Project Title

Optional Subtitle

Hermanni Hälvä [1]

MSc. Computational Statistics and Machine Learning

Supervisor: Prof. Bradley Love

Submission date: Day Month Year

[1]**Disclaimer:** This report is submitted as part requirement for the MSc CSML degree at UCL. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged

Abstract

Summarise your report concisely.

Acknowledgements

I would like to...

Contents

List of Figures

List of Tables

Chapter 1

Notes

- discuss how first trained on small data set
- one problem is that we are training only transformation to the leafs of the nodes. I wonder if we could includ basic level labels since those are the ones humans learn first and also some suggest on average capture most distinction (see the two papers)
- look at yolo 9000 for wordnet explanation
- Do I clearly state that CNNs are the state-of-the-art on image classification, give some example of current results / super human performance or when they wont competitions first time
- applications of zero-shot learning and better representation. e.g. autonomous vehicles need to make deductions.
- benthos example on wikipedia

Chapter 2

Intro/Literature Review Notes

Humans are able to extract rich semantic information from visual scenes. For instance, upon viewing a picture of a dog, we may also be able to identify it as a specific breed such as a golden retriever. Further, our understanding of the image benefits from semantic knowledge that is not captured explicitly in the visual features of a specific image. For example, we also know that the dog belongs in the category of mammals which, in turn, are animals and thus living entities. While this type of hierarchical semantic visual understanding comes to us effortlessly, it remains a challenging task for computer vision. In fact, most image classification models are trained on data sets with single, mutually exlusive labels and thus the learnt feature representations do not account, for example, for both cat and dog being four-legged animals. Why is this a problem? Generalization? An exception to this is the area of knowledge transfer and related tasks such as zero-shot learning in which the aim is to predict labels of previously unseen classes of images; a popular approach for zero-shot learning is to borrown strength from, say text data, to create a semantic space that embed all the possible labels of images including those not seen previously. Mapping between images and the semantic space is then learned using the 'seen' images. Once this mapping is learnt, it can be used to transform previously unseen images into the semantic space and

then apply some distance measure to label it as the category thats cloest in the embedding space. We postulate that the same idea could also be employed in the simple image classification tasks to achieve better generalization performance. In particular, we will train a deep learning model for image classification against a hierarchical semantic embedding derived from the WordNet database which, as we will show, will correspond to regularizing the model weights to account for these semantic relationships. We will use the recently developed poincare embeddings as the resulting embedding space can capture both the similarity between the possible labels as well as the hierarchical semantic relationships encoded by WordNet graphical model. The learnt model is then transferred to perform standard image classification by adding a final softmax classification layer and fine tuning this agains the ImageNet database. Our results show ... hopefully some improvements over training the model only against ImageNet which illustrates the benefit of incorporatig semantic knowledge..

Despite the many successes of deep learning in various computer vision tasks, most of these rely on well defined training and testing envionments and do not generalize well beyond them; we are thus stil far from general human-level performance.

PAPERS ON SEMANTIC KNOWLEDGE TRANSFER & ZERO-SHOT LEARNING

DeViSE: A Deep Visual-Semantic Embedding Model - Frome et al.

Overview

Typical object classification models treat all the categories unrelated, via N-way softmax. This leads to models that 'cannot transfer semantic information about learned labels to unseen words or phrases. Solution this paper proposes is to use both standard image data and then an unrelated large unannotated text data to learn semantic information. In particular, the model maps image inputs into this rich semantic embedding space. The results show similar performance to standard state-of-the-art DL models, but with a significant improvement in that much less 'semantically unreasonable' mistakes are made. Perhaps more importantly, they show this joint training allows the model to generalize to 20000 visual categories, despite having trained the model on just 1000 categories.

DeViSE extends work of Weston et al. (above) as it allows non-linearities and also capture semantics from text that's not contained in the image lables, hence allowing for zero-shot generalization. It also improves on the Socher et al. paper by using a deep model and avoid the trade-off which that paper has in predicting seen and unseen classes. Socher et al. also combine several different models whilst this one uses a unified model that only uses embeddings. Several other previous papers have used WordNet to build semantic representations; they authors here use a large unannotated text data which they claim is superior.

The modelling approach begins by first training the skip-gram model which predicts the adjacent terms in the text for each word and then creates an embedding on this. This model was trained on 5.4 billion word Wikipedia data set. Image-labels were then mapped into this vector representation. Next, a DNN was used to map images into this embedding space by removing the final softmax layer and instead using a similarity metric. More precisely, a combination of dot-product similarity and hinge rank loss was used (following

Weston et al.); this was found to be better than L2 loss. During model testing, image is projected into the embedding space and the nearest label is found using a hashing technique. The corresponding image-net synset is then found for this embedding.

The model is then used to perform zero-shot learning and compared against few baselines: state-of-the-art DL model and the authors' DeViSE model but using random embeddings rather than learnt word embeddings. The authors claim better results than standard DNN on standard classification task on imagenet, though I am really not convinced by this as the differences are tiny and doubt statistically significant. However, on zero-shot learning, DeViSE does really seem superior.

Thoughts:

- quite similar to our approach as we will also look to map image labels into word embedding space
- we will also follow this by removing the softmax layer from our choice of DNN and instead use some similarity metric
- we need to thus think what will be our final loss we will use, probably try several different
- need to think how to perform testing since predictions just give embedding location. Not sure if the hashing technique is the best way to do this.
- I really dont see why the use of the wikipedia data is superior to wordnet. Since wordnet seems to already correspond to how humans build a taxonomy of visual objects (see earlier 'psychology papers') then surely that would be the preferred approach? In particular, the wikipedia text is clearly larger but will have a lot of noise, and more importantly it may not be predictive of the relationship of how visual features correlate. For instance, the words 'bird' and 'sky' are likely to end up

near each other based on the wikipedia data, but the two dont visually resemble each other so perhaps it's not good for them to be close if the aim is to influence what visual features are to be learnt.

Zero-shot Recognition via Semantic Embeddings and Knowledge Graphs - Wang et al. 2018

Overview In previous literature, zero/few-shot learning has been achieved via knowledge transfer. Two different routes have been used for knowledge transfer. The first is to use implicit knowledge representations in the form of semantic embeddings create from some adjacent text data. Accoding to the authors, the generalization power of semantic models is limited, partly by the mapping models themselves. Further, there is no easy way to learn semantic embeddings from structured information such as knowledge graphs. Indeed, the second approach to zero-shot learning has been to use explicit knowledge transfer of rules and relationships. A simple example given is to learn a separate classifier for different compositional categories of a visual object.

This paper's novel contribution is to use both implicit knowledge representation (i.e. word embeddings) and explicit ones (i.e. knowledge graph) to learn a visual classifier. This is done by constructing a knowledge graph where the nodes of the knowledge graphs are the semantic word embedding representations, and are connected to each other by by edges that represent the relationships between the words. The node semantic embeddings are created using GloVe. Graph Convolution is used to pass messages in the graph between the categories according to the knowledge graphs, which in one of the experiments is just the WordNet sub-graph. Essentially this approach generates a new deep logistic classifier for each object. The visual features are extracted from inception V1/Resnet50 model, depending on the experiment. The results of the paper show very large improvements over DeVISE and other state-of-the-art in zero-shot classification.

Thoughts:

- the authors say its hard to learn semantic embeddings from structured information but actually Poincare embedding should allow this since they precisely try to represent hierarchical data in embedding e.g. wordnet.
- what exactly is wordnet SUB-graph? need to check
- the performance of this model is very impressive; I wonder if it we should also attempt how our model does in zero-shot learning

Chapter 3

Implementation Ideas

- which loss function to compare predicted and ground-truth embedding vectors. Dot product hinge loss as in DeVisE?
- at test time, how to predict labels i.e. to find nearest neighbour? Some tree or hashing e.g. DeViSE
- which baselines to use? Standard inception-v3, random embedding, word2vec embedding,
- what evaluations to make? resonableness of errors
- contrast method vs word2vec method
- is this truly zero-shot learning since we are using knowledge

Chapter 4

Background

4.1 Artificial Neural Networks & Deep Learning

Even though deep learning is often viewed as a new technique, in reality the recent breakthroughs are underpinned by decades, if not centuries of related research. For instance, earliest artificial neural networks, such as Rosenblatt's Perceptron [**?**] from the 1950s, are closely related to linear regressions dating back to Gauss [**?**]; these models are all of the form $f(\mathbf{x}, \mathbf{w}) = \mathbf{x}^t\mathbf{w}$ where $\mathbf{x}$ is a vector describing some input covariate and $\mathbf{w}$ model weights. Much like in linear regression, the aim is to learn a mapping $f(\mathbf{w}, \mathbf{x}) = y$ that defines the relationship between the input $\mathbf{x}$ and some category y it belongs to. In a simple problem, y would be binary and the weights $\mathbf{w}$ would be learnt such that the resulting hyperplane could linearly separate the data into the two classes based on the input features. These early models were largely restricted by their assuming linear separability and there was also no computationally feasible way of training the models at the time [**?**], [**?**] but they form the basis for nearly all Artificial Neural Networks (ANNs).

Several steps were taken to increase to complexity of these models and

to allow for non-linearities, many of them initially inspired to some degree by biological neurons [**?**]. For instance, real neurons typically only fire once action potential exceeds a specific threshold [**?**]. In ANNs, a reminiscent behaviour is attained via activation functions on top of a neuron outputs, most typically in the form of Rectified Linear Units (ReLU), which apply the following non-linearity: $f(\mathbf{x}, \mathbf{w}) = max\{0, \mathbf{x}^t\mathbf{w}\}$. ReLUs also improve the representational capacity of a network by imposing sparsity which can make it easier to disentangle the data [**?**] and hence lead to faster convergence of training [**?**].

Another insights borrowed from neuroscience was that intelligence stems from groups of neurons acting together rather than from the behaviour of individual neurons [**?**]; this idea is behind deep ANNs where several layers of neurons are connected to each other and numerous neurons are present in each layer. Below equations and Figure 4.1 give a simplified example of an ANN with two hidden layers:

$$\mathbf{H_1} = max\{0, \mathbf{WX} + \mathbf{B}\} \tag{4.1}$$

$$\mathbf{H_2} = max\{0, \mathbf{VH_1} + \mathbf{C}\} \tag{4.2}$$

$$\mathbf{F} = \mathbf{ZH_2} + \mathbf{D} \tag{4.3}$$

where $\mathbf{X}$ is an $D \times N$ input data matrix that holds the N different observations in columns and D is the dimension of a single observation, which for image data is often a flattened array of the image pixels. This matrix representation of input allows several datapoints to be fed through the network simultaneously, which is what is done in practice. $\mathbf{W}$, $\mathbf{V}$ and $\mathbf{Z}$ are the weights matrices of the two hidden layers and output layer respectively. The number of rows in each of these matrices is the number of neurons in that layer, whilst the number of columns corresponds to dimensionality of output from the previous layer. Notice also that constant bias matrices $\mathbf{B}$,

C, and **D** are added to the neuron outputs. These bias matrices are akin to intercepts in regression analysis and increase the representation ability of the model. Usually all the columns of the matrices are identical such that the same bias values are added to each input observation. We can see that mathematically these matrix operations are just affine transformations on the input, followed by ReLUs, which are applied elementwise. The resulting output of each neuron is thus a matrix, here **H1** and **H2**, where each row corresponds to an output from a specific neuron, calculated separately in the columns for each input vector. Notice also that ReLUs are not added on the output, rather the output layer transforms the representations of the hidden layer into a desired shape of output. For instance, for binary classification **F** would be of $2 \times N$, and the two output values for each input would reflect the relative likelihood of the two labels.

The ANN model we have described thus far is known as Feedforward Network or a Fully Connected Network, which refers to all neurons being connected to all other neurons in the preceding and succeeding layers. This is an important point as it fascilitates a hierarchical structure in which neurons in the later layers may combine features from earlier layers to create more complex features in turn. Relatedly, this architecture enables distributed representation in which several types features can be combined in different ways to represent an exponential number of different inputs [**?**]. As an example, if we had squares, rectangles and circles and each of them could be either red, green or blue, then we have 9 possible visual objects, yet all the possibilities could efficiently be represented by combining three color neurons with three shape neurons [**?**]. **see goodfellow ch 15**.

4.1.1 Training ANNs

In a typical classification problem we have n possible labels for each input and wish to predict a probability distribution over them. In these applications the outputs of ANN are transformed into 0 to 1 range. More formally, consider

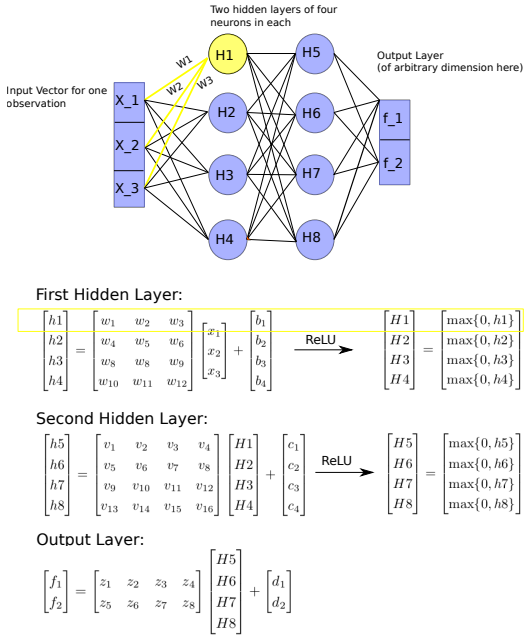

Figure 4.1: A simple Feed Forward ANN with two hidden layers. This figure illustrates a graphical model for Equation 4.1 - 3.3 as well as explicitly showing the matrix operations performed by the different neurons on a single vector input. The output dimension is set arbitraily to be a 2×1 vector, which could for instance be used as class scores in binary classification; in practice, the dimension will be problem dependent.

that the output from above ANN for a single input $\mathbf{x}$ is the $n \times 1$ vector $\mathbf{f}$; we then require, that each element of $\mathbf{f}$ is between 0 and 1 and that $\sum_1^n \mathrm{f}_i = 1$. The most common way of doing this is to assume that the output of the ANN are unnormalized (predicted) class log-probabilities, that is $f_i = \log \hat{P}(y = i|\mathbf{x})$ [**?**]. By taking exponents and normalizing across possible labels predicted class probabilities are calculated as per below - this is known as the softmax function:

$$\text{softmax}(\mathbf{f})_i = \frac{\exp(f_i)}{\sum_j \exp(f_j)} = q_i \tag{4.4}$$

where $q_i \in [0, 1]$ captures the predicted probability that the input into the ANN belongs to class i. One reason for the softmax layer's popularity is that it is easily compatabile with the cross entropy loss function defined as $L_i = -\sum_i p_i \log q_i$ [**?**]. In simple image classification tasks where the classes are mutually exclusive we have $p_i = 0 \forall i \neq C$ and $p_i = 1$ for $i = C$ denoting the correct class. Plugging Equation 4.4 into this equation gives:

$$L_i = -\log\left(\frac{\exp(f_C)}{\sum_j \exp(f_j)}\right) = -f_C + \log\left(\sum_j \exp(f_j)\right) \tag{4.5}$$

which is the loss incurred from one observation, and is clearly continuous and differentiable. The sequence of computation leading from model inputs all the way to the output of scalar loss is known as the forward pass. Usually forward pass is computed simultaneously for several inputs, known as a mini-batch, in which case the average loss across the mini-batch is typically used. The aim of training an ANN is based on learnng model weights that minimize some appropriate loss function, such as the cross-entropy loss above. Most supervised deep learning models, including the basic feedforward-network described above, are nowadays trained using the back-propagation algorithm [**?**] [**?**]. The main idea of this algorithm is to use the chain rule to decompose the gradient of a loss function so that it can be efficiently passed back through the network. More formally, assume the loss of an ANN is produced by a sequence of m nested operation:

$$L(y_i, x_i) = f^{(m)}(y_i, f^{(m-1)}(\dots f^{(2)}(f^{(1)}(x_i)))) \tag{4.6}$$

where y_i is the real label of the observation, x_i the input data, the different $f^{(i)}$ may for example represent different types of layers. Employing the chain rule recursively, a simple decomposition gives:

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial f^{(m)}} \frac{\partial f^{(m)}}{\partial f^{(m-1)}} \cdots \frac{\partial f^{(1)}}{\partial x_i} \tag{4.7}$$

These computations are done in the opposite order of the forward-pass operations; hence back-propagation. Above example is simplistic since usually in addition to inputs from preceding layer, each layer has also its own paramets which can be also be thought as inputs to that layer. The general idea of the chain rule still works in that situation as well, now however the gradient flow bifurcates at each layer; part of gradients flow into tha earlier layer while the others flow back into parameters; this is explained in more depth in Figure 4.2. By representing ANNs as computational graphs, and using the chain rule similar to above, it becomes easy to see how backpropagation can be used to send appropriate gradients to right places even in complex network architectures. Importantly, backpropagation does this efficiently since at each operation all upstream gradients are collated and then passed on to one layer down, which is a lot more efficient than considering every single path through an ANN individually. To see this, consider Figure 4.1: using back-propagation we can start from the output and, for instance, calculate the derivative of the output layer with respect to $H5$ neuron only once, and then pass this derivative to all neurons $H1$ - $H4$ simultaneously, which requires a lot less computation than considering all the paths that involve $H5$ separately.

After gradients are calculated using back-propagation, they are used by an optimization algorithm to change the model's parameters with the aim of minimizing the loss function. Here we consider stochastic gradient descent (SGD) [**?**] which is likely the most widely used optimization algorithm in deep learning. This algorithm is called stochastic because at each learning iteration only a subset, known as mini-batch, of the data is use to calculate gradients and to perform parameter updates; it has been shown that SGD

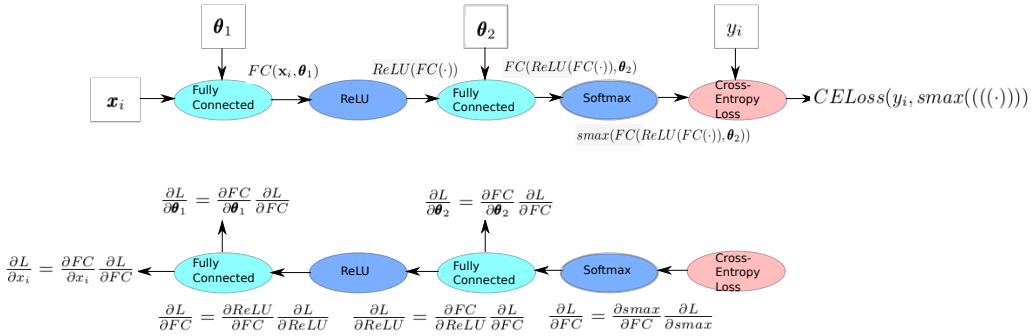

Figure 4.2: A simple example of forward (top) and backward passes (bottom) for a simple two-layer ANN, which illustrates how gradients flowing to early layers are calculated by multiplying the local gradient of a given layer with the gradient that flows back from the next layer

converges much faster than calculating gradients always on all available data; the time per update for the algorithm is independet of size of data as long as batch size is held constant [**?**]. The parameter updates are based on moving 'downhill' i.e. in the direction of negative gradients:

$$\boldsymbol{\theta}_{(t+1)} = \boldsymbol{\theta}_{(t)} - \eta \nabla_{\theta_{(t)}} L(\mathbf{X}, \mathbf{y}) \tag{4.8}$$

Most deep learning models are very sensitive to the choice of learning rate η; if it is too high, we are likely to miss minima and conversly there is a risk of local minima and slow training time when the parameter is set too small. Usually learning rate is reduced linearly with training, or in bigger steps at regular intervals, to reduce the impact of noise when we are near a minimum [**?**].

Another common addition to the vanilla SGD is momentum [**?**], which can accelerate learning when there is a lot of noise from SGD or when the Hessian of the loss (matrix of 2nd order derivatives) is ill-conditioned as shown in Figure 4.3. SGD with momentum ammends the original SGD by introducing a velocity term that accumulates gradients from previous iterations with

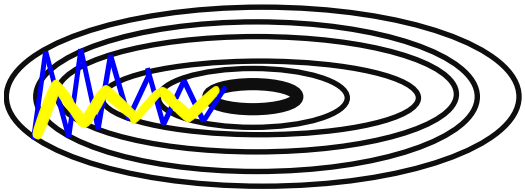

Figure 4.3: Example of ill conditioned Hessian and how momentum speeds up learning by accumulating the gradients of previous iterations such that velocity towards centre (lower loss) is established (shown in yellow). SGD without momentum keeps jumpin across the loss surface as if in a downward sloping canyon, but failing to utilize the slope (blue trace). The contours depict different levels of loss that decreases inwards

an exponential decay. Rumelhart and Hinton [**?**] described momentum as if dropping a ball-bearing on loss surface and letting momentum drive the ball. Further, the loss landscape can be imagined to be immersed in a liquid with a specified level of viscosity that defines how quickly the momentum fades. Algorithm 1 gives an example of full SGD momentum algorithm for learning model parameters. In practice, back-propagation and optimization can nowadays be done automatically by modern deep learning libraries such as PyTorch [**?**] and Tensorflow [**?**].

Algorithm 1 SGD with momentum (following [**?**])

Require: Learning rate η, Momentum decay parameter α
Require: Initial parameters $\boldsymbol{\theta}$, initial velocity ν
  while Convergence not met **do**
    Sample a minibatch of m observations from training data $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ and their
    Get corresponding targets from training data $\{\mathbf{y}_1, \ldots, \mathbf{y}_m\}$
    Compute gradient for the minibatch: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(\mathbf{x}_i|\boldsymbol{\theta}), \mathbf{y}_i)$
    Update velocity: $\boldsymbol{\nu} \leftarrow \alpha\boldsymbol{\nu} - \eta\mathbf{g}$
    Updata parameters: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{\nu}$
  end while

4.1.2 Convolutional Neural Networks (CNNs)

So far we have considered only simple feedforward ANNs with fully connected layers. Most of the ground-breaking accomplishments over the past decade in deep learning have however been achieved with convolutional neural networks (CNNs) [**?**] of some sort. This is particularly true for computer vision tasks such as image classifiction [**?**]. In fact, the term deep learning is often used synonymously with CNNs that have a large number of layers.

Unlike fully connected neurons in feedforward networks, convolutional layer has neurons, usually called kernels, which are connected only to some parts of the input it receives from its preceding layer - together many such neurons span the entire input data. Furthermore, there is weight sharing between the kernels to allow for the recognition of a particular feature anywhere in the input space [**?**]; convolution layers are thus particularly suited for data with locally correlated structures such as images. Consider an input image of 225x225, a typical convolution filter may have size 3x3 and, after having been trained on image data, could have learnt feature mapping that represents particular visual feature such as a vertical edge. This filter is then replicated over the entire 225x225 input range so that the model can detect that particular feature anywhere in the image. Usually each layer has a multitude of such kernels to detect different features in the input data. Mathematically, this feature detection corresponds to the convolution operation between input data X and convolution kernels K, which for discrete 2D data is given as $S(i,j) = (X * K) = (i,j) \sum_m \sum_n X(i-m, j-n)K(m,n)$ [**?**] where i and j denotes a particular image pixel location and m and n those of the kernel. Figure 4.4 gives a brief toy example to illustrate this process.

The reason why convolutions have proved so effective for image data is that natural visual scenes present strong local correlations - the world we view is not just a collection of randomly ordered pixels. Additionally, these visual features can appear at essentially anywhere in our visual scenes; thus the need for convolution layers to scan across the whole image [**?**]. Further,

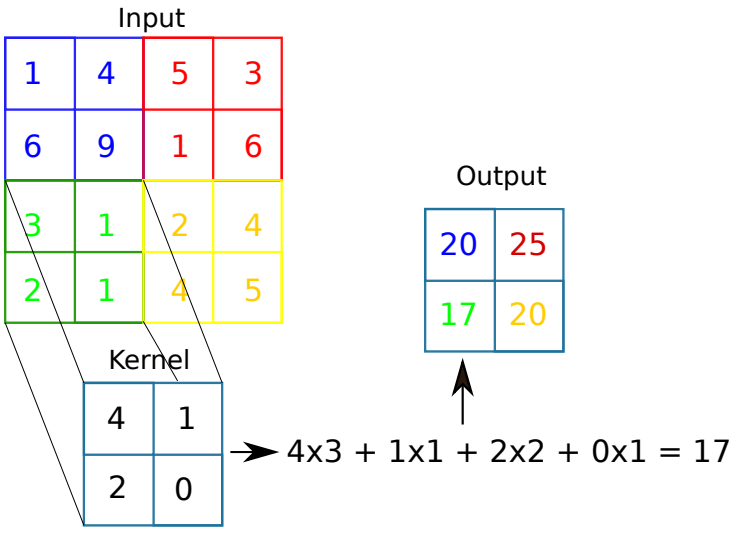

Figure 4.4: A simple example of the computations performed by a single 2x2 convolution kernel on a 4x4 input. Assuming stride length of two squares, the kernel will see each of the four corners and performs a convolution operation on each of them. This is shown explicitly for the bottom left green square. The computation is hence essentially a dot product similarity metric between the kernel and the local image areas.

CNNs usually have several layers of convolutions on top of each other. When such deep CNNs are trained on image data, the kernels across the different layers usually learn a hierarchy of visual features, such that the early layers detect edges and corners, which are then used by middle layers to create contours and parts of objects, and finally later layers build up more complex representations, possibly of complete objects [**?**]. This shares some similarities with mammals' visual cortex [**?**] in which earlier (V1) cells respond to edges and bars of different orientations [**?**] and later ones like V4 and the IT cortext to more complex shapes [**?**]. This idea of distributed hierarchical representations is crucial to our research. First, the abiliy to learn fundamental visual features, such as edges, should help to generalize models to novel visual scenes. Second, generalization is also improved by hierarchy of features: for example, if a CNN only trained on images of cats and apples, is shown a picture of a dog, it would be more likely to classify it as a cat than apple which is arguably the closer of the two. In the next chapter we show how an extension of this idea can be explotited to perform more accurate predictions for previously unseen classes of objects.

Convolution layers in CNNs are normally followed up by max-pooling functions. Usually this is just the $max()$ function applied to a grid output of its preceding convolution output. For example, in Figure 4.4 the max-pooling would only pass through the value of 25. The benefit of such down-samplign is local invariance: visual objects are not completely rigid and by passing on only the largest value, we are likely to produce the same output even if the values of the square changed around a bit [**?**] (imagine a letter that's slightly rotated between different views).

One of the most important features of CNNs is that they can be trained end-to-end using back-propagation and SGD. A consequence of paramount importance is that the type of visual features which convolution kernels learn to represent is fully determined by what best fits the data, and they can be learned directly from raw data. This is in sharp contrast to earlier computer

vision image classification that usually involved feature extraction techniques [?] such as SIFT [?] in which manually designed features are used. These features would then be passed to a separate classifier, for example Support Vector Machine [?]. CNNs with their automatic feature learning capability produced very large accuracy gains on this apprach [?] and following the breakthrough performance of the seminal AlexNet CNN [?] at the ImageNet 2012 competition, deep CNNs of various designs have become state-of-the-art in virtually all machine vision classification and detection tasks [?].

In addition to convolution and pooling layers, CNNs involve several other important architectural designs. For example, deep networks with a large number of convolution and pooling layers have been found to perform better [?] than shallow ones. This is exploited by several CNN models such as the ResNet architectures which can have more than 100 layers [?]. Theoretically, depth increases the representational capacity of the network [?]. Representational capacity is also increased by the use of ReLUs on top of convolution layers. All in all, modern deep CNNs can easily have hundres of millions of parameters and the ability to learn super-human performance on many image classification tasks [?]. Due to their large representational capactiy, these models can also easily overfit training data and the development of appropriate regularization techniques has played a large role in CNNs recent achievements. Perhaps the most popular regularization technique, and the one applied in this work, is DropOut [?]. During model training, dropout deactivates a each neuron in each iteration with a probability P. The consequence of this is that over the entire training period we essentially end up training an ensemble of different models, and the final model is an average of exponentially many sub-models [?]. This method regularizes the network as individual neurons can not rely too much on any other neurons and the output expected to get out of them. Durng test-time, dropout is turned off.

Many of the theoretical concepts of modern deep CNNs have been around for several decades [?]. The reason for their recent surge in popularity is to

a large extent that they have become a lot easier and faster to train as our computers have gotten more powerful and our datasets much larger. In particular, the ability to train deep models on Graphical Processing Units (GPUs) has cut training times by at least an order of magnitude [**?**]. However, we have only been able reap the benefits of faster compute and better models because of 'big data'. It has been suggested that CNNs with around 5000 labels per training category can on average start to match human performance [**?**] - over the past decade we have seen the advent of datasets that have up to tens of millions of training examples [**?**], [**?**], [**?**].

4.2 Exploiting Semantic Information in Computer Vision

4.2.1 Semantic Hierarchies and the Human Visual System

Upon observing almost any visual scene, humans are able to effortlessly cut it up into segments of distinct objects [1], even if we have never seen the scene before. This is a remarkable ability considering that the world we live in can present us with an infinite amount of visual variation to our retinas and yet we are able to easily recognize tens of thousands of distinct object categories [**?**] with invariance to various factors such as movement, lighting, shade, orientation and partial occlusion [**?**]. Rosch *et al.* [1] argue that this is because of the non-random, and hierarchical, structure of visual objects, for instance no breed of dogs will have wings but they will often share many features such as four legs with certain type of paws. In general entities that are closer to each other in a hierarchical semantic taxonomy will also typically share more visual features; dogs and birds are still more alike than dogs and vehicles since both belong to a higher level category of animals which all share certain visual attributes. Deselaers and Ferrari [**?**] veried these observations

in their study on the visual separability of different semantic categories using the ImageNet dataset.

Standard machine learning and deep learning models used for image classification do not exploit this semantic taxonomy since they are typically trained against one-hot encoded target vectors. For example, the 1K ImageNet data set [**?**] will have labels as 1000-dimensional vectors where there is 1 for the dimension indicating the correct class and zeros elsewhere. It follows that label vectors for different classes are thus orthogonal and normally a model trained on this data will therefore not account for any classes belonging to a common category at a higher level of the taxonomy. For example, in reality two different breeds of dog do fall under the same higher level categories of dogs, animals, mammals and so forth - all of these share certain amount of visual features. The chosen level of hierarchy of a one-hot encoding is also usually arbitrary; should an image be labelled a dog or a labrador retriever? Optimally both would be taken into account, we argue. The number of machine learning algorithms that do take this into account is limited, however; below we give an overview of them.

4.2.2 Semantic Hierarchies for Image Classification

The use of semantic hierarchies in computer vision dates back to early work on image retrieval [**?**], [**?**], [**?**] [**?**], where it has been used to expand potential query terms and to impose a more general association between image features and corresonding labels. These ideas were subsequently utilized to improve image annotation and classification [**?**], [**?**] [**?**]. For example, Marszalek and Schmid [**?**] use the WordNet database to create a lexical hierarchy of their image labels and train an SVM classifier [**?**] that acts at each node of the hierarchy. The resulting model proves very flexible and performs well under uncertainty - if the model is unsure about what dog breed is in an image, it can then move up one level and just predict a 'dog'. It should be noted that the dataset used by the authors is very simple by today's standards, however.

More recently, Redmon and Farhadi [**?**] employed a similar idea but in the context of deep learning. In particular, for each image, its label was taken to be the full path from the root node of the WordNet tree to the originl singular label. The model was then trained with multiple softmax functions, one for each level of hierarchy in order produce a chain of conditional probabilities from the root node to leaves. The authors' purpose for building this model was to learn simultaneously from two datasets of unequal hierarchies and to jointly optimize classification and detection (locating an object) on the two datasets. However, other possible benefits of such a flexible model, like generalization to new classes, were not explored in detail. In their research on human categorization, Rosch *et al.* [1] defined the concept of 'basic level'. This is described as the most fundamental level of a specific category in the semantic taxonomy and is the one first learnt usually by children and also objects are identified the quickest at this level. The level below basic level is called 'subordinate level'. An example of this structure would be the basic level label of 'fox' and its subordinate 'arctic fox'. The importance of the basic level seems to stem from this level having the highest ratio of visual variation between categories to variation within categories [1], [2]. It results that it has been especially practical for humans to identify and label objects at this level.

Hillel and Weinshall [**?**] implemented the idea of basic and subordinate levels by building a two-stage classifier where the first stage generates a vector representing the different parts of an objects and the second stage classifies instances based on this vector. On average, this strategy outperformed a traditional one-step algorithm. Wang and Cottrell [**?**] took these ideas to the deep learning-era an trained a CNN on both basic and subordinate labels of the ImageNet 2012 data [**?**]. They found that this leads to an improved performance on the standard top-5 classification accuracy. Another simiar study [**?**] explored a dataset which mostly had coarse labeled images (similar to the basic level) and only some fine grained examples. The authors showed

that their custom CNN, which used both hierarchies, could predit fine-level labels better than a model that's only trained on fine level. This suggests that the fine-grained model is able to borrow strength from coarse labels and consequently generalize better. Peterson *et al.* extended this line of research by exploring the type of representations learnt by a CNN trained on both 'basic' and 'sub-ordinate' labels. First, the authors found that including the basic-level labels in pre-training or fine-tuning led to a much more clustered representation of the feature spaces extracted from the final layer of the CNN (e.g. the features vectors of different breeds of dogs were now bundeled up together whilst if trained on just subordinate labels, then there was no clustering of similar categories). The features also had learn separate hierarchies for natural and man made objects. Finally, they illustrated the generalization power of these representations by running a zero-shot learning experiment in which the model was shown only a few examples of either sub- or basic-level objects and then tries to find all other images from the data set belonging to the given label. Interestingly, the results show that the supplementary basic-level training had led to a bias to label objects at this level which is congruent with corresponding studies in humans [5].

Despite the important contribution of above works, they are limited in that they only consider two levels of a semantic taxonomy. It has been shown that, despite the importance of the basic level, humans posses and utilize a much more complex multi-level semantic hierarchy [2]. We are not aware of any deep learning papers that explore the type of representations and the consequent generalization properties that would be learnt by considering a multi-level semantic taxonomy. We attempt to fill this void by exploiting a full set of hierarchical taxonomy via semantic embeddings built on a large-scale lexical database WordNet [**?**].

4.2.3 Semantic Embeddings and Zero-shot Learning

Above studies illustrate the implicit connection between text and visual data processing in humans. Joliceur *et al.* [2] explored the dependence of our visual system on semantic understanding in more depth and illustrated this with several experiments. For instance, upon viewing a picture of a chair we can use our semantic knowledge base to generalize it to the category of furniture even though the image itself doesn't reveal that such an abstraction even exists. Similary, we can seamlessly access our visual memory to imagine semantic concepts, even ones we may never have seen such as 'a cat wearing ice skates'.

Zero-shot learning (ZSL) [**?**] is concerned with making predictions about previously unseen classes of images. Formally if a classifier f is trained on features X_{TR} and training labels Y_{TR} such that $f : X_{TR} \rightarrow Y_{TR}$, then in ZSL for the possible class labels we have that $Y_{TR} \cup Y_{ZSL} = \emptyset$ where Y_{ZSL} represents the test labels of any ZSL dataset. A fundamental idea in this area was to exploit the semantic relatedness of visually linked categories [**?**], [**?**], [**?**] by projecting input data into a lower dimensional semantic embedding space. This can be represnted by the following equations [**?**] (see Figure 4.5 for a graphical explanation):

$$\mathcal{S} : X^d \rightarrow F^P \tag{4.9}$$

$$\mathcal{L} : F^P \rightarrow Y \tag{4.10}$$

Here we assume that we have a full collection of all pairs $\{f, y\}_{1:M}$ of both seen and unseen classes where f is the point in the semantic embedding space F^P that corresponds to the label y. In the training phase we can take the data for the seen classes $\{X, y\}_{1:N}$, where $N << M$, and replace each y with its embedding vector f and thus learn the mapping $\mathcal{S}$. Labels for objects of unseen classes can then be predicted by first mapping them into F^P through $\mathcal{S}$ and then employing $\mathcal{L}$ to find the most appropriate y based

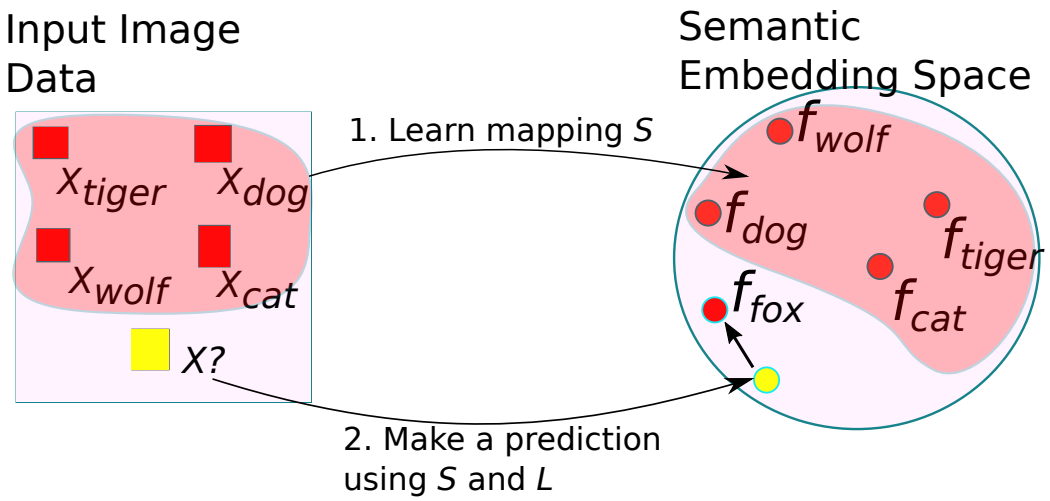

Figure 4.5: A graphical illustration of zero-shot learning where the left square contains all input images. Red ones are the training classes whcih are used to learn a mapping $\mathcal{S} : X^d \rightarrow F^P$. The circle represents the semantic embedding space. Note how we can access embedding vector for also classes we dont have visual training data for. Upon seeing a novel image (yellow square) we predict its embedding vector (yellow dot) and then find nearest plausible classe (here fox) using some function $\mathcal{L}$. Novel image is hence classified as a fox even though the model has never seen one before

on the predicted semantic embedding; this could for instance be a nearest neighbor classifier. The trick is thus having the semantic embedding space F^P to contain vectors for all seen and unseen classes. To be clear, the term 'zero-shot' thus refers to the trained model never having seen $M - N$ of the classes but in our semantic knowledge base we do know about the existence of those zero-shot classes and their names. The challenge is thus in how to construct the semantic embedding space and what type of model to use to learn $\mathcal{S}$; the following paragraphs will cover the existing approaches.

In one of the early works on the topic, Socher *et al.* [**?**] created word embedding vectors for all seen and unseen class labels from freely available Wikipedia data using a methodology [**?**] that uses local and global context of the words to place similar words next to each other in the embedding

space. A two-layer feedforward network was then used to learn a mapping from images features into the embedding manifold. Outlier detection was used to determine if the predicted embedding was one of the known or unknown classes. Only 8 known classes and 2 unknown classes were considered however. To address this and other limitations, Frome *et al.* [**?**] developed the Deep Visual-Semantic Embedding Model (DeViSE), which was trained on 1000 classes of the ImageNet dataset and its ZSL abilities were tested on the remaining 20,000 classes. This model structure is similar to ours in that it takes a pre-trained deep learning model, AlexNet (which won the 2012 ImageNet competition) [**?**], and then removes the softmax layer and replaces it with a projection layer that attempts to predict the word vector embedding of each image label. The word embeddings, in turn, were extracted from a skip-gram model trained on 5.4billion words of text from Wikipedia. The skip-gram model [**?**], [**?**] learns to place vectors of similar terms near to each other by attempting to predict adjacent words. This model succesfully used the semantic information to make correct guesses on thousands of unseen classes, and it still provides a good baseline for new methods as we will show in our results section.

The authors of DeViSE worried however that the model was overfitting the transformation from the image representations into the semantic feature space. To overcome this they introduced another novel model called Convex Combination of Semantic Embeddings (ConSE) [**?**]. The idea is as follows: take a previously unseen image class and simply feed this into a pretrained deep learning model which has been trained on, say 1000 classes. The model will consequently try to predict this new image as one of the known thousand classes and outputs respective 1000 predicted class probabilities $p_0(y|\mathbf{x})$ such that $\sum_{y=1}^{1000} p_0(y|\mathbf{x}) = 1$. The predicted semantic embedding for the new class can then be taken as the convex combination of the known classes weighed

by the predicted probabilities from the previous step. More formally [**?**]:

$$f(x) = \frac{1}{Z} \sum_{1}^{T} p(\hat{y}(\mathbf{x}, t)|\mathbf{x}) \cdot \mathcal{L}(\hat{y}(\mathbf{x}, t)) \tag{4.11}$$

where $\hat{y}(\mathbf{x}, t)$ is the t^{th} most likely class-label for image $\mathbf{x}$ out of the known labels, with $p()$ giving the respective probability. $\mathcal{L}$ maps each of these labels deterministically to their embedding. This model was shown to generalize slightly better to novel classes than DeViSE when both used the same image features and embedding vectors.

Bibliography

[1] E. Rosch, C. B. Mervis, W. D. Gray, D. M. Johnson, and P. Boyes-Braem, "Basic objects in natural categories," *Cognitive Psychology*, vol. 8, no. 3, pp. 382–439, 1976.

[2] P. Joliceur, M. A. Gluck, and S. M. Kosslyn, "Pictures and Naming: Making the Connection," *Cognitive Psychology*, vol. 16, pp. 243–275, 1984.

[3] J. C. Peterson, P. Soulos, A. Nematzadeh, and T. L. Griffiths, "Learning Hierarchical Visual Representations in Deep Neural Networks Using Hierarchical Linguistic Labels," 2018.

[4] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," 2015.

[5] F. Xu and J. B. Tenenbaum, "Word learning as Bayesian inference.," *Proceedings of the 22nd Annual Meeting of the Cognitive Science Society*, pp. 517–522, 2000.