

Integration of Image and Word Embeddings for Descriptive Image Similarity

David Gustafsson, Lund University
Tobias Lindberg, Lund University

June 2017

Abstract

Many people today possess a private digital photo collection. Such collections are often just chronologically sorted. One way to make photo browsing more interesting would be to suggest semantically related photos to a currently viewed photo, and if, in addition, such a relationship could be justified in words, that would create extra value for the user.

To find a solution to this problem, the approach of this thesis is to bridge the domains of images and language by creating vector embeddings for images in an already existing semantic vector space for words. Transfer learning with an image as input and the vector representation of a corresponding human-created caption as sought output is applied to a convolutional neural network originally trained for object detection. The transfer and training is carried out in the machine learning framework TensorFlow.

The described approach shows promising performance in general and a thorough comparison of different layouts is carried out. The best model is tested, qualitatively as well as quantitatively through a task-specific custom evaluation scheme and on common benchmark datasets. The conclusion based on these results is that the proposed system is well suited for the given tasks, and that it opens up for a number of interesting extensions.

Acknowledgments

We would like to thank Microsoft for having given us the opportunity to write this thesis in collaboration with them. Special thanks go to our supervisors at the Microsoft office in Lund, Jakob Moberg and Fredrik Linåker, for providing us with an interesting use case, insight into a truly inspiring company, and for always being there with support and good advice.

We would also like to thank our supervisor at Lund University, Karl Åström, for sharing his great knowledge in computer vision and for valuable feedback along the way.

Contents

1	Introduction	1
1.1	Background	1
1.2	Problem Formulation	2
1.3	Related Work	2
1.4	Structure	4
2	Theory	6
2.1	Machine Learning	6
2.1.1	Neural Networks	7
2.1.2	Convolutional Neural Networks	8
2.1.3	Convolutional Layer	9
2.1.4	Pooling Layer	11
2.2	Training and Optimization	11
2.2.1	Backpropagation	12
2.2.2	Batch Normalization	14
2.2.3	Transfer Learning	16
2.2.4	Adam Optimizer	17
2.3	Information Retrieval	19
2.3.1	Mean Average Precision	20
2.3.2	R@k	20
2.4	InceptionV3	20
2.4.1	Design Principles	21
2.4.2	Inception Layer	21
2.4.3	Architecture	24
2.5	Word2vec	24
2.5.1	Continuous Bag-of-Words	25
2.5.2	Word Representation	26
3	Method	27
3.1	Approach	27
3.2	Model	30
3.2.1	Hyperparameters	30
3.2.2	Additional Architecture	33

3.2.3	Loss Function	34
3.2.4	Optimizer	34
4	Data	36
4.1	Training Data	36
4.2	Validation Data	38
4.3	Test Data	39
5	Experimental Setup	41
5.1	Hyperparameters	41
5.2	Training	41
6	Results	43
6.1	Training	43
6.2	Validation	45
6.2.1	Loss function	45
6.2.2	Hyperparameters	46
6.3	Test	50
6.3.1	SUN	50
6.3.2	Benchmark	51
6.4	Demo	52
7	Discussion	53
7.1	Future Work	55
8	Conclusion	58
9	Work Division	59
	Appendices	60
A	SUN Scene Names	60
B	Full Results	61
	Bibliography	61

Chapter 1

Introduction

This chapter serves to introduce our work and to put it into practical context by giving a background to the scenario behind the problem and an intended use case for the derived model. The general problem is then concretized into four research questions, which have outlined our work throughout this project and which we answer in the following chapters. We give a brief overview of the most closely related research and mention how our approach is differentiated from earlier work. Lastly, the structure of the remainder of this report is given.

1.1 Background

As the amount and the sizes of private digital photo collections are growing by the minute, there is an increased demand for smart, scalable solutions for both storage and management. Such collections have historically often just been sorted chronologically, although more sophisticated solutions, such as exploitation of geographical information, have emerged as alternative ways to create smart structures. However, the late successes in machine learning and computer vision have opened up for even more advanced applications directed towards photo management, and this thesis explores one such possibility, namely a deep learning approach to content-based image retrieval.

Content-based image retrieval (CBIR) has been an active field of research over the last decades. The word content could refer to anything between high-level concepts such as *person* and *car*, to low-level features such as shapes and colors, although lately more interest have been directed towards high-level CBIR. The query in the retrieval problem typically consists of an image or a short text, to which one or more semantically similar images should be retrieved.

In our scenario, we imagine a home computer user with a private digital photo collection. Our aim is to, for a currently viewed photo, present a small number of semantically similar photos with a short description of how they

are similar. That means that the query will be an image and that the underlying mathematical distance function for retrieval should be constructed such that a human would agree that the query image and the retrieved images are semantically similar with respect to the given description. This is a very complex and not particularly well-posed problem, which we thus need to reformulate before it can be further approached.

We find inspiration in earlier work, which has shown that it is possible to bridge the domains of images and language by the construction of integrated vector spaces, since the twofold problem of retrieving both related images and a description requires just a model that is able to associate the two domains. There exist high quality vector embeddings for the English vocabulary that are freely available. We then need to construct a model that is able to create relevant vector embeddings for a user's private photos, and a way to integrate these with the word embeddings. Earlier work has shown that deep neural networks, and in particular convolutional neural networks (CNNs), can provide such vector embeddings, and we build on that in our thesis.

1.2 Problem Formulation

From the background and context given above, we have settled on the following problem formulations:

- How can we define a metric to match the human perception of semantic image similarity?
- How can we construct an algorithm that creates vector representations of images and uses these representations to retrieve the most semantically similar images for a given query image?
- How can we equip the presented similar images with a description of the similarity?
- Can we implement a system like this?

1.3 Related Work

As mentioned above, CBIR has attracted much interest for a long time. Consequently, a number of surveying academic papers have been published on this topic at different points in time. Two of the more recent, which both focus on high-level features and cover a number of neural network-approaches are [1] and [2], where the latter goes into even more detail about modern techniques utilizing CNNs.

In [2] and the context of CNNs, distinctions are made between *pre-trained CNN models*, *fine-tuned CNN models*, and *hybrid methods*. All three types

build on the extraction of features from layers in neural networks but the division is based on how the respective networks are trained. Image retrieval models where the features are extracted from “existing” neural networks are classified as *pre-trained CNN models*. Examples of such networks are AlexNet [3] and ResNet [4], which are both trained for object detection and will therefore likely struggle with generalization. It is also as an example pointed out that pre-trained CNN models can often not discriminate between different famous landmarks even though the underlying network is able to classify images as containing buildings correctly. A *fine-tuned CNN model* would instead be a model where the output layer of a pre-trained network is updated to have dimensionality corresponding to the number of different landmarks in a dataset. It is thus still trained for classification, but only with more relevant classes. (Transfer learning is a more general and common term for this.) Lastly, the *hybrid methods* rely on a sliding window approach, or similar, to create smaller patches, which are then processed with standard low-level image processing techniques before the results are fed as input to a CNN. The features for image retrieval are then again obtained from a later layer in the CNN, but the advantage of this approach is that the separated low-level processing of patches could make the system more scale invariant.

The impact of transfer learning on CNNs for image retrieval is explored in [5] among other works. The authors evaluate the performance when the semantic representation is obtained from the third-to-last, second-to-last, and the very last layer of a pre-trained AlexNet, respectively. The conclusion is that the best overall performance is obtained when using the second-to-last layer. Experiments with dimensionality reduction of the extracted features are also carried out, indicating that some reduction can be made without significant loss in retrieval performance. That is however nothing we will focus on in our work.

A number of approaches for creating meaningful multimodal (image and word) representations have been presented lately. In [6], a model, which concatenates image feature vectors extracted from a CNN with word embedding vectors, is proposed. This does indeed integrate the two domains but the approach is not directly suitable for our task, since we would like to be able to generate vector representations for unseen images that are not already equipped with words.

The works in [7], [8] and [9] are all centered around the idea of an integrated word and image space. In the former, the idea is to embed images into an existing word vector space. That is done by taking a CNN trained for object recognition and fine-tuning it to map the input image to a static vector representation of a caption that describes the image. The aim of the work is to improve object detection by exploiting the inherent knowledge of the underlying word space to both make more semantically reasonable misclassifications and to make it possible for the model to correctly classify unseen objects, i.e. to enable zero-shot prediction.

A similar approach is taken in [8]. Image embeddings are obtained from the values in the middle layer of an autoencoder, where the first half of the network corresponds to a CNN. The training objective is partly to minimize the reconstruction error, but also partly to minimize the distance between the image embeddings and embeddings of corresponding captions. The model is designed to perform well on image captioning framed as a ranking task over a fixed set of captions, as well as on text-to-image retrieval.

In [9], the aim is to find the best matching caption for a given image. This is done by measuring the distances between image and word embeddings in the integrated vector space and rank the captions based on that. The image embeddings are again obtained from a layer in a CNN, while a multi-layer perceptron is introduced and trained to project vector representations of captions onto the described image space. The method can thus in some sense be seen as the inversion of the method in [7], where image embeddings are projected onto an existing word vector space.

Our work is in some parts reminiscent of [7], but there are important differences. Firstly, we build and fine-tune our model with image-to-image and many-images-to-word retrieval as main objectives, whereas image-to-label, i.e. classic object recognition, is the main focus in [7]. Moreover, a number of design parameters, such as optimizer, embedding dimensionality, and loss function, are chosen differently in the two works. We also perform a more thorough analysis of different network layouts than we have seen in any previous work.

1.4 Structure

The remainder of this thesis is structured in the following way:

- **Chapter 2** builds theoretical foundations for the concepts used in the following chapters.
- **Chapter 3** describes the core ideas behind the approach as well as the overarching layout of the desired system.
- **Chapter 4** gives a motivation to why different data has been used for training and evaluation, and describes the two datasets.
- **Chapter 5** details and motivates the experimental setup used to evaluate different models.
- **Chapter 6** provides quantitative results and comparisons between a set of models with different architectures.
- **Chapter 7** contains a discussion about the chosen approach and mentions possible directions in which it would be interesting to extend the research.

- **Chapter 8** summarizes the work and findings of this thesis.
- **Chapter 9** gives a description of how the different subtasks have been divided between the authors during this work.

Chapter 2

Theory

This chapter provides theory for the most important tools that are later used to construct the model. First comes an introduction to machine learning and neural networks, which is followed by theory for optimization of neural networks and examples of a few such algorithms. The next section gives a more detailed description of Google’s InceptionV3 network and lastly, an overview of the vector embedding model for words, `word2vec`, is provided.

The reader is expected to have basic knowledge within the field of machine learning and neural networks, as some basic terms will only be explained concisely in order to make room for more relevant and complex concepts. For a more thorough introduction to the basic concepts, the reader is referred to [10].

2.1 Machine Learning

Artificial intelligence has for a long time intrigued humans and the quest to achieve it dates back to before the age of computers. A famous example is an autonomous chess machine called the “Mechanical Turk”, created in the 1770s, which was later found to be a hoax as the machine was manually controlled by a chess master. Still, the search for artificial intelligence continued and the main approach for many years was to implement models of hard coded rules that the creator defined.

A relevant example in computer vision is [11] from 1978, where the authors tried to create an autonomous system performing semantic segmentation of objects in an image. Each object to be segmented had its own set of manually created rules which can be seen in the example of window detection in figure 2.1. However, the results were extremely poor compared to the standard of today, and one major flaw was that the system quickly grew to be extremely complex as the number of rules grew linearly with the number of objects. Furthermore, the task of segmenting objects by finding a set of manually created rules which would generalize to all possible scenarios was

next to impossible.

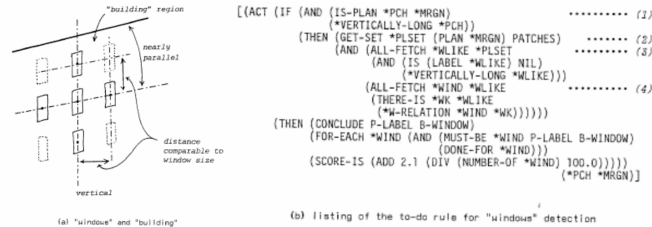


Figure 2.1: Handmade rules created in [11] for segmenting windows.

In order to combat this problem, another approach emerged that assigned the rule-finding task to the computer, which is today known as machine learning. There is a well known quote defining machine learning by Tom M. Mitchell in [12]:

“A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.“

The segmentation sub-problem of window detection described above can be stated in accordance with this quote by viewing the task T as performing window detection, the experience E as gained from being presented images along with the true segmentation (i.e. training data), and the performance measure P as created by evaluating the segmentation produced by the model compared to the ground truth segmentation through a loss function.

2.1.1 Neural Networks

The fields of cognitive science and artificial intelligence have always been closely related as one tries to understand human intelligence and the other tries to mimic the same. This has among other things led to the interdisciplinary study of neural networks, which have their origins in attempts to find mathematical representations of biological information processing, as shown in [13] and [14]. The general idea of a neural network is shown in figure 2.2, where there are layers consisting of nodes and connections between layers which enables the flow of information from the input to the output layer. An analogy can now be established to how the brain processes information by exchanging the nodes for neurons and the connections for synapses.

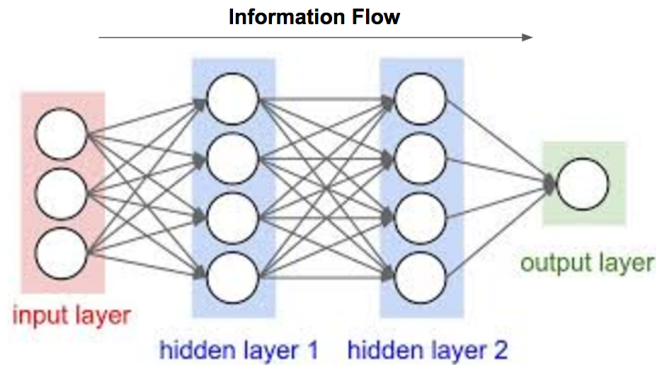


Figure 2.2: General idea of a neural network.

The original and most intuitively idea of a neural network is the multilayer perceptron, where each node in one layer is connected by a weight to each node in the next layer, and the two layers are consequently said to be fully connected. In order for the network to not collapse into a linear model, hence the hidden layers would serve no purpose, a non-linear activation function is added after the linear combination of the previous layer's nodes:

$$z_j = \sigma\left(\sum_{i=1}^N w_{i,j}x_i\right), \quad j = 1, 2, \dots, M,$$

where x_i is the value of node i in the previous layer, z_j is the value of node j in the current layer, $w_{i,j}$ is the unique weight connecting node i and j in the previous and current layer, and σ is the non-linear activation function. For the specific example in the formula above, there are N nodes in the previous layer and M nodes in the current layer, and thus there exists $N \times M$ weights connecting both layers.

2.1.2 Convolutional Neural Networks

A convolutional neural network is a special type of neural network influenced by the biological visual system. These networks have been a driving force in the fields of computer vision and machine learning since Alex Krizhevsky et al. showed their prominence when winning the image classification competition ILSVRC 2012 [15].

It all begun in the 1960s when the authors of [16] performed a famous experiment with the visual cortex system of cats. They observed how neurons fired when presented with different orientations of a pattern and concluded that there were neurons which responded to individual smaller regions of the visual field, called receptive fields.

Inspired by the advancement in [16], a new type of neural networks emerged in the 1990s called convolutional networks, which had the primary

purpose of performing image classification. LeNet-5, proposed by LeCun et al. in 1988 [17], was one of the very first convolutional networks, which mainly performed character recognition tasks such as zip-code scanning.

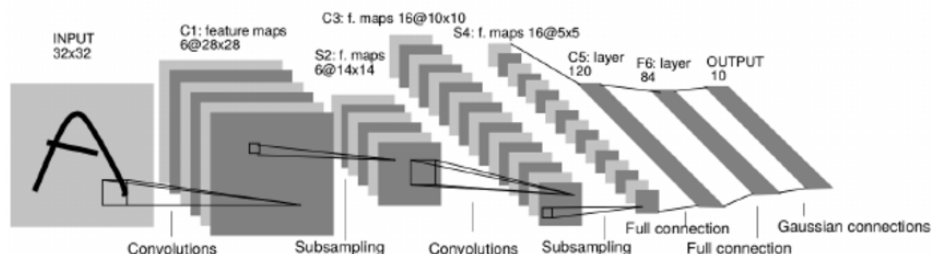


Figure 2.3: Architecture of LeNet-5 proposed in [17].

The network architecture of LeNet-5, shown in figure 2.3, was revolutionary as it proposed two new types of layers called convolutional layers (convolutions) and pooling layers (subsampling). These layers tries to mimic the effect of neurons having receptive fields that was found in [16].

2.1.3 Convolutional Layer

From a technical standpoint, the convolutional layer has two major improvements over the fully connected layer in image classification. Firstly, an image has local connectivity given a task, e.g. face recognition only needs the information from a smaller region of the image containing the face. This translates to nodes connected to a sub-part of the image, called filters. Secondly, in most cases the local connectivity is stationary over the image, for instance an apple can be detected with the same filter regardless of location in the image. This indicates that sliding one filter should be sufficient instead of having a unique filter at each local segment of the image performing the same task. These concepts are illustrated in figure 2.4.

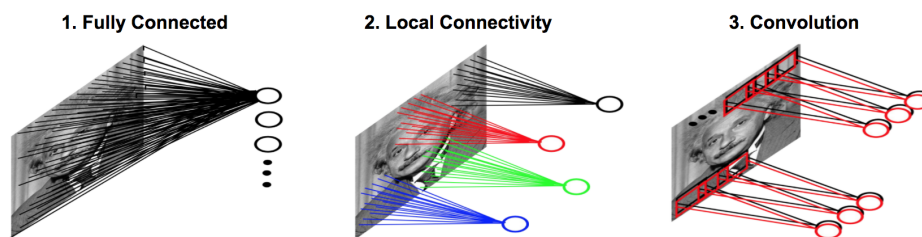


Figure 2.4: Illustration of the characteristics behind full connectivity, local connectivity and convolution.

Using a convolutional layer greatly reduces the complexity, as measured by the number of free parameters in the layer, which can be shown by a

simple example:

Image size: 200×200 , Nodes: $4 \cdot 10^4$
Filter size: 10×10 , Nr. of Filters: 100
Fully connected $\implies 200 \cdot 200 \cdot 4 \cdot 10^4 = 16 \cdot 10^8$ parameters
Locally connected $\implies 10 \cdot 10 \cdot 4 \cdot 10^4 = 4 \cdot 10^6$ parameters
Convolution $\implies 10 \cdot 10 \cdot 100 = 10^4$ parameters

A traditional convolutional layer is defined by three variables: the number of filters, the filter size and the stride. Both the number of filters and the filter size was used in the example above and are intuitive to comprehend. The stride, on the other hand, controls how the filters are slid across the image, and is depicted in figure 2.5.

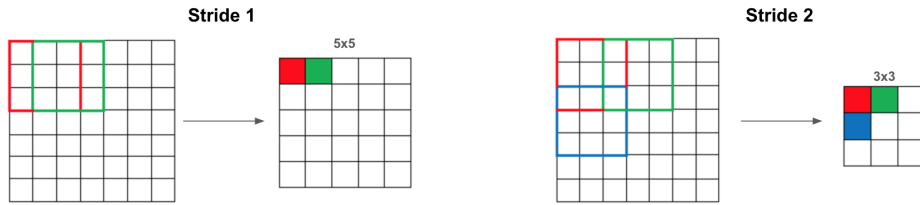


Figure 2.5: Convolution with one filter of size 3×3 on a 7×7 input with stride 1 and 2.

There are some dependencies between the variables in order for this operation to work out. Denote the input size \mathbf{W} , the filter size \mathbf{F} , and the stride \mathbf{S} . The output size \mathbf{O} is then calculated as

$$\mathbf{O} = \frac{\mathbf{W} - \mathbf{F}}{\mathbf{S}} + 1, \quad \mathbf{O} \in \mathbb{Z}^2.$$

Note that the division is done separately in each dimension and since \mathbf{O} has to be in \mathbb{Z}^2 , the right hand side of the equation has some restrictions. Padding could be used to solve the problem but it is however generally favorable to avoid as it makes assumptions about how the input expands outside its border. A concluding example of a convolutional layer can be seen in figure 2.6.

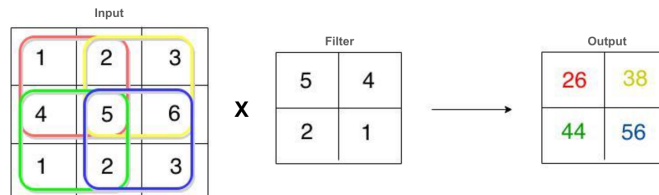


Figure 2.6: Example of convolution with input size 3×3 , filter size 2×2 and stride 1.

2.1.4 Pooling Layer

It is common practice to periodically integrate pooling layers when constructing a convolutional network. A pooling layer is a convolutional layer with a static operation, e.g. maximum value, mean value, or L^2 norm. Thus it is not possible to perform any update of the pooling layer during training. However, the pooling layer will reduce the amount of parameters in the network. For example, a pooling layer with filter size 2×2 and stride 2 will reduce the number of parameters by 75%. This greatly reduces the computation in the network and helps to control overfitting as it reduces the complexity.

A more practical motivation for the use of pooling layers is that a filter in a convolutional layer is tailored to detect a specific feature, e.g. an edge or an object such as an apple. Thus, if such a feature exists in the input to the convolutional layer, the output will have active nodes, i.e. high values, for the specific positions where the filter and the feature align. However there will be some activation in the nodes where the filter partly overlap the feature. In most cases the main purpose of the convolutional layer is to detect whether a feature is present or not, hence there is not any gain in information by having these half overlapping nodes. A pooling layer with for instance the max-operation will keep the high values, i.e. the most activated nodes, but in a more compressed form and, as the complexity is greatly reduced, this tends to be favorable. A concrete example of a pooling layer is given in figure 2.7.

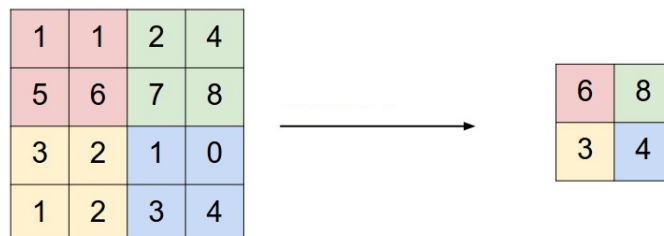


Figure 2.7: Example of a pooling layer with max-operation, filter size 2×2 and stride 2.

2.2 Training and Optimization

In order for a machine learning model to learn a specific task, a loss function is developed, which gives numerical feedback on how a model performs and allows the model to update itself to increase its performance. These functions usually have two input parameters, the prediction of the model, $\hat{\mathbf{y}}$, and the ground truth, \mathbf{y} . In most cases the goal is to get $\hat{\mathbf{y}}$ to equal \mathbf{y} , and therefore

in regression problems it is common to minimize the L^2 distance,

$$L(\mathbf{y}, \hat{\mathbf{y}}) = \|\mathbf{y} - \hat{\mathbf{y}}\|_2.$$

This poses a minimization problem given the loss function and the parameters in the model. For the majority of neural networks this becomes a non-convex problem where gradient descent based optimizers are commonly used.

2.2.1 Backpropagation

When training a neural network, the goal is to update the weights in each layer such that the network minimizes the loss function. Thus a gradient descent based optimizer wants to find the gradients with respect to each weight \mathbf{W}_i :

$$\frac{\partial L(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{W}_i}, \quad i = 1, 2, \dots, N. \quad (2.1)$$

These gradients need to be found in a precise and computationally efficient manner. Finding a gradient is commonly done by either of the following two methods.

- **Numerical gradient:**

Find the gradient approximatively through the limit

$$\frac{df(\mathbf{x})}{d\mathbf{x}} = \frac{f(\mathbf{x} + \mathbf{h}) - f(\mathbf{x})}{\mathbf{h}}, \quad \mathbf{h} \rightarrow \mathbf{0}.$$

This solution is computationally expensive as finding these limits in the most basic implementation requires several passes through the network when calculating $f(\mathbf{x})$ and $f(\mathbf{x} + \mathbf{h})$.

- **Analytical gradient:**

Calculate the exact analytical gradient for each weight. This solves the problem of being approximative, but leads to complex expressions when working with deep neural networks. Computationally it will also be expensive as each layer of weights will need a pass through the network in order to calculate their analytical gradients.

Exploiting the structure of a neural network provides an alternative approach to calculate the gradients in a precise manner without having to derive the exact expression for the gradients. This exploitation will be described in three steps; by finding the gradient in equation 2.1 for i equal to N , $N - 1$, and for a general i , respectively.

Notation

For a fully connected feed forward network, denote the weights \mathbf{W}_i and the activation function σ_i for layer i . Then, for input \mathbf{x} , the output $\hat{\mathbf{y}}$ is calculated as

$$\hat{\mathbf{y}} = \sigma_N(\mathbf{W}_N \sigma_{N-1}(\mathbf{W}_{N-1} \dots \sigma_2(\mathbf{W}_2 \sigma_1(\mathbf{W}_1 \mathbf{x})) \dots)). \quad (2.2)$$

Furthermore, denote the intermediate results during a forward pass \mathbf{h}_i and \mathbf{z}_i before and after activation respectively. Then

$$\begin{aligned} \mathbf{z}_i &= \sigma_i(\mathbf{W}_i \sigma_{i-1}(\mathbf{W}_{i-1} \dots \sigma_2(\mathbf{W}_2 \sigma_1(\mathbf{W}_1 \mathbf{x})) \dots)), & \mathbf{z}_i &= \sigma_i(\mathbf{W}_i \mathbf{z}_{i-1}), \\ \mathbf{h}_i &= \mathbf{W}_i \sigma_{i-1}(\mathbf{W}_{i-1} \dots \sigma_2(\mathbf{W}_2 \sigma_1(\mathbf{W}_1 \mathbf{x})) \dots), & \mathbf{h}_i &= \mathbf{W}_i \sigma_{i-1}(\mathbf{h}_{i-1}), \\ & & \mathbf{z}_i &= \sigma_i(\mathbf{h}_i), & \mathbf{h}_i &= \mathbf{W}_i \mathbf{z}_{i-1}. \end{aligned}$$

Differentiation w.r.t. \mathbf{W}_N

By storing each intermediate result during a single forward pass (note that $\hat{\mathbf{y}} = \mathbf{z}_N$) one can exploit the chain rule to calculate

$$\frac{\partial L(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{W}_N} = \frac{\partial L(\mathbf{y}, \mathbf{z}_N)}{\partial \mathbf{z}_N} \frac{\partial \mathbf{z}_N}{\partial \mathbf{W}_N} = \frac{\partial L(\mathbf{y}, \mathbf{z}_N)}{\partial \mathbf{z}_N} \frac{\partial \mathbf{z}_N}{\partial \mathbf{h}_N} \frac{\partial \mathbf{h}_N}{\partial \mathbf{W}_N}. \quad (2.3)$$

The first factor $\frac{\partial L(\mathbf{y}, \mathbf{z}_N)}{\partial \mathbf{z}_N}$ and second factor $\frac{\partial \mathbf{z}_N}{\partial \mathbf{h}_N} = \frac{\partial \sigma_N(\mathbf{h}_N)}{\partial \mathbf{h}_N}$ will only depend on its respective function, L and σ_N , and can be analytically calculated. The last factor can be simplified to $\frac{\partial \mathbf{h}_N}{\partial \mathbf{W}_N} = \frac{\partial \mathbf{W}_N \mathbf{z}_{N-1}}{\partial \mathbf{W}_N}$, where the numerator is a vector and the denominator is a matrix. This gradient will become a tensor and interestingly, if calculated, one will realize that the sparsity makes it possible to collapse the tensor into a matrix. Finally, it can now be concluded that each of these gradients will only depend on \mathbf{y} , \mathbf{z}_N , \mathbf{h}_N and \mathbf{z}_{N-1} , which are all stored in the forward pass.

Differentiation w.r.t. \mathbf{W}_{N-1}

Proceeding to find the gradient with respect to \mathbf{W}_{N-1} gives

$$\frac{\partial L(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{W}_{N-1}} = \frac{\partial L(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}_{N-1}} \frac{\partial \mathbf{z}_{N-1}}{\partial \mathbf{h}_{N-1}} \frac{\partial \mathbf{h}_{N-1}}{\partial \mathbf{W}_{N-1}},$$

where the two last factors can be found in the same way as when differentiating with respect to \mathbf{W}_N . The last factor needs to be expanded more in order for it to be found:

$$\frac{\partial L(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}_{N-1}} = \frac{\partial L(\mathbf{y}, \mathbf{z}_N)}{\partial \mathbf{z}_N} \frac{\partial \mathbf{z}_N}{\partial \mathbf{z}_{N-1}} = \frac{\partial L(\mathbf{y}, \mathbf{z}_N)}{\partial \mathbf{z}_N} \frac{\partial \mathbf{z}_N}{\partial \mathbf{h}_N} \frac{\partial \mathbf{h}_N}{\partial \mathbf{z}_{N-1}}.$$

It can now be noted that $\frac{\partial L(\mathbf{y}, \mathbf{z}_N)}{\partial \mathbf{z}_N} \frac{\partial \mathbf{z}_N}{\partial \mathbf{h}_N}$ has already been calculated in equation 2.3 and that $\frac{\partial \mathbf{h}_N}{\partial \mathbf{z}_{N-1}} = \frac{\partial \mathbf{W}_N \mathbf{z}_{N-1}}{\partial \mathbf{z}_{N-1}} = \mathbf{W}_N$. Hence the derivative of $L(\mathbf{y}, \hat{\mathbf{y}})$ with respect to \mathbf{z}_{N-1} can be efficiently calculated by passing the already calculated error $\frac{\partial L(\mathbf{y}, \mathbf{z}_N)}{\partial \mathbf{z}_N} \frac{\partial \mathbf{z}_N}{\partial \mathbf{h}_N}$ from the previous step.

Differentiation w.r.t. \mathbf{W}_i

It has now been shown that calculation of the derivatives of $L(\mathbf{y}, \hat{\mathbf{y}})$ with respect to \mathbf{W}_N and \mathbf{W}_{N-1} in descending order can be done in a precise and efficient manner. Finally, the method can be generalized to an arbitrary layer i by finding

$$\frac{\partial L(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{W}_i} = \frac{\partial L(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}_i} \frac{\partial \mathbf{z}_i}{\partial \mathbf{h}_i} \frac{\partial \mathbf{h}_i}{\partial \mathbf{W}_i}, \quad (2.4)$$

where, once again, the two last factors can be found in the same manner as previously described, and for the first factor, one can again exploit the chain rule:

$$\frac{\partial L(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}_i} = \frac{\partial L(\mathbf{y}, \mathbf{z}_N)}{\partial \mathbf{z}_{i+1}} \frac{\partial \mathbf{z}_{i+1}}{\partial \mathbf{z}_i} = \frac{\partial L(\mathbf{y}, \mathbf{z}_N)}{\partial \mathbf{z}_{i+1}} \frac{\partial \mathbf{z}_{i+1}}{\partial \mathbf{h}_{i+1}} \frac{\partial \mathbf{h}_{i+1}}{\partial \mathbf{z}_i}. \quad (2.5)$$

The two first factors in equation 2.5 are calculated and passed down from the previous step when calculating $\frac{\partial L(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{W}_{i+1}}$, and the last factor is found as $\frac{\partial \mathbf{h}_{i+1}}{\partial \mathbf{z}_i} = \frac{\partial \mathbf{W}_{i+1} \mathbf{z}_i}{\partial \mathbf{z}_i} = \mathbf{W}_{i+1}$.

Final algorithm

The backpropagation algorithm for a fully connected feed forward neural network can now be stated as follows.

1. Forward pass

Perform a forward pass through the network and cache \mathbf{h}_i and \mathbf{z}_i for $i = 1, 2, \dots, N$.

2. Backward pass

Find the gradients $\frac{\partial L(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{W}_i}$ for $i = N, N-1, \dots, 1$ in descending order by the following steps.

- (a) Calculate $\frac{\partial L(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}_i}$, $\frac{\partial \mathbf{z}_i}{\partial \mathbf{h}_i}$ and $\frac{\partial \mathbf{h}_i}{\partial \mathbf{W}_i}$ from \mathbf{z}_i , \mathbf{h}_i , \mathbf{W}_{i+1} , and $\frac{\partial L(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}_{i+1}} \frac{\partial \mathbf{z}_{i+1}}{\partial \mathbf{h}_{i+1}}$.
- (b) Form the desired gradient $\frac{\partial L(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{W}_i} = \frac{\partial L(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}_i} \frac{\partial \mathbf{z}_i}{\partial \mathbf{h}_i} \frac{\partial \mathbf{h}_i}{\partial \mathbf{W}_i}$.
- (c) Calculate $\frac{\partial L(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}_i} \frac{\partial \mathbf{z}_i}{\partial \mathbf{h}_i}$ and pass to the next step $i-1$.

2.2.2 Batch Normalization

From the backpropagation algorithm it is clear that the gradients of the loss function with respect to the weights \mathbf{W}_i will become a product of more and more factors as i goes from the top layer N to the bottom layer 1. It will therefore behave as an exponential function and, depending on the value of all previous gradients, it can in some cases either explode or vanish, similar to how an exponential function $f(x) = a^x$ will either explode or vanish

depending on the value of a . Consequently, one wants the gradients to be reasonably large such that they do not vanish, but not too large such that they explode.

Generally, the most common problem is vanishing gradients, which is due to the behavior of the activation functions since, for each iteration of the back propagation algorithm, one multiplies with the factor $\frac{\partial z_i}{\partial \mathbf{h}_i} = \frac{\partial \sigma_i(\mathbf{h}_i)}{\partial \mathbf{h}_i}$, which only depends on the choice of activation function. By examining two common choices of activation functions, the sigmoid and the hyperbolic tangent (tanh) which is depicted in figure 2.8, it is clear that, as the input reaches a large positive or negative value, the gradient will tend to zero. This creates a region of interest, the area between the vertical dotted lines in figure 2.8, where one during training wish to have the pre-activation values \mathbf{h}_i in order to get a non-zero gradient.

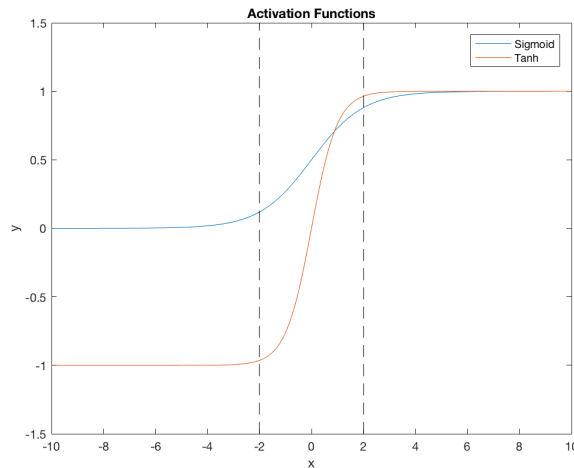


Figure 2.8: Plot of the sigmoid and tanh function.

For a long time this was solved by carefully initializing the weights in the network, commonly from a normal distribution with zero mean, such that \mathbf{h}_i , given the input \mathbf{x} , was initialized in the region of interest. However, a more sophisticated approach of forcing the pre-activation values \mathbf{h}_i into the region of interest is called batch normalization and was introduced in [18]. The idea is to force \mathbf{h}_i to take on a unit normal distribution during training by feeding a batch of inputs \mathbf{x} , and as the input propagates through the network, each \mathbf{h}_i is normalized based on the batch that was fed through. By denoting the batch of size M at the input $\mathbf{X} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(M)}\}$, and at the activation units $\mathbf{H}_i = \{\mathbf{h}_i^{(1)}, \mathbf{h}_i^{(2)}, \dots, \mathbf{h}_i^{(M)}\}$, the batch normalization

can be written as

$$\hat{\mathbf{h}}_i^{(k)} = \frac{\mathbf{h}_i^{(k)} - \mathbb{E}[\mathbf{H}_i]}{\sqrt{\text{Var}[\mathbf{H}_i]}}, \quad k = 1, 2, \dots, M, \quad (2.6)$$

where $\hat{\mathbf{H}}_i = \{\hat{\mathbf{h}}_i^{(1)}, \hat{\mathbf{h}}_i^{(2)}, \dots, \hat{\mathbf{h}}_i^{(M)}\}$ are the normalized values, which are fed into the activation function.

The authors of [18] expand even further on this idea by, instead of assuming unit normal distribution, letting the network scale and shift the normalized pre-activation values $\hat{\mathbf{H}}_i$. This is seen as a learning process and learnable scale variables γ_i and shift variables β_i are introduced. Building on equation 2.6, the complete algorithm becomes

1.

$$\hat{\mathbf{h}}_i^{(k)} = \frac{\mathbf{h}_i^{(k)} - \mathbb{E}[\mathbf{H}_i]}{\sqrt{\text{Var}[\mathbf{H}_i]}}, \quad k = 1, 2, \dots, M$$

2.

$$\bar{\mathbf{h}}_i^{(k)} = \gamma_i \hat{\mathbf{h}}_i^{(k)} + \beta_i, \quad k = 1, 2, \dots, M,$$

where $\bar{\mathbf{H}}_i = \{\bar{\mathbf{h}}_i^{(1)}, \bar{\mathbf{h}}_i^{(2)}, \dots, \bar{\mathbf{h}}_i^{(M)}\}$ are the normalized values, which are fed into the activation function. An overview of how batch normalization can fit into the architecture of a neural network is given in figure 2.9.

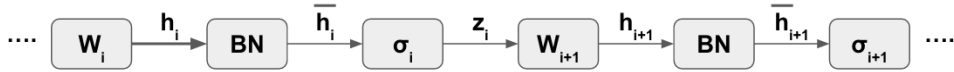


Figure 2.9: Illustration of how batch normalization (BN) fits into the architecture of a neural network.

2.2.3 Transfer Learning

A common problem in machine learning, and especially in the supervised case, is the limitation of training data. Thus, to compensate for a complex task and a small amount of training data, one can utilize a model that is pre-trained on similar data and a similar task as a basis for the new task. The approach is called transfer learning and has proven to yield increased performance for a number of problems [19]. It is based on the idea of transferring knowledge between similar tasks. For instance, a programmer who can code in Java but wants to learn Python can utilize a lot of already possessed experience to learn quicker. Most machine learning methods have their unique ways of efficiently performing this transfer of knowledge, and below is a description of how it is generally done for neural networks.

As information flows through a neural network it extracts and creates features from the input in order to finally form the output. More precisely,

one usually distinguishes between features created in earlier layers as low-level features, i.e. general features such as corners and edges, and features from the final layers as high-level features, i.e. specific features for the task to be performed. The idea is that, given two similar tasks, the low-level features will be the same while the high-level features will differ. Thus, if denoting two similar tasks A and B, this will translate to first training the network on task A and then use the pre-trained network to only update the final layers for task B, which is depicted in figure 2.10.

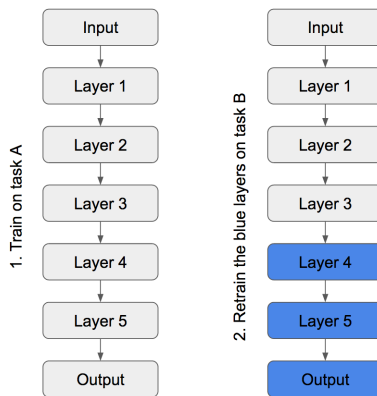


Figure 2.10: Transfer learning for a neural network on two tasks A and B.

In addition to only updating the final layers, new, randomly initialized layers could be added at the end. There can also be an increase in performance by branching from a different layer, i.e. going from an intermediate layer to the output.

2.2.4 Adam Optimizer

Stochastic gradient descent (SGD) has played a major role in the late success of machine learning as most problems can be cast as a minimization problem of some scalar loss function. The update rule for original stochastic gradient descent is given by

$$\Theta_{t+1} = \Theta_t - \eta \Delta_{\Theta} f(\Theta_t), \quad (2.7)$$

where η is the learning rate, Θ_t is the parameter at time step t , and f is the function to be minimized. Over the years, several improvements have been built upon the original implementation of SGD, where the primary focus has been to speed up the time it takes to find a local optimum. One of the most recent improvements is called adaptive moment estimation (Adam) optimizer [20]. The algorithm is based upon the combined advantages of two other optimizers; RMSProp [21] and Momentum [22].

Momentum

A common problem with the SGD optimizer is when the gradient is much steeper in some dimensions than others, which is often the case around a local minimum. This leads to oscillations when approaching the minimum, which in turn delays the convergence. In order to tackle this, one can update the parameters by an exponentially decaying average of the gradients:

$$\begin{aligned}\mathbf{m}_t &= \beta\mathbf{m}_{t-1} + \eta\Delta_{\Theta}f(\Theta_t), \\ \Theta_{t+1} &= \Theta_t - \mathbf{m}_t.\end{aligned}$$

Here, in addition to the notation in equation 2.7, β is the momentum decay and \mathbf{m}_t is the momentum term at time step t . In figure 2.11, the effect of momentum can be seen as the optimizer stores previous gradients in the momentum term.

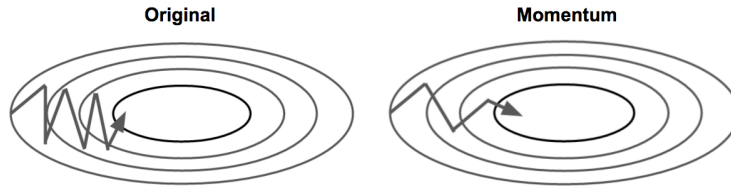


Figure 2.11: Original SGD vs. Momentum optimizer.

RMSProp

SGD optimizers tend to be sensitive to the initial learning rate, which is what RMSProp was built to improve upon. RMSProp is in turn an upgraded version of another optimizer called AdaGrad [23]. The idea is to reduce the step size of larger and more recurrent updates and increase it for smaller and less recurrent updates. Thus, if the optimizer performs a too large or small update, depending on the learning rate, it will compensate for that. This is done by keeping an exponentially decaying average of the squared gradients and divide the learning rate by this factor:

$$\begin{aligned}\mathbf{v}_t &= \gamma\mathbf{v}_{t-1} + (1 - \gamma)(\Delta_{\Theta}f(\Theta_t))^2, \\ \Theta_{t+1} &= \Theta_t - \frac{\eta}{\sqrt{\mathbf{v}_t + \epsilon}}\Delta_{\Theta}f(\Theta_t).\end{aligned}$$

Here \mathbf{v}_t is the squared gradient, $\gamma \in [0, 1)$ is the decay factor for the squared gradient, and ϵ (usually around 10^{-8}) prevents division by zero. Note that all calculations are performed element-wise. It can now be seen that if an update in one dimension is large or occurs often, that dimension will become large in \mathbf{v}_t and, consequently, the step size in that dimension will be greatly reduced by the division. Vice versa is true for dimensions with small and less recurrent gradients.

Adam

The Adam optimizer finally combines the two main ideas from Momentum and RMSProp. It also proposes a solution to the bias created when initializing \mathbf{v}_0 and \mathbf{m}_0 to zero. The update rules for Adam are the following:

$$\begin{aligned}\mathbf{m}_t &= \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \Delta_{\Theta} f(\Theta_t), \\ \mathbf{v}_t &= \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) (\Delta_{\Theta} f(\Theta_t))^2, \\ \hat{\mathbf{m}}_t &= \frac{\mathbf{m}_t}{1 - \beta_1^t}, \\ \hat{\mathbf{v}}_t &= \frac{\mathbf{v}_t}{1 - \beta_2^t}, \\ \Theta_{t+1} &= \Theta_t - \frac{\eta}{\sqrt{\hat{\mathbf{v}}_t + \epsilon}} \hat{\mathbf{m}}_t.\end{aligned}$$

Here $\hat{\mathbf{m}}_t$ and $\hat{\mathbf{v}}_t$ are bias-corrected terms, and β_1 and β_2 are decay factors.

The two bias-corrected terms are derived in the same way so it suffices to present the derivation for the momentum \mathbf{m}_t only. First, \mathbf{m}_t can be expressed as a sum of gradients at previous time-steps,

$$\mathbf{m}_t = (1 - \beta_1) \sum_{i=1}^t \beta_1^{t-i} \Delta_{\Theta} f(\Theta_i).$$

It is then assumed that the gradients $\Delta_{\Theta} f(\Theta_i)$ are stationary and therefore their expected value will not change over time. (In [20], the authors show that the theory holds even without this assumption.) Now, one wishes to find the discrepancy between $\mathbb{E}[\mathbf{m}_t]$ and $\mathbb{E}[\Delta_{\Theta} f(\Theta_t)]$, in order to compensate for it:

$$\begin{aligned}\mathbb{E}[\mathbf{m}_t] &= \mathbb{E}[(1 - \beta_1) \sum_{i=1}^t \beta_1^{t-i} \Delta_{\Theta} f(\Theta_i)] = (1 - \beta_1) \sum_{i=1}^t \beta_1^{t-i} \mathbb{E}[\Delta_{\Theta} f(\Theta_i)] \\ &= \mathbb{E}[\Delta_{\Theta} f(\Theta_t)] (1 - \beta_1) \sum_{i=1}^t \beta_1^{t-i} = \mathbb{E}[\Delta_{\Theta} f(\Theta_t)] (1 - \beta_1^t).\end{aligned}$$

It can now be seen that the discrepancy is a factor $(1 - \beta_1^t)$ and thus, dividing \mathbf{m}_t by this factor will do the compensation. It should also be noted that as t increases and as \mathbf{m}_t is based on more and more past gradients, the bias-correcting factor $1/(1 - \beta_1^t)$ will tend to 1 as expected.

2.3 Information Retrieval

Information retrieval is the task of retrieving relevant items from a database given a query, e.g. web-page search. Commonly, all items in the database are ranked given their similarity to the query and the most similar items are

then retrieved. Systems building on this are usually evaluated on datasets where relevant items that should be retrieved for each query are marked. Given such a dataset, there are a couple of measures at hand to evaluate a model. Two of these are used in this work; the mean average precision and R@k.

2.3.1 Mean Average Precision

The Mean Average Precision (MAP) is based on the precision-at- k measure, which is defined as the precision of a query when only considering the first k items in a ranked database, i.e.

$$P(k) = \frac{tp(k)}{tp(k) + fp(k)} = \frac{tp(k)}{k}, \quad (2.8)$$

where $tp(k)$ is the number of true positives within the first k items, and $fp(k)$ is analogously the number of false positives within the first k items. All instances above and including k are considered positive predictions, and hence it follows that $tp(k) + fp(k) = k$.

The average precision of a query can then be calculated as

$$AP = \frac{\sum_{k=1}^n P(k)\delta(k)}{tp(n)}, \quad (2.9)$$

where n is the total number of items in the database, and $\delta(k)$ is an indicator function that equals 1 if the element at rank k is a true positive and 0 otherwise.

Finally, the MAP is simply the mean AP over a number of queries, q , i.e.

$$MAP = \frac{\sum_{q=1}^Q AP(q)}{Q}, \quad (2.10)$$

where Q is the total number of queries.

2.3.2 R@k

The R@k measure checks whether any relevant item is present among the first k ranked items. If that is the case, the query is marked as positive, and if not as negative. An average is then formed over all queries, e.g. if there are ten queries and three of them have at least one relevant item among the first k ranked items, the R@k score becomes 0.3.

2.4 InceptionV3

The computer vision competition ImageNet Large Scale Visual Recognition Contest (ILSVRC) [15] was in 2012 won with a convolutional neural network

called AlexNet [3]. This success sparked new interest for the use of CNNs in the field of computer vision, and applications were found in many different areas. In the following years, ILSVRC saw many models with both deeper and smarter architecture improving on the work that AlexNet initialized. One of the more recent networks that has shown very good performance to a small computational cost is InceptionV3, which is described in this section.

2.4.1 Design Principles

The InceptionV3 network is designed around four core principles, which are all intuitively motivated from qualitative results by the authors.

1. Avoid bottlenecks. The information flowing through every possible cut should be close to equal. The flow of information is measured as the number of connections between nodes that are cut.
2. Increase the number of dimensions as information passes through the network. The output of each layer should have a higher dimensionality than the input.
3. Lowering the resolution of the input before a convolutional layer will not lead to any significant loss in information gain.
4. Balance the width and depth of the network as it has generally proved to increase the performance of neural networks.

2.4.2 Inception Layer

The inception layer is a clever use of convolutions in order to achieve the same results as a standard convolutional layer but in a more efficient manner. The first idea is that, given the same input to several convolutional layers with different filter sizes, each layer will produce unique features as the filters can find spatially different connections. It therefore suggests that convolutional layers can be arranged parallelly and that the output can be concatenated, which aligns well with principle 4 since the network width can then be increased. Apart from stacking convolutional layers, pooling layers can also be added as it will pass raw information forward to the next layer. The authors of InceptionV3's precursor GoogLeNet [24] decided to restrict the filter sizes to 1×1 , 3×3 , and 5×5 which gives the layer layout depicted in figure 2.12.

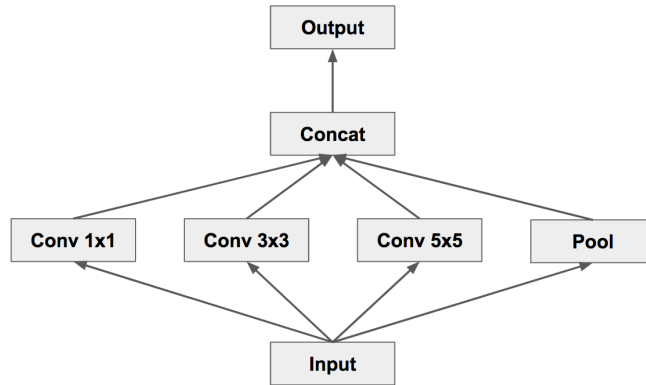


Figure 2.12: Parallel stacking of convolutional and pooling layers.

Furthermore, in accordance with design principle 3, the dimensionality of the input to the computationally expensive 3×3 and 5×5 layers can be reduced. That is possible by adding 1×1 convolutional layers before these layers. To understand why, denote the input size $[W, H, N]$ where W and H are the width and height, and N is the number of feature maps, which correspond to the number of filters in the previous layer. Then, given a convolutional layer with filter size 1×1 , K filters and stride 1, the output will become $[W, H, K]$ at a relatively small computational cost of $\mathcal{O}(KN)$. Choosing $K \ll N$ will thus greatly reduce the number of dimensions. This idea is shown in figure 2.13.

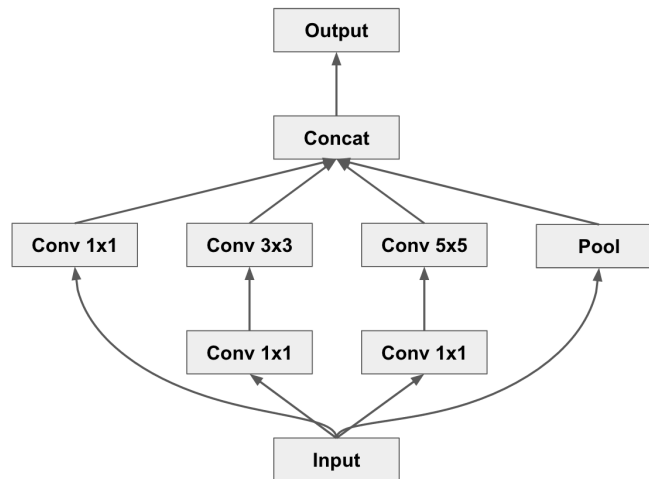


Figure 2.13: Dimensionality reduction before convolution of size 3×3 and 5×5 .

The final idea is that a convolutional layer can be described by several smaller layers. For instance, as shown in figure 2.14, two 3×3 layers can

be used to capture the behavior of a 5×5 layer. In figure 2.14 the two top tiles, which are the outputs of the convolutions, will depend on all 5×5 tiles in the input, and hence both solutions should be able to capture the same information. Replacing a 5×5 convolution with two 3×3 convolutions provides a way to reduce the computational cost by a factor $\frac{2 \times 3 \times 3}{5 \times 5} = \frac{18}{25} = 0.72$. Note that this can be applied in many variations and a general method that will always work is to replace an $n \times n$ layer by two layers of size $n \times 1$ and $1 \times n$.

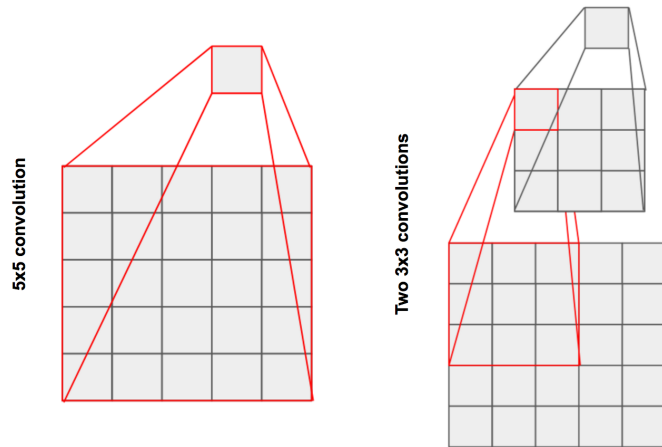


Figure 2.14: Convolution with one filter of size 5×5 and two filters of size 3×3 , respectively.

Adding the result shown in figure 2.14 to the inception layer will produce the graph seen in figure 2.15. All the basic ideas behind the inception layer presented above can be combined and modified in many ways to produce numerous different inception layers.

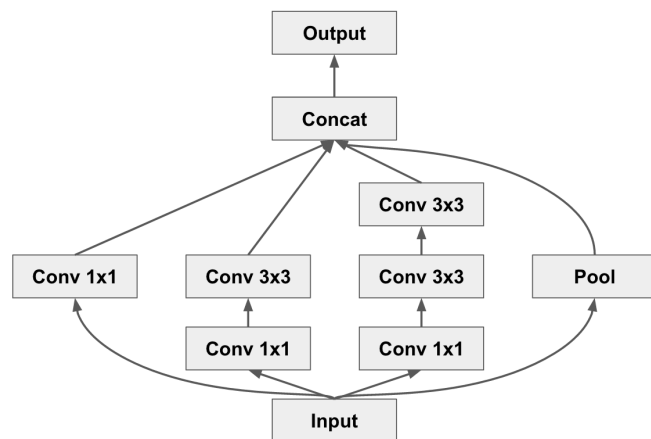


Figure 2.15: Final inception layer which include all presented ideas.

2.4.3 Architecture

InceptionV3 has a simple architecture, given the knowledge of an inception layer, which is shown in figure 2.16. The core pieces are three inception layers with slightly differing setups, but all are similar to the one presented in the previous section. Note that the inception layers are repeated a couple of times and that they all follow design principle 2, such that the dimensionality of the output is higher than for the input to the layer.

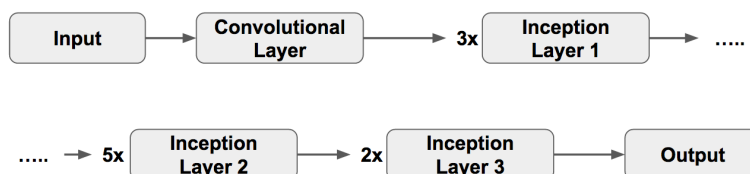


Figure 2.16: Architecture of InceptionV3.

2.5 Word2vec

The complex problem of natural language modeling can be addressed in many ways. Models building mainly on hand-written rules were on a large scale replaced by statistical ones in the late 1980s, and the field has since then been intimately connected to machine learning, benefiting from the huge advances in that area. Lately, the construction of vector spaces for text modeling has emerged as a popular angle of approach, and, among such models, the word2vec model first presented in [25] has gained a great amount of attention.

The word2vec model builds on the distributional hypothesis [26] that words that appear in the same contexts will be closely related and often have similar meanings. Based on that, either of two different learning algorithms can be utilized to find vector embeddings for words appearing in a training dataset. The first algorithm is called Continuous Bag-of-Words (CBOW) and aims to predict a single word from a given surrounding. The objective in the other algorithm is, contrariwise, to predict the context when a single word is given as input. This approach is denoted the skip-gram model.

It is difficult to make a precise statement about how the performance of the two algorithms compare in general but CBOW seems to give better accuracy for frequent words while skip-gram handles infrequent words better. Since they are roughly the inversion of each other, only CBOW is detailed in this chapter.

2.5.1 Continuous Bag-of-Words

The setup for the CBOW model is a neural network with one input layer, one hidden layer and one output layer, as depicted in figure 2.17. During training, an input sentence is split up into a middle word, v_t , and a context of $2k$ surrounding words, $c = \{v_{t-k}, v_{t-k+1}, \dots, v_{t-1}, v_{t+1}, \dots, v_{t+k-1}, v_{t+k}\}$. The network takes a binary vector that is the sum of the one-hot encodings of the surrounding words, i.e. the “bag” of the surrounding words, as input. Consequently, the bag of words does not contain any information about the order of the words. This sparse but high-dimensional input is projected onto a hidden layer of lower dimensions. (A vocabulary could consist of $10^5 - 10^7$ words [27], while the hidden layer is commonly chosen to have 300 nodes.) No activation function is used at the hidden layer.

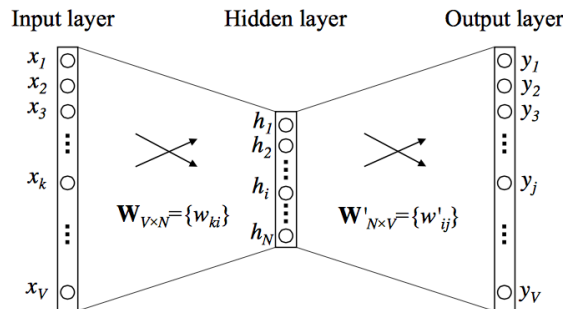


Figure 2.17: The architecture for the neural network used in the word2vec algorithm, where $x_i = 1$ if word i in the vocabulary is part of the context, and $x_i = 0$ otherwise.

Finally, the output layer has the same dimensions as the input layer, and the sought output is the one-hot encoding of the missing middle word, v_t . By using a softmax function at the final layer, the output can be interpreted as the probability for each word, v , in the vocabulary being the middle word given context c , i.e.

$$p(v|c) = \frac{e^{y(c)^T x(v)}}{\sum_{\hat{v} \in V} e^{y(c)^T x(\hat{v})}}, \quad (2.11)$$

where $x(\alpha)$ is the binary one-hot input encoding for word α while $y(\alpha)$ is the network output layer value, and where V is the whole vocabulary.

The objective is then to maximize the probability for the correct middle word and to minimize it for all other words. However, in order to increase training efficiency, a form of Noise Contrastive Estimation [28] called Negative Sampling is introduced in [27]. The core idea behind that approach is to sample only a smaller set of incorrect words at each iteration in order to not have to update the whole weight matrix between the hidden and output

layer each time. Negative Sampling has proven to not only increase training efficiency, but also to improve the quality of the word vectors after training.

2.5.2 Word Representation

When the model has finished the training over a large number of natural language sentences (in [27] a set of one billion words from news articles has been used), the output layer is of less interest. The sought semantic embedding of a word is retrieved from the hidden layer values when the one-hot encoding of the word is fed as input, or, equivalently, the corresponding column in the weight matrix between the input layer and the hidden layer.

By looking at how the derived vector representations are related for related words, it is possible to find rather clear structures in the embeddings. The most basic pattern is that words with similar meaning have vectors that lie close to each other in the vector space, which is depicted in figure 2.18. Moreover, if one for example, based on the cosine distance between the vectors, rank the nearest vectors to the result of the arithmetic expression

$$h(\textit{Paris}) - h(\textit{France}) + h(\textit{Italy}),$$

the top result would be $h(\textit{Rome})$, i.e. the vector corresponding to the word *Rome*. This type of results can be found, not only for semantic relations like the one above, but also for syntactic relations such as verb forms [25].

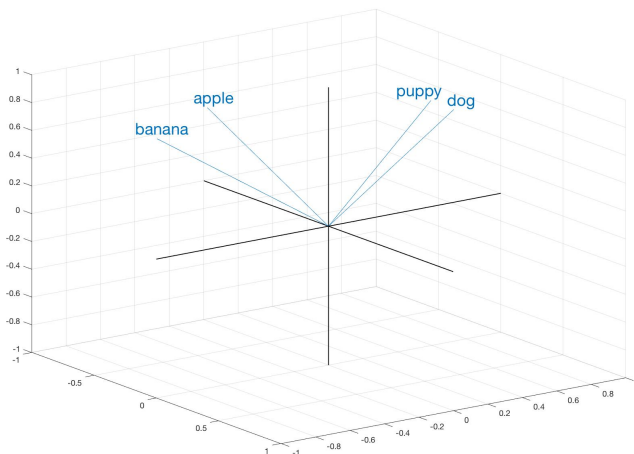


Figure 2.18: A 3-dimensional vector space where similar words have vectors lying close to each other.

Chapter 3

Method

This chapter serves to describe how the theory that was introduced in the previous chapter has been applied to form a deep learning module with aim to provide the best possible semantic image retrieval and similarity description. The first part presents the idea to incorporate the semantic image representations into an already existing word embedding space. It is followed by a description of the overarching layout of the desired model as well as justifications for the design choices.

3.1 Approach

The problem we are facing is twofold in that we want to find both related images and descriptions of the relatedness. By filling the gap in “*More ... photos*” with the best fitting single word, a simple similarity description that would work well in a future product can be obtained. The first part of the problem then corresponds to classic image-to-image retrieval while the second could be entitled many-images-to-word retrieval. Considering these two retrieval tasks, and especially the multimodality of the latter, it is logical to look for ways to connect the domains of images and language. Such an approach would let us solve the two parts of the problem with a single model instead of treating them as separate, which would typically call for two models.

As accounted for in section 1.3, previous work has presented different ways of bridging the domains of images and language. Our system is reminiscent of the one in [7] but was derived independently, and the process is therefore explained in detail from the bottom up.

It is shown in [5] that image vector representations of dimensionality between the second and fourth order of magnitude can be used for image retrieval. Such a vector is merely a column of real numbers, where each value represents how much of the corresponding feature the image contains. The features could be known, i.e. manually labeled, describing how likely it

is that the corresponding image depicts e.g. winter, water, or animals. The features could also be unknown, i.e. difficult for a human to interpret but not for the machine that produced them.

Word2vec, which was introduced in section 2.5, is a 300-dimensional vector space for word embeddings with unknown features. Since images can be (partly) described by words, it should be possible to create image embeddings in the existing word2vec space. Our goal then becomes to construct a model that is able to create meaningful vector embeddings for new images. That is, for an unseen image, the proposed model should assign a 300-dimensional vector in the word2vec space such that the image is surrounded by similar images and words that describe the image. This will generate an integrated vector space containing both word embeddings and image embeddings, which is depicted (in three dimensions) in figure 3.1.

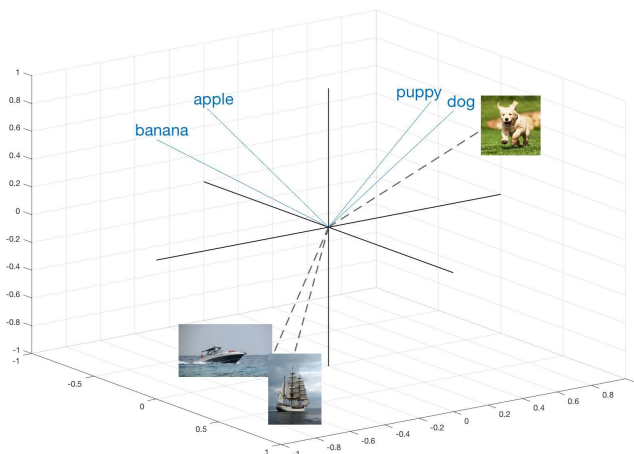


Figure 3.1: A 3-dimensional integrated vector space containing both image and word embeddings.

We also need to address the question of how to define image similarity, which is now equivalent with how to embed images into the word2vec space. Earlier image retrieval systems have built on tags, i.e. concepts that are recognized in images by a trained concept detection model. Such approaches directly provide robust and intuitive similarity metrics by simply finding the number of common concepts for two images. The methods could also be extended to integrate images and language by representing each image as the sum of the word2vec vectors corresponding to the present concept tags, and the similarity between two images could then be determined by some distance in the word2vec space. The approach does however come with a couple of drawbacks. Firstly, a network trained for concept detection is

limited to the finite number of concepts it is shown during training. It is infeasible to train a network on all possible concepts and, secondly, since these concepts are often objects such as *book* and *spoon* and we are interested in *semantic* image similarity, not all concepts will be relevant for our task anyway.

Another idea is that the semantic essence of an image, which is precisely what we are interested in, is better captured by a short natural language image caption. An example is given in figure 3.2, where we mean that “*a bunch of friends having a good time conversing at home*” captures the semantic essence in a better way than the tags, which are: potted plant, couch, dining table, person, wine glass, remote, book, cup, chair, cell phone, vase. We therefore want to build our model on natural language captions rather than tags.



Figure 3.2: An example indicating that the semantic essence of an image is better captured by a short natural language caption than by tags.

In order to construct a model that is able to create relevant embedding vectors in the word2vec space for unseen images, the task is formulated as a supervised machine learning problem, and hence we need a training dataset containing images along with the desired ground truth vectors in the word2vec space. Each image could of course be described in many ways, which in turn would lead to multiple possible ground truth vectors. Thus, in order to train a machine learning model on such a problem, it becomes very important to maintain consistency throughout the training process, which is motivated further in chapter 4.

By taking this approach, we allow the model to find a very intuitive

and direct connection between images and language. Figure 3.3 outlines our method and shows how the proposed model fits into the overarching system.

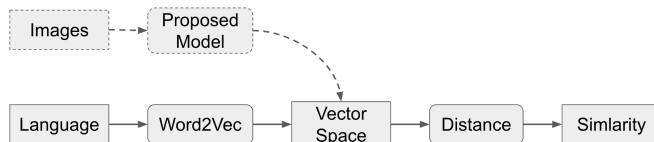


Figure 3.3: Overview of the system, where the dashed part corresponds to the novel connection moving images into the pre-defined word2vec vector space.

3.2 Model

As described in section 1.3, trained convolutional neural networks can often generate good vector representations for images, which we utilize in our model. Training a CNN for a computer vision task from scratch is however very computationally expensive and it can take many days to reach convergence on modern hardware [24]. Thus, to be able to explore multiple network layouts and hyperparameter settings, we use a neural network that is trained on a different but related task and apply transfer learning to exploit some of the learned concepts as a basis for the training towards our specific goal.

More precisely, the network layout was obtained from the TensorFlow-Slim implementation of Google’s InceptionV3 architecture and the pre-trained weights were loaded from the corresponding 2016-08-28 checkpoint [29]. An overview of the network layout can be seen in figure 3.4.

3.2.1 Hyperparameters

Performing transfer learning for a neural network gives rise to a number of model-level hyperparameters, as detailed below. The loaded network was originally trained on the ImageNet classification task, as described in [30], and the output is a 1,000-dimensional vector answering to the probabilities that a given input image contains the respective objects. This high-level object information could of course be used for image retrieval, but it is not necessarily the representation that gives the best retrieval performance out of all the extractable representations in the network, as shown in [31]. There is therefore reason to further explore how the performance on our specific problem depends on the choice of layer to “branch” at, i.e. where in the pre-trained network to extract the information.

The dimensions of the extracted information will depend on the branching layer, which follows from the description of the network in section 2.4.1. Then, in order to incorporate the semantic image representations into the

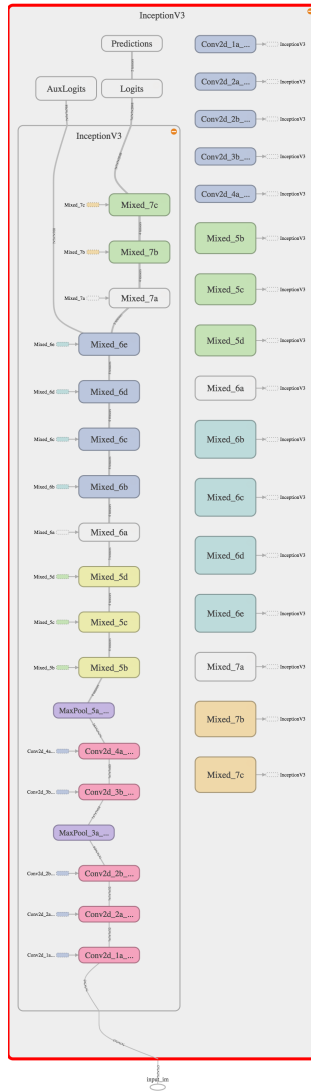


Figure 3.4: TensorBoard graph of the TensorFlow-Slim InceptionV3 implementation, where the inception layers are denoted as Mixed_XX.

word2vec embedding space, the extracted information needs to be transformed. This could be achieved by inserting an additional layer of weights from the branching layer to a 300-dimensional output layer. However, inserting more than one fully connected layer would increase the model complexity and could allow the model to learn more complex relations between the extracted information and the sought output. Thus we consider the number of additional layers to be a tunable variable in our model, which we would like to find an optimal value for.

To apply transfer learning by extracting an intermediate vector representation from the InceptionV3 network and using it as input to a smaller fully connected network is in theory likely to imply some limitations to the learning abilities of the resulting network layout. That is, a network architecture identical to the compound architecture described above that is trained from scratch could possibly give better performance on the new task, as it would be able to adjust the weights all the way back through the InceptionV3 network.

As a measure to move the model in this direction, one can let a number of the pre-trained layers be trainable, i.e. the weights in these layers are initialized with values found during ImageNet training, but they are then updated through backpropagation of the errors on the new task. This approach can thus be seen as a compromise between just extracting the values from the pre-trained network and training a new network from scratch. By letting the number of trainable InceptionV3 layers vary as a hyperparameter, we will hopefully be able to conclude whether it affects the overall performance or not.

In summary, there are three model-level parameters to optimize: the branching layer, the number of additional layers, and the number of trainable layers. An example illustrating these parameters can be seen in figure 3.5.

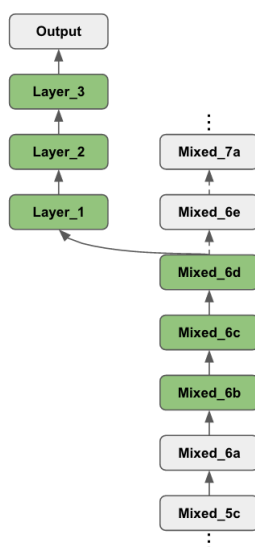


Figure 3.5: Graph of a model branching at layer 6d, having three additional layers and three trainable layers. Green indicates that the weights at that layer will be updated.

3.2.2 Additional Architecture

The greater part of the model comes from the loaded InceptionV3 network, whose architecture is depicted in figure 3.4. However, in order to integrate InceptionV3 into our overarching system and since the loaded network expects input on a certain format, a preprocessing step needs to be added in addition to the additional trailing layers described in the previous section. This is illustrated in figure 3.6.



Figure 3.6: Illustration of the different steps in the implemented model.

Preprocessing

Since we want to be able to feed images of varying size in both monochrome and color, there is a need to normalize the data before feeding it into the neural network - especially in order to be able to perform some form of mini-batch gradient descent. The preprocessing consists of three steps. Each image is first resized to 299×299 pixels using bilinear interpolation. It is then formatted to have three color channels (RGB), and, finally, a normalization is performed to ensure that the pixel values are in the range $[-1, 1]$, as expected by the TensorFlow-Slim model.

Additional Layers

The additional architecture consists of N standard fully-connected layers, and the input dimension to the first additional layer, which depends on the branching layer as described in section 2.4, is denoted D . Then, with the goal to obtain a 300-dimensional output vector, we choose the intermediate dimensions as the values on the line between D and 300. More concretely and with the notation introduced above, the input and output dimensions of layer $i \in [1, N]$ becomes

$$(\text{input}_i, \text{output}_i) = \left(D - (i - 1) \frac{D - 300}{N}, D - i \frac{D - 300}{N} \right).$$

Furthermore, for all additional layers except the last one, the hyperbolic tangent function, \tanh , is used as activation function along with batch normalization, which is described in section 2.2.2. There is no activation function at the last layer since that would only make it more difficult for the network to output good vector embeddings. Finally, the weights for all additional layers are initialized from a normal distribution with zero mean and standard deviation 10^{-5} .

3.2.3 Loss Function

The loss function serves as an interpretation of how the model performs on the given task, where we want our model to be able to “match the human perception of semantic image similarity” and “equip the presented similar images with a description of the similarity”, as stated in section 1.2. Thus, a natural candidate for loss function would be the cosine distance between the predicted 300-dimensional vector representation of the input image and its corresponding 300-dimensional ground truth vector, since it has proven to be a good similarity measurement used for words in the word2vec model as mentioned in section 2.5. Minimizing this quantity during training should make the network learn structures that generate image vectors at meaningful positions in the shared embedding space, and it should allow for good image retrieval.

However, if minimizing the cosine distance alone, the whole notion of vector norm is disregarded. That is definitely not optimal for scenarios other than just one-to-one retrieval tasks, i.e. tasks other than to retrieve the single closest vector to a single query vector as measured by the cosine distance. In the similarity description task (and other possible future extensions) where multiple vector representations of images are added together, disregarding the vector norms could lead to very strange behavior. The rationale is that the length of each caption vector, which comes from the summation of the word vectors, is in itself a vital piece of information. If the only objective is to get the angle correct during training, there is no property ensuring that the resultant of two image vectors will make any sense, or even be the same between two training sessions.

Following this reasoning it would make more sense to minimize the L^2 norm, which implicitly takes cosine distance into consideration as

$$\|\mathbf{y} - \hat{\mathbf{y}}\|_2 = 0 \Rightarrow \mathbf{y} = \hat{\mathbf{y}} \Rightarrow (1 - \cos(\mathbf{y}, \hat{\mathbf{y}})) = 0, \quad (3.1)$$

i.e. the minimum point of the L^2 norm is also a minimum point for the cosine distance. This does however not completely reflect the fact that it for our purposes is more important to get the angle correct than to get the length correct, and to compensate for this characteristic, the loss function is established as

$$L(\mathbf{y}, \hat{\mathbf{y}}) = \alpha (1 - \cos(\mathbf{y}, \hat{\mathbf{y}})) + (1 - \alpha) \|\mathbf{y} - \hat{\mathbf{y}}\|_2, \quad (3.2)$$

where α is left as a hyperparameter.

3.2.4 Optimizer

Our problem boils down to minimization of the scalar $L(\mathbf{y}, \hat{\mathbf{y}})$ and thus, as described in section 2.2.4, a gradient descent based algorithm is well suited for solving this. There are a number of such available but as a matter of

fact it appears that much of the recently published research within machine learning does not devote a lot of energy to the choice of algorithm.

The comparison in [32] does however give a very good overview of the most popular methods including Momentum, RMSProp, and Adam, and it concludes that adaptive algorithms, where the step size is allowed to vary between dimensions depending on how frequently and how much they have been updated in the past, in many scenarios are able to find a minimum significantly faster than non-adaptive methods.

RMSProp and Adam are then the two main candidates. InceptionV3 was originally trained using RMSProp but, considering the advantages of Adam such as bias-correction, we decide to implement Adam optimization in our model.

Chapter 4

Data

This chapter contains a thorough description of the data used in this project. All steps in the collection and cleaning phases are explained. Following the machine learning conventions, there are three datasets; training data to train the models, validation data to choose the best model, and test data to evaluate the best model on unseen data. The validation and test datasets are differently structured than the training dataset, which is also justified below.

4.1 Training Data

As described in section 3.1, we wish to build our system on natural language image captions, following the hypothesis that such short snippets are able to capture the semantic essence of an image in a good way. Thus, we need a consistent dataset containing images along with short, descriptive captions.

The training dataset used in this project is the publicly available MSCOCO (Microsoft Common Objects in COntext) [33]. It is a set of 123,287 Flickr-images collected with intent to have an as high proportion of non-iconic images as possible. A non-iconic image can be thought of as an image containing more than one object or, at least, an object in real-world context. An example of an iconic and a non-iconic image of a horse can be seen in figure 4.1. After some qualitative examination, we conclude that the images in MSCOCO are good representations of what an average private photo collection could look like, which serves our purposes in this project well.



Figure 4.1: An iconic and a non-iconic image of a horse.

Each row in MSCOCO consists of an image along with a list of objects that are present in the image, the segmentations of these objects, and five short natural language captions. This is depicted in figure 4.2.



Figure 4.2: One row in the MSCOCO dataset containing an image, a list of present objects, and five short captions.

To extract the semantic information in a caption, a vector representation is computed from the sum of the word2vec embeddings of the individual words. The word addition is motivated by the linear structure in the word2vec model described in [27]. In order to improve the quality of the caption vectors, stop words are excluded and certain recurring bigrams are taken care of prior to the summation. The whole pre-processing chain of a caption is illustrated in figure 4.3.

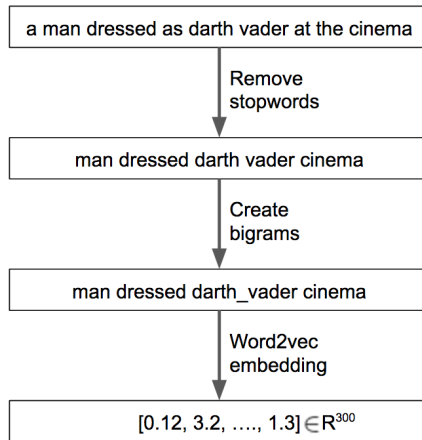


Figure 4.3: The pre-processing chain of a caption.

The training dataset then consists of 82,783 images (1/3 of the original MSCOCO data is set aside for validation and testing), each associated with five 300-dimensional vectors obtained from the five image captions. These vectors are the sought semantic representations for the images. All five vectors are stored in the dataset but for each training iteration only one is chosen as the sought output, which is meant to introduce a form of regularization on the dataset level.

4.2 Validation Data

The validation dataset is collected independently from the training dataset and is structured differently, which is justified by the following reasoning. Firstly, the goal is to build a model that can provide good semantic image retrieval and similarity description, and thus this is what we would like to measure during validation. We could measure the loss function defined for training on unseen data, for instance on the 40,500 images in the MSCOCO validation set, and hope that a smaller loss implies better performance on the two tasks. However, as the loss function is set aside as a hyperparameter, we cannot use the loss function as a measurement to choose a model since different α -values will lead to non-comparable losses. Also, since MSCOCO does not explicitly contain any information about relations between images or any classes, it is not well suited for image retrieval evaluation.

Instead we find the SUN database [34] useful for this as it consists of images with similar characteristics as in MSCOCO and as the images are there sorted into 908 classes. Some of the SUN classes are however almost identical, such as soccer stadium and soccer arena, and to avoid penalizing a model that retrieves images from two distinct but very similar classes, there is a need to manually select a subset of classes. At the same time,

we rename some of the classes to have names that, according to our best discretion, represent the classes in the best way and that do also exist in the word2vec space. The full list of the selected classes and their revised names can be found in appendix A.

After this process, the validation dataset consists of 87 classes containing 20 images each. It is then possible to evaluate the image retrieval task based on how the model, given a specific query image, ranks the images from the same class. For example, given a query image from the class freeway, how high will the remaining 19 freeway images be ranked among all images in the dataset? Moreover, when given a small set of images from the same class, the performance on the similarity description task can be evaluated based on how high the correct class name is ranked among all 87 class names. The MAP score introduced in section 2.3.1 is used for both tasks.

4.3 Test Data

The purpose of the test data is to assess the model that is chosen as the best, based on the validation. Making conclusions about the performance only from the validation process is highly inappropriate since that would allow for overfitting towards the validation data. In order to get a good picture of the performance for both image retrieval and similarity description, we use three different datasets. These are described below along with motivations to how they can be used to assess the model.

SUN

Analogously to how the validation dataset was created, we form a SUN test dataset consisting of the same 87 classes but 10 new images per class. This dataset serves the purpose of evaluating how the system could solve the original task to present semantically similar images along with a similarity description. However, since we designed this dataset and the evaluation procedure ourselves, there does not exist any benchmark results and it is hence difficult to conclude whether a test score is good or bad.

INRIA Holidays

To further evaluate the performance on the image retrieval part of the problem, we use the INRIA Holidays dataset [35]. It consists of 1,491 images, of which 500 are query images and 991 corresponding relevant images (1-3 per query image). The image retrieval can then be assessed by letting a trained model rank the 991 images given their similarity to each of the 500 query images and measure where the 1-3 relevant images are ranked.

The relevant images in the INRIA Holidays dataset tend to be visually similar to their respective query image, which makes this dataset less relevant

for our purposes. Our aim is to produce a model that is able to find high-level semantic similarities, e.g. an image of a lake might be similar to other images of lakes but does not necessarily need to be more similar if it is the same lake. However, this dataset gives us a benchmark for the image retrieval performance of our model as we can compare to the results in [5], where the MAP score is the chosen measurement.

MSCOCO

Performing similarity description for a group of images is a less explored task and we were not able to find any benchmark datasets for this task. Text retrieval is however a closely related task where, in short, the goal is to retrieve a describing text for a single query image, and for which it exists benchmark datasets. We choose to reuse the MSCOCO dataset for this, where a comparable result is given in [36].

The procedure in [36] is to randomly pick 1,000 and 5,000 images (respectively for two tests) from the 40,500 images in the MSCOCO validation dataset, where each image is also accompanied by five captions. The images are then used as queries and all the captions are ranked. The performance is measured by the R@k score that was introduced in section 2.3.2 and, in addition, the median rank of the first relevant caption is also reported.

Chapter 5

Experimental Setup

This chapter contains a description of the experimental setup, detailing the specific model-level hyperparameters as well as the training procedure.

5.1 Hyperparameters

In chapter 3, four model-level hyperparameters (in contrast to training-level hyperparameters such as e.g. batch size and learning rate) were presented as free variables, which we wanted to find an optimal value for. These were the branching layer (BL), the number of additional fully connected layers (AL), the number of trainable InceptionV3 layers (TL), and the linear combination coefficient in the loss function (α).

In our experiments, the branching is performed after layer Mixed_7b, Mixed_6d, or Mixed_6a respectively. The number of additional layers is either 1 or 3. We let none (0), 3, or all of the layers (∞) preceding the branching layer be trainable, and we let alpha equal 0 or 0.95. That is,

- $BL \in \{7b, 6d, 6a\}$,
- $AL \in \{1, 3\}$,
- $TL \in \{0, 3, \infty\}$,
- $\alpha \in \{0, 0.95\}$,

which results in a total of $3 \cdot 2 \cdot 3 \cdot 2 = 36$ models. An example can be seen in figure 3.5, where $BL=6d$, $AL=3$, $TL=3$, and $\alpha = 0.95$. (The α parameter is contained in the loss function and cannot be deduced from the graph.)

5.2 Training

Contrarily to the recommended value $\epsilon = 10^{-8}$ for the Adam optimizer in [20], we experience better results with $\epsilon = 0.01$. (The great influence of ϵ in

Adam optimization is discussed in [37] as well.) As suggested in [32], we run Adam with the learning rate initialized to 10^{-3} and no annealing schedule.

Each model is trained for 25 epochs over the MSCOCO training dataset containing 82,783 images, and the data is shuffled between the epochs. All models use mini-batches of size 32. The optimizer learning rate is initialized to 0.001 and in accordance with [32], no learning rate decay is applied. Evaluation during training to prevent overfitting is performed by measuring the average loss for the first 200 images in the MSCOCO validation dataset.

The training was run on an Ubuntu desktop machine with a six-core i7-5820K CPU @ 3.30 GHz, 32 GB of main memory, and a GeForce GTX 1080 GPU with 8 GB of memory.

Chapter 6

Results

This chapter aims to answer the fourth question in section 1.2; “Can we implement a system like this?”. We are also interested in how good the model can get following this approach and in order to explore that, a grid search over the four model-level hyperparameters is performed. The results are presented for the training, validation, and test setups respectively, which is followed by some examples from our simple demo. A complete list of both training and evaluation results for all models can be found in appendix B.

6.1 Training

As accounted for in section 3.2.3, the loss function is a linear combination of the L^2 distance and the cosine distance between the predicted vector representation for an image and the corresponding caption vector resultant. Since it is not relevant to compare the loss function values between runs when α is varying, we instead account for the two components separately in the following.

In figure 6.1, the cosine distance and the L^2 distance are plotted as functions of the number of training images presented to the network (over 25 epochs), respectively. The orange and the red curves correspond to the training and validation loss for the $\{\text{BL}=7\text{b}, \text{AL}=3, \text{TL}=\infty, \alpha = 0.95\}$ model, while the turquoise and the blue curves correspond to the training and validation loss for the $\{\text{BL}=7\text{b}, \text{AL}=1, \text{TL}=3, \alpha = 0.95\}$ model.

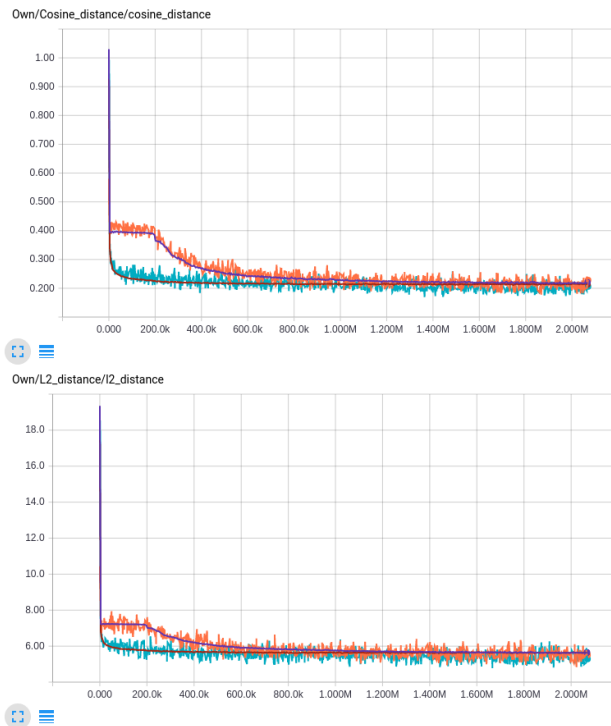


Figure 6.1: TensorBoard summary for the $\{\text{BL}=7\text{b}, \text{AL}=3, \text{TL}=\infty, \alpha = 0.95\}$ model (orange for train loss and blue for validation loss) and for the $\{\text{BL}=7\text{b}, \text{AL}=1, \text{TL}=3, \alpha = 0.95\}$ model (turquoise for train and red for validation).

We can draw a number of conclusions from the graphs in figure 6.1. Firstly, both the cosine and the L^2 part of the loss function show nice behavior in that they quickly adjust from the randomly initialized weights for the additional layers to a more reasonable setting, which is then slowly improved upon.

Secondly, we cannot see any signs of overfitting. The validation curves follows the training curves closely the whole way.

Thirdly, it seems like both models have almost reached convergence. This is based on the looks of the loss functions and holds true for all 36 models. As in all machine learning applications, concluding convergence is however a delicate matter, since there is no established method that always works.

Fourthly, we notice a clear saddle point between approximately 0 and $2 \cdot 10^5$ training instances for the $\{\text{BL}=7\text{b}, \text{AL}=3, \text{TL}=\infty, \alpha = 0.95\}$ model. It is reasonable that the model with more additional layers and more re-trainable InceptionV3 layers is the one encountering a saddle point. That is since, in general, the more complex the model is, the longer it will need in order to adjust its weights to the problem.

6.2 Validation

This section presents the MAP scores for the image retrieval and similarity description tests on the validation part of the SUN dataset described in 4.2. The appropriateness of the chosen loss function is evaluated, and the results are then conditioned on each of the free model parameters in order to investigate their impact.

6.2.1 Loss function

As explained in section 3.2.3, the loss function was derived to be a proxy for what we actually wanted to minimize but which was not possible to integrate in the backpropagation framework. In order to evaluate this choice of loss function, we look at the correlation between it and the MAP scores based on the validation data. We wish that if a model performs well (gets a low value) for the loss function, that should imply that it will perform well (get a high value) on the validation data, as measured by the MAP scores. It should be noted that this also indirectly evaluates the quality of the structure of the training data, since a small loss means that the model is indeed able to map the images to the embeddings of the captions in the word2vec space.

One way of confirming that the chosen loss function is a good proxy for image retrieval and similarity description is by looking at the table in appendix B. It is sorted decreasingly by image retrieval MAP, and we see the tendency that the better models w.r.t. image retrieval and similarity description reach lower values for the loss function (with some exceptions). It is also clear that the two tasks of image retrieval and similarity description are correlated since models that perform well on one task also perform well on the other.

Another way to visualize this tendency is by plotting the MAP scores as a function of the loss values, which is done in figure 6.2. The cases for $\alpha = 0$ and $\alpha = 0.95$ are separated since the magnitude of the loss function is heavily affected by α . The straight line that best fits the data is also added to each plot to show the trend. Classic statistical hypothesis testing of whether the slope is negative confirms that there indeed exists a relationship on the significance level 0.05 for all four cases.

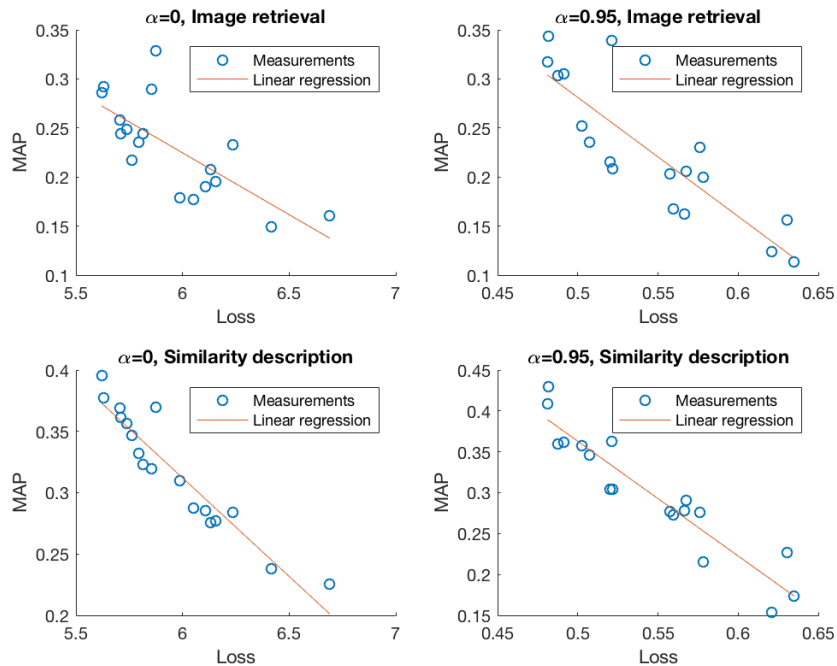


Figure 6.2: The MAP score for the image retrieval and similarity description tests plotted as a function of the loss.

6.2.2 Hyperparameters

From the results presented in appendix B, patterns can be found between some of the model parameters and the performance. Below, the results are listed conditioned on each of the four model parameters, which helps in order to find such suggested implications between the parameter values and the model performance. Possible explanations to these effects are given when they can be found.

Branching Layer

When conditioning on the layer at which we branch from the pre-trained InceptionV3 network, we obtain the results in table 6.1. As can be seen, BL=7a consistently outperforms BL=6d, which in turn almost consistently gives better performance than BL=6a.

Since we are striving for a model that is able to find *semantic* similarities between images, this is not a surprising result. The high-level features in the later layers of the InceptionV3 network should be more useful in this regard than low-level features in the earlier layers.

Table 6.1: Influence of branching layer on the MAP score for similarity description and image retrieval. Bold text indicates best value for that row in the respective test.

	Image retrieval			Similarity description		
	BL=6a	BL=6d	BL=7b	BL=6a	BL=6d	BL=7b
AL=1, TL=0, $\alpha=0$	0.1604	0.2326	0.3284	0.2251	0.2842	0.3692
AL=1, TL=0, $\alpha=0.95$	0.1558	0.2303	0.3395	0.2272	0.2763	0.3623
AL=3, TL=0, $\alpha=0$	0.1488	0.2073	0.2898	0.2378	0.2753	0.3196
AL=3, TL=0, $\alpha=0.95$	0.1133	0.1237	0.1998	0.1736	0.1536	0.2156
AL=1, TL=3, $\alpha=0$	0.1953	0.2488	0.2863	0.2769	0.3561	0.3957
AL=1, TL=3, $\alpha=0.95$	0.2061	0.2355	0.3438	0.2902	0.3455	0.4301
AL=3, TL=3, $\alpha=0$	0.1768	0.2353	0.2921	0.2872	0.3324	0.3777
AL=3, TL=3, $\alpha=0.95$	0.1623	0.2085	0.3051	0.2779	0.3045	0.3620
AL=1, TL= ∞ , $\alpha=0$	0.1903	0.2168	0.2444	0.2852	0.3467	0.3615
AL=1, TL= ∞ , $\alpha=0.95$	0.2028	0.2522	0.3174	0.2772	0.3577	0.4092
AL=3, TL= ∞ , $\alpha=0$	0.1787	0.2438	0.2585	0.3096	0.3233	0.3688
AL=3, TL= ∞ , $\alpha=0.95$	0.1674	0.2150	0.3037	0.2733	0.3047	0.3600

Additional Layers

In table 6.2, the results are conditioned on the number of additional layers added after branching. We can see that one additional layer in general outperforms three additional layers for both image retrieval and similarity description, with more significant differences in MAP scores.

One explanation could be that as the InceptionV3 network was trained to predict the presence of objects and since the task of embedding an image into the word2vec space is somewhat similar, there is no need to increase the complexity by adding more than one additional layer. Another explanation could be that due to the complexity of having three additional layers, longer training could be needed to achieve similar results. As mentioned, we seem to reach convergence for all models, but it is of course possible that it is a question about saddle points for the more complex models.

Table 6.2: Influence of the number of additional layers on the MAP scores.

	Image retrieval		Similarity description	
	AL=1	AL=3	AL=1	AL=3
BL=7b, TL=0, $\alpha=0$	0.3284	0.2898	0.3695	0.3196
BL=7b, TL=0, $\alpha=0.95$	0.3395	0.1998	0.3623	0.2156
BL=7b, TL=3, $\alpha=0$	0.2863	0.2921	0.3957	0.3777
BL=7b, TL=3, $\alpha=0.95$	0.3438	0.3051	0.4301	0.3620
BL=7b, TL= ∞ , $\alpha=0$	0.2444	0.2585	0.3615	0.3688
BL=7b, TL= ∞ , $\alpha=0.95$	0.3174	0.3037	0.4092	0.3600
BL=6d, TL=0, $\alpha=0$	0.2326	0.2073	0.2842	0.2753
BL=6d, TL=0, $\alpha=0.95$	0.2303	0.1237	0.2763	0.1536
BL=6d, TL=3, $\alpha=0$	0.2488	0.2353	0.3561	0.3324
BL=6d, TL=3, $\alpha=0.95$	0.2355	0.2085	0.3455	0.3045
BL=6d, TL= ∞ , $\alpha=0$	0.2168	0.2438	0.3467	0.3233
BL=6d, TL= ∞ , $\alpha=0.95$	0.2522	0.2150	0.3577	0.3047
BL=6a, TL=0, $\alpha=0$	0.1604	0.1488	0.2251	0.2378
BL=6a, TL=0, $\alpha=0.95$	0.1558	0.1133	0.2272	0.1736
BL=6a, TL=3, $\alpha=0$	0.1953	0.1768	0.2769	0.2872
BL=6a, TL=3, $\alpha=0.95$	0.2061	0.1623	0.2902	0.2779
BL=6a, TL= ∞ , $\alpha=0$	0.1903	0.1787	0.2852	0.3096
BL=6a, TL= ∞ , $\alpha=0.95$	0.2028	0.1674	0.2772	0.2733

Loss function α

The results conditioned on the two loss functions are shown in table 6.3. Compared to the previous two parameters, the difference between the two loss functions is minimal and could simply be due to random initialization. However, we can see that for BL=7b, it seems like the loss function behaves as discussed in section 3.4.2, where $\alpha = 0.95$ performs significantly better on image retrieval.

Table 6.3: Influence of loss function parameter α on the MAP scores.

	Image retrieval		Similarity description	
	$\alpha=0$	$\alpha=0.95$	$\alpha=0$	$\alpha=0.95$
BL=7b, TL=0, AL=1	0.3284	0.3395	0.3695	0.3623
BL=7b, TL=0, AL=3	0.2898	0.1998	0.3196	0.2156
BL=7b, TL=3, AL=1	0.2863	0.3438	0.3957	0.4301
BL=7b, TL=3, AL=3	0.2921	0.3051	0.3777	0.3620
BL=7b, TL= ∞ , AL=1	0.2444	0.3174	0.3615	0.4092
BL=7b, TL= ∞ , AL=3	0.2585	0.3037	0.3688	0.3600
BL=6d, TL=0, AL=1	0.2326	0.2303	0.2842	0.2763
BL=6d, TL=0, AL=3	0.2073	0.1237	0.2753	0.1536
BL=6d, TL=3, AL=1	0.2488	0.2355	0.3561	0.3455
BL=6d, TL=3, AL=3	0.2353	0.2085	0.3324	0.3045
BL=6d, TL= ∞ , AL=1	0.2168	0.2522	0.3467	0.3577
BL=6d, TL= ∞ , AL=3	0.2438	0.2150	0.3233	0.3047
BL=6a, TL=0, AL=1	0.1604	0.1558	0.2251	0.2272
BL=6a, TL=0, AL=3	0.1488	0.1133	0.2378	0.1736
BL=6a, TL=3, AL=1	0.1953	0.2061	0.2769	0.2902
BL=6a, TL=3, AL=3	0.1768	0.1623	0.2872	0.2779
BL=6a, TL= ∞ , AL=1	0.1903	0.2028	0.2852	0.2772
BL=6a, TL= ∞ , AL=3	0.1787	0.1674	0.3096	0.2733

Training layers

Finally, the results when conditioning on the number of training layers are shown in table 6.4. It seems like it is advantageous to allow the network to update some layers before the branching, while it in most cases is less important if only three or all layers are updated. This behavior could have a potential explanation in the fact that the earliest layers represent very basic features, which there is no reason to relearn between different tasks, and so it is sufficient to re-train the last three layers before branching.

Table 6.4: Influence of the number of training layers on the MAP scores.

	Image retrieval			Similarity description		
	TL=0	TL=3	TL= ∞	TL=0	TL=3	TL= ∞
BL=7b, AL=1, $\alpha=0$	0.3284	0.2863	0.2444	0.3695	0.3957	0.3615
BL=7b, AL=1, $\alpha=0.95$	0.3395	0.3438	0.3174	0.3623	0.4301	0.4092
BL=7b, AL=3, $\alpha=0$	0.2898	0.2921	0.2585	0.3196	0.3777	0.3688
BL=7b, AL=3, $\alpha=0.95$	0.1998	0.3051	0.3037	0.2156	0.3620	0.3600
BL=6d, AL=1, $\alpha=0$	0.2326	0.2488	0.2168	0.2842	0.3561	0.3467
BL=6d, AL=1, $\alpha=0.95$	0.2303	0.2355	0.2522	0.2763	0.3455	0.3577
BL=6d, AL=3, $\alpha=0$	0.2073	0.2353	0.2438	0.2753	0.3324	0.3233
BL=6d, AL=3, $\alpha=0.95$	0.1237	0.2085	0.2150	0.1536	0.3045	0.3047
BL=6a, AL=1, $\alpha=0$	0.1604	0.1953	0.1903	0.2251	0.2769	0.2852
BL=6a, AL=1, $\alpha=0.95$	0.1558	0.2061	0.2028	0.2272	0.2902	0.2772
BL=6a, AL=3, $\alpha=0$	0.1488	0.1768	0.1787	0.2378	0.2872	0.3096
BL=6a, AL=3, $\alpha=0.95$	0.1133	0.1623	0.1674	0.1736	0.2779	0.2733

6.3 Test

Based on the validation scores presented in the previous section, we find the $\{BL=7b, AL=1, TL=3, \alpha = 0.95\}$ model, i.e. the one branching at layer 7b, having one additional layer, three re-trainable InceptionV3 layers and using the mixed loss function that takes both the cosine and the L^2 distances into account, to be the best model. It gets a MAP of 0.3438 on the image retrieval and 0.4301 on the similarity description, which is the highest in each category for all models. This section contains an evaluation of this model, first on the SUN test dataset and then on the two benchmark datasets introduced in section 4.3.

6.3.1 SUN

Table 6.5 shows the MAP scores on both the SUN test dataset and the SUN validation dataset. The scores are lower for the test dataset which could be due to the generalization error when picking the best out of 36 models on a certain dataset, and which then motivates the construction of a separate SUN test dataset with unseen images.

Table 6.5: Validation compared to test MAP score on the SUN dataset.

	Image retrieval	Similarity description
Validation	0.3438	0.4301
Test	0.3269	0.3974

To give a better intuition for the rather non-transparent MAP metric, figure 6.3 contains two histograms showing the number of times the relevant

images and labels are ranked at each position, respectively. These are aggregated over all queries for the two tests but the image retrieval histogram is cut off at 250 on the x-axis to improve legibility. As can be seen, the raw numbers behind the MAP scores indicate that the model shows good behavior and reasonable performance in that it ranks relevant instances more frequently at the positions closer to the top.

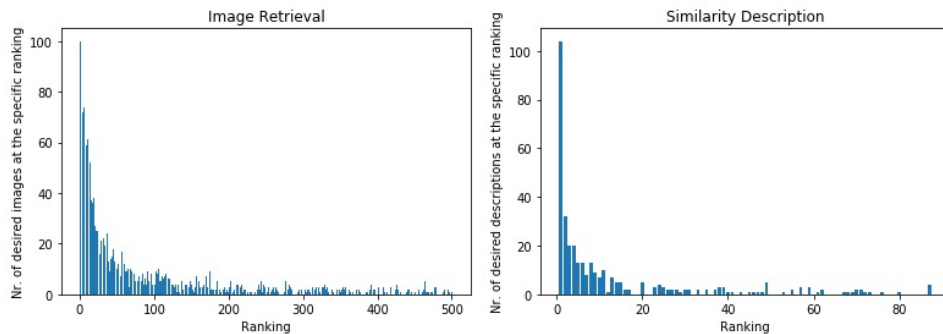


Figure 6.3: Histogram showing the ranking of relevant images (left) and labels (right) aggregated over all queries in respective test.

6.3.2 Benchmark

The model is finally evaluated on the two benchmark datasets introduced in 4.3; INRIA Holidays and MSCOCO. In table 6.6, the results on the INRIA dataset and the benchmark score from [5] can be seen. Our model has a lower MAP score which is expected as the model in [5] was designed specifically for this type of image retrieval. However, if examining the properties of the MAP score, one can see that it rewards a model more for moving a relevant images from rank 3 to 2 than from rank 10 to 9. Furthermore, if looking at a query image with one relevant image in the database, the ranking of the relevant image would be $\frac{1}{\text{MAP}}$ in order to reproduce the MAP score, which gives an average ranking of 1.5 for [5] and around 1.75 for our model. With this reasoning, our results on the INRIA dataset are not bad, considering that our model is designed to also solve the similarity description task simultaneously.

Table 6.6: Benchmark of image retrieval on the INRIA dataset.

	Babenko et al. [5]	Our model
MAP	0.75	0.57

The results on the MSCOCO dataset are presented in table 6.7, where [36] is used as a benchmark result. We can again see that our model is slightly inferior, which once again is expected since it is not trained for the task that

is tested for here. It should also be noted that the median is calculated from the average of the five best models in [36], which explains why it is not an integer.

Table 6.7: Benchmark of text retrieval on the MSCOCO dataset for 1,000 and 5,000 images respectively.

	1,000 images				5,000 images			
	R@1	R@5	R@10	Median	R@1	R@5	R@10	Median
Karpathy et al. [36]	29.4	62.0	75.9	2.5	11.8	32.5	45.4	12.2
Our model	27.8	57.9	72.0	4	11.1	28.4	38.7	19

6.4 Demo

In order to give a final picture of the performance of our model, figure 6.4 depicts a simple demo, where the unseen images in the MSCOCO validation dataset are used to represent a user’s private photo collection. The bigger pictures represent the queries and the smaller images at the bottom are the most semantically similar images retrieved by our model. The bold words between the queries and the retrieved images are the similarity descriptions found by the model.



Figure 6.4: A simple demo showing the created system in action when fed with unseen images that should represent a user’s private photo collection.

Chapter 7

Discussion

This chapter contains reflections on the work process and the results. We try to mention potential limiting factors and drawbacks for the derived model while also justifying some of the decisions made during the modeling phase. Lastly, a discussion on possible directions for related future work is included.

Firstly, the chosen approach to try to solve both the image retrieval and the similarity description problem with a single model does not guarantee the best possible performance. So far, humanity has been able to construct weak AIs (models trained on narrow tasks) with very convincing results, but building strong AIs (models able to handle multiple tasks) has proven to be much more challenging. Therefore, one could argue that breaking a complex problem into multiple more concrete subtasks and training separate models for these would be a more reasonable approach. However, from a business and implementational point of view, there is a value in having fewer models. In addition, we were appealed by the neatness and simplicity of the chosen approach.

Secondly, each image in MSCOCO is associated with five captions, as depicted in figure 4.2. These captions are of course similar since they describe the same image but they often capture different details and nuances. A reasonable approach could have been to sum the vectors for all words in all captions belonging to the same image and to use this resultant as the single representation for that image. Instead of doing that, we associate each image with all five vectors and choose one at random during training. This should also introduce an intrinsic form of regularization as it makes it impossible for the network to learn an exact representation for each image, but rather forces it to learn patterns that generalize better to unseen images.

Also regarding the training data, it should be mentioned that a substantial portion of the MSCOCO dataset was not used in this project. When downloading the data it was divided into 82,783 training images and 40,504 validation images. In order to speed up the training, only 200 of the validation images were used, and approximately 6,000 more were used for testing.

Thus the remaining validation images could have been used for training, which in theory could have improved the performance of the model. Since we do not seem to be limited by overfitting we do however not expect this to have affected too much.

As mentioned in chapter 6.1, it is always difficult to conclude that a model has converged. However, a four times longer training session using all the available data (cf. previous paragraph) showed that there was no further decrease in validation error. Logging and studying more quantities, such as the magnitude of the weight updates, throughout training could nonetheless have contributed to this view.

We constructed a new validation and test dataset according to the particular characteristics of our problem. It served the purpose to allow reliable comparisons between different models very well, but it made it on the other hand difficult to make comparisons with previous research. The absence of ways to directly compare or interpret results is in general of course not desirable and could open up for questions about the difficulty of the tests and, by extension, the actual performance. That is why we decided to also include two benchmark datasets and the demo in the evaluation.

With the twofold problem formulation to both find semantically similar photos as well as to describe this similarity, it was hard to find suitable benchmark datasets, especially for the similarity description part. For the two datasets we decided to use, we had to compare our results with those of models that were specifically designed for either image or text retrieval and these test datasets. This brought about a risk to mislead the reader, but it was a necessary step in order to obtain some sort of reality connection. In the end, it also indicated that our model performed at least decently on each dataset, while, as expected, not outperforming the benchmark models.

It is interesting to note that model seems to be able to distinguish between red and green buses (or possibly between single- and double-deckers) in the two lower examples in figure 6.4. We are able to find more examples like this, and it is supposedly thanks to the nuances included in the captions. Thus, for a model built on tags to be able to do this, it would require that there were specific classes in the training dataset for both the number of decks and for colors.

In order to get a better understanding for where the model fails and why the MAP scores are not even higher, figure 7.1 depicts four image retrieval queries. The leftmost image in each row is the query image and the following four are the most similar images in the presented order, according to model $\{\text{BL}=7\text{b}, \text{AL}=1, \text{TL}=3, \alpha = 0.95\}$. The associated ground truth class is stated right above each image and shows that the models face a challenging evaluation and that there is sometimes a logical explanation behind the misclassifications.

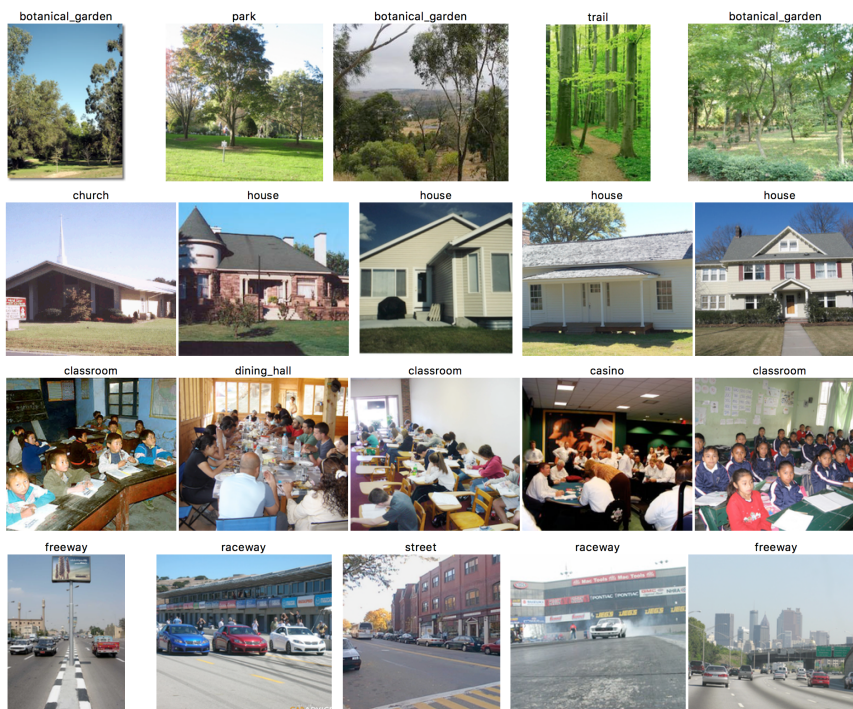


Figure 7.1: Four image retrieval queries from the SUN test dataset, where the leftmost image in each row is the query image and the following four are the most similar images according to model $\{BL=7b, AL=1, TL=3, \alpha = 0.95\}$.

In total, based on all the presented results, we mean that the approach to construct an integrated embedding space for words and images was well suited for the given problem and that the described implementation solves it.

7.1 Future Work

We have in this thesis mainly focused on the information retrieval task with images as queries while retrieving either images or words. However, by integrating the image and word embeddings, it immediately becomes possible to also retrieve images from textual queries, sometimes called text based image retrieval. An example of this is shown in figure 7.2, where the images come from the unseen MSCOCO validation data. It would be very interesting to further explore and improve this part of the system.

With the approach to embed images into the pre-defined word2vec space, we have also inherited many of the properties of this space, e.g. the cosine distance as similarity measure, which were first stated in [25]. This raises questions about whether the integrated vector space possesses other prop-



Figure 7.2: Showcase of text as query for retrieving images, the text to the left is the query for the five images to the right.

erties that could be exploited to further enhance the performance on the mentioned tasks. It is especially interesting for the similarity description task as it directly builds on finding a connection between images and a language. This could be done in many other ways than by simply choosing the closest word to a group of images, and much work could be put into this subtask.

As discussed in chapter 3, we have treated the word2vec space as static and thus we did not evaluate whether changing the setup of it could improve the integration of images. Based on the idiom that “a picture is worth a thousand words”, it could be reasoned that if a 300-dimensional vector embedding is suitable for words, a higher dimensionality might be favorable for image embeddings. It would be interesting to investigate how different embedding dimensions affects the performance on the two tasks and if it introduces a trade-off between them as a too high dimensionality possibly could reduce the word embedding quality.

Finally, we believe that the work in this thesis opens up for new and innovative solutions to problems related to both the image and word domains. One such example is automatic album generation and labeling through cluster analysis, which would be very easy to implement with the presented solution. The additive property of the introduced vector space could also be used to obtain representations of a user’s browsing and search histories,

which in turn could be used to create a more personalized photo management experience. Some of these ideas are depicted in figure 7.3 but that is of course only a fraction of the possible extensions to our model.

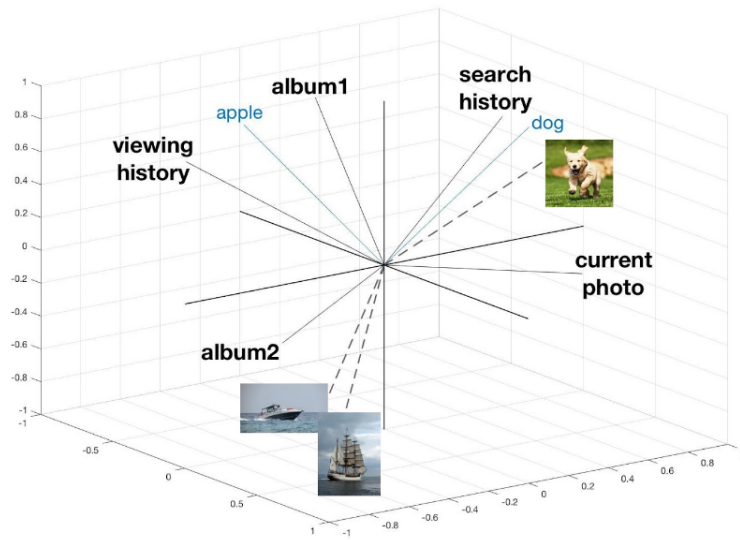


Figure 7.3: Some of the potential extensions to the presented solution.

Chapter 8

Conclusion

The problem faced in this thesis emerged from the need of an intelligent way to improve a home user’s photo browsing experience by suggesting images from their own private collection. It was formulated as to, given a currently viewed image, present semantically similar images along with a short similarity description, which was then in turn converted into four direct research questions.

In response to the multimodal nature of our problem, we decided to build our system on an integrated image and word embedding model. The idea was to create a structure where similarity was directly related to distance so that we, by evaluating distances between the embedded images and words, would be able to solve both the image retrieval and the similarity description part of the problem in parallel. As word2vec already possessed this property for word embeddings, the approach became to construct a model that could embed images into this already pre-defined 300-dimensional vector space.

The problem was stated as a supervised learning problem with images as input and the word2vec embeddings of short, manually annotated captions as sought output. A convolutional neural network was chosen as model based on the complexity of the problem and the method’s recent successes in computer vision. More specifically, a pre-trained object recognition network called InceptionV3 was used as a base and transfer learning was then applied towards the specific embedding task.

A validation dataset and scheme was manually created and used to investigate the impact of four different model-level hyperparameters. This showed among other things an increase in performance when branching at a later layer for transfer learning, and a best model was also chosen based on the validation. This model was then thoroughly evaluated, both on the manually created scheme and on two benchmark datasets for image and text retrieval, as well as qualitatively through a simple demo. Based on the combined results of these evaluations, we were able to conclude that the proposed model was well suited for the given problem.

Chapter 9

Work Division

It has throughout the whole project been important to both of us authors to be involved with each part of the work as far as possible. As a result, both authors have contributed equally to the full work. However, when we deemed that we would achieve better results from assigning a certain task to one person, we ensured that the other person at least retained full insight into every detail of the process. An example of such a task is the coding part of the implementation, where David has written most of the code for the preprocessing of the data, while Tobias has written most of the code for setting up and training the network.

Appendix A

SUN Scene Names

The 87 scenes that were selected from the SUN database [34] are listed below with their old names to the left and their new names to the right. To find the discarded scenes, compare with the 908 scenes at <http://vision.princeton.edu/projects/2010/SUN/explore/>.

airport/airport	airport	ice_cream_parlor	ice_cream_parlor
alley	alley	industrial_area	industry
amusement_park	amusement_park	jail/indoor	prison
art_gallery	art_gallery	kiosk/outdoor	kiosk
bakery/shop	bakery	kitchen	kitchen
bar	bar	labyrinth/outdoor	labyrinth
basketball_court/indoor	basketball	lake/natural	lake
bathroom	bathroom	library/indoor	library
beach	beach	market/outdoor	market
beer_garden	outdoor_seating	mini_golf_course/outdoor	minigolf
bicycle_racks	bike	mosque/outdoor	mosque
botanical_garden	botanical_garden	mountain	mountain
bowling_alley	bowling	mountain_path	hiking_trail
boxing_ring	boxing	movie_theater/indoor	cinema
brewery/indoor	brewing	office	office
bridge	bridge	park	park
bus_depot/outdoor	bus	pasture	pasture
canal/urban	canal	patio	patio
casino/indoor	casino	playground	playground
castle	castle	plaza	square
cathedral/indoor	cathedral_nave	raceway	raceway
cemetery	cemetery	restaurant	restaurant
chalet	chalet	river	river
church/outdoor	church	sauna	sauna
city	city	sea_cliff	cliff
classroom	classroom	skatepark	skateboarding
desert/sand	desert	ski_slope	skiing
dining_hall	dining_hall	sky	sky
discotheque	nightclub	stadium/baseball	baseball
dock	dock	stadium/outdoor	arena
driving_range/outdoor	golf	stage/outdoor	concert
factory/indoor	factory	staircase	stairs
field/wild	field	street	street
field_road	highroad	swimming_pool/outdoor	swimming_pool
fishpond	pond	tennis_court/outdoor	tennis
florist_shop/indoor	florist	tent/outdoor	tenting
forest_path	trail	tower	tower
fountain	fountain	train_station/platform	station
freeway	freeway	underwater/ocean_shallow	underwater
gymnasium/indoor	gym	waterfall/plunge	waterfall
harbor	harbor	wine_cellar/bottle_storage	wine_cellar
hospital_room	hospital	ziggurat	ziggurat
hot_tub/outdoor	hot_tub	zoo	zoo
house	house		

Appendix B

Full Results

The results for training and validation for all models are listed below and sorted in descending order by image retrieval score.

BL	TL	AL	alpha	cos	L2	mixed	Image Retrieval	Similarity Description
7b	3	1	0,95	0,2124	5,597	0,4816	0,3438	0,4301
7b	0	1	0,95	0,2372	5,916	0,5212	0,3395	0,3623
7b	0	1	0	0,2376	5,876	5,876	0,3284	0,3695
7b	all	1	0,95	0,2120	5,596	0,4812	0,3174	0,4092
7b	3	3	0,95	0,2193	5,655	0,4911	0,3051	0,3620
7b	all	3	0,95	0,2170	5,629	0,4876	0,3037	0,3600
7b	3	3	0	0,2182	5,631	5,631	0,2921	0,3777
7b	0	3	0	0,2370	5,854	5,854	0,2898	0,3196
7b	3	1	0	0,2132	5,621	5,621	0,2863	0,3957
7b	all	3	0	0,2254	5,795	5,705	0,2585	0,3688
6d	all	1	0,95	0,2259	5,758	0,5025	0,2522	0,3577
6d	3	1	0	0,2260	5,739	5,739	0,2488	0,3561
7b	all	1	0	0,2233	5,710	5,71	0,2444	0,3615
6d	all	3	0	0,2353	5,814	5,814	0,2438	0,3233
6d	3	1	0,95	0,2285	5,802	0,5072	0,2355	0,3455
6d	3	3	0	0,2332	5,796	5,796	0,2353	0,3324
6d	0	1	0	0,2723	6,237	6,237	0,2326	0,2842
6d	0	1	0,95	0,2748	6,303	0,5762	0,2303	0,2763
6d	all	1	0	0,2277	5,762	5,762	0,2168	0,3467
6d	all	3	0,95	0,2390	5,867	0,5204	0,2150	0,3047
6d	3	3	0,95	0,2403	5,871	0,5219	0,2085	0,3045
6d	0	3	0	0,2636	6,131	6,131	0,2073	0,2753
6a	3	1	0,95	0,2691	6,235	0,5674	0,2061	0,2902
6a	all	1	0,95	0,2628	6,155	0,5574	0,2028	0,2772
7b	0	3	0,95	0,2778	6,287	0,5783	0,1998	0,2156
6a	3	1	0	0,2622	6,154	6,154	0,1953	0,2769
6a	all	1	0	0,2581	6,109	6,109	0,1903	0,2852
6a	all	3	0	0,2521	5,986	5,986	0,1787	0,3096
6a	3	3	0	0,2574	6,051	6,051	0,1768	0,2872
6a	all	3	0,95	0,2660	6,143	0,5598	0,1674	0,2733
6a	3	3	0,95	0,2710	6,182	0,5665	0,1623	0,2779
6a	0	1	0	0,3236	6,691	6,691	0,1604	0,2251
6a	0	1	0,95	0,3147	6,634	0,6307	0,1558	0,2272
6a	0	3	0	0,2944	6,416	6,416	0,1488	0,2378
6d	0	3	0,95	0,3078	6,568	0,6208	0,1237	0,1536
6a	0	3	0,95	0,3185	6,646	0,6349	0,1133	0,1736

Bibliography

- [1] M. Yasmin, S. Mohsin, and M. Sharif, “Intelligent image retrieval techniques: a survey,” *Journal of applied research and technology*, vol. 12, no. 1, pp. 87–103, 2014.
- [2] L. Zheng, Y. Yang, and Q. Tian, “Sift meets cnn: a decade survey of instance retrieval,” *arXiv preprint arXiv:1608.01807*, 2016.
- [3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
- [4] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.
- [5] A. Babenko, A. Slesarev, A. Chigorin, and V. Lempitsky, “Neural codes for image retrieval,” in *European conference on computer vision*. Springer, 2014, pp. 584–599.
- [6] D. Kiela and L. Bottou, “Learning image embeddings using convolutional neural networks for improved multi-modal semantics.” in *EMNLP*. Citeseer, 2014, pp. 36–45.
- [7] A. Frome, G. S. Corrado, J. Shlens, S. Bengio, J. Dean, T. Mikolov *et al.*, “Devise: A deep visual-semantic embedding model,” in *Advances in neural information processing systems*, 2013, pp. 2121–2129.
- [8] H. Pham, “Implications of multimodal deep learning for textual and visual data.”
- [9] J. Dong, X. Li, and C. G. Snoek, “Word2visualvec: Cross-media retrieval by visual feature prediction,” *arXiv preprint arXiv:1604.06838*, 2016.
- [10] C. M. Bishop, “Pattern recognition,” *Machine Learning*, vol. 128, pp. 1–58, 2006.

- [11] Y.-i. Ohta, T. Kanade, and T. Sakai, “An analysis system for scenes containing objects with substructures,” in *Proceedings of the Fourth International Joint Conference on Pattern Recognitions*, 1978, pp. 752–754.
- [12] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.
- [13] H. Landahl, W. S. McCulloch, and W. Pitts, “A statistical consequence of the logical calculus of nervous nets,” *Bulletin of Mathematical Biology*, vol. 5, no. 4, pp. 135–137, 1943.
- [14] B. Widrow, M. E. Hoff *et al.*, “Adaptive switching circuits,” in *IRE WESCON convention record*, vol. 4, no. 1. New York, 1960, pp. 96–104.
- [15] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, “Imagenet large scale visual recognition challenge,” *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.
- [16] D. H. Hubel and T. N. Wiesel, “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex,” *The Journal of physiology*, vol. 160, no. 1, pp. 106–154, 1962.
- [17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [18] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.
- [19] S. J. Pan and Q. Yang, “A survey on transfer learning,” *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.
- [20] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [21] T. Tieleman and G. Hinton, “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude,” *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, 2012.
- [22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

- [23] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.
- [24] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.
- [25] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *arXiv preprint arXiv:1301.3781*, 2013.
- [26] Z. S. Harris, “Distributional structure,” *Word*, vol. 10, no. 2-3, pp. 146–162, 1954.
- [27] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in *Advances in neural information processing systems*, 2013, pp. 3111–3119.
- [28] M. U. Gutmann and A. Hyvärinen, “Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics,” *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 307–361, 2012.
- [29] N. Silberman, N. Wu, A. Kurakin, P. Ledbetter, J. Tunney, D. J. Lasiman, and L. Lee. (2017) Tensorflow-slim image classification library. [Online]. Available: <https://github.com/tensorflow/models/blob/master/slim/README.md>
- [30] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.
- [31] J. Yue-Hei Ng, F. Yang, and L. S. Davis, “Exploiting local features from deep networks for image retrieval,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2015, pp. 53–61.
- [32] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv preprint arXiv:1609.04747*, 2016.
- [33] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft coco: Common objects in context,” in *European Conference on Computer Vision*. Springer, 2014, pp. 740–755.

- [34] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba, “Sun database: Large-scale scene recognition from abbey to zoo,” in *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*. IEEE, 2010, pp. 3485–3492.
- [35] H. Jégou, M. Douze, and C. Schmid, “Hamming embedding and weak geometry consistency for large scale image search-extended version,” 2008.
- [36] A. Karpathy and L. Fei-Fei, “Deep visual-semantic alignments for generating image descriptions,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3128–3137.
- [37] S. Funk. (2015) Rmsprop loses to smorms3 - beware the epsilon! [Online]. Available: <http://sifter.org/~simon/journal/20150420.html>