Doctoral Thesis

# Zero-Shot Recognition of Generic Objects

Tristan HASCOET

Graduate School of System Informatics

Kobe University

A thesis submitted for PhD. degree

June 2019

# Doctoral Thesis

## Zero-Shot Recognition of Generic Objects

Tristan HASCOET

# Contents

# List of Figures

# List of Tables

# Declaration

I herewith declare that I have produced this paper without the prohibited assistance of third parties and without making use of aids other than those specified; notions taken over directly or indirectly from other sources have been identified as such. This paper has not previously been presented in identical or similar form to any other german or foreign examination board.

The related contents in this thesis have been previously published or submitted for publication by the author. A complete list of publications can be found on *pp.* xi-xii.

# Publication List

## Journal Papers

1. Tristan Hascoet, Yasuo Ariki, and Tetsuya Takiguchi: "Semantic embeddings of generic objects for zero-shot learning", *EURASIP Journal on Image and Video Processing*, 2019.1 (2019): 13.

2. Tristan Hascoet, Quentin Febvre, Yasuo Ariki, and Tetsuya Takiguchi: "Reversible designs for extreme memory cost reduction of CNN training", *EURASIP Journal on Image and Video Processing*, Under review.

## International Conference Papers

1. Tristan Hascoet, Yasuo Ariki, and Tetsuya Takiguchi: "Semantic web and zero-shot learning of large scale visual classes", *Proceedings of the First WOrkshop on Symbolic Neural Learning, July 2017, Nagoya*

2. Tristan Hascoet, Yasuo Ariki, and Tetsuya Takiguchi: "Investigation of the correlations between CNN visual features and word embeddings", *Second Workshop on Closing the Loop Between Vision and Language, International Conference on Computer Vision, Deceber 2017, Venice.*

3. Tristan Hascoet, Yasuo Ariki, and Tetsuya Takiguchi: "Zero-shot learning using dictionary definitions", *International Workshop on Frontiers of Computer Vision. March 2018, Seoul.*

4. Tristan Hascoet, Yasuo Ariki, and Tetsuya Takiguchi: “On zero-shot recognition of generic objects”, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. June 2019, Long Beach*, XXX.

## Domestic Conference Papers

1. Tristan Hascoet, Yasuo Ariki, and Tetsuya Takiguchi: “Visually grounded word embeddings for zero-shot learning of visual categories”, *Computer Vision and Image Media, March 2018, Kyoto.*

2. Tristan Hascoet, Yasuo Ariki, and Tetsuya Takiguchi: “Knowledge graph embeddings for Zero-Shot Learning”, *National Symposium on Image Processing and Understanding, August 2018, Sapporo.*

# Glossary

| | |
|---|---|
| **ZSL** | Zero-Shot Learning |
| **CNN** | Convolutional Neural Network |
| **NLP** | Natural Language Processing |
| **GCN** | Graph Convolution Network |

# Chapter 1

# Introduction

## 1.1 Background

## 1.2 Approaches

## 1.3 Purpose of This Thesis

### 1.3.1 Novelties of This Thesis

## 1.4 Outline

# Chapter 2

# Redefining Generic object ZSL

The related publications for this chapter are [].

## 2.1 Introduction

Datasets play a leading role in computer vision research. Perhaps the most striking example of the impact a dataset can have on research has been the introduction of Imagenet [3]. The new scale and granularity of Imagenet's coverage of the visual world has paved the way for the success and wide spread adoption of CNN [4, 5] that have revolutionized generic object recognition.

The current best-practice for the development of a practical object recognition solution consists in collecting and annotating application-specific training data to fine-tune a large Imagenet-pretrained CNN on. This data annotation process can be prohibitively expensive for many applications which hinders the widespread usage of these technologies. ZSL models generalize the recognition ability of traditional image classifiers to unknown classes, for which no image sample is available for training. The promise of ZSL for generic object recognition is huge: to scale up the recognition capacity of image classifiers beyond the set of annotated training classes. Hence ZSL has the potential to be of great practical impact as they would considerably ease the deployment of object recognition technologies by eliminating the need for expensive task-specific data collection and fine-tuning processes.

Despite its great promise, and after a decade of active research [6], the accuracy of ZSL models on the standard Imagenet benchmark [7] remain far too low for practical applications. To better understand this lack of progress, we analyzed the errors of several ZSL baselines. Our analysis leads us to identify two main factors impacting the accuracy of ZSL models: structural flaws in the standard evaluation protocol and poor quality of both semantic and visual samples. On the bright side of things, we show that once these flaws are taken into account, the actual accuracy of existing ZSL models is much higher than was previously thought.

On the other hand, we show that a trivial solution outperforms most existing ZSL models by a large margin, which is upsetting. To explain this phenomenon, we introduce the notion of structural bias in ZSL datasets. We argue that ZSL models should aim to develop compositional reasoning abilities, but the presence of structural bias in the Imagenet benchmark favors solutions based on a trivial one to one mapping between training and test classes. We come to the conclusion that a new benchmark is needed to address the different problems identified by our analysis and, in the last section of this paper, we detail the semi-automated construction of a new benchmark we propose.

To structure our discussion, we first briefly review preliminaries on ZSL in Section 3. Section 4 details our analysis of the different factors impacting the accuracy of ZSL models on the standard benchmark. Section 5 introduces the notions of structural bias, and propose a way to measure and minimize its impact in the construction of a new benchmark. Finally, Section 6 summarizes the construction of our proposed benchmark. For space constraint, we only include the main results of our analysis in the body of this paper. We refer interested readers to the supplementary material for additional results and details of our analysis.

## 2.2 Related Work

### 2.2.1 ZSL datasets

Early research on ZSL has been carried out on relatively small scale or domain specific benchmarks [8, 9, 10], for which human-annotated visual attributes are proposed as semantic representations of the visual classes. On the one hand, these benchmarks have provided a controlled setup for the development of theoretical models and the accurate tracking of ZSL progress. On the other hand, it is unclear whether approaches developed on such dataset would generalize to the more practical setting of zero-shot generic object recognition. For instance, in generic object recognition, manually annotating each and every possible visual class of interest with a set of visual attributes is impractical due to the diversity and complexity of the visual world.

The Imagenet dataset [3] consists of more than 13 million images scattered among 21,845 visual classes. Imagenet relies on Wordnet [11] to structure its classes: each visual class in Imagenet corresponds to a concept in Wordnet. Frome *et al.*[7] proposed a benchmark for ZS generic object recognition based on the Imagenet dataset, which has been widely adopted as the standard evaluation benchmark by recent works [1, 2, 12, 13, 14, 15, 16]. Using word embeddings as semantic representations, they use the 1000 classes of the ILSVRC dataset as training classes and propose different test splits drawn from the remaining 20,845 classes of the Imagenet dataset based on their distance to the training classes within the Wordnet hierarchy: the *2-hops*, *3-hops* and *all* test splits.

Careful inspection of these test splits revealed a confusion in their name: The *2-hops* test split actually consists of the set of 1589 test classes directly connected to the training set classes in Wordnet, i.e; within *1 hop* of the training set. Similarly, the *3-hops* test set actually corresponds to the test classes within *2-hops*. In this paper, we will refer to the standard test splits by the name of their true configuration: *1-hop*, *2-hops* and *all*, as illustrated in Figure 1.

### 2.2.2 Dataset bias

Bias in datasets can take many forms, depending on the specific target task. Torralba *et al.*[17] investigates bias in generic object recognition. The notion of structural bias we introduce in Section 5 is closely related to the notion of negative set bias they analyze.

As more complex tasks are being considered, more insidious forms of bias sneak into our datasets. In VQA, the impressive results of early baseline models have later been shown to be largely due to statistical biases in the question/answers pairs [18, 19, 20]. Similar to these works, we will show that a trivial solution leveraging structural bias in the Imagenet ZSL benchmark outperforms early ZSL baselines.

Xian *et al.*[1] identify structural incoherences in small-scale ZSL benchmarks and proposes new test splits to remedy them. Closely related to our work, they also observe a correlation between test class sample population and classification accuracy in the Imagenet ZSL benchmark. However, their analysis mainly focuses on small-scale benchmarks and the comparison of existing ZSL models, while we analyze the ZSL benchmark for generic object recognition in more depth.

## 2.3 Preliminaries

ZSL models aim to recognize unseen classes, for which no image sample is available to learn from. To do so, ZSL models use descriptions of the visual classes, i.e., representations of the visual classes in a semantic space shared by both training and test classes. To evaluate the out-of-sample recognition ability of models, ZSL benchmarks split the full set of classes $C$ into disjoint training and test sets. ZSL benchmarks are fully defined by three components: a set of training and test classes $(C_{tr}, C_{te})$, a set of labeled images $X$, and a set of semantic representations $Y$:

$$C_{tr} \cup C_{te} \subset C \tag{2.1a}$$

$$C_{tr} \cap C_{te} = \emptyset \tag{2.1b}$$

$$Y = \{y_c \in \mathbb{R}^d \quad \forall c \in C\} \tag{2.1c}$$

$$X = \{(x, c) \in \mathbb{R}^{3\times h\times w} \times C\} \tag{2.1d}$$

$$Tr = \{(x, y_c) \mid c \in C_{tr}\} \tag{2.1e}$$

$$Te = \{(x, y_c) \mid c \in C_{te}\} \tag{2.1f}$$

ZSL models are typically trained to minimize a loss function $\mathcal{L}$ over a similarity score $E$ between image and semantic features of the training sample set with respect to the model parameters $\theta$.

$$\theta^* = argmin_\theta \mathbb{E}_{(x,y)\in Tr} \mathcal{L}(E_\theta(x,y) + \Omega(\theta)) \tag{2.2}$$

In the standard ZSL setting, test samples $x_{te}$ are classified among the set of unseen test classes by retrieving the class description $y$ of highest similarity score:

$$c = argmax_{c\in C_{te}} E(x_{te}, y_c) \tag{2.3}$$

In the generalized ZSL setting, test samples are classified among the full set of training and test classes:

$$c = argmax_{c\in C} E(x_{te}, y_c) \tag{2.4}$$

Xian *et al.*[13] have shown that many ZSL models can be formulated within a same linear model framework, with different training objectives and regularization terms. More recently, Wang *et al.*[16] have proposed a Graph Convolutional Network (GCN) model that has shown impressive improvements over the previous state of the art. In our study, we will present results obtained with both a baseline linear model [14] and a state of the art GCN model [2, 16].

## 2.4 Error analysis

In the previous section, we have mentioned that ZSL benchmarks are fully defined by three components: a set of labeled images $X$, a set of semantic representations $Y$, and the set of training and test classes $(C_{tr}, C_{te})$. In this section, we analyze each of the standard benchmark components individually: We first highlight inconsistencies in the configuration of the different test splits and show that these inconsistencies lead to many false negatives in the reported evaluation of ZSL models outputs. Next, we identify a number of factors impacting the quality of the word embeddings of visual classes and argue that visual classes with poor semantic representations should be excluded from ZSL benchmarks. We then observe that the Imagenet dataset contains many ambiguous image samples. We define what a *good* image sample means in the context of ZSL and propose a method to automatically select such images.

### 2.4.1 Structural flaws

Figure 1 illustrates the configuration of test classes of the standard test splits within the Wordnet hierarchy. This configuration leads to an obvious contradiction: test sets include visual classes of both parents and their children concepts. Consider the problem of classifying images of birds within the *hop-1* test split as in Figure 1. The standard test splits give rise to two possibly inconsistent scenarios:

**Figure 2.1:** Illustration of the standard test splits configuration

A ZSL model may classify an image of the children class *Cathartid* as its parent class *Raptor*. The standard benchmark considers such cases as classification errors, while the classification is semantically correct.

A ZSL model may classify an image of the parent class *Raptor* as one of its children class: *Cathartid*. Classification may be semantically correct or incorrect, depending on the specific breed of raptor in the image, but we have no way to automatically assess it without additional annotation. The standard benchmark considers such cases as classification errors, while the classification is semantically undefined.

We refer to both of the above cases as false negatives. Figure 2 illustrates the distribution of ZSL classification outputs among these different scenarios on the 1-hop test split. On the standard ZSL task for instance, the reported accuracy of the GCN model is 21.8% while the actual (semantically correct) accuracy should be somewhere in between 27.8% and 40.4%.

The ratio of false negatives per accuracy increases dramatically in the generalized ZSL setting. The linear baseline reported accuracy is only 1.9%, while

**Figure 2.2:** Distribution of the classification outputs of different ZSL models on the *1-hop* test split. An image $x$ can be either be classified into its actual label $c$, the parent class of $c$, one of its children class, or an unrelated class. Only the latter case constitutes a definitive error.

the actual (semantically correct) accuracy lies between 16.0% and 41.1%. This is due to the fact that ZSL models tend to classify test images into their parent or children training class: for example, *Cathartid* images tend to be classified as *Vulture*. Appendix A of the supplementary material presents results on the other standard splits on which we show that the ratio of false negative per reported accuracy further increases with with larger test splits.

### 2.4.2 Word embeddings

In this section, we identify two factors impacting the quality of word embeddings and analyse their affect on ZSL accuracy: polysemy and occurrence frequency. These problems naturally arise in the definition of large scale object categories so they are inherent problems of ZS recognition of generic objects. However, we argue that ZSL benchmarks should provide a curated environment with high quality, unambiguous, semantic representations and that solutions to tackle the special case of polysemous and rare words should be separately investigated in the future.

#### 2.4.2.1 Occurrence frequency

Word embeddings are learned in an unsupervised manner from the co-occurrence statistics of words in large text corpora. Common words are learned from plentiful statistics so we expect them to provide more semantically meaningful representations than rare words, which are learned from scarce co-occurrence statistics. We found many Imagenet class labels to be rare words (see Appendix B of the supplementary materials), with as many as 33.7% of label words appearing less than 50 times in Wikipedia. Here, we question whether the few co-occurrence statistics from which such rare word embeddings are learned actually provide any visually discriminative information for ZSL.

To answer this question, we evaluate ZSL models on different test splits of 100 classes: we split the Imagenet classes into different subsets based on the occurrence frequency of their label word. We independently evaluate the accuracy of our model on each of these splits and report the ZSL accuracy with respect to the average occurrence frequency of the visual class labels in Figure 3.

**Figure 2.3:** Each dot in these figures represent the top-1 accuracy (y-axis) of a 100 classes test split with respect to the test split characteristics (x-axis): Left: Mean occurrence frequency of the test class labels. Right: test classes of primary meaning, such as cairn (monument), or secondary meaning, such as cairn (dog)

Our results highlight a strong correlation ($r = 0.89$) between word frequency and the Linear baseline accuracy as test splits made of rare words strikingly underperform test splits made of more common words, although accuracy remains well above chance (1%), even for test sets of very rare words. Results are more nuanced for the GCN model (correlation coefficient $r = 0.74$), which can be explained by the fact that GCN uses the Wordnet hierarchy information in addition to word embeddings.

#### 2.4.2.2 Polysemy

The English language contains many polysemous words, which makes it difficult to uniquely identify a visual class with a single word. We found that half of the ImageNet word labels are shared with at least one other Wordnet concept, and that 38% of ImageNet classes share at least one word label with other visual classes. Figure 4 illustrates the example of the word "cairn". Two visual classes share the same label "cairn": One relates to the meaning of cairn as a stone memorial, while the other refers to a dog breed. This is problematic as both of these visual classes share the same representation in the label space, so they are essentially defined as the same class although they correspond to two visually very distinct concepts.

To deal with polysemy, we assume that all words have one primary meaning, with possibly several secondary meanings. We consider word embeddings to reflect the semantics of their primary meaning exclusively, and discard visual classes associated with the secondary meanings of their word label. To automatically identify the first meaning of visual class labels. we implement a solution based on both Wordnet and word embeddings statistics detailed in the supplementary material.

**Figure 2.4:** Illustration of polysemous words. Each color represents the 100 nearest neighbors of a given word. "Cairn" and its closest neighbors are clustered around the stone and monument related vocabulary, far away from dog-related vocabulary so we assign the top visual class as primary meaning of the word cairn.

We conduct an experiment to assess both the impact of polysemy on ZSL accuracy and the efficiency of our solution. As in the previous section, we evaluate our ZSL models on different test splits of 100 classes: We separately evaluate test classes identified as the primary meaning of their word label and test classes corresponding to the secondary meaning of their word label. Figure 3 reports the accuracy obtained on these different test splits. We can see a significant boost in the ZSL accuracy of test classes whose word labels are identified as primary meanings. In comparison, test splits made exclusively of secondary meanings performed poorly. This confirms that polysemy does indeed impact ZSL accuracy, and suggests that our solution for primary meaning identification allows addressing this problem.

### 2.4.3 Image samples

The ILSVRC dataset consists of a high-quality curated subset of the Imagenet dataset. The current ZSL benchmark uses ILSVRC classes as training classes and classes drawn from the remainder of the Imagenet dataset as test sets, assuming similar standards of quality from these test classes. Upon closer inspection, we found these test classes to contain many inconsistencies and ambiguities. In this section, we detail a solution to automatically filter out ambiguous samples so as to only select quality samples for our proposed benchmark.

#### 2.4.3.1 Class-wise selection

Xian *et al.*[13] have first identified a correlation between the sample population of visual classes and their classification accuracy. They conjecture that small population classes are harder to classify because they correspond to fine-grained visual concepts, while large population classes correspond to easier, coarse-grained concepts. Manual inspection of these classes lead us to a different interpretation: First, we found no significant correlation between sample population and concept granularity (Appendix C). For example, fine-grained concepts such as specific species of birds or dogs tend to have high sample populations. On the other hand, we found many visually ambiguous concepts such as ”ringer”, ”covering” or ”chair of state” to have low sample populations. Such visually ambiguous concepts are harder for crowd-sourced annotators to reach consensus on labeling, resulting in lower population counts.

In Figure 5, we report the ZSL accuracy of our models on different test splits with respect to their average population counts. This figure shows a clear correlation between the sample population and the accuracy of both models, with low accuracy for low sample population classes. We use the sample population as a rough indicator to quickly filter out ambiguous visual classes and only consider classes with sample population superior to 300 images as valid candidate classes in our proposed dataset.

**Figure 2.5:** ZSL accuracy with respect to sample population sizes. Left: Distribution of Imagenet class population size. 6.1% of Imagenet classes have less than 10 samples, 21.1% have less than 100 samples. Right: ZSL accuracy of different test splits with respect to their mean sample population size.

#### 2.4.3.2 Sample-wise selection

Even among the selected classes, we found many inconsistent and ambiguous images to remain (Appendix C), so we would like to further filter quality test images sample-wise. But what makes a good candidate image for a ZSL benchmark? How can we measure the quality of a sample? We argue that ZSL benchmarks should only reflect the *zero-shot ability* of models: ZSL benchmarks should evaluate the accuracy of ZSL models *relatively to the accuracy of standard non-ZSL models.* Hence, we define a good ZSL sample as an image unambiguous enough to be correctly classified by standard image classifiers trained in a supervised manner.

To automatically filter such quality samples, we fine-tune and evaluate a standard CNN in a supervised manner on the set of candidate test classes. We consider consistently miss-classified samples to be too ambiguous for ZSL and only select samples that were correctly classified by the CNN Details of this selection process are presented in Appendix C of the supplementary material.

### 2.4.4 Dataset Summary

Figure 6 summarizes the impact of the different factors we analyzed on the top-1 classification error of both our baseline models on the "1-hop" test split. The error rate of the Linear model on the standard ZSL setting drops from 86% to 61% after removing ambiguous images, semantic samples, and structural flaws. The error rate of the GCN model on the generalized setting drops from 90% to 47%.

**Figure 2.6:** Estimation of the impact of different factors on the reported error of existing models on the 1-hop test split

The GCN model is particularly sensitive to the structural flaws of the standard benchmark, but less sensitive to noisy word embeddings than the linear baseline. This can be easily explained by the fact that GCN models rely on the explicit Wordnet hierarchy information as semantic data in addition to word embeddings. Additional results and details on the methodology of our analysis are given in Appendix D of the supplementary material.

## 2.5 Structural bias

ZSL models are inspired by the human ability to recognize unknown objects from a mere description, as it is often illustrated by the following example: Without having ever seen a zebra, a person would be able to recognize one, knowing that zebras look like horses covered in black and white stripes. This example illustrates the human capacity to *compose* visual features of *different* known objects to define and recognize previously unknown object categories.

Standard image classifiers encode class labels as *local representations* (one-hot embeddings), in which each dimension represents a different visual class, as illustrated in Figure 8. As such, no information is shared among classes in the label space: visual class embeddings are equally distant and orthogonal to each other. The main idea behind ZSL models is to instead embed visual classes into *distributed representations*: In label space, visual classes are defined by multiple visual features (horse-ish shape, stripes, colors) shared among classes. Distributed representations allow to define and recognize unknown classes by *composition* of visual features shared with known classes, in a similar manner as the human ability described above.

The embedding of visual classes into distributed feature representations is especially powerful since it allows to define a combinatorial number of test classes by composition of a possibly small set of features learned from a given set of training classes. Hence, we argue that the key challenge behind ZSL is to achieve ZS recognition of unknown classes by composition of known visual features, following their original inspiration of the human ability, and as made possible by distributed feature representations. In this section, we will see that not all ZSL problems require such kind of compositional ability. On the standard benchmark, we show that a trivial solution based on local representations of visual classes outperform existing approaches based on word embeddings. We show that this trivial solution is made possible by the specific configuration of the standard test splits and introduce the notion of structural bias to refer to the existence of such trivial solutions in ZSL datasets.

### 2.5.1 Toy example

Figure 7 illustrates a toy ZSL problem in which, given a training set of *Horse* and *TV monitor* images, the goal is to classify images of *Zebra* and *PC laptop*. Let's consider training an image classifier on the training set and directly applying it to images from the test set. We can safely assume that most zebra images will be classified as horses, and most laptop samples as TV monitors. Hence, a trivial solution to this problem consists in defining a one to one mapping between test classes and their closest training class: *Horse=Zebra* and *TV monitor=PC laptop*. This example makes it fairly obvious that not all ZSL problems require the ability to compose visual features to solve.

**Figure 2.7:** Illustration of the toy example. Left: Wordnet-like class hierarchy. Training classes are shown in red and test class in green. Right: Illustration of image samples. The black captions represent the distance between classes as their shortest path length.

Classification problems define a close-world assumption: As all test samples are known to belong to one of the test classes, classifying an image $x$ into a given test class $c$ means that $x$ is more likely to belong to $c$ than other classes of the test set. In other words, classification is performed relatively to a negative set of classes [17]. What made this trivial ZSL solution possible is the fact that test classes of our toy example are very similar to one of the training class, relatively to their negative set. This allowed us to identify a one-to-one mapping by similarity between training and test classes. We refer to this trivial solution as a similarity-based solution, in opposition to solutions based on the composition of visual features.

As illustrated in Figure 8, the similarity mapping between test and training classes can be directly embedded in the semantic space using local representations. The trivial solution consists in assigning to test classes the exact same semantic representation as their most similar training class. Consider applying these semantic embeddings within a ZSL framework to our toy problem: classifying a test image $x$ as a Horse relatively to the negative set of TV within the training set becomes strictly equivalent to classifying $x$ as Zebra relatively to its

**Figure 2.8:** Illustration of local (one-hot, on the left) and distributed (right) representations of visual classes. The similarity-based solution encodes both training and test classes as local representations. Composition-based solutions need distributed representations.

negative set PC within the test set. Hence, any existing ZSL model using these local embeddings instead of distributed representations like word embeddings $Y$ would converge to the same solution.

### 2.5.2 Standard benchmark

Besides our toy example, how well would this trivial solution perform on the standard benchmark? To implement it, we used the Linear baseline model [14] with local representations inferred from the Wordnet hierarchy (see Appendix E), but any model would essentially converge to a similar solution. Table 1 compares the accuracy of this trivial solution to state of the art models as reported in [1, 2]. The trivial similarity-based solution outperforms existing ZSL models by a significant margin. Only GCN-based models [2], which we discuss in the next section, seem to outperform our trivial solution.

### 2.5.3 Measuring structural bias

In our toy example, we have hinted at the fact that structural bias emerges for test sets in which test classes are relatively similar to training classes, while being

**Table 2.1:** Top-1 accuracy on the standard test splits (top) as reported for linear baselines in [1], (middle) as reported for GCN-based models in [2] and (down) obtained by our trivial solution

| model | 1-hop | 2-hops | all |
|---|---|---|---|
| SYNC [15] | 9.26 | 2.29 | 0.96 |
| CONSE [12] | 7.63 | 2.18 | 0.95 |
| ESZSL [14] | 6.35 | 1.51 | 0.62 |
| LATEM [13] | 5.45 | 1.32 | 0.5 |
| DEVISE[7] | 5.25 | 1.29 | 0.49 |
| CMT [21] | 2.88 | 0.67 | 0.29 |
| GCNZ [16] | 19.8 | 4.1 | 1.8 |
| ADGPM [2] | **26.6** | **6.3** | **3.0** |
| Trivial | 20.27 | 3.59 | 1.53 |

comparably more dissimilar to each other (to their negative set). To confirm this intuition, we define the following structural ratio:

$$r(c) = \frac{min_{c' \in C_{tr}} d(c, c')}{min_{c' \in C_{te}} d(c, c')} \tag{2.5a}$$

$$R(C_{te}) = \frac{1}{|C_{te}|} \sum_{c \in C_{te}} r(c) \tag{2.5b}$$

In which $c$ represents a visual class, $C_{te}$ and $C_{tr}$ represent test and training sets respectively, and $d$ is a distance reflecting similarity between two classes. Here, $r(c)$ represents the ratio of the distance between $c$ and its closest training class to the distance between $c$ and its closest test class. In our experiments, we use the the shortest path length between two classes in the Wordnet hierarchy as a measure of distance $d$, although different metrics would be interesting to investigate as well. We compute the structural ratio of a test set $R(C_{te})$ as the mean structural ratio of its individual classes. Figure 9 shows the top-1 accuracy achieved by baseline models on different test sets with respect to their structural ratio $R$. As for previous experiments, we report our results on test splits of 100 classes.

**Figure 2.9:** ZSL accuracy on different test sets with respect to their structural ratio $R(C_{te})$.

On test splits of low structural ratio, the trivial solution performs remarkably well, on par with the state of the art GCN model. Such test splits are similar to the toy example in which each test class is closely related to a training class while being far away from other test classes in the Wordnet hierarchy. As an example, the structural ratio of the test split in our toy example is $R(C_{te}) = 1/2 \times (2/4 + 2/4) = 0.5$, which corresponds to the highest accuracies achieved by the trivial solution. We say that such test split is structurally biased towards similarity-based trivial solutions.

However, the accuracy of the similarity-based trivial solution decreases sharply with the structural ratio until it reaches near chance accuracy for the highest ratios. Hence maximizing the structural ratio of test splits seems to be an efficient way to minimize structural bias. Although their accuracy decrease with larger structural ratios, both GCN and Linear models remain well above chance. These results suggest that ZSL models based on word embeddings are indeed capable of compositional reasoning. At the very least, they are able to perform more complex ZSL tasks than the trivial similarity-based solution. Interestingly, as the trivial solution converges towards chance accuracy, the GCN model accuracy seems to converge towards the accuracy of the ZSL baseline. This suggests that the main reason behind the success of GCN models is that they efficiently leverage the Wordnet hierarchy to exploit structural bias.

The *1-hop* and *2-hops* test splits of the standard benchmark consist of the set of test classes closest to the training classes within the Wordnet hierarchy. This leads to test splits of very low structural ratio, similar to our toy example. For instance, the *1-hop* test split has a structural ratio of 0.55. It is an example of structural bias even more extreme than our toy example as test classes are either children or parent classes of a training class. In the next section, we propose a new benchmark with maximal structural ratio in order to minimize structural bias.

## 2.6 New Benchmark

### 2.6.1 Proposed Benchmark

In this section, we briefly detail the semi-automated construction of a new benchmark designed to fix the different flaws of the current benchmark highlighted by our analysis. For space constraints, a number of minor considerations could not be properly presented in this paper. We detail these additional considerations in Appendix F of the supplementary material. Appendix F also provides additional details regarding the different parameters and the level of automation of each of the construction process. Appendix G provides details on the code and data we release. Following Frome *et al.*[7], we use the ILSVRC dataset as training set, and propose a new test set. The selection of this new test set proceeds in two steps:

In a first step, we select a subset of candidate test classes $C' \subset C$ from the remaining 20,845 Imagenet classes based on the statistics of image samples and word labels: We first filter out semantic samples $Y' \subset Y$ corresponding to rare or polysemous words of secondary meaning (Section 4.2). We then discard visual classes of low sample population and filter out ambiguous image samples using supervised learning to select $X' \subset X$ (Section 4.3). The set of candidate test classes is the subset of visual classes $C' \subset C$ for which sufficiently high quality image and semantic samples were selected.

In a second step, we define the test split $C_{te} \subset C'$ as a *structurally consistent* set of *minimal structural bias*: The test set was carefully selected so as to contain no overlap among its own classes nor with the training classes in order to provide a structurally consistent test set for the generalized ZSL setting. This test set consists of 500 classes of maximal structural ratio $R(C_{te})$ so as to minimize structural bias.

### 2.6.2 Evaluation

**Table 2.2:** Evaluation on the proposed benchmark. Accuracy in the generalized ZSL setting are reported as harmonic means over training and test accuracy following [1]

| Model | ZSL | | G-ZSL | |
|---|---|---|---|---|
| | @1 | @5 | @1 | @5 |
| Trivial | 1.2 | 3.9 | 0 | 0 |
| CONSE [12] | 10.65 | 25.10 | 0.12 | 19.34 |
| DEVISE [7] | 11.15 | 29.52 | **7.87** | 26.10 |
| ESZSL [14] | 13.54 | 32.61 | 4.59 | 25.53 |
| GCN-6 [16] | 9.58 | 27.19 | 4.81 | 23.35 |
| GCN-2 [2] | 14.09 | 35.12 | 4.96 | **30.35** |
| ADGPM [2] | **14.10** | **36.03** | 4.90 | 29.96 |

Table 2 presents the evaluation of a number of baseline models on the newly proposed benchmarks. A few notable results stand out from this table: First, different from the standard benchmark, CONSE [12] performs worse than DEVISE [7]. The relatively high accuracy reported by the CONSE model on the standard benchmark is most likely due to the fact that word embeddings of test classes are statistically close to the word embedding of their parent/children test classes so that CONSE results more closely fit the trivial similarity-based trivial solution. We expect model averaging methods to benefit the most from the structural bias in the standard benchmark.

Second, the impressive improvements reported by GCN-based models over linear baselines are significantly reduced, although GCN models still outperform linear baselines. This result corroborates the observation, in Section 5, that GCN models tend to converge towards the results of linear baseline models for high structural ratio.

## 2.7 Conclusion and Discussion

ZSL has the potential to be of great practical impact for object recognition. However, as for any computer vision task, the availability of a high quality benchmark is a prerequisite for progress. In this paper, we have shown major flaws in the standard generic object ZSL benchmark and proposed a new benchmark to address these flaws. More importantly, we introduced the notion of structural bias in ZSL dataset that allows trivial solutions based on simple similarity matching in semantic space. We encourage researchers to evaluate their past and future models on our proposed benchmark. It seems likely that sound ideas may have been discarded for their poor performance relative to baseline models that benefited most from structural bias. Some of these ideas may be worth revisiting today.

Finally, we believe that a deeper discussion on the goals and the definition of ZSL is still very much needed. There is a risk in developing complex models to address poorly characterized problems: Mathematical complexity can act as a smokescreen of complexity that obfuscates the real problems and key challenges behind ZSL. Instead, we believe that practical considerations grounded in common sense are still very much needed at this stage of ZSL research. The identification of structural bias is a first step towards a sound characterization of ZSL problems. One practical way to continue this discussion would be to investigate structural bias in other ZSL benchmarks.

# Chapter 3

# Visual Feature Extraction

The related publications for this chapter are [].

## 3.1 Introduction

Over the last few years, Convolutional Neural Networks (CNN) have enabled unprecedented progress on a wide array of computer vision tasks. One disadvantage of these approaches is their resource consumption: Training deep models within a reasonable amount of time requires special Graphical Processing Units (GPU) with numerous cores and large memory capacity. Given the practical importance of these models, a lot of research effort has been directed towards algorithmic and hardware innovations to improve their resource efficiency such as low-precision arithmetic [22], network pruning for inference [23], or efficient stochastic optimization algorithms [24].

In this paper, we focus on a particular aspect of resource efficiency: optimizing the memory cost of training CNNs. We envision several potential benefits from the ability to train large neural networks within limited memory:

**Democratization of Deep Learning research:** Training large CNN requires special GPUs with large memory capacity. Typical desktop GPUs memory capacity is too small for training large CNNs. As a result, getting into deep learning research comes with the barrier cost of either buying specialized hardware or renting live instances from cloud service providers. Reducing the memory cost of deep model training would allow training deep nets on standard graphic cards without the need for specialized hardware, effectively removing this barrier cost. In this paper, we demonstrate efficient training of a CNN on the CIFAR10 dataset (93.3% accuracy within 67 minutes) on an Nvidia GTX750 with only 1GB of memory.

**On-device training:** With mobile applications, a lot of attention has been given to optimize inference on edge devices with limited computation resources. Training state-of-the-art CNN on embedded devices, however, has still received little attention. Efficient on-device training is a challenging task for the underlying power efficiency, computation and memory optimization challenges it involves. As such, CNN training has thus far been relegated to large cloud servers, and trained CNNs are typically deployed to embedded device fleets over the network. On-device training would allow bypassing these server-client interactions over the network. We can think of several potential applications of on-device training, including:

- Life-long learning: Autonomous systems deployed in evolving environments like drones, robots or sensor networks might benefit from continuous life-long learning to adapt to their changing environment. On-device training would enable such application without the expensive communication burden of having edge devices continuously sending their data to remote servers over the network. It would also provide resilience to network failures in critical application scenarios.

- In privacy-critical applications such as biometric mobile phone authentication, users might not want to have their data sent over the network. On-device training would allow fine-tuning recognition models on local data without sending sensitive data over the network.

In this work, we propose an architecture with minimal training memory cost requirements which enables training within the tight memory constraints of embedded devices.

**Research in optimization:** Recent works on stochastic optimization algorithms have highlighted the benefits of large batch training [25, 26]. For example, in Imagenet, linear speed-ups in training have been observed with increasing batch sizes up to tens of thousands of samples [26]. Optimizing the memory cost of CNN training may allow further research on the optimization trade-offs of large batch training. For small datasets like MNIST or CIFAR10, we are able to process the full dataset in 14 and 18 GB of memory respectively. Although large batch training on such small dataset is very computationally inefficient with current stochastic optimization algorithms [26], the ability to process the full dataset in one pass allows to easily train CNNs on the true gradient of the error. Memory optimization techniques have the potential to facilitate research on optimization techniques outside the realm of Stochastic Gradient Descent to be investigated.

In this paper, we build on recent works on reversible networks [27, 28] and ask the question: how far can we reduce CNN training memory cost using reversible designs with minimal impact on the accuracy and computational cost? To do so, we take as a starting point the Resnet-18 architecture and analyze its training memory requirements. We then analyze the memory cost reduction of invertible designs successively introduced in the RevNet and iRevNet architectures. We identify the memory bottleneck of such architectures, which leads us to introduce a layer-wise invertible architecture. However, we observe that layer-wise invertible networks accumulate numerical errors across their layers, which leads to numerical instabilities impacting model accuracy. We characterize the accumulation of numerical errors within long chains of revertible operations and investigate their effect on model accuracy. To mitigate the impact of these numerical errors on the model accuracy, we propose both a reparameterization of invertible layers and a hybrid architecture combining the benefits of layer-wise and residual-block-wise reversibility to stabilize training.

Our main result is to present a new architecture that allows to efficiently train a CNN with the minimal memory cost of 352 bytes per pixel. We demonstrate the efficiency of our method by efficiently training a model to 93.3% accuracy on the CIFAR10 dataset within 67 minutes on a low-end Nvidia GTX750 with only 1GB of VRAM.

## 3.2 Related Work

### 3.2.1 Reversibility

Reversible network designs have been proposed for various purposes including generative modeling, visualization, solving inverse problems, or theoretical analysis of hidden representations.

Flow-based generative models use analytically invertible transformations to compute the change of variable formula. Invertibility is either achieved through channel partitioning schemes (NICE [29] Real-NVP [30]), weight matrix factorization (GLOW [31]) or constraining layer architectures to easily invertible unitary operations (Normalization flows [32])

Neural ODEs [33] take a drastically different take on invertibility: They leverage the analogy between residual networks and the Euler method to define continuous hidden state systems. The conceptual shift from a finite set of discrete transformations to a continuous regime gives them invertibility for free. The computational efficiency of this approach, however, remains to be demonstrated.

The RevNet model [27] was inspired by the Real-NVP generative model. They adapt the idea of channel partitioning and propose an efficient architecture for discriminative learning. The iRevNet [28] model builds on the RevNet architecture: they propose to replace the irreversible max-pooling operation with an invertible operation that reshapes the hidden activation states so as to compensate the loss of spatial resolution by an increase in the channel dimension. By preserving the volume of activations, their pooling operation allows for exact reconstruction of the inverse. In their original work, the authors focus on the analysis of the representations learned by invertible models rather than resource efficiency. From a resource optimization point of view, one downside of their method is that the proposed invertible pooling scheme drastically increases the number of channels in upper layers. As the size of the convolution kernel weights grows quadratically in the number of channels, the memory cost associated with storing the model weights becomes a major memory bottleneck. We address this issue in our proposed architecture. In [34], the authors use these reversible architectures to study undesirable invariances in feature space.

In [35], the authors propose a unified architecture performing well on both generative and discriminative tasks. They enforce invertibility by regularizing the weights of residual blocks so as to guarantee the existence of an inverse operation. However, the computation of the inverse operation is performed with power iteration methods which are not optimal from a computational perspective.

Finally, [36] propose to reconstruct the input activations of normalization and activation layers using their inverse function during the backward pass. We propose a similar method for layer-wise invertible networks. However, as their model does not invert convolution layers, it does not feature long chains of invertible operations so that they do not need to account for numerical instabilities. Instead, our proposed model features long chains of invertible operations so that we need to characterize numerical errors in order to stabilize training.

### 3.2.2 Resource efficiency

Research into resource optimization of CNNs covers a wide array of techniques, most of which are orthogonal to our work. We briefly present some of these works:

On the architectural side, Squeezenet [37] was first proposed as an efficient neural architecture reducing the number of model parameters while maintaining high classification accuracy. MobileNet [38] uses depth-wise separable convolutions to further reduce the computational cost of inference for embedded device applications.

Network pruning [23] is a set of techniques developed to decrease the model weight size and computational complexity. Network pruning works by removing the network weights that contribute the least to the model output. Pruning deep models has been shown to drastically reduce the memory cost and computational cost of inference without significantly hurting model accuracy. Although pruning has been concerned with optimization of the resource inference, the recently proposed lottery ticket hypothesis [39] has shown that specifically pruned networks could be trained from scratch to high accuracy. This may be an interesting and complementary line of work to investigate in the future to reduce training memory costs.

Low precision arithmetic has been proposed as a mean to reduce both memory consumption and computation time of deep learning models. Mixed precision training [40] combines float16 with float32 operations to avoid numerical instabilities due to either overflow or underflow. For inference, integer quantization [22, 41] has been shown to drastically improve the computation and memory efficiency and has been successfully deployed on both edge devices and data centers. Integrating mixed-precision training to our proposed architecture would allow us to further reduce training memory costs.

Most related to our work, gradient checkpointing was introduced as a mean to reduce the memory cost of deep neural network training. Gradient checkpointing, first introduced in [42], trades off memory for computational complexity by storing only a subset of the activations during the forward pass. During the backward pass, missing activations are recomputed from the stored activations as needed by the backpropagation algorithm. Follow-up work [43] has since built on the original gradient checkpointing algorithm to improve this memory/computation trade-off. However, reversible models like RevNet have been shown to offer better computational complexity than gradient checkpointing, at the cost of constraining the model architecture to invertible residual blocks.

## 3.3 Preliminaries

In this section, we analyze the memory footprint of training architectures with different reversibility patterns. We start by introducing some notations and briefly review the backpropagation algorithm in order to characterize the training memory consumption of deep neural networks. In our analysis, we use a Resnet-18 as a reference baseline and analyze its training memory footprint. We then gradually augment the baseline architecture with reversible designs and analyze their impact on computation and memory consumption.

### 3.3.1 Backpropagation & Notations

Let us consider a model $F$ made of $N$ sequential layers trained to minimize the error $e$ defined by a loss function $\mathcal{L}$ for an input $x$ and ground-truth label $\bar{y}$:

$$F : x \rightarrow y \tag{3.1a}$$

$$F : x \rightarrow f_N \circ ... \circ f_2 \circ f_1(x) \tag{3.1b}$$

$$e = \mathcal{L}(y, \bar{y}) \tag{3.1c}$$

During the forward pass, each layer $f_i$ takes as input the activations $z_{i-1}$ from the previous layer and outputs activation features $z_i = f_i(z_{i-1})$, with $z_0 = x$ and $z_N = y$ being the input and output of the network respectively.

During the backward pass, the gradient of the loss with respect to the hidden activations are propagated backward through the layers of the networks using the chain rule as:

$$\frac{\delta \mathcal{L}}{\delta z_{i-1}} = \frac{\delta \mathcal{L}}{\delta z_i} \times \frac{\delta z_i}{\delta z_{i-1}} \tag{3.2}$$

Before propagating the loss gradient with respect to its input to the previous layer, each parameterized layer computes the gradient of the loss with respect to its parameters. In vanilla SGD, for a given learning rate $\eta$, the weight gradients are subsequently used to update the weight values as:

$$\frac{\delta \mathcal{L}}{\delta \theta_i} = \frac{\delta \mathcal{L}}{\delta z_i} \times \frac{\delta z_i}{\theta_i} \tag{3.3a}$$

$$\theta_i \leftarrow \theta_i - \eta \times \frac{\delta \mathcal{L}}{\delta \theta_i} \tag{3.3b}$$

However, the analytical form of the weight gradients are functions of the layer's input activations $z_{i-1}$. In convolution layers, for instance, the weight gradients can be computed as the convolution of the input activation by the output's gradient:

$$\frac{\delta \mathcal{L}}{\delta \theta_i} = z_{i-1} \star \frac{\delta \mathcal{L}}{\delta z_i} \tag{3.4}$$

Hence, computing the derivative of the loss with respect to each layer's parameters $\theta_i$ requires knowledge of the input activation values $z_{i-1}$. In the standard backpropagation algorithm, hidden layers activations are stored in memory upon computation during the forward pass. Activations accumulate in live memory buffers until used for the weight gradients computation in the backward pass. Once the weight gradients computed in the backward pass, the hidden activation buffers can be freed from live memory. However, the accumulation of activation values stored within each parameterized layer along the forward pass creates a major bottleneck in GPU memory.

The idea behind reversible designs is to constrain the network architecture to feature invertible transformations. Doing so, activations $z_i$ in lower layers can be recomputed through inverse operations from the activations $z_{j>i}$ of higher layers. In such architectures, activation do not need to be kept in memory during the forward pass as they can be recomputed from higher layer activations during the backward pass, effectively freeing up the GPU live memory.

### 3.3.2 Memory footprint

We denote the memory footprint of training a neural network as a value $\mathcal{M}$ in bytes. Given an input $x$ and ground truth label $\bar{y}$, the memory footprint represents the peak memory consumption during an iteration of training including the forward and backward pass. We divide the total training memory footprint $\mathcal{M}$ into several memory cost factors: the cost of storing the model weights, the hidden activations, and the gradients:

$$\mathcal{M} = M_\theta + M_z + M_g \tag{3.5}$$

In the following subsections, we detail the memory footprint of existing architectures with different reversibility patterns. To help us formalize these memory costs, we further introduce the following notations: let $n(x)$ denote the number of elements in a tensor $x$, i.e.; if $x$ is an $h \times w$ matrix, then $n(x) = h \times w$. Let $bpe$ be the memory cost in bytes per elements of a given precision so that the actual memory cost for storing an $h \times w$ matrix is $n(x) \times bpe$. For instance, float32 tensors have a memory cost per element $bpe = 4$. We use $bs$ to denote the batch size, and $c_i$ to denote the number of channels at layer $i$.

### 3.3.3 Vanilla ResNet

The architecture of a vanilla ResNet-18 is shown in Figure 1. Vanilla ResNets do not use reversible computations so that the input activations of all parameterized layers need to be accumulated in memory during the forward pass for the computation of the weight gradients to be done in the backward pass.

Hence the peak memory footprint of training a vanilla ResNet happens at the beginning of the backward pass when the top layer's activation gradients need to be stored in memory in addition to the full stack of hidden activation values.

Let us denote by $P \subset N$ the subset of parameterized layers of a network $F$ (i.e.; convolutions and batch normalization layers, excluding activation functions and pooling layers). The memory cost associated with storing the hidden activation values is given by:

$$M_z = \sum_{i \in P} n(z_i) \times bpe \tag{3.6a}$$

$$M_z = \sum_{i \in P} bs \times c_i \times h_i \times w_i \times bpe \tag{3.6b}$$

Where $h_i$ and $w_i$ represent the spatial dimensions of the activation values at layer $i$. $h_i$ and $w_i$ are determined by the input image size $h \times w$ and the pooling factor $p_i$ of layer $i$, so we can factor out both the spatial dimensions and the batch size from this equation, yielding a memory cost per input pixel $M'_z$:

$$M_z = \sum_{i \in P} bs \times h \times w \times p_i \times c_i \times bpe \tag{3.7a}$$

$$M_z = bs \times h \times w \times \sum_{i \in P} p_i \times c_i \times bpe \tag{3.7b}$$

$$M'_z = M_z / (bs \times h \times w) \tag{3.7c}$$

$$M'_z = \sum_{i \in P} p_i \times c_i \times bpe \tag{3.7d}$$

The memory footprint of the weights is given by:

$$M_\theta = \sum_{i \in P} n(\theta_i) \times bpe \tag{3.8}$$

The memory footprint of the gradients correspond to the size of the gradient buffers at the time of peak memory usage. In a vanilla ResNet18 model, this peak memory usage happens during the backward pass through the last convolution of the network. Hence, the memory footprint of the gradients correspond to the memory cost of storing the gradients with respect to either the input or the output of this layer, which also depends on the input pixel size:

$$M_g = max(n(g_{i-1}), n(g_i)) \times bpe \tag{3.9a}$$
$$M_g = h \times w \times bs \times p_i \times max(c_{i-1}, c_i) \tag{3.9b}$$
$$M'_g = p_i \times max(c_{i-1}, c_i) \tag{3.9c}$$

Figure 1 illustrates the peak memory consumption of a ResNet-like architecture. For a ResNet parameterized following Table 1, the peak memory consumption can then be computed as:

$$\mathcal{M} = M_\theta + M_z + M_g \tag{3.10a}$$
$$\mathcal{M} = (M_\theta + (M'_z + M'_g) \times (h \times w \times bs)) \times bpe \tag{3.10b}$$
$$\mathcal{M} = (12.5 * 10^6 + 1928 \times (h \times w \times bs)) \tag{3.10c}$$
$$\tag{3.10d}$$

For example, a training iteration over a typical batch of 32 images of resolution $240 \times 240$ requires 12.5 MB of memory to store the model weights and 3.8 GB of memory to store the hidden layers activations and gradients for a total of $\mathcal{M} = 3.81$ GB of VRAM. The memory cost of the hidden activations is thus the main memory bottleneck of CNN training as the cost associated with the model weights is negligible in comparison.

### 3.3.4 RevNet

The RevNet architecture introduces reversible blocks as drop-in replacements of the residual blocks of the ResNet architecture. Reversible blocks have analytical inverses that allow for the computation of both their input and hidden activation values from the value of their output activations. Two factors create memory bottlenecks in training RevNet architectures, which we refer to as the local and global bottlenecks.

First, the RevNet architecture features non-volume preserving max-pooling layers, for which the inverse cannot be computed. As these layers do not have analytical inverses, their input must be stored in memory during the forward pass for the reconstruction of lower layer's activations to be computed during the backward pass. We refer to the memory cost associated with storing these activations as the global bottleneck, since these activations need to be accumulated during the forward pass through the full architecture.

The local memory bottleneck has to do with the synchronization of the reversible block computations: While activations values are computed by a forward pass through the reversible block modules, gradients computations flow backward through these modules so that the activations and gradient computations cannot be performed simultaneously. Figure 2 illustrates the process of backpropagating through a reversible block: First, the input activation values of the parameterized hidden layers within the reversible blocks are recomputed from the output. Once the full set of activation have been computed and stored in GPU memory, the backpropagation of the gradients through the reversible block can begin. We refer to the accumulation of the hidden activation values within the reversible block as the local memory bottleneck.

For a typical parameterization of a RevNet, as summarized in Table 1, the local bottleneck of lower layers actually outweighs the global memory bottleneck introduced by non-reversible pooling layers. Indeed, as the spatial resolution decreases with pooling operations, the cost associated with storing the input activations of higher layers becomes negligible compared to the cost of storing activation values in lower layers. Hence, surprisingly, the peak memory consumption of the RevNet architecture, as illustrated in Figure 3, happens in the backward pass through the first reversible block, in which the local memory bottleneck is maximum. For the architecture described in Table 1, the peak memory consumption can be computed as:

$$\mathcal{M} = M_\theta + M_z + M_g \tag{3.11a}$$

$$\mathcal{M} = (M_\theta + (M'_z + M'_g) \times (h \times w \times bs)) \times bpe \tag{3.11b}$$

$$\mathcal{M} = (12.7 \times 10^6 + 640 \times (h \times w \times bs)) \tag{3.11c}$$

$$\tag{3.11d}$$

Following our previous example, a RevNet architecture closely mimicking the ResNet-18 architecture requires $\mathcal{M} = 1.19$ GB of VRAM for a training iteration over batch of 32 images of resolution $240 \times 240$.

Finally, the memory savings allowed by the reversible block come with the additional computational cost of computing the hidden activations during the backward pass. As noted in the original paper, this computational cost is equivalent to performing one additional forward pass.

### 3.3.5 iRevNet

The iRevNet model builds on the RevNet architecture: they replace the irreversible max-pooling operation with an invertible operation that reshapes the hidden activation states so as to compensate for the loss of spatial resolution by an increase in the channel dimension. As such, the iRevNet architecture is fully invertible, which alleviates the global memory bottleneck of the RevNet architecture.

This pooling operation works by stacking the neighboring elements of the pooling regions along the channel dimension, i.e.; for a 2D pooling operation with $2 \times 2$ pooling window, the number of output channels is four times the number of input channels. Unfortunately, the size of a volume-preserving convolution kernel grows quadratically in the number of input channels:

$$M(\theta) = c_{in} \times c_{out} \times k_h \times k_w \tag{3.12a}$$

$$M(\theta) = c^2 \times k_h \times k_w \tag{3.12b}$$

Consider an iRevNet network with initial channel size 32. After three levels of $2 \times 2$ pooling, the effective channel size becomes $32 \times 4^3 = 2048$. A typical $3 \times 3$ convolution layer kernel for higher layers of such network would have $n(\theta) = 2048^2 \times 3 \times 3 = 37M$ parameters. At this point, the memory cost of the network weights $M_\theta$ becomes an additional memory bottleneck.

Furthermore, the iRevNet architecture does not address the local memory bottleneck of the reversible blocks. Figure 4 illustrates such architecture. For an initial channel size of 32, as summarized in Table 1, the peak memory consumption is given by:

$$\mathcal{M} = M_\theta + M_z + M_g \tag{3.13a}$$
$$\mathcal{M} = (M_\theta + (M'_z + M'_g) \times (h \times w \times bs)) \times bpe \tag{3.13b}$$
$$\mathcal{M} = (171 \times 10^6 + 640 \times (h \times w \times bs)) \tag{3.13c}$$
$$\tag{3.13d}$$

Training such an architecture for an iteration over batches of 32 images of resolution $240 \times 240$ would require $\mathcal{M} = 1.35$GB of VRAM. In the next section, we introduce both layer-wise reversibility and a variant on this pooling operations to address the local memory bottleneck of reversible blocks and the weight memory bottleneck respectively.

## 3.4 Method

RevNet and iRevNet architectures implement reversible transformations at the level of residual blocks. As we have seen in the previous section, the design of these reversible blocks create a local memory bottleneck as all hidden activations within a reversible block need to be computed before the gradients are back-propagated through the block. In order to circumvent this local bottleneck, we introduce layer-wise invertible operations. However, these invertible operations introduce numerical error, which we characterize in the following subsections. In Section 5, we will show that these numerical errors lead to instabilities that degrade the model accuracy. Hence, in section 4.2, we propose a hybrid model combining layer-wise and residual block-wise reversible operations to stabilize training while resolving the local memory bottleneck at the cost of a small additional computational cost.

### 3.4.1 Layer-wise Invertibility

In this section, we present invertible layers that act as drop-in replacement for convolution, batch normalization, pooling and non-linearity layers. We then characterize the numerical instabilities arising from the invertible batch normalization and non-linearities.

#### 3.4.1.1 Invertible batch normalization

As batch normalization is not a bijective operation, it does admit an analytical inverse. However, the inverse reconstruction of a batch normalization layer can be realized with minimal memory cost. Given first and second order moment parameters $\beta$ and $\gamma$, the forward $f$ and inverse $f^{-1}$ operation of an invertible batch normalization layer can be computed as follows:

$$y = f(x) = \gamma \times \frac{x - \hat{x}}{\sqrt{\dot{x}} + \epsilon} + \beta \tag{3.14a}$$

$$x = f^{-1}(y, \hat{x}, \dot{x}) = (\sqrt{\dot{x}} + \epsilon) \times \frac{y - \beta}{\gamma} + \hat{x} \tag{3.14b}$$

Where $\hat{x}$ and $\dot{x}$ represent the mean and standard deviation of $x$ respectively. Hence, the input activation $x$ can be recovered from $y$ through $f^{-1}$ at the minimal memory cost of storing the input activation statistics $\hat{x}$ and $\dot{x}$.

Let us consider the accumulation of numerical errors arising from the inverse computation of an invertible batch normalization layer. During the backward pass, the invertible batch norm layer is supposed to compute its input $x = f^{-1}(y, \hat{x}, \dot{x})$ from the output $y$. In reality, however, the output recovered by upstream invertible layers is a noisy estimate $\hat{y} = y + \epsilon_y$ of the true output due to numerical errors introduced by upstream layers. Let us define the signal to noise ratio (SNR) of the input and output signal as follows:

$$snr_o = \frac{|y|^2}{|\epsilon^y|^2} \tag{3.15a}$$

$$snr_i = \frac{|x|^2}{|\epsilon^x|^2} \tag{3.15b}$$

We are interested in characterizing the factor $\alpha$ of reduction of the SNR through the inverse reconstruction:

$$\alpha = \frac{snr_i}{snr_o} \tag{3.16}$$

To illustrate the mechanism through which the batch normalization inverse operation reduces the SNR, let us consider a toy layer with only two channels and parameters $\beta = [0, 0]$ and $\gamma = [1, \rho]$. For simplicity, let us consider an input signal $x$ independently and identically distributed across both channels with zero mean and standard deviation 1 so that, in the forward pass, we have:

$$y =[y_0, y_1] \tag{3.17a}$$

$$y =[x_0, x_1 \times \rho] \tag{3.17b}$$

$$|y|^2 =|x_0|^2 + |x_1|^2 \times \rho^2 \tag{3.17c}$$

$$|y|^2 =\frac{1}{2} \times |x|^2 + \frac{1}{2} \times |x|^2 \times \rho^2 \tag{3.17d}$$

$$|y|^2 =\frac{|x|^2}{2} \times (1 + \rho^2) \tag{3.17e}$$

During the backward pass, the noisy estimate $\tilde{y} = y + \epsilon^y$ is fed back as input to the inverse operation. Similarly, let us suppose a noise $\epsilon^y$ identically distributed across both channels so that we have:

$$\tilde{y} = [\tilde{y}_0, \tilde{y}_1] \tag{3.18a}$$
$$\tilde{y} = [x_0 + \epsilon^y_0, x_1 \times \rho + \epsilon^y_1] \tag{3.18b}$$
$$\tilde{x} = [\tilde{y}_0, \frac{\tilde{y}_1}{\rho}] \tag{3.18c}$$
$$\tilde{x} = [x_0 + \epsilon^y_0, x_1 + \frac{\epsilon^y_1}{\rho}] \tag{3.18d}$$
$$\epsilon^x = \tilde{x} - x \tag{3.18e}$$
$$\epsilon^x = [\epsilon^y_0, \frac{\epsilon^y_1}{\rho}] \tag{3.18f}$$
$$|\epsilon^x|^2 = |\epsilon^y_0|^2 + \frac{|\epsilon^y_1|^2}{\rho^2} \tag{3.18g}$$
$$|\epsilon^x|^2 = \frac{1}{2} \times |\epsilon^y|^2 + \frac{1}{2} \times \frac{|\epsilon^y|^2}{\rho^2} \tag{3.18h}$$
$$|\epsilon^x|^2 = \frac{|\epsilon^y|^2}{2} \times (1 + \frac{1}{\rho^2}) \tag{3.18i}$$

Using the above formulation, the SNR reduction factor $\alpha$ can be expressed as:

$$\alpha = \frac{snr_i}{snr_o} \tag{3.19a}$$
$$\alpha = \frac{|x|^2}{|\epsilon^x|^2} \times \frac{|\epsilon^y|^2}{|y|^2} \tag{3.19b}$$
$$\alpha = \frac{4}{(1 + \frac{1}{\rho^2}) \times (1 + \rho^2)} \tag{3.19c}$$

Figure 5 shows the expected evolution of $\alpha$ through our toy layer for different values of the factor $\rho$. To validate our formula, we empirically evaluate $\alpha$ for normal Gaussian inputs $x$ and output noise $\epsilon^y$ and find it to closely match the theoretical results given by equation 19.

In essence, numerical instabilities in the inverse computation of the batch normalization layer arise from the fact that the signal across different channels $i$

and $j$ are amplified by different factors $\gamma_i$ and $\gamma_j$. While the signal amplification in the forward and inverse path cancel out each other ($x = f^{-1}(f(x))$), the noise only gets amplified in the backward pass.

In the above demonstration, we have used a toy parameterization of the invertible batch normalization layer to illustrate the mechanism behind the SNR degradation. For arbitrarily parameterized batch normalization layers, the SNR degradation factor becomes:

$$\alpha = \frac{snr_i}{snr_o} \tag{3.20a}$$

$$\alpha = \frac{|x|^2}{|\epsilon^x|^2} \times \frac{|\epsilon^y|^2}{|y|^2} \tag{3.20b}$$

$$\alpha = \frac{|x|^2}{|y|^2} \times \frac{|\epsilon^y|^2}{|\epsilon^x|^2} \tag{3.20c}$$

$$\tag{3.20d}$$

Assuming a noise $\epsilon^y$, equally distributed across all channels, the noise ratio can be computed as follows:

$$\tilde{y}_i = \gamma \times \frac{x_i - \hat{x}_i}{\sqrt{\dot{x}_i} + \epsilon_i} + \beta_i + \epsilon_i^y \tag{3.21a}$$

$$\tilde{x}_i = (\sqrt{\dot{x}_i} + \epsilon) \times \frac{y_i - \beta_i}{\gamma_i} + \hat{x}_i \tag{3.21b}$$

$$\tilde{x}_i = x_i + \frac{\gamma_i}{\dot{x}_i} \times \epsilon_i^y \tag{3.21c}$$

$$\epsilon_i^x = \tilde{x}_i - x_i \tag{3.21d}$$

$$\epsilon_i^x = \frac{\gamma_i}{\dot{x}_i} \times \epsilon_i^y \tag{3.21e}$$

$$\frac{|\epsilon^y|^2}{|\epsilon^x|^2} = \frac{|\epsilon^y|^2}{\frac{|\epsilon^y|^2}{c} \times \sum_i \frac{\dot{x_i}^2}{\gamma_i^2}} \tag{3.21f}$$

$$\frac{|\epsilon^y|^2}{|\epsilon^x|^2} = \frac{c}{\sum_i \frac{\dot{x_i}^2}{\gamma_i^2}} \tag{3.21g}$$

Assuming input $x$ following a Gaussian distribution with channel-wise mean and standard deviation $\hat{x_i}$ and $\dot{x_i}$ respectively, the formula for the SNR reduction factor $\alpha$ becomes:

$$\frac{|x|^2}{|y|^2} = \frac{\sum_i |x_i|^2}{\sum_i |y|^2} \tag{3.22a}$$

$$\frac{|x|^2}{|y|^2} = \frac{\sum_i (\hat{x}_i^2 + \dot{x_i}^2)}{\sum_i (\gamma_i^2 + \beta_i^2)} \tag{3.22b}$$

$$\alpha = \frac{\sum_i (\hat{x}_i^2 + \dot{x_i}^2)}{\sum_i (\gamma_i^2 + \beta_i^2)} \times \frac{c}{\sum_i \frac{\dot{x_i}^2}{\gamma_i^2}} \tag{3.22c}$$

We verify the validity of equation (22c) by empirically evaluating the different $\alpha$ ratio yielded by randomly initialized batch normalization layers with input values and output noise randomly drawn from Gaussian distribution. Figure 5 plots the empirically found $\alpha$ values against the theoretical values predicted by our formula and show a close match.

Finally, we propose the following modification, introducing the hyperparameter $\epsilon_i$, to the invertible batch normalization layer:

$$y = f(x) = |\gamma + \epsilon_i| \times \frac{x - \hat{x}}{\sqrt{\dot{x}} + \epsilon} + \beta \tag{3.23a}$$

$$x = f^{-1}(y) = (\sqrt{\dot{x}} + \epsilon) \times \frac{y - \beta}{|\gamma + \epsilon_i|} + \hat{x} \tag{3.23b}$$

The introduction of the $\epsilon_i$ hyper parameter serves two purposes: First, it stabilizes the numerical errors described above by lower bounding the smallest $\gamma$ parameters. Second, it prevents numerical instabilities that would otherwise arise from the inverse computation as $\gamma$ parameters tend towards zero.

### 3.4.1.2 Invertible activation function

A good invertible activation function must be bijective (to guarantee the existence of an inverse function) and non-saturating (for numerical stability). For these properties, we focus our attention on Leaky ReLUs whose forward $f$ and inverse $f^{-1}$ computations are defined, for a negative slope parameter $n$, as follow:

$$y = f(x) = \begin{cases} x, & \text{if } x > 0 \\ x/n, & \text{otherwise} \end{cases} \tag{3.24a}$$

$$x = f^{-1}(y) = \begin{cases} y, & \text{if } y > 0 \\ y \times n, & \text{otherwise} \end{cases} \tag{3.24b}$$

The analysis of the numerical errors yielded by the invertible Leaky ReLU follows a similar reasoning as the toy batch normalization example with an additional subtlety: Similar to the toy batch normalization example, we can think of the leaky ReLU as artificially splitting the input x across two different channels, one channel leaving the output unchanged and one channel that divides the input by a factor $n$ during the forward pass and multiplies its output by a factor $n$ during the backward pass.

However, these artificial channels are defined by the sign of the input and output during the forward and backward pass respectively. Hence, we need to consider the cases in which the noise flips the sign of the output activations, which leads to different behaviors of the invertible Leaky ReLU across four cases:

$$y = \begin{cases} y_{nn} \text{ if } \hat{y} < 0 & \text{and } y < 0 \\ y_{np} \text{ if } \hat{y} >= 0 & \text{and } y < 0 \\ y_{pp} \text{ if } \hat{y} >= 0 & \text{and } y >= 0 \\ y_{pn} \text{ if } \hat{y} < 0 & \text{and } y >= 1 \end{cases} \tag{3.25a}$$

Where the index $np$, for instance, represents negative activations whose reconstructions have become positive due to the added noise. The signal to noise ratio of the input and outputs can be expressed respectively as:

In the case where $y >> \epsilon_y$, the probability of sign flips ($y_{np}$, $y_{pp}$) is negligible, so that the output signal $y$ is evenly splited along $y_{pp}$ and $y_{nn}$. In this regime, the degradation of the SNR obeys a formula similar to the toy batch normalization example:

$$y = [y_{pp}, y_{nn}] \tag{3.26a}$$
$$y = [x_{pp}, \frac{x_{nn}}{n}] \tag{3.26b}$$
$$|y|^2 = \frac{1}{2} \times |x|^2 + \frac{1}{2} \times \frac{|x|^2}{n^2} \tag{3.26c}$$
$$|y|^2 = \frac{|x|^2}{2} \times (1 + \frac{1}{n^2}) \tag{3.26d}$$

$$\tilde{y} = [\tilde{y}_{pp}, \tilde{y}_{nn}] \tag{3.27a}$$
$$\tilde{y} = [x_{pp} + \epsilon^y_{pp}, \frac{x_{nn}}{n} + \epsilon^y_{nn}] \tag{3.27b}$$
$$\tilde{x} = [\tilde{y}_{pp}, \tilde{y}_{nn} \times n] \tag{3.27c}$$
$$\tilde{x} = [x_{pp} + \epsilon^y_{pp}, x_{nn} + \epsilon^y_{nn} \times n] \tag{3.27d}$$
$$\epsilon^x = \tilde{x} - x \tag{3.27e}$$
$$\epsilon^x = [\epsilon^y_{pp}, \epsilon^y_{nn} \times n] \tag{3.27f}$$
$$|\epsilon^x|^2 = \frac{1}{2} \times |\epsilon^y|^2 + \frac{1}{2} \times |\epsilon^y|^2 \times n^2 \tag{3.27g}$$
$$|\epsilon^x|^2 = \frac{|\epsilon^y|^2}{2} \times (1 + n^2) \tag{3.27h}$$

Using the above formulation, the signal to noise ration reduction factor $\alpha$ can be expressed as:

$$\alpha = \frac{snr_i}{snr_o} \tag{3.28a}$$
$$\alpha = \frac{|x|^2}{|\epsilon^x|^2} \times \frac{|\epsilon^y|^2}{|y|^2} \tag{3.28b}$$
$$\alpha = \frac{4}{(1 + \frac{1}{n^2}) \times (1 + n^2)} \tag{3.28c}$$

Hence numerical errors can be controlled by setting the value of the negative slope $n$. As $n$ tends towards 1, $\alpha$ converges to 1, yielding minimum signal degradation. However, as $n$ tends towards 1, the network tends toward a linear behavior, which hurts the model expressivity. Figure 6 shows the evolution of the SNR degradation $\alpha$ for different negative slopes $n$; and, in Section 5.1, we investigate the impact of the negative slope parameter on the model accuracy.

When the noise reaches an amplitude similar to or greater than the activation signal, the effects of sign flips complicate the equation. However, in this regime, the signal to noise ratio becomes too low for training to converge, as numerical errors prevent any useful weight update, so we leave the problem of characterizing this regime open.

#### 3.4.1.3 Invertible convolutions

Invertible convolution layers can be defined in several ways. The inverse operation of a convolution is often referred to as deconvolution, and is defined for a subspace of the kernel weight space.

However, deconvolutions are computationally expensive and subject to numerical errors. Instead, we choose to implement invertible convolutions using the channel partitioning scheme as the reversible block design for its simplicity, numerical stability and computational efficiency. Hence, invertible convolutions, in our architecture, can be seen as minimal reversible blocks in which both modules consist of a single convolution. Gomez *et al.*[27] found the numerical errors introduced by reversible blocks to have no impact on the model accuracy. Similarly, we found reversible blocks extremely stable yielding negligible numerical errors compared to the invertible batch normalization and Leaky ReLU layers.

#### 3.4.1.4 Pooling

In [28], the authors propose an invertible pooling operation that operates by stacking the neighboring elements of the pooling regions along the channel dimension. As noted in Section 3.5, the increase in channel size at each pooling level induces a quadratic increase in the number of parameters of upstream convolution, which creates a new memory bottleneck.

To circumvent this quadratic increase in the memory cost of the weight, we propose a new pooling layer that stacks the elements of neighboring pooling regions along the batch size instead of the channel size. We refer to both kind of pooling as channel pooling $\mathcal{P}_c$ and batch pooling $\mathcal{P}_b$ respectively, depending on the dimension along which activation features are stacked. Given a $2 \times 2$ pooling region and an input activation tensor $x$ of dimensions $bs \times c \times h \times w$, where $bs$ refers to the batch size, $c$ to the number of channels and $h \times w$ to the spatial resolution, the reshaping operation performed by both pooling layers can be formalized as follows:

$$\mathcal{P}_c : x \to y \tag{3.29a}$$
$$\mathcal{P}_c : \mathbb{R}^{bs \times c \times h \times w} \to \mathbb{R}^{bs \times 4c \times \frac{h}{2} \times \frac{w}{2}} \tag{3.29b}$$
$$\mathcal{P}_b : x \to y \tag{3.29c}$$
$$\mathcal{P}_b : \mathbb{R}^{bs \times c \times h \times w} \to \mathbb{R}^{4bs \times c \times \frac{h}{2} \times \frac{w}{2}} \tag{3.29d}$$

Channel pooling gives usa way to perform volume-preserving pooling operations while increasing the number of channels at a given layer of the architecture, while batch pooling gives us a way to perform volume-preserving pooling operations while keeping the number of channel constant, By alternating between channel and batch pooling, we can control the number of channels at each pooling level of the model's architecture.

#### 3.4.1.5 Layer-wise invertible architecture

Putting together the above building blocks, Figure 7 illustrates a layer-wise invertible architecture. For an input channel size of 32, The peak memory usage for an iteration of training this architecture can be computed as follows:

$$\mathcal{M} = M_\theta + M_z + M_g \tag{3.30a}$$
$$\mathcal{M} = (M_\theta + (M'_z + M'_g) \times (h \times w \times bs)) \times bpe \tag{3.30b}$$
$$\mathcal{M} = (29.6 \times 10^6 + 320 \times (h \times w \times bs)) \tag{3.30c}$$
$$\tag{3.30d}$$

Training an iteration over a typical batch of 32 images with resolution $240 \times 240$ would require $\mathcal{M} = 590$MB of VRAM. Similar to the RevNet architecture, the reconstruction of the hidden activations by inverse transformations during the backward pass comes with an additional computational cost similar to a forward pass.

### 3.4.2 Hybrid architecture

In section 3, we saw that layer-wise activation and normalization layers degrade the signal to noise ratio of the reconstructed activations. In section 5.1, we will quantify the accumulation of numerical errors through long chains of layer-wise invertible operations and show that numerical errors negatively impact model accuracy.

To prevent these numerical instabilities, we introduce a hybrid architecture, illustrated in Figure 8, combining reversible residual blocks with layer-wise invertible functions. Conceptually, the role of the residual level reversible block is to reconstruct the input activation of residual blocks with minimal errors, while the role of the layer-wise invertible layers is to efficiently recompute the hidden activations within the reversible residual blocks at the same time as the gradient propagates to circumvent the local memory bottleneck of the reversible module.

The backward pass through these hybrid reversible blocks is illustrated in Figure 9 and proceeds as follows: First, the input $x$ is computed from the output $y$ through the analytical inverse of the reversible block. These computations are made without storing the hidden activation values of the sub-modules. Second, the gradient of the activations are propagated backward through the reversible of the block modules. As each layer within these modules is invertible, the hidden activation values are computed using the layer-wise inverse along the gradient.

The analytical inverse of the residual level reversible blocks is used to propagate hidden activations with minimal reconstruction error to the lower modules, while layer-wise inversion allows us to alleviate the local bottleneck of the reversible block by computing the hidden activation values together with the backward flow of the gradients. As layer-wise inverses are only used for hidden feature computations within the scope of the reversible block, and reversible blocks are made of relatively short chains of operations, numerical errors do not accumulate up to a damaging degree, The peak memory consumption of our proposed architecture, as illustrated in Figure 8 and parameterized in Table 1, can be computed as

$$\mathcal{M} = M_\theta + M_z + M_g \tag{3.31a}$$

$$\mathcal{M} = (M_\theta + (M'_z + M'_g) \times (h \times w \times bs)) \times bpe \tag{3.31b}$$

$$\mathcal{M} = (14.8 \times 10^6 + 352 \times (h \times w \times bs)) \tag{3.31c}$$

$$\tag{3.31d}$$

Training an iteration over batch of 32 images of resolution $240 \times 240$ would require $\mathcal{M} = 648$MB of VRAM.

It should be noted, however, that this architecture adds an extra computational cost as both the reversible block inverse and layer-wise inverse need to be computed. Hence, instead of one additional forward pass, as in the RevNet and layer-wise architectures, our hybrid architecture comes with a computational cost equivalent to performing two additional forward passes during the backward pass.

## 3.5 Experiments and results

We use the CIFAR10 dataset as a benchmark for our experiments. The CIFAR10 dataset is complex enough to require efficient architectures to reach high accuracy, yet small enough to enable us to rapidly iterate over different architectural designs. We start by analyzing numerical errors arising in layer-wise invertible and hybrid architectures, and outline their impact on accuracy. This analysis motivates our choice of architecture and hyperparameter. We then summarize the benefits and drawbacks of our proposed architecture in comparison to different baseline architectures.

### 3.5.1 Impact of Numerical stability

#### 3.5.1.1 Layer-wise Invertible Architecture

In this section, we quantify the accumulation of numerical errors in layer-wise invertible architectures and analyze their impact on the accuracy. The architecture of these models is illustrated in Figure 7. We investigate the evolution of numerical errors, and their impact on accuracy, for networks of different depth and different hyper-parameter values. Figure 10 illustrates the degradation of the signal-to-noise ration along the layers of one such model.

We found the two most impacting parameters to be the depth $N$ of the network and the negative slope $n$ of the activation function. Figure 11 shows the evolution of the numerical errors with both of these parameters.

Next, we investigate the impact of numerical errors on the accuracy. In order to isolate the impact of the numerical errors, we compare the accuracy reached by the same architecture with and without inverse reconstruction of the hidden layers activations. Without reconstruction, the hidden activation values are stored along the forward pass and the gradient updates are computed from the true, noiseless activation values, so that the only difference between both settings is the noise introduced by the inverse reconstructions.

In Figure 12, we compare the evolution of the accuracy in both settings for different depth and negative slopes. For small depths (or high negative slopes), in which the numerical errors are minimum, both models yield similar accuracy. However, as the numerical errors grow, the accuracy of the model goes down, while the accuracy of the ideal baseline keeps increasing, which can be seen with both depth and negative slopes. This loss in accuracy is the direct result of numerical errors, which prevent the model from converging to higher accuracies.

#### 3.5.1.2 Hybrid Invertible Architecture

In section 4.2, we introduced a hybrid architecture, illustrated in Figure 8, to prevent the impact of numerical errors on accuracy. Figure 13 shows the propagation of the signal to noise ratio through the layers of such hybrid architecture. As can be seen in this figure, the hybrid architecture is much more robust to numerical errors as activations are propagated from one reversible block to the other using the reversible block inverse computations instead of layer-wise inversions.

Figure 14 shows the evolution of the SNR with increasing depth $N$ and for different values of negative slope $n$. This figure shows a much more stable evolution of the signal to noise ratio than the layer-wise architecture.

| Model | Accuracy | #Params | Channels | Pooling | $M_\theta$ | $M'_z$ | $\mathcal{M}$ |
|---|---|---|---|---|---|---|---|
| Resnet | 94.7 | $3.1M$ | $32-64-128-256$ | Max Pooling | $12.4M$ | 1928 | $1.01G$ |
| RevNet | 94.5 | $3.1M$ | $40-80-256-320$ | Max Pooling | $12.4M$ | 640 | $348M$ |
| i-RevNet | 93.8 | $42.8M$ | $32-128-512-2048$ | $\mathcal{P}_c-\mathcal{P}_c-\mathcal{P}_c$ | $171M$ | 640 | $500M$ |
| Ours | 93.3 | $3.7M$ | $32-128-512-512$ | $[\mathcal{P}_c,\mathcal{P}_c,\mathcal{P}_b]$ | $14.8M$ | 352 | $200M$ |

**Table 3.1:** Summary of architectures with different levels of reversibility

Figure 15 compares the evolution of the accuracy reached by this hybrid architecture with noisy activations and noiseless ideal activations as depth and negative slope increase. The negative impacts of numerical errors observed in the layer-wise architecture are gone, confirming that the numerical stability brought by the hybrid architecture effectively stabilizes training.

### 3.5.2 Model comparison

Table 1 summarizes our main results. In this table, we compare architectures with different patterns of reversibility. To allow for a fair comparison, we have tweaked each architecture to keep the number of parameters as close as possible, with the notable exception of the i-RevNet architecture. The i-Revnet pooling scheme enforces a quadratic growth of its parameters with each level of pooling. In order to keep the number of parameters of the i-RevNet close to the other baselines, we would have to drastically reduce the number of channels of lower layers, which we found yield poor performance. Furthermore, it should be noted that the i-RevNet architecture we present slightly differs from the original i-Revnet model as our implementation uses RevNet-like reversible modules with one module per channel split for similarity with the other architecture we evaluate instead of the single module used in the original architecture.

All models were trained for 50 epochs of stochastic gradient descent with cyclical learning rate and momentum [44] with minimal image augmentation.

The parameters of our proposed architecture are given in Table 1. This architecture was selected as the best performing architecture from an extensive architecture search on a constrained weight budget. Compared to the original ResNet architecture, our model drastically cuts the memory cost of training. These drastic memory cuts come at the cost of a small degradation in accuracy.

| GPU | Time | Accuracy |
|---|---|---|
| GTX750 | 93.3 | 35$min.$ |
| GTX 1080Ti | 93.3 | 67$min.$ |

**Table 3.2:** Training statistics on different hardware

Furthermore, our hybrid architecture requires the computational equivalent of two additional forward passes within each backward pass. The computational complexity, however, remains reasonable: In Table 2, we compare the time of training our proposed architecture to 93.3% on a high-end Nvidia GTX 1080Ti and a low-end Nvidia GTX750. The GTX750 only has 1GB of VRAM, which results in roughly 400MB of available memory after the initialization of various frameworks. Training a vanilla ResNet with large batch sizes on such limited memory resources is impractical, while our architecture allows for efficient training.

## 3.6 Conclusion

Convolutional Neural Networks form the backbone of modern computer vision systems. However, the accuracy of these models come at the cost of resource intensive training and inference procedures. While tremendous efforts have been put into the optimization of the inference step on resource-limited device, relatively little work have focused on algorithmic solutions for limited resource training. In this paper, we have presented an architecture able to yield high accuracy classifications within very tight memory constraints. We highlighted several potential applications of memory-efficient training procedures, such as on-device training, and illustrated the efficiency of our approach by training a CNN to 93.3% accuracy on a low-end GPU with only 1GB of memory.

# Chapter 4

# Semantic Feature Extraction

The related publications for this chapter are [].

## 4.1 Introduction

Recent successes in generic object recognition have largely been driven by the successful application of Convolutional Neural Networks (CNN) trained in a supervised manner on large image datasets. One main drawback of these approaches is that they require a large amount of annotated data to successfully generalize to unseen image samples. The collection and annotation of such dataset for custom applications can be prohibitively complex and/or expensive which hinders their applications to many real world practical scenarios. To reduce the number of training samples needed for efficient learning, *few-shot learning* techniques are being actively researched. The zero-shot learning (ZSL) paradigm represents the extreme case of few-shot learning in which recognition models are trained to recognize instances of a set of target classes without any training sample to learn from.

To recognize unseen classes, ZSL models use descriptions of the visual classes, i.e., representations of the visual classes in a non-visual modality. Research in ZSL has been driven by relatively small scale benchmarks [8, 45] for which human-annotated visual attributes are available as visual class descriptions. In the case of generic object recognition, however, manually annotating each and every possible visual class of interest with a set of visual attributes is impractical. Hence, generalizing the zero-shot learning approaches developed on such benchmarks to the more practical case of generic object recognition comes with the additional challenge of collecting suitable descriptions of the visual classes.

Finding such description presents two challenges: first, the collection of these descriptions must be automated so as to not require an expensive human annotation process. Second, the collected descriptions must be visually discriminative enough to enable the zero-shot recognition of generic objects. Word embeddings are learned in an unsupervised manner from large text corpora so that they can be collected in a large scale without human supervision. Furthermore, their successful application to a number of Natural Language Processing (NLP) tasks have shown that word embedding representations encode a number of desirable semantic features, which have been naturally assumed to generalize to vision tasks. For these desirable properties, word embeddings have become the standard visual class descriptions used by recent zero-shot generic object recognition models([1, 7, 12, 15, 46]).

In this paper, we first question the current consensus on using word embeddings for zero-shot recognition of visual classes and discuss the relevance and limitations of the application of word embeddings to vision tasks. We then argue that generic objects can also be described by either text documents or knowledge graph data that satisfy our requirements: these descriptions both contain visually discriminative information and are automatically collectible from the web in a large scale, without requiring human intervention. We then present state of the art methods to learn feature representations of both graph and text documents. Finally, we use a simple ZSL baseline [14] model to evaluate the visually discriminative power of these representations on a zero-shot generic object recognition benchmark. In the remainder or this paper, we refer to the descriptions of visual classes in different modalities as different *levels* of descriptions: either as word-, document-, or graph-level descriptions.

Our investigation highlights large differences in the ZSL accuracy of our baseline model using different embeddings. First, we find all word embedding models to not be equal: using GloVe [47] representations instead of the Word2vec [48] vectors used by previous works almost doubles the accuracy of our baseline model on a standard test split, effectively outperforming state-of-the-art results obtained by more sophisticated ZSL models. Second, we find that different levels of descriptions are better suited to different application settings: graph embeddings tend to outperform word and document embeddings for small test class sets that are semantically close to the training classes according to the Wordnet hierarchy. In fact, we show that embedding a simple taxonomy of visual classes with a recently proposed model [49] slightly outperforms the best performing word embeddings. Embedding the full Wordnet knowledge graph, we are able to further improve on the state-of-the-art by a relative 45%. On the other hand, we found graph embeddings to perform poorly on larger, more diverse test sets. On such difficult test sets, document embeddings tend to perform the best, although classification accuracy remains very low.

To summarize, the contributions of our work are as follows:

- We give insights into the relevance (Section 4.2) and limitations (Section 4.3) of word embeddings for zero-shot recognition of generic objects. We conduct a set of experiments to quantify the extent of these limitations (Section 8.2).

- We show how Linked Open Data can be used to automatically collect rich descriptions of generic objects at the graph and document levels (Section 5).

- We review the literature on graph (Section 6) and document (Section 7) embedding.

- We conduct the first, to the best of our knowledge, large-scale investigation of semantic representations for zero-shot recognition of generic objects (Section 8.3).

- We make our code and data openly available for future research.

Section 2 reviews the literature for related work and Section 3 presents the methodology used in our investigation.

## 4.2 Related Work

While the majority of works on zero-shot generic object recognition have used word embeddings as semantic features, some works have explored the use of different semantic features, which we present in this section.

In [50], the authors use different linguistic resources to derive semantic similarity scores between classes, between classes and attributes, and to automatically mine attribute-classes correspondence. Similar to our work, they automate the acquisition of semantic data from knowledge bases, but they focus on deriving semantic similarity scores and part attributes while we evaluate graph embedding models.

[51] uses visual class co-occurrence statistics to perform ZSL. Given a training set of multi-labeled images and similarity scores between known and unknown labels, they use the co-occurrence distribution of known labels to predict the occurrence of unknown labels in test images. Their multi-label classification setting differs from the ZSL setting in which input images are classified into a unique class.

[52] questions the limits of using a single data point (word embedding vectors) as semantic representations of visual classes because this setting does not allow the representation of intra-class variance of semantic concepts. They used Gaussian distributions to model both semantic and visual feature distributions of the visual classes.

More related to our work, [53] investigates different semantic representations for zero-shot action recognition. They compare different representations of documents and videos, while we investigate the application of word, document and knowledge graph embeddings to zero-shot recognition of generic objects.

A series of works of [13, 54, 55] compares the zero-shot classification accuracy obtained with semantic representations derived from words, taxonomy and manual attribute annotations on fine-grain or small scale ZSL benchmarks. Our investigation differs in that we are concerned with the more practical task of generic object recognition and we investigate a broader class of semantic features.

## 4.3 Method

The general architecture of ZSL models can be seen as the combination of three modules $\{V, S, E\}$, as illustrated in Figure 1. The visual module $V$ extracts high-level visual features $V(x)$ from raw input images $x$; the semantic module $S$ extracts semantic features $S(y)$ from raw descriptions $y$ of the visual classes and the core ZSL module $E$ computes a similarity score $E(V(x), S(y))$ between semantic and visual features.

ZSL models aim to generalize the classification ability of traditional image classifiers to out-of-sample classes for which no image sample is available to learn from. To evaluate the out-of-sample recognition ability of models, ZSL benchmarks split the full set of classes $C$ into disjoint training and test sets.

$$C_{train} \cup C_{test} \subset C \tag{4.1a}$$

$$C_{train} \cap C_{test} = \emptyset \tag{4.1b}$$

$$Tr = \{(x, y), y \in C_{train}\} \tag{4.1c}$$

$$Te = \{(x, y), y \in C_{test}\} \tag{4.1d}$$

Learning is performed by minimizing a loss function $\mathcal{L}$ over the regularized similarity score of the set of training sample $Tr$ with respect to the model's parameters $W$.

$$W^* = argmin_W \mathbb{E}_{(x,y) \in Tr} \mathcal{L}(E(V(x), S(y)) + \Omega(W) \tag{4.2}$$

At test time, an image $x_{test}$ can be classified among the set of unseen test classes by retrieving the description $y$ of highest similarity score.

$$y^* = argmax_{y \in C_{test}} E(V(x_{test}), S(y)) \tag{4.3}$$

The visual and semantic modules can either be learned jointly with the core ZSL module in an end-to-end procedure by back-propagation of the error signal from the core ZSL module to the two lower modules, or they can be learned independently on unsupervised or auxiliary supervised tasks (e.g., pretraining the visual module on the ILSVRC classification task and pretraining the semantic module as an unsupervised word embedding model).

Our work focuses exclusively on the semantic module: we question what raw descriptions $y$ and embedding module $S$ provide semantic features $S(y)$ that are most visually discriminative so as to enable zero-shot recognition of generic objects. We restrict our study to embedding models $S$ learned independently from other modules, without visual supervision from the ZSL module.

To conduct our study, we are heavily dependent on the data available to us in the form of image-description pairs $(x, y)$. We use the ImageNet dataset as our starting point as it has become the standard evaluation benchmark for generic object ZSL. In ImageNet, visual classes are indexed by Wordnet concepts, which are defined by three components that correspond to the three levels of representations we investigate: their lemmas (a set of synonym words that refer to the concept), a definition in natural language, and a node connected by a set of predicate edges to other concept nodes of the Wordnet knowledge graph. Figure 2 illustrates the different levels of descriptions provided by Wordnet.

In section 5, we will show how the semantic web can be used to automatically collect document- and graph-level descriptions of visual classes that are order of magnitudes larger than the descriptions provided by Wordnet. We first discuss the relevance and limitations of word embeddings in the following section.

## 4.4 Word Embeddings

### 4.4.1 Overview

Distributional Semantic Models (DSM) and neural word embeddings are two related classes of models that learn continuous distributed representations of words. These models implement the distributional hypothesis that states that the meaning of words can be defined by the context in which they occur. DSMs explicitly factorize matrices of word co-occurrence statistics while neural word embedding models learn word representations by stochastic optimization methods. The latter typically sample individual words and their context from large text corpora, maximizing a similarity score between co-occurring words. These approaches have been extensively studied both theoretically [56] and practically [57]. In

[56], the authors show that the skip-gram word2vec model with negative sampling implicitly factorizes a shifted PMI matrix, suggesting that both approaches are qualitatively similar. For the sake of the following discussion, we consider that word embedding models do implicitly factorize matrices derived from word co-occurrence statistics following [56]. While qualitatively similar, the empirical study of [57] showed that neural embedding approaches tend to outperform DSM models on standard benchmarks. In Section 8.2, we evaluate three state-of-the-art embedding models on our ZSL benchmark: GloVe [47], FastText [58] and word2vec [48].

### 4.4.2 Relevance

Word embeddings have been shown to efficiently encode lexical and semantic similarities between words. Their successful application to NLP tasks has prompted word embeddings as standard semantic representations for zero-shot learning, while there has been little discussion as to the relevance and limitations of their application to vision task.

Slightly different from the original distributional hypothesis, ZSL models using word embeddings as semantic features make the assumption that the *appearance of generic objects* can be characterized by the context in which their lemmas occur. Table 1 shows the co-occurrence frequency of a few common visual class lemmas with words that explicitly characterize visual attributes. This table shows that visual class lemmas tend to share high co-occurrence frequency with either their part attributes (i.e., both car and truck co-occur more frequently with wheel than bird and cassowary do) or action verbs that implicitly impacts the shape of these objects (i.e. both cars and trucks are “drivable” vehicles). This suggests that the co-occurrence patterns of words in large text corpora indeed contain information regarding distinctive visual features shared among classes. Hence, the latent space learned by the implicit factorization of such matrix embeds visually discriminative information so that word embeddings provide suitable semantic representations for zero-shot learning applications. A particular exception that stands out from Table 1 is the word cassowary, which we discuss in the following section.

**Table 4.1:** Lemmas co-occurrence with visually discriminative words

| | car | truck | bird | cassowary |
|---|---|---|---|---|
| wheel | $1.3 \times 10^{-4}$ | $1.6 \times 10^{-4}$ | $2.0 \times 10^{-6}$ | 0.0 |
| drive | $1.0 \times 10^{-3}$ | $1.0 \times 10^{-3}$ | $2.2 \times 10^{-5}$ | $2.4 \times 10^{-3}$ |
| wings | $4.0 \times 10^{-6}$ | 0.0 | $1.6 \times 10^{-4}$ | 0.0 |
| beak | 0.0 | 0.0 | $5.2 \times 10^{-5}$ | 0.0 |
| occ. | $4.7 \times 10^{5}$ | $7.8 \times 10^{4}$ | $1.4 \times 10^{5}$ | $7.9 \times 10^{2}$ |

Statistics presented in this table were gathered from the English Wikipedia corpus with a context window size of 5 words. Columns correspond to visual class lemmas and rows correspond to visually discriminative words. The last row shows the number of occurrence of the visual class lemmas in the corpus. Upper rows show the frequency of occurrence of visually discriminative words within the context of visual class lemmas. For example, the upper left value denotes $p(wheel|car)$.

### 4.4.3 Limitations

Consider describing a small set of naturally occurring generic objects such as the "car", "truck" and "bird" classes presented in Table 1. Coarse-grain visual classes seem to be well defined by such common words, for which rich co-occurrence statistics can be easily collected from large text corpora. However, humans can identify categories well beyond the limited scope of such coarse-grain visual classes. As one considers larger visual class sets of finer grain, several complications arise:

**N-grams.** Different from coarse grain concepts, fine-grain concepts are often not best described by single words but by composition of words (e.g. n-grams such as "polar bear" or "blue jeans" vs. their unigram parent class "bear" and "trousers"). We found that 54.2% of ImageNet class lemmas are not single words but n-grams. N-grams representations can be computed (e.g., by averaging of their individual words) but we question whether n-gram embeddings can be as visually discriminative as single word embeddings. In section 7.2, we investigate the impact of n-gram composition on ZSL accuracy.

**Rare words.** We found that fine-grain visual classes that are correctly defined by a single word tend to be defined by rare words (e.g. the rare lemma "cassowary" vs. the common lemma "bird" of its parent class). As discussed in the previous section, word embeddings are learned from their co-occurrence statistics in large text corpora. While visual clues are embedded in words co-occurrence patterns, a considerable amount of noise (i.e. non visually discriminative information) stems from random word co-occurrences. The example of Cassowary is illustrated in Table 1. The word "cassowary" only occurs 792 times in the English Wikipedia corpus in which it does not co-occur once with the visual bird-like attributes "wings" or "beak". Instead, "cassowary" randomly co-occurs once with the word "drive". We conjecture that frequently occurring words provide more visually discriminative representations than rare words because of the higher "visual signal to noise ratio" of their co-occurrence statistics. We found that 9.7% of ImageNet lemmas appear less than 10 times in Wikipedia. Figure 4a shows the occurrence count distribution of ImageNet lemmas in the English Wikipedia corpus and, in Section 8.2, we investigate the impact of lemma occurrence frequency on ZSL accuracy.

**Homonyms.** Natural languages contain many homonyms which makes it difficult to uniquely identify visual classes with a single word. For example, a "(river) bank" and a "(financial) bank" share similar representations in a word embedding space while being two different visual concepts. The consequences of homonymy are two folds: first, the semantic representation of homonym classes is learned from the co-occurrence statistics of the different meanings of the lemma which results in noisy embeddings. Second, a mechanism to break ties between homonym visual classes must be given. We found that 13% of the ImageNet lemmas are shared with at least one other class and 38% of ImageNet classes share a lexical form with at least one other class. A rigorous evaluation of the impact of homonymy on ZSL accuracy involves evaluating different heuristics to break ties between homonym classes which is beyond the scope of this work so we do not evaluate the impact of homonymy as we only mention it for completeness.

**Table 4.2:** Comparison of knowledge base statistics

| | Documents | | Graphs | | |
|---|---|---|---|---|---|
| | doc | w/doc | nodes | edges | triples |
| Wordnet | 117k | 10 | 117k | 20 | 372k |
| Babelnet | - | - | 15M | 2.3k | 1.3G |
| Wikipedia | 5.6M | 630 | - | - | - |

(left) Number of document and average size (word per documents) of Wordnet definitions vs. Wikipedia articles. (right) number of nodes, edge types and triples of Wordnet vs. Babelnet knowledge graphs

## 4.5 Data Augmentation

The limitations of word-level representations highlighted in the previous section motivate us to investigate graph- and document-level descriptions. In this section, we focus on the automated acquisition of raw descriptions $y$ to augment the Wordnet definitions and knowledge graph.

An interesting feature of WordNet is its integration to the Linked Open Data (LOD) cloud. Linked Data [59] refers to a set of best practices for publishers to integrate their data into a web of data; i.e, the semantic web. The LOD cloud references openly published datasets that follow the Linked Data best practices. Datasets of the LOD cloud contain links from their resources to resources of other LOD datasets. These links define equivalences between Wordnet concepts and resources of larger, richer knowledge bases. In particular, Wordnet concepts have been fully mapped to entities of the Babelnet [60] knowledge graph. Babelnet entities are also linked to external knowledge bases such as DBPedia or Freebase. Following the links of the LOD cloud, as illustrated in Figure 3, we are able to collect descriptions of ImageNet classes orders of magnitude larger than the graph and document descriptions provided by Wordnet in a fully automated process. In our experiments, we used the Babelnet knowledge graph as augmented graph-level descriptions and Wikipedia articles as augmented document-level descriptions. Table 2 summarizes statistics of these datasets to illustrate the scale of this data augmentation.

## 4.6 Graph Embeddings

A knowledge graph can be formalized as a set of facts $\mathcal{G} = \{(s, p, o) \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}\}$. Each fact in the graph consists of a predicate (edge) $p \in \mathcal{R}$ and two entities (nodes) $s, o \in \mathcal{E} \times \mathcal{E}$, respectively referred to as the subject and object of the triple. Each triple denotes a relationship of type $p$ between the subject $s$ and the object $o$. Learning distributed representations of knowledge graph nodes and edges has been extensively studied in the framework of Statistical Relational Learning (SRL) [61]. SRL models are concerned with knowledge base completion tasks in which models aim to recover missing facts from large and incomplete knowledge graphs. In the following subsection, we review a subset of the literature on knowledge graph embeddings and, in Section 8.3, we evaluate the representations learned by these models on our ZSL benchmark. We refer the reader to [62] for more in-depth coverage of state-of-the-art models.

In addition to knowledge graph embeddings we consider methods from network embeddings; i.e., embeddings of non-typed edge graphs. Recently, two concurrent works ([63], [49]) have shown the benefits of hyperbolic space properties for embedding tree-like hierarchical data. In particular, [49] has shown impressively low reconstruction errors of hyperbolic embeddings of the Wordnet hierarchy. Follow-up work [64] have shown that the Wordnet hierarchy can be embedded perfectly (i.e. with zero reconstruction error) in a two-dimensional Poincarre disk. Because of their impressive success, we include the model introduced in [49] in our evaluation in section 8.3. We briefly present this work in Section 6.2. At the time of this writing, hyperbolic embedding models have not yet been extended to graph structures with typed edges like knowledge graphs. Hence, we apply [49] to the Wordnet hierarchy (the subset of $\mathcal{G}$ considering only the Hypernym-Hyponym predicates).

### 4.6.1 Knowledge graph embeddings

Knowledge graph embedding models include tensor decomposition and neural embedding models. In [65], the authors show that simple neural embedding baselines such as DistMult [66] tend to outperform more sophisticated approaches on several benchmark knowledge base completion tasks, which leads us to focus on

**Table 4.3:** Neural embedding scoring and loss functions

| Model | $\psi(e_s, r_p, e_o)$ | $\mathcal{L}$ |
|---|---|---|
| *TransE* [67] | $\|e_s + r_p - e_o\|$ | $L_2$ |
| *DistMult* [66] | $\langle e_s, r_p, e_o \rangle$ | *Ranking* |
| *ConvE* [68] | $f(vec(f([e_s; r_p] * w))W)e_o$ | *BCE* |
| $TransE^*$ | $\langle e_s + r_p, e_o \rangle$ | *Triplet* |

baseline neural embedding models. Neural embedding models learn $d$-dimensional vector representations of entities $\{e_i \in \mathbb{R}^d, \forall i \in \mathcal{E}\}$ and relations $\{r_i \in \mathbb{R}^d, \forall i \in \mathcal{R}\}$ by maximizing a scoring function $\psi(e_s, r_p, e_o)$ for triples $(s, p, o) \in G$. Learning is performed stochastically by minimizing a loss function $\mathcal{L}$ over the score of randomly sampled triples:

$$e^*, r^* = argmin\Big(\mathbb{E}_{(s,p,o)\in\mathcal{G}}\mathcal{L}\left(\psi(e_s, r_p, e_o)\right)\Big) \tag{4.4}$$

Different embedding models differ in their choice of scoring function $\psi(e_s, r_p, e_o)$ and loss function $\mathcal{L}$ used for training. Table 3 summarizes the scoring function of popular models we evaluate in section 8.3. TransE [67] proposes to model relations $r$ as translations in a Euclidean space, and considers the standard euclidean distance between translated subject and object embeddings as scoring function. DistMult [66] uses the trilinear dot product $\langle x, y, z\rangle = \sum_i x_i y_i z_i$ scoring function with a margin-based ranking loss function. ConvE [68] introduces depth in the scoring function. Their model consists of a convolution layer over the concatenation $[e_s, r_r]$ of the subject and predicate representations followed by a linear layer with ReLu activations. They use the dot product between the output of the network and the object embedding $e_o$ as similarity score and the Binary Cross Entropy as loss function. While experimenting with these models, we found that representing relations as translations (similar to TransE) with a sigmoid dot product similarity score (similar to DistMult) yield highest accuracy. We also found the triplet margin loss to improve the quality of our embeddings. This variation is shown as $TransE^*$ in Table 3 and evaluated in section 8.3 together with other baselines.

### 4.6.2 Hyperbolic taxonomy embedding

The work of [49] proposed an original method called Poincarre embeddings to learn continuous embedding of symbols organized in a latent hierarchy. Instead of considering distances in a Euclidean space, the Poincarre model embeds symbols in a hyperbolic space in which the distance from data point $u$ to $v$ can be expressed as:

$$d(u,v) = arcosh\Big(1 + 2\frac{\|u-v\|^2}{(1-\|u\|^2)(1-\|v\|^2)}\Big) \tag{4.5}$$

Given a set of untyped edges $\mathcal{D} = \{(u,v)\}$, embeddings are learned by stochastic gradient descent so as to minimize the following loss function:

$$\mathcal{L}(\mathcal{D}) = \sum_{(u,v)\in\mathcal{D}} log \frac{e^{(-d(u,v))}}{\sum_{v'\in\mathcal{N}(u)} e^{-d(u,v')}} \tag{4.6}$$

Where $v' \in \mathcal{N}(u)$ represents a set of negative sample nodes $v'$ that are not connected by an edge to $u$. The representations learned by this model have shown to better capture the hierarchical structure of taxonomies.

## 4.7 Document Embeddings

In this section, we review the literature for learning embeddings from text documents. Different models are concerned with documents of different scale, so we separately present embedding models for short, sentence-like documents (i.e. Wordnet definitions) and models concerned with full-text documents (i.e. Wikipedia articles). Sentence-level embeddings have been extensively studied for NLP applications, such as natural language inference or sentiment analysis, whereas full-text document embeddings have mainly been studied for information retrieval applications.

### 4.7.1 Sentence level embedding

The spectrum of different meanings that can be expressed by sentences is combinatorially more complex than the spectrum of meaning covered by individual words so that there is no agreed-upon universal sentence embedding model as there are for word embeddings. Instead, different models have been shown to be better suited to different NLP tasks.

Learning sentence embedding has been a very active topic of research over the last few years which has lead to a variety of architectures and training procedures. In this section, we discuss most relevant works, that we evaluate in Section 8.3.

Let $S = [w_0, w_1, ..., w_T]$ be a sentence of $T$ words. Sentence embedding models produce a fixed-length representation $f(S)$ from variable-length sequences of words $S$. Different models differ in the choice of architecture and training signal used by the model.

#### 4.7.1.1 Architectures

Most state-of-the-art models use either bag of words (BoW) or recurrent (Rec) architectures. BoW architectures represent sentences as the mean of transformations $g(w_i)$ applied to their individual words $w_i \in S$.

$$BoW(S) = \frac{1}{T}\sum_{i=1}^{T} g(w_i) \tag{4.7}$$

The BoW models we investigate use either the identity function $g(x) = x$ or linear projections $g_W(x) = Wx$. We respectively refer to these models as $BoW_{id}$ and $BoW_{lin}$. Recurrent architectures sequentially process the words of a sentence by maintaining an internal state vector $s_t$ at each step $t$ of the processing, following equations (5). The exact formulation of functions $g$ and $f$ depends on the particular architecture used (LSTM, GRU or RNN).

$$s_t = g(s_{t-1}, w_t) \tag{4.8a}$$

$$o_t = f(s_t, w_t) \tag{4.8b}$$

$$Rec(S) = o_T = f(s_T, w_T) \tag{4.8c}$$

#### 4.7.1.2 Training signals

Sentence embedding models can either be trained on supervised or unsupervised tasks. Supervised models are trained on corpora of labeled sentences, and use sentence labels as training signal. Unsupervised models either use the context information of neighboring sentences (inter-sentence objectives) in a corpus of ordered sentences as training signal or only use the information of their own words (intra-sentence objectives).

**Supervised objectives.** In [69], the authors argue that models trained on the task of natural language inference (NLI) learn universal representations of sentences that generalize well to other NLP tasks. Their model, InferSent, uses recurrent architectures trained on the MultiNLI dataset.

DictRep [70] is trained to map dictionary definitions of words to their word embeddings. They investigate both $BoW_{lin}$ and $LSTM$ architectures.

Using similar architectures, the same work proposes CaptRep to learn visually grounded sentence representations by mapping image captions to image features. They use the MS-COCO dataset and extract visual features using a CNN.

**Unsupervised inter-sentence objectives.** The SkipThought model [71] uses a $LSTM$ sequence-to-sequence architecture. Given a corpus of ordered sentences $[S_0, S_1, ..., S_N]$, the model is trained to predict neighboring sentences $S_{i-1}$, $S_{i+1}$ from a sentence $S_i$.

The FastSent model [72] uses adjacent sentences as prediction target similar to SkipThought. Unlike SkipThought, they use a $BoW_{id}$ model with a log bilinear objective function to speed up the training process.

**Unsupervised intra-sentence objectives.** The Paragraph2vec model [73] extends the original word2vec model to sentences and document. To do so, [73] proposes to jointly learn sentence (or paragraph) representations as context words together with the word embeddings.

The Sent2vec [74] model proposes a different extension of the word2vec model: different from Paragrah2vec, the Sent2vec does not explicitly learn sentence representations. Instead, they model sentences with the $BoW_{id}$ architecture. Sent2vec is trained to predict each individual word of a sentence given the BoW representation of the other words of the sentence.

Similar to SkipThought, Sequential Denoising Autoencoder (SDAE, [72]) uses a recurrent sequence-to-sequence architecture. Different from Skipthough, SDAE injects noise in the input sentences and uses the reconstruction loss of the output as training signal.

### 4.7.2 Full document embedding

Embedding full-length document has mainly been studied for document retrieval applications for which two of the most popular methods are Latent Semantic Indexing [75] and Latent Dirichlet Allocation [76]. Latent Semantic Indexing performs singular value decomposition on a matrix of term/document occurrence matrix. The TF-IDF model was introduced as a weighting factor to reduce the impact of frequently occurring words and has been shown to improve search results on document retrieval tasks. In section 8.3, we evaluate the TF-IDF representations of Wikipedia articles on our ZSL benchmark.

In addition to image retrieval, the processing of full-text documents has recently attracted the attention of the machine learning community for problems including abstractive text summarization [77] and open question answering [78]. These models typically use attention mechanisms to attend to specific segments of the document relevant to the task at hand. However, these models are not concerned with extracting fixed-length representations of the document content so that their integration to ZSL models is not straightforward. Integration of such models to the ZSL pipeline using the Wikipedia articles we provide represents one interesting direction for future research.

## 4.8 Experiments & Results

### 4.8.1 Methodology

In the following experiments, we used the 1,000 classes of the ILSVRC2012 image classification dataset as a training set. We use a ResNet-50 [79] as the visual module to extract 2048-dimension feature vectors from raw images. We use the ESZSL model proposed in [14] as the core ZSL module. We evaluate the classification accuracy of our model using different semantic modules on different test splits as described in the following subsections.

When openly available, we always prefer the reference implementation of the semantic modules we evaluate. We re-implement the models for which no open implementation has been released. All implementations are made available on the github page of the project[1]. Detailed hyper-parameters and training settings are also accessible on the project page. Experiments with word embeddings were performed using the pretrained vectors as released by the original papers.

### 4.8.2 Word embedding limitations

In this section, we quantify the limitations of word embeddings discussed in Section 4. Results presented in this section were obtained with pretrained FastText embeddings as we found FastText vectors to provide a better coverage of ImageNet lemmas than GloVe and word2vec (very rare lemmas are often missing from pretrained vectors).

**Rare words.** In Section 4, we conjecture that frequent words, learned from dense co-occurrence statistics, provide more discriminative embeddings than rare words, learned from scarce co-occurrence statistics. Figure 4a shows the distribution of occurrence counts of the visual class lemmas in the full Wikipedia English corpus. We split the ImageNet dataset class-wise into 5 artificial subsets. Each split $k$ contains visual classes whose lemmas appear between $10^k$ and $10^{k+1}$ times in the English Wikipedia corpus. We evaluate each split individually. For each split, we randomly sample 200 classes, and evaluate the accuracy of our model on a 200 class classification problem. As different randomly sampled test class sets yield different results, we repeat this operation 20 times per split with different randomly sampled test class sets, and we present the mean, first quartiles, and extrema of each split's top-1 accuracy in Figure 4b.

Our results highlight a strong correlation between lemmas frequency and classification accuracy. The first two splits (rare words) strikingly under-perform mid-frequency terms with 6% and 7% mean accuracy compared to the 14% accuracy of the best performing splits. On the other hand, we found that very frequent words also show lower classification accuracy, which was unexpected.

**N-grams.** In this experiment, we evaluate the impact of n-grams on the classification accuracy of our model. Figure 5a shows the distribution of ImageNet lemmas as n-grams and unigrams. We split the ImageNet dataset into unigram and n-gram lemmas class sets. N-gram representations were computed as the

[1]https://github.com/TristHas/ZSL-semantics

mean of their individual word embeddings. As lemma scarcity might be correlated to their being n-gram or unigram, we only used classes whose lemmas appear between 1000 and 100,000 times within the English Wikipedia corpus to reduce the influence of lemma scarcity. Figure 5b shows the top-1 accuracy of each split, following the evaluation protocol described for the lemma scarcity evaluation.

Surprisingly, ZSL accuracy does not seem to suffer from the "n-gram-ness" of visual class lemmas as n-gram lemmas even outperform single word lemmas by an average of 2%.

### 4.8.3 Standard evaluation

Table 4 presents the results of our evaluation on standard test splits used in previous works [1, 7, 12, 15, 46]. The hop-2 split consists of the $1,509$ classes within two hops of the training classes in the Wordnet hierarchy. The hop-3 split consists of the $7,678$ classes within 3 hops and the "all" dataset considers the full set of $20K$ classes.

**General Observations.** The bottom section of the table presents the state-of-the art results obtained using word2vec embeddings as reported in [1]. Note that the result they report for the ESZSL model using word2vec semantic vectors slightly outperform ours, which might be due to the different visual features we use. This indicates that the higher accuracies can not be attribute to our use of better visual features but are the results of more discriminative semantic features.

**Word embeddings.** Table 4 highlights striking differences in performance between the word2vec embeddings used in previous works ([1, 7, 12, 15, 46]) and both GloVe and FastText embeddings. Using GloVe embeddings with the ESZSL model (10.96% top-1 accuracy on hop-2 split), we are able to improve on the state-of-the-art (SYNC, 9.26%) by an absolute 1.70% on the top-1 accuracy of the hop-2 split.

**Sentence embeddings.** We expected sentence embedding to provide strong representations as they contain explicit references to fine-grain visual features of visual classes as illustrated by the definition of Cassowary given in Figure 2. Despite being a very active research topics, sentence embedding models performed surprisingly poorly on the hop-2 test split. We observe that models trained with supervised training signals tend to perform better than modes trained in an unsupervised manner. Among unsupervised models, models trained with intra-sentence signals like Sent2vec seem to perform better than SkipThought or FastSent that use inter-sentence training signal. It is worth noting that, despite their relatively low accuracy compared to the best performing graph and word embeddings, the InferSent model still outperform the word2vec embeddings used in previous works. Furthermore, on the more challenging hop-3 and "all" splits, InferSent tend to outperform graph embeddings.

**Graph embeddings.** Graph embeddings performed remarkably well on the hop-2 dataset. In particular, the TransE model with our proposed modifications outperformed the best performing word embedding model by an absolute 2.5% in top-1 accuracy (13.49% vs. 10.96%).

The Poincarre embeddings are learned from the Wordnet hierarchy alone so that they do not embed explicit visual information such as object part attributes. It is remarkable that such embeddings performed on par with the best performing word embeddings.

ConvE embeddings performed poorly in contrast. This result seems to make sense: the ConvE model uses a non-linear similarity function which is not designed to learn linearly separable embeddings whereas the ESZSL module [14] we used as ZSL module uses a regularized bilinear form as similarity measure between visual and semantic features.

The accuracy of graph embeddings tends to decrease dramatically with larger test splits. Visual classes of the hop-3 and all test splits have more distant shortest path to the training classes in the Wordnet knowledge graph. Hence, the graph embeddings of training and test classes within these split are likely to share less information which might explain this result.

**Data augmentation.** Wikipedia documents performed better than sentence embeddings which is encouraging. However, one major limitation of this approach is that many links between Wordnet concepts and Wikipedia articles could not be recovered from LOD data. Only 60% of visual classes were successfully matched to a Wikipedia article. Hence, we manually recovered the missing links for the hop-2 set (623 missing classes), but we did not recover the missing links for classes of larger test splits as manual linking is very time consuming.

Surprisingly, augmenting the Wordnet knowledge graph with Babelnet did not improve on the model's accuracy. The Babelnet knowledge graph is orders of magnitude larger than both Wordnet and standard SRL benchmarks against which knowledge graph embedding models are usually evaluated. This leads us to believe that models of greater capacity than the linear baselines we evaluated might be needed to extract more discriminative representations from such large knowledge bases.

## 4.9 Conclusion & Discussion

In this paper, we stressed out the importance of the semantic module on the accuracy of ZSL models. In particular, we showed that using better word embeddings with a simple baseline model is enough to outperform the state-of-the-art accuracy achieved by more sophisticated models.

More importantly, we discussed the relevance of word embeddings for ZSL and highlighted some limitations that lead us to believe that other levels of descriptions might be needed to achieve zero-shot recognition of generic objects.

We argued that, in addition to words, visual classes can be defined by text documents in natural language, which we referred to as document level descriptions, or by structured data as made available by large knowledge graphs, which we referred to as graph-level descriptions.

We showed that rich descriptions of visual concepts can be automatically scrapped from the web using linked open data and made this data openly available. We then reviewed the literature concerned with learning representations from such data and found that representations learned from explicit descriptions as made available by documents and knowledge graphs can indeed further outperform the best performing word embeddings on some zero-shot object recognition benchmark.

Architectures for the processing of graphs and documents are being actively researched. As research on graph and document embedding gains momentum, and given the limitations of word-level descriptions we outlined in Section 4, we expect semantic modules defined at the document and graph level to prove increasingly more efficient and we hope that the data we provide can prove useful for future research.

Our work opens the way for interesting future research directions: In this paper, we restricted our investigation to semantic modules learned independently from the ZSL model. Extending the models we presented to a visually supervised setting represents one interesting direction.

Another interesting direction is to efficiently combine the semantic representations learned from different levels of descriptions. In particular, it would be interesting to combine the information of structured knowledge bases with that of word embeddings as both representations perform remarkably well while being of very different nature.

**Table 4.4:** Results on the standard ImageNet ESZSL test splits.

| | | | | 2-hop | | | 3-hop | |
|---|---|---|---|---|---|---|---|---|
| ZSL module | description | semantic module | top-1 | top-5 | top-10 | top-1 | top-5 | top-1 |
| ESZSL | Wordnet lemmas | *word2vec* | 5.84 | 18.57 | 27.70 | 1.45 | 5.04 | 8.08 |
| | | *FastText* | 9.29 | 27.15 | 37.41 | 1.86 | 6.13 | 9.82 |
| | | *GloVe* | 10.96 | 30.51 | 41.25 | **2.31** | **7.73** | **12.2** |
| ESZSL | Wordnet hierarchy | *Poincarre* | 10.99 | 32.34 | 44.17 | 1.19 | 4.51 | 7.70 |
| ESZSL | Wordnet graph | *TransE* | 5.58 | 12.37 | 15.76 | 0.35 | 1.40 | 2.07 |
| | | *DistMult* | 10.84 | 38.02 | 49.57 | 1.34 | 4.89 | 7.92 |
| | | *ConvE* | 4.23 | 10.12 | 13.46 | 0.15 | 0.43 | 0.67 |
| | | *TransE*$^*$ | **13.49** | **40.51** | **51.99** | 1.67 | 6.29 | 9.89 |
| ESZSL | Babelnet graph | *TransE* | 3.71 | 11.08 | 14.61 | 0.10 | 0.32 | 0.51 |
| | | *DistMult* | 5.96 | 17.40 | 25.46 | 1.00 | 3.79 | 6.36 |
| | | *TransE*$^*$ | 11.31 | 31.12 | 42.63 | 2.13 | 6.68 | 10.3 |
| ESZSL | Wordnet definitions | *InferSent* | 6.68 | 21.42 | 32.25 | 2.10 | 7.07 | 11.4 |
| | | *DictRep* | 6.44 | 22.17 | 33.40 | 1.71 | 6.53 | 10.7 |
| | | *CapRep* | 4.16 | 14.39 | 23.02 | 0.97 | 3.85 | 6.62 |
| | | *Sent2vec* | 4.09 | 15.47 | 24.75 | 1.24 | 4.79 | 8.01 |
| | | *SDAE* | 4.15 | 13.97 | 21.23 | 1.41 | 4.80 | 7.56 |
| | | *SkipThought* | 1.72 | 4.02 | 5.65 | 0.48 | 1.57 | 2.35 |
| | | *FastSent* | 1.90 | 9.81 | 15.65 | 0.70 | 3.91 | 6.82 |
| ESZSL | Wikipedia articles | *TFIDF* | 9.03 | 26.53 | 37.31 | - | - | - |
| | | | State-of-the-art | | | | | |
| SYNC [15] | Wordnet lemmas | *word2vec* | **9.26** | - | - | **2.29** | - | - |
| CONSE [12] | | | 7.63 | | | 2.18 | | |
| ESZSL [14] | | | 6.35 | | | 1.51 | | |
| ALE [55] | | | 5.38 | | | 1.32 | | |
| LATEM [13] | | | 5.45 | | | 1.32 | | |
| SJE [54] | | | 5.31 | | | 1.33 | | |
| DEVISE[7] | | | 5.25 | | | 1.29 | | |
| CMT [21] | | | 2.88 | | | 0.67 | | |

The upper part of the table shows our results using the ESZSL model with different semantic representations. The bottom part of the table shows state-of-the-art results as reported in [1]

# Chapter 5

# Conclusions

In []

# References

[1] Yongqin Xian, Bernt Schiele, and Zeynep Akata. Zero-shot learning-the good, the bad and the ugly. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4582–4591, 2017. ix, 5, 6, 17, 18, 21, 52, 68, 72

[2] Michael Kampffmeyer, Yinbo Chen, Xiaodan Liang, Hao Wang, Yujia Zhang, and Eric P Xing. Rethinking knowledge graph propagation for zero-shot learning. *arXiv preprint arXiv:1805.11724*, 2018. ix, 5, 7, 17, 18, 21

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009. 3, 5

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 3

[5] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 3

[6] Hugo Larochelle, Dumitru Erhan, and Yoshua Bengio. Zero-data learning of new tasks. In *AAAI*, volume 1, page 3, 2008. 4

[7] Andrea Frome, Greg S Corrado, Jon Shlens, Samy Bengio, Jeff Dean, Tomas Mikolov, et al. Devise: A deep visual-semantic embedding model. In *Advances in neural information processing systems*, pages 2121–2129, 2013. 4, 5, 18, 20, 21, 52, 68, 72

[8] Christoph H Lampert, Hannes Nickisch, and Stefan Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *Computer Vision and*

*Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 951–958. IEEE, 2009. 4, 51

[9] Genevieve Patterson and James Hays. Sun attribute database: Discovering, annotating, and recognizing scene attributes. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2751–2758. IEEE, 2012. 4

[10] Peter Welinder, Steve Branson, Takeshi Mita, Catherine Wah, Florian Schroff, Serge Belongie, and Pietro Perona. Caltech-ucsd birds 200. 2010. 4

[11] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995. 5

[12] Mohammad Norouzi, Tomas Mikolov, Samy Bengio, Yoram Singer, Jonathon Shlens, Andrea Frome, Greg S Corrado, and Jeffrey Dean. Zero-shot learning by convex combination of semantic embeddings. 5, 18, 21, 52, 68, 72

[13] Yongqin Xian, Zeynep Akata, Gaurav Sharma, Quynh Nguyen, Matthias Hein, and Bernt Schiele. Latent embeddings for zero-shot classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 69–77, 2016. 5, 7, 12, 18, 54, 72

[14] Bernardino Romera-Paredes and Philip Torr. An embarrassingly simple approach to zero-shot learning. In *International Conference on Machine Learning*, pages 2152–2161, 2015. 5, 7, 17, 18, 21, 52, 66, 69, 72

[15] Soravit Changpinyo, Wei-Lun Chao, Boqing Gong, and Fei Sha. Synthesized classifiers for zero-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5327–5336, 2016. 5, 18, 52, 68, 72

[16] Xiaolong Wang, Yufei Ye, and Abhinav Gupta. Zero-shot recognition via semantic embeddings and knowledge graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6857–6866, 2018. 5, 7, 18, 21

[17] Antonio Torralba and Alexei A Efros. Unbiased look at dataset bias. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1521–1528. IEEE, 2011. 5, 16

[18] Allan Jabri, Armand Joulin, and Laurens van der Maaten. Revisiting visual question answering baselines. In *European conference on computer vision*, pages 727–739. Springer, 2016. 6

[19] Kushal Kafle and Christopher Kanan. An analysis of visual question answering algorithms. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 1983–1991. IEEE, 2017. 6

[20] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 1988–1997. IEEE, 2017. 6

[21] Richard Socher, Milind Ganjoo, Christopher D Manning, and Andrew Ng. Zero-shot learning through cross-modal transfer. In *Advances in neural information processing systems*, pages 935–943, 2013. 18, 72

[22] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018. 23, 28

[23] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016. 23, 27

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 23

[25] Christopher J Shallue, Jaehoon Lee, Joe Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E Dahl. Measuring the effects of data parallelism on neural network training. *arXiv preprint arXiv:1811.03600*, 2018. 24

[26] Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018. 24, 25

[27] Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. In *Advances in Neural Information Processing Systems*, pages 2214–2224, 2017. 25, 26, 43

[28] Jörn-Henrik Jacobsen, Arnold Smeulders, and Edouard Oyallon. i-revnet: Deep invertible networks. *arXiv preprint arXiv:1802.07088*, 2018. 25, 26, 43

[29] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014. 26

[30] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016. 26

[31] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10215–10224, 2018. 26

[32] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015. 26

[33] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6571–6583, 2018. 26

[34] Jörn-Henrik Jacobsen, Jens Behrmann, Richard Zemel, and Matthias Bethge. Excessive invariance causes adversarial vulnerability. *arXiv preprint arXiv:1811.00401*, 2018. 26

[35] Jens Behrmann, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. *arXiv preprint arXiv:1811.00995*, 2018. 27

[36] Samuel Rota Bulò, Lorenzo Porzi, and Peter Kontschieder. In-place activated batchnorm for memory-optimized training of dnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5639–5647, 2018. 27

[37] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016. 27

[38] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 27

[39] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018. 27

[40] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017. 28

[41] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. Training and inference with integers in deep neural networks. *arXiv preprint arXiv:1802.04680*, 2018. 28

[42] James Martens and Ilya Sutskever. Training deep and recurrent networks with hessian-free optimization. In *Neural networks: Tricks of the trade*, pages 479–535. Springer, 2012. 28

[43] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016. 28

[44] Leslie N Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. *arXiv preprint arXiv:1708.07120*, 2017. 48

[45] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011. 51

[46] Elyor Kodirov, Tao Xiang, and Shaogang Gong. Semantic autoencoder for zero-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3174–3183, 2017. 52, 68

[47] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014. 52, 57

[48] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013. 52, 57

[49] Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. In *Advances in Neural Information Processing Systems*, pages 6341–6350, 2017. 53, 61, 63

[50] Marcus Rohrbach, Michael Stark, György Szarvas, Iryna Gurevych, and Bernt Schiele. What helps where–and why? semantic relatedness for knowledge transfer. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 910–917. IEEE, 2010. 54

[51] Thomas Mensink, Efstratios Gavves, and Cees GM Snoek. Costa: Co-occurrence statistics for zero-shot classification. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 2441–2448. IEEE, 2014. 54

[52] Tanmoy Mukherjee and Timothy Hospedales. Gaussian visual-linguistic embedding for zero-shot recognition. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 912–918, 2016. 54

[53] Qian Wang and Ke Chen. Alternative semantic representations for zero-shot human action recognition. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 87–102. Springer, 2017. 54

[54] Zeynep Akata, Scott Reed, Daniel Walter, Honglak Lee, and Bernt Schiele. Evaluation of output embeddings for fine-grained image classification. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 2927–2936. IEEE, 2015. 54, 72

[55] Zeynep Akata, Florent Perronnin, Zaid Harchaoui, and Cordelia Schmid. Label-embedding for image classification. *IEEE transactions on pattern analysis and machine intelligence*, 38(7):1425–1438, 2016. 54, 72

[56] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185, 2014. 56, 57

[57] Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 238–247, 2014. 56, 57

[58] Armand Joulin, Edouard Grave, and Piotr Bojanowski Tomas Mikolov. Bag of tricks for efficient text classification. *EACL 2017*, page 427, 2017. 57

[59] Liyang Yu. Linked open data. In *A Developers Guide to the Semantic Web*, pages 409–466. Springer, 2011. 60

[60] Roberto Navigli and Simone Paolo Ponzetto. Babelnet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. *Artificial Intelligence*, 193:217–250, 2012. 60

[61] Lise Getoor and Ben Taskar. *Introduction to statistical relational learning.* MIT press, 2007. 61

[62] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017. 61

[63] Benjamin Paul Chamberlain, James R Clough, and Marc Peter Deisenroth. Neural embeddings of graphs in hyperbolic space. 61

[64] Christopher De Sa, Albert Gu, Christopher Ré, and Frederic Sala. Representation tradeoffs for hyperbolic embeddings. 2018. 61

[65] Rudolf Kadlec, Ondrej Bajgar, and Jan Kleindienst. Knowledge base completion: Baselines strike back. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 69–74, 2017. 61

[66] Bishan Yang, Scott Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. 2015. 61, 62

[67] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795, 2013. 62

[68] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. 2017. 62

[69] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680, 2017. 65

[70] Felix Hill, KyungHyun Cho, Anna Korhonen, and Yoshua Bengio. Learning to understand phrases by embedding the dictionary. *Transactions of the Association for Computational Linguistics*, 4:17–30, 2016. 65

[71] Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302, 2015. 65

[72] Felix Hill, Kyunghyun Cho, and Anna Korhonen. Learning distributed representations of sentences from unlabelled data. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1367–1377, 2016. 65

[73] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196, 2014. 65

[74] Matteo Pagliardini, Prakhar Gupta, and Martin Jaggi. Unsupervised learning of sentence embeddings using compositional n-gram features. Technical report, 2017. 65

[75] Susan T Dumais. Latent semantic analysis. *Annual review of information science and technology*, 38(1):188–230, 2004. 66

[76] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003. 66

[77] Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Caglar Gulcehre, and Bing Xiang. Abstractive text summarization using sequence-to-sequence rnns and beyond. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, 2016. 66

[78] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1870–1879, 2017. 66

[79] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 66

# Appendix

## some section

# Acknowledgements

First, I would like to thank my supervisors, Emeritus Professor Yasuo Ariki and Associate Professor Tetsuya Takiguchi at Kobe University, who have given me helpful advice and continued support during my research and writing up. Their broad knowledge in the field and his down-to-earth attitude has been of great help to my study. I also thank Professor Ohkawa, Professor Tamaki, and Professor Matoba for their constructive comments and valuable suggestions, which helped to improve this thesis.

Meanwhile, I would like to thank the past and the present members in CS 17 Media Lab., where we have done efforts together and shared joys and sorrows of research life.

Finally, to my family and my friends, it cannot be described in words, yet I would like to show my full gratitude for their love, encouragements, and unconditional support throughout my study.

# BibTeX Citation for This Thesis

```
@article   {TristanThesis2019,
             title={{Zero Shot Recognition of Generic Objects
             journal={Doctoral Thesis},
             author={Tristan Hascoet},
             institution={Kobe University},
             month={Oct.},
             year={2019}
           }
```

Doctor Thesis, Kobe University
“Voice Conversion Based on Non-negative Matrix Factorization and Its Application to Practical Tasks”, XXX pages
Submitted on June, 6, 2019.
The date of publication is printed in the cover of repository version published in Kobe University Repository Kernel.