxes.

Using the model trained on Faster R-CNN and predicted with random boxes, figure 6.11 shows how ground truth and average precision correlates. The average precision seems to have highest score when images have around five ground truth labels, and from there it gets lower and lower as the image have more labels connected to it. An example of this can be seen in figure 6.9 where the model fails to predict most of the 22 ground

truth labels.

Overall, with the approach chosen to solve the multi-label scenario, we realize that the NUS-WIDE data might not have been the best dataset to train or test on. Our architecture may have higher accuracy for ZSL if trained and tested on a data where the labels are more object focused. On data with object oriented labels, each image will have the same amount of labels as objects and could help our model make the connection between label and image-box and increase the training and prediction accuracy.

Our model performed worse than the comparative alternatives, and as we followed the same unseen and seen split as Zhang et al. we expected similar results. This shows the limitations of our architecture, however with further experimenting the accuracy might improve, bringing us closer to their result.

## 6.3 Limitations

In this section we will go through the limitations of our thesis, why we have such limitations and also what that could potentially decrease our limitations.

- Looking at the datasets NUS-WIDE and Imagenet, we only introduced 81 and 200 unseen labels. To enable our model to be used in a real-life situation it needs to also be tested on a larger number of unseen labels, and our results show that our architecture is not prune to noise, and adding more words in the language model decreases the accuracy. If including all the words from the language models, meaning all words from Wikipedia, the accuracy would likely drop a lot or approach zero. One suggestion to solve this is to train the model on a larger set of object categories and test it on an even larger set of unseen classes.
- The image labels in NUS-WIDE are tags from Flickr, which means it can be quite noisy with both bad labeled data and labels. Therefore performing ZSL on another dataset is needed to further investigate our architecture.
- Our architecture relies on the labels for the images can be connected to a certain object in these images. Some words, such as countries and history, can be hard to connect to a certain object and thus suffer from this. This could be fixed training the model on different data where each of the labels are object focused.
- The language model uses text from the English Wikipedia, and is relates words in a textual relationship. Meaning, words that are semantically similar, are placed close together. However, since our problem focuses on visual objects, some objects that are visually similar are not necessary semantically similar. A suggestion for improving this could be to train these models on image descriptions.

- With the architecture being complex, and with a large amount of training and testing data. The architecture has not been tested with many different parameters. For example, only one optimizer was tested and the batch size was kept constant at 32. In future work, we suggest to test the model on a variety of parameters.
- Same as with the parameters, our projection network have a fixes number of layers and hidden units. Different number of layers and hidden units might give different score, and investigating the architecture to find the most optimal network would also help with the results.
- In our experiments we have trained Word2vec using the CBOW algorithm. However, Word2Vec also contains another algorithm, Skip-Gram, that in some cases have reported better score than the CBOW, and thus have we included Skip-Gram for future work.





# Chapter 7

## Conclusion

Through developing a model capable of predicting unseen objects for single- and multi-label images, we have learned that these models still have a long way to go before being able to recognize new objects the same way human does, especially if expecting the model to predict correctly on the first guess. We have also learned that sometimes it can be hard to validate such a model as the model might make a good guess, but if the guess does not correspond to the ground truth label, the label made by humans, it will be counted as a miss.

With the goal to investigate *Zero-Shot Learning* techniques in multi-label scenarios, we presented in section 1.2 three research questions that we investigated. In terms of these, the results shows:

**Research Question 1:** *How can the knowledge from text be used to help an image-classifier predict unseen classes?*

To investigate this research question, we trained language models on two different algorithms and in different dimensions. These models were trained using the English Wikipedia as text-corpus and the results indicate that knowledge from text can be used to solve a ZSL problem. However, some words seems to be easier to predict than others, which can indicate that the knowledge from text can only be used to a certain extent and that it will be limited on the text corpus the language models are trained on.

In our experiments, increasing the language model dimensions improved the accuracy, and it would be interesting to investigate with even higher dimensions to see if the accuracy continues to improve or if it flattens.

**Research Question 2:** *How does our Zero-Shot learning model scale when increas-*

*ing the number of available classes?*

The ZSL model suffered when increasing the number of words possible for the model to predict on. This suggest that when increasing the number of words available to the model, it generates noise for the model which makes it harder to predict correctly.

**Research Question 3:** *How can Zero-Shot learning be applied to multi-label scenarios?*

As the results indicates, making prediction for multi-label scenarios in ZSL can be successful when splitting the image into a set of boxes and using the knowledge from a model pre-trained on single-label ZSL to choose a correct label for the given box. However, this requires that the labels for each image is connected to a specific object on that image. We learned from our experiments that this is not always the case, with labels focusing on the full context and not objects.

When making these boxes, we have learned that the best approach for training a model is not necessarily the best approach for making predictions. Our results showed that randomly generated boxes performed well for making predictions, but when training on with this approach the model suffered a drop in the accuracy.

Overall the answers to the research questions have positive indications, but still needs further research in more scenarios and on different data before giving a concluding answer. This thesis only gives indicative answers and is not able to tell if the results can be further improved using the same architecture or approach, or if the accuracy flattens.

## 7.1 Contributions

The following bullets describe the main contributions of our thesis:

- We combined two ZSL architectures and can now predict both multi-label and single-label. We updated the CNN to the state-of-the-art network inception-v3. To generate bounding boxes for the multi-label experiments we used three different approaches: randomly placed bounding boxes, and the two different ODFs: YOLOv2 and Faster-RCNN.
- We performed experiments to choose the best performing language model and to choose the best performing dimension. We also ran one experiment to choose between four different loss functions.
- We believe that in the future ZSL systems can be used in for example search technology or automatic tag generation on social media. This could help a user

to search in a visual manner instead of by image name and the system does not need to see all tags it suggests. Multi-label ZSL is quite new, and is not ready to be put into production yet, but to push the system even further we have made some suggestions of future work. We have made our code available at [https://github.com/thomasSve/Msc\\_Multi\\_label\\_ZeroShot](https://github.com/thomasSve/Msc_Multi_label_ZeroShot).

- We performed tests on the models with language spaces containing different number of candidate labels, with one space containing only the unseen labels and the other including the seen with the unseen.

## 7.2 Future Work

As work with the system has progressed and the results have been analyzed, ideas for future work have appeared. All the experiments took a long time to complete, and with limited time we had to stop experimenting to finish writing the thesis. These ideas are presented in the following bullets:

- Higher dimensional language models reported higher accuracy, and it would be interesting to train a language model with more than 1000 dimensions to see if the accuracy continued to increase.
- Train the Word2vec model using the Skip-Gram algorithm instead of CBOW as Skip-Gram has been the chosen algorithm among the two in related papers, and it could improve the language model further.
- Test the model on a different multi-label dataset than NUS-WIDE that focuses more on object categories rather than tags. The NUS-WIDE labels is tags extracted from Flickr and therefore it has quite noisy labels. A possible multi-label dataset to try is open images [54] provided by Google.
- Experiment with different parameters such as different batch sizes and optimizers. As our neural network architecture was large and took a large amount of RAM on the GPU memory, we used a batch size of 32 which is standard in Keras. Further experimenting with higher batch sizes such as 64 or even higher might give better results. Choosing the right optimizer is also important, so running experiments to determine the optimizer for the network might further increase the results.
- Experiment with different numbers of layers for the projection network and also the dimension of the neural networks. For this thesis, the projection network used the same neural network architecture except the last layer that had different dimensions depending on the language model dimension.

- Try changing the loss function to one similar as used by Zhang et al. [32] since it reported better results. The loss function used by Zhang et al. accepted positive and negative feedback, which means that the network can be trained with both correct labels and wrong labels. This might yield better vectors and makes sense to try to push the accuracy of the model even further.
- In the multi-label scenario, include co-occurrence statistics for test set. This could help the model combine the predicted labels with co-occurrence statistics to include words that often occur together. For example cup and table might often occur together in the image labels, but not the language model, and this could potentially help the model. One thing to investigate in the future would be to train the language model on a different text corpus than the English Wikipedia. The language model could be trained on a collection of image descriptions or labels. Possibly train it first on the full English Wikipedia, before tuning it on the image descriptions set or image labels. This could possibly help the training to get classes that occur often together closer, even if this is not the case in Wikipedia. This way we hope to generate language models that puts words for objects that are visually similar closer together, instead of relying on the textual similarity.
- Flip the model around and change the projection network to project the word vector to the image features instead of image features to word vector. This would mean that the input of the projection network would be the word vector label, that was now projected to the dimension native to the image features. It would be interesting to see if this flipped approach would result in a higher score, as the image features have a higher dimension.
- To get better results it would be interesting to use an ensemble of a number of CNN architectures to extract features. In our architecture we used the inception-v3 with weights pre-trained on ImageNet [40]. Since the release of this architecture, newer architectures have been released that report having a higher accuracy. Combining multiple models into an ensemble with both different CNN architectures, but also different projection layers would most likely yield a higher accuracy. This would require a lot of computation time and was therefore not possible during this thesis.

# Bibliography

- [1] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” in *Advances in neural information processing systems*, 2015, pp. 91–99.
- [2] J. Redmon and A. Farhadi, “Yolo9000: Better, faster, stronger,” *arXiv preprint arXiv:1612.08242*, 2016.
- [3] I. Biederman, “Recognition-by-components: a theory of human image understanding,” *Psychological review*, vol. 94, no. 2, p. 115, 1987.
- [4] B. Yao, A. Khosla, and L. Fei-Fei, “Combining randomization and discrimination for fine-grained image categorization,” in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE, 2011, pp. 1577–1584.
- [5] L. Fei-Fei, R. Fergus, and P. Perona, “One-shot learning of object categories,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 4, pp. 594–611, 2006.
- [6] A. Frome, G. S. Corrado, J. Shlens, S. Bengio, J. Dean, T. Mikolov *et al.*, “Devise: A deep visual-semantic embedding model,” in *Advances in neural information processing systems*, 2013, pp. 2121–2129.
- [7] X. Li, S. Liao, W. Lan, X. Du, and G. Yang, “Zero-shot image tagging by hierarchical semantic embedding,” 2015.
- [8] M. Palatucci, D. Pomerleau, G. E. Hinton, and T. M. Mitchell, “Zero-shot learning with semantic output codes,” in *Advances in neural information processing systems*, 2009, pp. 1410–1418.
- [9] M. Norouzi, T. Mikolov, S. Bengio, Y. Singer, J. Shlens, A. Frome, G. Corrado, and J. Dean, “Zero-shot learning by convex combination of semantic embeddings,” *CoRR*, vol. abs/1312.5650, 2013. [Online]. Available: <http://arxiv.org/abs/1312.5650>

- 
- [10] S. C. Shapiro, “Artificial intelligence,” 1982. [Online]. Available: <https://www.cse.buffalo.edu/~shapiro/Papers/ai-eofcs.pdf>
- [11] T. B. M. V. Association and S. for Pattern Recognition, “What is computer vision?” accessed: 2017-02-27. [Online]. Available: <http://www.bmva.org/visionoverview>
- [12] S. J. Pan and Q. Yang, “A survey on transfer learning,” *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.
- [13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [14] E. G. Miller, N. E. Matsakis, and P. A. Viola, “Learning from one example through shared densities on transforms,” in *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, vol. 1. IEEE, 2000, pp. 464–471.
- [15] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, pp. 115–133, 1943. [Online]. Available: <http://dx.doi.org/10.1007/BF02478259>
- [16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, pp. 533–536, 1986.
- [17] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006, ch. 4.2: Probabilistic Generative Models.
- [18] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, “Object recognition with gradient-based learning,” in *Shape, contour and grouping in computer vision*. Springer, 1999, pp. 319–345.
- [19] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” *CoRR*, vol. abs/1311.2901, 2013. [Online]. Available: <http://arxiv.org/abs/1311.2901>
- [20] M. Díez Buil, “Non-maxima suppression,” 2011. [Online]. Available: <http://hdl.handle.net/2454/4626>
- [21] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” *CoRR*, vol. abs/1310.4546, 2013. [Online]. Available: <http://arxiv.org/abs/1310.4546>
- [22] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *CoRR*, vol. abs/1301.3781, 2013. [Online]. Available: <http://arxiv.org/abs/1301.3781>

- 
- [23] Y. Goldberg and O. Levy, “word2vec explained: Deriving mikolov et al.’s negative-sampling word-embedding method,” *arXiv preprint arXiv:1402.3722*, 2014.
- [24] J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation.” in *EMNLP*, vol. 14, 2014, pp. 1532–43.
- [25] X. Li, Y. Guo, and D. Schuurmans, “Semi-supervised zero-shot classification with label representation learning,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 4211–4219.
- [26] Z. Akata, F. Perronnin, Z. Harchaoui, and C. Schmid, “Label-embedding for attribute-based classification,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 819–826. [Online]. Available: <http://dx.doi.org/10.1109/CVPR.2013.111>
- [27] Z. Akata, S. Reed, D. Walter, H. Lee, and B. Schiele, “Evaluation of output embeddings for fine-grained image classification,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2927–2936. [Online]. Available: <http://dx.doi.org/10.1109/CVPR.2015.7298911>
- [28] M. Bucher, S. Herbin, and F. Jurie, “Improving semantic embedding consistency by metric learning for zero-shot classification,” in *European Conference on Computer Vision*. Springer, 2016, pp. 730–746.
- [29] C. H. Lampert, H. Nickisch, and S. Harmeling, “Learning to detect unseen object classes by between-class attribute transfer,” in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 951–958. [Online]. Available: <http://dx.doi.org/10.1109/CVPR.2009.5206594>
- [30] S. Changpinyo, W.-L. Chao, B. Gong, and F. Sha, “Synthesized classifiers for zero-shot learning,” *arXiv preprint arXiv:1603.00550*, 2016. [Online]. Available: <http://arxiv.org/abs/1603.00550>
- [31] Y. Fu and L. Sigal, “Semi-supervised vocabulary-informed learning,” *CoRR*, vol. abs/1604.07093, 2016. [Online]. Available: <http://arxiv.org/abs/1604.07093>
- [32] Y. Zhang, B. Gong, and M. Shah, “Fast zero-shot image tagging,” in *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*. IEEE, 2016, pp. 5985–5994.
- [33] Y. Fu, Y. Yang, T. M. Hospedales, T. Xiang, and S. Gong, “Transductive multi-label zero-shot learning,” *CoRR*, vol. abs/1503.07790, 2015. [Online]. Available: <http://arxiv.org/abs/1503.07790>
-

- 
- [34] Z. Ren, H. Jin, Z. L. Lin, C. Fang, and A. L. Yuille, “Multi-instance visual-semantic embedding,” *CoRR*, vol. abs/1512.06963, 2015. [Online]. Available: <http://arxiv.org/abs/1512.06963>
- [35] R. Girshick, “Fast r-cnn,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1440–1448.
- [36] G. A. Miller, “Wordnet: a lexical database for english,” *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.
- [37] M. Baroni, G. Dinu, and G. Kruszewski, “Don’t count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors.” in *ACL (1)*, 2014, pp. 238–247.
- [38] R. Socher, M. Ganjoo, C. D. Manning, and A. Ng, “Zero-shot learning through cross-modal transfer,” in *Advances in neural information processing systems*, 2013, pp. 935–943.
- [39] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*, 2012. [Online]. Available: <http://www.cs.toronto.edu/~hinton/absps/imagenet.pdf>
- [40] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.
- [41] C. Szegedy, S. Ioffe, and V. Vanhoucke, “Inception-v4, inception-resnet and the impact of residual connections on learning,” *CoRR*, vol. abs/1602.07261, 2016. [Online]. Available: <http://arxiv.org/abs/1602.07261>
- [42] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” *CoRR*, vol. abs/1409.4842, 2014. [Online]. Available: <http://arxiv.org/abs/1409.4842>
- [43] F. Chollet, “Keras,” 2015. [Online]. Available: <https://github.com/fchollet/keras>
- [44] J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation,” in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: <http://www.aclweb.org/anthology/D14-1162>



- 
- [45] R. Řehůřek and P. Sojka, “Software Framework for Topic Modelling with Large Corpora,” in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, May 2010, pp. 45–50, <http://is.muni.cz/publication/884893/en>.
- [46] R. Mudge, “Generating a plain text corpus from wikipedia.” [Online]. Available: <https://blog.afterthedeathline.com/2009/12/04/generating-a-plain-text-corpus-from-wikipedia/>
- [47] Spotify, “Annoy (approximate nearest neighbors oh yeah).” [Online]. Available: <https://github.com/spotify/annoy>
- [48] L. v. d. Maaten and G. Hinton, “Visualizing data using t-sne,” *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.
- [49] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from [tensorflow.org](http://tensorflow.org/). [Online]. Available: <http://tensorflow.org/>
- [50] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng, “Nus-wide: a real-world web image database from national university of singapore,” in *Proceedings of the ACM international conference on image and video retrieval*. ACM, 2009, p. 48.
- [51] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [52] T. Mensink, J. Verbeek, F. Perronnin, and G. Csurka, “Metric learning for large scale image classification: Generalizing to new classes at near-zero cost,” *Computer Vision—ECCV 2012*, pp. 488–501, 2012.
- [53] L. Zhang, T. Xiang, and S. Gong, “Learning a deep embedding model for zero-shot learning,” *arXiv preprint arXiv:1611.05088*, 2016.
- [54] I. Krasin, T. Duerig, N. Alldrin, A. Veit, S. Abu-El-Haija, S. Belongie, D. Cai, Z. Feng, V. Ferrari, V. Gomes, A. Gupta, D. Narayanan, C. Sun, G. Chechik, and K. Murphy, “Openimages: A public dataset for large-scale multi-label and multi-class image classification.” *Dataset available from <https://github.com/openimages>*, 2016.