Unsupervised Pretraining of Neural Networks with Multiple Targets using Siamese Architectures

Von der Fakultät für Mathematik und Informatik
der Universität Leipzig
angenommene

DISSERTATION

zur Erlangung des akademischen Grades

DOCTOR RERUM NATURALIUM
(Dr. rer. nat.)

im Fachgebiet

Informatik

Vorgelegt
von Herr M. Sc. Maximilian Bryan
geboren am 31. Mai 1988 in Bremen, Deutschland.

Die Annahme der Dissertation wurde empfohlen von:

1. Prof. Dr. Gerhard Heyer (Universität Leipzig)
2. Prof. Dr. Antoine Doucet (La Rochelle Université, Frankreich)

Die Verleihung des akademischen Grades erfolgt mit Bestehen der Verteidigung am 05. Oktober 2021 mit dem Gesamtprädikat *cum laude*.

Contents

List of Figures

List of Tables

Abstract

A model's response for a given input pattern depends on the seen patterns in the training data. The larger the amount of training data, the more likely edge cases are covered during training. However, the more complex input patterns are, the larger the model has to be. For very simple use cases, a relatively small model can achieve very high test accuracy in a matter of minutes. On the other hand, a large model has to be trained for multiple days. The actual time to develop a model of that size can be considered to be even greater since often many different architecture types and hyper-parameter configurations have to be tried.

An extreme case for a large model is the recently released *GPT-3* model. This model consists of 175 billion parameters and was trained using 45 terabytes of text data. The model was trained to generate text and is able to write news articles and source code based only on a rough description. However, a model like this is only creatable for researchers with access to special hardware or immense amounts of data. Thus, it is desirable to find less resource-intensive training approaches to enable other researchers to create well performing models.

This thesis investigates the use of pre-trained models. If a model has been trained on one dataset and is then trained on another similar data, it faster learns to adjust to similar patterns than a model that has not yet seen any of the task's pattern. Thus, the learned lessons from one training are *transferred* to another task. During pre-training, the model is trained to solve a specific task like predicting the next word in a sequence or first encoding an input image before decoding it. Such models contain an encoder and a decoder part. When transferring that model to another task, parts of the model's layers will be removed. As a result, having to discard fewer weights results in faster training since less time has to be spent on training parts of a model that are only needed to solve an auxiliary task.

Throughout this thesis, the concept of *siamese architectures* will be discussed since when using that architecture, no parameters have to be discarded when transferring a model trained with that approach onto another task. Thus, the siamese pre-training approach positively impacts the need for resources like time and energy use and drives the development of new models in the direction of Green AI. The models trained with this approach will be evaluated by comparing them to models trained with other pre-training approaches as well as large existing models. It will be shown that the models trained for the tasks in this thesis perform as good as externally pre-trained models, given the right choice of data and training targets:

It will be shown that the number and type of training targets during pre-training impacts a model's performance on transfer learning tasks. The use cases presented in this thesis cover different data from different domains to show that the siamese training approach is widely applicable. Consequently, researchers are motivated to create their own pre-trained models for data domains, for which there are no existing pre-trained models.

Zusammenfassung

Die Vorhersage eines Models hängt davon ab, welche Muster in den während des Trainings benutzen Daten vorhanden sind. Je größer die Menge an Trainingsdaten ist, desto wahrscheinlicher ist es, dass Grenzfälle in den Daten vorkommen. Je größer jedoch die Anzahl der zu lernenden Mustern ist, desto größer muss jedoch das Modell sein. Für einfache Anwendungsfälle ist es möglich ein kleines Modell in wenigen Minuten zu trainieren um bereits gute Ergebnisse auf Testdaten zu erhalten. Für komplexe Anwendungsfälle kann ein dementsprechend großes Modell jedoch bis zu mehrere Tage benötigen um ausreichend gut zu sein. Die tatsächliche Zeit um so ein Modell zu entwickeln ist meist noch größer, da nur in den seltensten Fällen die erste Version eines Modells die richtige ist. Vielmehr werden in der Praxis erst mehrere Architekturen und Konfigurationen ausprobiert werden müssen um gute Ergebnisse zu erzielen.

Ein Extremfall für ein großes Modell ist das kürzlich veröffentlichte Modell mit dem Namen *GPT-3*. Dieses Modell besteht aus 175 Milliarden Parametern und wurde mit Trainingsdaten in der Größenordnung von 45 Terabyte trainiert. Das Modell wurde trainiert Text zu generieren und ist in der Lage Nachrichtenartikel und Quelltext zu generieren, basierend auf einer groben Ausgangsbeschreibung. Solch ein Modell können nur solche Forscher entwickeln die Zugang zu entsprechender Hardware und Datenmengen haben. Es demnach von Interesse das Trainingsvorgehen daher zu verbessern, dass auch mit wenig vorhandenen Ressourcen Modelle für komplexe Anwendungsfälle trainiert werden können.

Diese Arbeit beschäfigt sich mit dem Vortrainieren von neuronalen Netzen. Wenn ein neuronales Netz auf einem Datensatz trainiert wurde und dann auf einem zweiten Datensatz weiter trainiert wird, lernt es die Merkmale des zweiten Datensatzes schneller, da es nicht von Grund auf Muster lernen muss sondern auf bereits gelerntes zurückgreifen kann. Man spricht dann davon, dass das Wissen *transferiert* wird. Während des Vortrainierens bekommt ein Modell häufig eine Aufgabe wie zum Beispiel, im Fall von Bilddaten, die Trainingsdaten erst zu komprimieren und dann wieder herzustellen. Bei Textdaten könnte ein Modell vortrainiert werden, indem es einen Satz als Eingabe erhält und dann den nächsten Satz aus dem Quelldokument vorhersagen muss. Solche Modelle bestehen dementsprechend aus einem *Encoder* und einem *Decoder*. Der Nachteil bei diesem Vorgehen ist, dass der Decoder lediglich für das Vortrainieren benötigt wird und für den späteren Anwendungsfall nur der Encoder benötigt wird. Da beim Vortrainieren der Decoder

jedoch mit trainiert wird, ist der Trainingsprozess bei diesem Encoder-Decoder-Vorgehen langsam.

Zentraler Bestandteil in dieser Arbeit ist deswegen das Untersuchen der Vorteile und Nachteile der *siamesische* Modellarchitektur. Diese Architektur besteht lediglich aus einem Encoder, was dazu führt, dass das Vortrainieren kostengünstiger ist, da weniger Gewichte trainiert werden müssen. Der wesentliche wissenschaftliche Beitrag liegt darin, dass die siamische Architektur ausführlich verglichen wird mit vergleichbaren Ansätzen. Dabei werden bestimmte Nachteile gefunden, wie zum Beispiel dass die Auswahl einer Ähnlichkeitsfunktion oder das Zusammenstellen der Trainingsdaten große Auswirkung auf das Modelltraining haben. Es wird erarbeitet, welche Ähnlichkeitsfunktion in welchen Kontexten empfohlen wird sowie wie andere Nachteile der siamischen Architektur durch die Anpassung der Trainingsziele ausgeglichen werden können. Die entsprechenden Experimente werden dabei auf Daten aus unterschiedlichen Domänen ausgeführt um zu zeigen, dass der entsprechende Ansatz universell anwendbar ist. Die Ergebnisse aus konkreten Anwendungsfällen zeigen außerdem, dass die innerhalb dieser Arbeit entwickelten Modelle ähnlich gut abschneiden wie extern verfügbare Modelle, welche mit großem Ressourcenaufwand trainiert worden sind. Dies zeigt, dass mit Bedacht erarbeitete Architekturen die benötigten Ressourcen verringern können.

Chapter 1

Introduction

1.1 Introduction

In this day and age, neural networks are used in many different contexts, for example, speech recognition on a smartphone or visual object detection for self-driving cars. Their advantage is that they can process complex information in a very short period of time, giving the impression of immediate response. A model's response for a given input pattern in production depends on the seen patterns in the training data. The larger the amount of training data, the more likely edge cases are covered during training. However, the more complex input patterns are, the larger the model has to be. For very simple use cases like the commonly used MNIST dataset[1], a relatively small model can achieve very high test accuracy in a matter of minutes. On the other hand, a large model, like the commonly used BERT model in the NLP context, has been trained for multiple days [32]. The actual time to develop a model of that size can be considered to be even greater since often many different architecture types and hyper-parameter configurations have to be tried.

An extreme case for a large model is the recently released *GPT-3* model [14]. This model consists of 175 billion parameters and was trained using 45 terabytes of text data. The model was trained to generate text and is able to write news articles[2] and source code based only on a rough description[3]. The capabilities of such a model are impressive. However, a model like this is only creatable for researchers with access to special hardware or immense amounts of data. The analysis of the needed resources, time, and data are called *Red AI*, and *Green AI* [116]. Red AI focuses on aspects such as accuracy (or similar measures) by increasing a model's training time, training data, or the number of parameters. In contrast, Green AI focuses on efficiency, i.e., creating comparably good results by decreasing the needed resources.

[1] The commonly used MNIST dataset is considered as too easy for neural networks and has been replaced by *Fashion MNIST* [135].

[2] `https://www.theguardian.com/commentisfree/2020/sep/08/robot-wrote-this-article-gpt-3`, accessed 2020-12-21.

[3] `https://gpt3.website/`, accessed 2020-12-21.

Ideally, creating a neural network consists of two basic steps: *building* the network by creating its architecture, and *training* the network. However, the two steps impact each other since it is unlikely that the very first built model is perfectly suited for the given task. During training, some architectural design decisions may prove to be disadvantageous for the given task and available data. In such cases, training has to be canceled, the architecture or the way data is given into the model has to be revised, and training has to be restarted. This relates to the common phrase used in agile programming, being *fail early, fail often* (John C. Maxwell). The earlier such problems can be identified, the faster the model's overall development will be. Since neural network training times are mostly large, some problems may only be found after a long training time.

When training a neural network, the most important aspect is changing the weights using gradient descent [74]. There exist approaches like Batch normalization [56] or different kinds of optimizers [58, 104, 142] to speed up the process of gradient descent. In this thesis, a different aspect of speeding up the training process shall be investigated through the use of pre-trained models. If a model has been trained on one dataset and is then trained on another similar data, it faster learns to adjust to similar patterns than a model that has not yet seen any of the task's pattern. Thus, the learned lessons from one training are *transferred* to another task.

Transfer learning has been proven successful ([125, 34, 120, 29], see also sections 4.1 and 4.3.4). Transfer learning means a neural network is trained twice. First, it is trained on a dataset with typically lots of samples. After this initial training, the network is trained on another dataset. That second dataset typically has significantly fewer samples compared to the first dataset. The network has already been trained on data, which means it can identify common patterns for the given use case. The advantage of this transfer learning approach is even more significant, the more similar the data is for the two training phases. The larger dataset may contain the same patterns in different contexts, which reduces the problem of model overfitting. In some cases, the model can learn patterns that are even not present at all in the second dataset but are crucial when using the model in production. That way, using a pretrained model for a task does not only decrease training time but can also increase the overall performance. If the two datasets are unrelated, transfer learning cannot be used.

Critical design decisions have to be made when pre-training a model before transferring it to another task. During pre-training, the model is trained to solve a specific task like predicting the next word in a sequence or first encoding an input image before decoding it. Such models contain an encoder and a decoder part (often called *autoencoders*, see section 2.2.2 for details). When transferring that model to another task, parts of the model's layers will be removed. The model will most likely not be used to solve the same task as during the first training step. However, if the transfer learning task's data strongly differs from the first training phase's data, the learned features cannot be used, and thus the pre-training efforts are wasted. As a result, having to discard fewer weights results in faster training

since less time has to be spent on training parts of a model that are only needed to solve an auxiliary task.

Throughout this thesis, the concept of *siamese architectures* (see section 2.2.5) is discussed since when using that architecture, no parameters have to be discarded when transferring a model trained with that approach onto another task. Thus, the siamese pre-training approach positively impacts the need for resources like time and energy use and drives the development of new models in the direction of Green AI. The models trained with this approach will be evaluated by comparing them to models trained with other pre-training approaches as well as large existing models. It will be shown that the models trained for the tasks in this thesis will perform as good as externally pre-trained models, given the right choice of data and training targets. Also, the use cases presented in this thesis cover different data from different domains to show that the siamese training approach is widely applicable. This again shall motivate other researchers to not blindly use an existing pre-trained model but also to create a custom pre-trained model for use cases for which there exists no suited pre-trained model.

1.2 Research focus

To enable researchers to create well-performing models while having only limited resources, like small amounts of data or not the latest hardware, this thesis motivates the use of pre-trained models for various machine learning problems. Among the different kinds of pre-training approaches, the siamese training approach will be investigated thoroughly and compared against other approaches since that architecture type is most efficient when considering the number of parameters during pre-training. However, when dealing with the siamese training approach, the following questions have to be investigated in detail:

1. What are the effects of different hyper-parameter decisions for the siamese pre-training approach? When using the siamese training approach, several design decisions have to be made which are unique to this training approach. One of those is with which similarity or distance function to compare the created encodings. Popular similarity functions like cosine, euclidian, or Jaccard all lead to acceptable results. However, these similarity functions require encoding values to be in a specific range. For example, Jaccard similarity requires all values to be in between 0 and 1, while cosine similarity works best if the values are centered around 0. These value range requirements affect a model's performance when the encoding is used for additional training tasks. That phenomenon, as well as the general effect of different similarity functions, will be investigated in section 3.1. Created encodings from language data are used in transfer learning tasks in section 3.1 and 3.3, as well as for a concrete use case in section 4.3. For all these encodings, it has to be discussed what kind of features these encodings contain and whether these features are suited for the given use case. In section 4.3 it is

shown that the encodings are not suited, and an adjustment for the training process is presented and discussed.

2. How does the number and type of pre-training targets affect a model's performance on transfer learning tasks? Neural networks always have to be trained with a training target. This also applies to unsupervised pre-training since no gradients can be calculated without a training target. A straight-forward classification task uses a different type of architecture compared to a network that is trained in an unsupervised manner. Unsupervised training targets can be, for example, the reconstruction of input data. This is typically done when dealing with images: An image is encoded to a vector representation, which is passed through a decoder that has to reconstruct the input image. This kind of architecture is called an *autoencoder* (see section 2.2.2). For textual data, a model could be trained to encode an input sentence before decoding it again. For both cases, the trained encoder would learn about the most occurring features in the input data, which again is advantageous when transferring the encoder onto a different task. An encoder-decoder architecture is suitable for this task.

Training an autoencoder is expensive since the model consists of two parts, of which, in most cases, only one part is needed for transfer learning tasks. The *siamese architectures* (see section 2.2.5), on the other hand, only consists of an encoder part and does not need a decoder. When training a siamese model, pairs are created from the data and are either positive or negative. The model has to create an encoding for each entry of the pair. If the pair is positive, the encodings have to be similar. If the pair is negative, the encodings have to be dissimilar. Training these kinds of models may be cheaper due to the need for fewer weights. However, this thesis investigates the effect the selection of positive and negative pairs have on a siamese model's performance (see for example section 3.2). Also, it has to be investigated how siamese models perform on transfer learning tasks in comparison to models that have been pre-trained with other techniques (see section 4.3.4). The concept of *multitask learning* (see section 2.1.4) is discussed in detail by showing how a model's results are affected by training with multiple training targets.

3. What is the relation between a slim encoder like a small siamese model and a large general-purpose model like BERT? The usage of pre-trained models is common in many fields: When dealing with visual data, it is common to use a pre-trained model like *VGG16* [122] or *ResNet* [45]. For NLP tasks, there exist general-purpose models like *BERT* [32]. All these mentioned models have been trained on huge datasets and contain many parameters. They have learned a wide variety of features, which again makes them easily transferable to nearly any use case from the same domain. However, while those models' size is one of their strengths, it is also one of their weaknesses. If those models are fine-tuned for a specific use case, it may be very resource intensive to train them. This creates

the question of whether it is necessary to use a resource-intensive model and fine-tune it or whether it might be better to train a small model from scratch using a smaller dataset. Sections 3.1 and 3.3 compare how to train models that are relatively small compared to their large counterparts, sections 4.1 and 4.3 describe use cases in which both model types have been used and discussed the advantages and disadvantages of both types.

1.3 Structure

The mentioned research questions from section 1.2 will be tackled by executing multiple experiments. By doing that, the effects of hyper-parameters like distance functions, architectures, and training data choices can be discussed. Since unsupervised pre-training does not require labels, the trained models are compared qualitatively. In the context of text data, learned word and sentence embeddings will be compared. Additionally, the models will be applied to a transfer learning task for quantitive analysis. The experiments will cover data with different dimensionality and from different domains to show the different effects of the hyper-parameter choices and to also show that the siamese training approach can be applied to several different use cases. The result will be a set of general guidelines on how to pre-train neural networks using the siamese training approach.

Chapter 2 gives an overview of learning paradigms to embed different use cases for how neural networks can be used. Here, the focus will be on those paradigms that can be applied to unsupervised learning (see section 2.1), for example, those for which no labels are needed. After that, neural network architectures for unsupervised learning tasks are explained (see section 2.2). One property that will be discussed in detail is how complex it is to train these architectures. For example, an autoencoder first encodes input data before decoding it, whereas a siamese architecture only encodes data.

Having explained different architectures for unsupervised pre-training, chapter 3 covers the actual pre-training. These models and all other models in this thesis have been created using Keras[4] in combination with Tensorflow[5]. For hardware, GPUs provided by the clara cluster at Leipzig University have been used [6]. The experiments use different kinds of data to show that the concept of pre-training is universal and not bound to a specific use case or type of data. Experiments using visual data are shown first (see section 3.2) since those data can be used to visualize effects of architecture design choices regarding autoencoders and siamese architectures. Next, word embeddings are trained (see section 3.1), where the effect of different similarity functions like cosine or euclidian similarity is discussed. Also, the amount of data used for pre-training is discussed by training multiple models using different amounts of data before applying them onto transfer learning tasks.

[4] https://keras.io/
[5] https://www.tensorflow.org/
[6] https://git.sc.uni-leipzig.de/sc/

In section 3.3, sentence embeddings are trained. The trained sentence embeddings are used on transfer learning tasks. It is shown that the simple approach results in worse performance compared to using a popular model like BERT. It is shown that the worse performance is caused by low feature richness. It will be shown how using multiple training targets significantly improves the models' performance.

After having created multiple guidelines for what to consider when pre-training neural networks with different data types, that knowledge is applied to multiple practical use cases in chapter 4. The first use case is a classification task using tweets (see section 4.1). For this task, only a limited amount of labeled training data is available, which again shows the advantages of using pretrained models with a large amount of unlabeled data. Section 4.3.4 also covers a use case with a small amount of labeled training data, whereas the data type is visual data. Section 4.3 covers a study in which n-gram embeddings created from one corpus are used to find similar n-grams in a different corpus. The main problem to overcome is that the learned patterns during unsupervised pre-training are not suited to solve the given task. It is shown that the architecture has to be slightly changed to be adjusted for that problem.

Finally, chapter 5 gives an overview of all the learned lessons throughout this thesis.

1.4 Related publications

Results from the case studies presented in chapter 4 have already been published in the peer-reviewed proceedings of various computer science and computational humanities conferences:

- Bryan, M., Philipp, J.: Unsupervised pre-training for text classification using siamese transfer learning. In: Cappellato, L., Ferro, N., Losada, D., Müller, H. (eds.) CLEF 2019 Labs and Workshops, Notebook Papers. CEUR-WS.org (Sep 2019) - *This article presents a siamese architecture for pre-training recurrent networks on textual data. The network has to map pairs of sentences onto a vector representation. After having pretrained that network, it was enhanced and trained on a smaller dataset to have it classify textual data. I was shown that using this kind of approach for pre-training results in better results comparing to doing no pre-training or only using pre-trained embeddings when doing text classification for a task with only a small amount of training data. For evaluation, the bots and gender profiling dataset provided by PAN 2019 was used.*

- Bryan, M., Heyer, G., Philipp, N., Rehak, M., Wiedemann, P.: Convolutional attention on images for locating macular edema. In: Zheng, Y., Williams, B.M., Chen, K. (eds.) Medical Image Understanding and Analysis. pp. 391–398. Springer International Publishing, Cham (2020) - *Neural networks have become a standard for classifying images. However, by their*

very nature, their internal data representation remains opaque. To solve this dilemma, attention mechanisms highlight regions in input data that have been used for a network's classification decision. This article presents two attention architectures for the classification of medical images. First, a simple architecture is explained, which creates one attention map that is used for all classes. After that, an architecture that creates an attention map for each class is shown. The presented architectures well meet the baseline of standard convolutional classifications while at the same time increasing their explainability.

- Bryan, M., Burghardt, M., Molz, J.: A computational expedition into the undiscovered country - evaluating neural networks for the identification of hamlet text reuse. In: CHR (2020) - *This article describes a two-step processing pipeline for identifying text reuse of Shakespeare's Hamlet in a corpus of postmodern fiction by comparing n-grams from both sources. A key feature of that approach lies in a pre-filtering step, in which target sentences in the fiction corpus are selected that are potential candidates for Hamlet text reuse. In a second filtering step, potential text reuse pairs are compared by their vector representation using a neural network trained in an unsupervised manner. It was found that using the vector similarity resulted in undesired results since the similarity is based on different textual features compared to what is actually desired.*

Chapter 2

Machine learning

Chapter 1 motivated the usage of using pretrained models when tackling machine learning tasks by introducing the two-step process called *transfer learning*. It was also mentioned that using auxiliary training targets with possibly misleading labels can be used to pre-train a model before transferring it onto another task. This section defines several different learning approaches and relates *transfer learning* and *weak supervision* to them. At first, different learning paradigms are compared. These learning paradigms describe how learning experiments are to be done with different types of labeled data. For that, different types of machine learning models are mentioned, whereas only neural networks are looked at in detail. For these types of models, different architectures and configurations shall be investigated. Of these different architectures, *siamese models* are looked at very closely. That type of architecture has the advantage that none of its parameters have to be discarded when transferring a model to another task.

2.1 Learning paradigms

This section shall investigate different paradigms for machine learning experiments. Machine learning can be defined as using information gathered in the past to improve performance make predictions about events in the future [81]. Mathematically speaking, machine learning comprises of a set of input samples $X = x_0, x_1, ...x_n$ and a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ which describes a mapping from training samples onto a desired representation [141, 23]. Each input sample can contain one or more numerical features, which can be things like the height of a person or the number of occurrences of a specific word in a text. If an input is not numerical, like a word or an image, it has to be transformed into a numerical representation. For textual data, this can be done by creating a vector of a length corresponding to the number of words in a given dictionary. The occurrence of a specific word would be represented by setting a value in the vector to 1, whereas the vector's position corresponds to the word's position in the used dictionary. For visual data, images can be transformed into a numerical representation using each

pixel's color values, typically between 0 and 255. Also, since the function f contains a set of parameters to be changed, f is called a *model*. Different parameters with which the training process is controlled are called *hyperparameters*. In the training process of a machine learning experiment, that function f has to be trained such that its output Z is as close as possible to a predefined output Y. Training is done in pairs (X_i, Y_i), whereas Y_i is called a label or target.

The aspect with which to separate the different paradigms is the providing of labels for training. When a target is given for each input sample, that paradigm is called *supervised learning*. When labels only partially exist or when labels are possibly incorrect, these approaches are called *semi-supervised learning* (also called incomplete supervision [140, 139]). When there are no labels given, the task changes such that not a function from input samples onto different classes shall be learned, but a model shall find patterns in the data. Such findings are not predefined, which is why that paradigm is called *unsupervised learning*.

2.1.1 Supervised learning

For *supervised learning* experiments ([85] among others), the data comprises of pairs with labels for each input sample: $L = (x_1, y_1), (x_2, y_2), ..., (x_{|L|}, y_{|L|})$. When training, at all times, the current performance of the model can be evaluated by calculating the distance between all current predictions Z for the input samples and the desired targets Y. As metric for evaluating such distance, mean squared error like $MSE(Y, Z) = \sum N(Y_n - Z_n)^2/N$, whereas N represents the number of all training samples. The values of Y can be of two forms, which are either categorical or real-valued.

If the values in Y contain categorical data, the labels represent class information for the input data. In that case, the learning experiment is called a *classification task*. When there are two classes to predict, it is called *binary classification*. Technically, two classes can be represented by one output value. If the number of classes is larger than two, it is called *multiclass classification*. Exemplary use cases for supervised learning experiments could be a data set with images of objects, and the labels contain information which object on these images can be found. An exemplary data set for this is the *MNIST*[1], which contains about 60,000 black-and-white images the size of $28x28$ pixels, whereas each image contains one of ten possible digits. For textual data, classifying email into the categories spam or no-spam is a common use case.

When having real-valued labels, the model has to give a numerical prediction, which could be something like predicting temperature values or prices. In that case, the learning experiment is called a *regression task*. When training a model in a supervised manner, the data is split into training and test data. That way, the model is trained on training data, and its performance is evaluated using the test data. That way, the problem of overfitting can be prevented: When the model is

[1] Available at http://yann.lecun.com/exdb/mnist/

trained only on one set of data, it adjusts itself to that training data that when using it in production, i.e., having the model predict input samples that have not been seen when training the model, such predictions can be very bad if they contain patterns that have not been seen during training. When training a model using training and test data, the training error on both sets of data is expected to decrease during training. If the test data's error is rising again, while the training data's error continues to decrease, overfitting is happening, and training should be stopped.

The training process for supervised training is easy to control because of the ability to evaluate the model's performance. The downside of such an approach lies in what needs to be done before training: the creation of training pairs (X_i, Y_i), which can be very expensive, whereas, at the same time, the amount of unlabeled data is much higher [85].

When evaluating a model's learning progress, it is common to use a larger subset of the data as training data and a smaller subset as test data ([74] among others). While the training data is used by the model to adjust its parameters to give a better output for the given input-output-pairs from the training data, the test data is used to monitor the model's performance on non-training data. If the model's performance on the training data is significantly better compared to its performance on the test data, the model may be overfitting to the training data and thus may perform badly on unseen data when used in production. In some cases, the split between training and test data may be such that, by chance, some patterns are only included in the test data and not in the training data, leading to the model not being able to learn those patterns. To overcome this downside of a random data split, it is common to do several splits of training and test data. Here, the data can be split into *K* subsets, of which 1 is used for testing and *K-1* are used for training. The training is done *K* times, whereas each time a different subset is used for testing, and at the end, all test accuracies are averaged. That approach is called *K-fold cross-validation* ([74] among others).

In this thesis, supervised learning is used for all experiments, at least indirectly: Neural networks always have to be trained with data samples consisting of input data and a label. Otherwise, gradient descent could not be computed. However, in the case of unsupervised learning, the training target with which a model is trained is only used during that unsupervised training phase, while the model will be used differently. Actual supervised learning experiments are executed in sections 4.1 and 4.3.4, where models are trained to classify tweets or patients' images.

2.1.2 Transfer learning

Transfer learning [89, 42] describes a two-step approach where a model is trained on two different data sets. Intuitively, transfer learning is done when considering that it is easier to learn about apples when already having learned about pears[2]. When training a model, mathematically, it is assumed that input samples come

[2]Example taken from [89].

from a certain feature space and have the same distribution as the input samples in the test or production phase, respectively. If a model is reused for a different task, it is assumed that the new input samples and their features are from a similar distribution.

Transfer learning can be used when for a specific task, only a small set of training data exists. At first, the model will be trained on a data set with input samples from a similar domain compared to the desired task. In the first training phase, the model can learn generic patterns. In the second phase, the model will be trained on the smaller but desired data set. With the learned patterns from the first data set, training may be easier than only training the model in the smaller dataset. Also, instead of pre-training a model oneself, a pre-trained model from a different source can be used. For example, when dealing with neural networks, there exist several pre-trained models. For visual tasks, famous pre-trained models are *Densenet* [54], *Resnet* [45] or *VGG16* [122]. These three models have been trained on large image data sets and have learned to recognize very different visual features. When using such a model on a different use case, the very last layer for classification has to be removed in favor of a custom classification layer, whereas all the other weights are kept. When a model is then trained, it does not need to learn common visual patterns from scratch but can reuse such learned patterns. For textual data, pretrained models exist for word embeddings ([78, 91, 11] among others) or sentence embeddings ([59, 71, 32] among others). All these models have in common that they have been trained on large text corpora, whereas the auxiliary training tasks differ among the different models.

In experiments as done by [50] and others, a network was first trained to reduce the dimensionality of data and then classify it. This approach is close to transfer learning, but since the classification training was done on the same data used in the first training phase, it is not called such.

A practical use case may be in medical images where there could be a dataset of medical images that either contain symptoms of one or more diseases or no disease (see section 4.3.4). If that data set is very small, a model first could be trained on a dataset containing images of the same type with similar diseases. The model then could learn visual patterns, which can then be reused when training on the smaller dataset.

2.1.3 Active learning

Active learning (also called *self-learning* [117, 37, 2] or *query-learning* [119]) defines an iterative learning process, where there are two different data sets, of which there is one data set L with labeled input samples, as is with supervised learning, and a second data set $U = x_1, x_2, ..., x_{|U|}$ with unlabeled training samples and falls under *incomplete supervision*. The model is trained on the data set L, which contains input samples with corresponding targets. It can predict values for all unlabeled input samples from dataset U when having trained the model. These predictions can then be checked by a human annotator (also called *oracle* [140]).

The model's prediction can be seen as confidences with which to label unlabeled data. Predictions with high values can easily be annotated as correct such that the training set can be extended. When the model's prediction is not polarized such that one class gets a high value, these cases can be seen as edge cases, for which there have not been enough samples in the training set so far, such that the model was not able to give a clear prediction. It has been shown that some newly labeled samples provide more information than other samples [119]. After having labeled data, the number of samples in data set U has decreased and has increased in data set L. The model can then be trained such that it adjusts itself to the now enlarged training set to give better predictions to the samples in the data set U.

Supervised-learning depends on labels being existent for all input samples, of which the creation can be expensive since a human annotator has to make decisions for all existing input samples. Only a subset of input samples needs to be labeled when doing active learning before training a model. Unlabeled data still has to be labeled, but such a process is more straightforward since an annotator just has to confirm predictions with very high confidence and only has to make more challenging decisions on edge-cases.

Active learning is not used thoroughly in this thesis. However, in section 4.3, an approach is presented in which n-gram pairs created by a model are annotated. These annotations are then used to continue training that model. That approach can be considered being the initial step of an active learning process.

2.1.4 Multitask learning

In the previously mentioned supervised learning experiments, a model is trained to solve a single task, meaning learning a function $f : \mathcal{X} \rightarrow \mathcal{Y}$. That function f can be trained such that for one given input sample, several tasks can be solved at once: $f : \mathcal{X} \rightarrow \mathcal{Y}_1, \mathcal{Y}_2, ...\mathcal{Y}_n$. That pattern is called *multitask learning* [22]. It shall be noted that the different tasks' targets are independent of each other. For example, when there is an image data set containing images of either cats and dogs, a model has to predict a cat's occurrence and a dog's occurrence. If both occurrences are mutually exclusive, these two predictions are the same task. On the other hand, a different task could be something unrelated to the first task, such as recognizing the presence of different objects in the background.

When a model learns features to do its classification task, these learned features can be shared by all tasks. This type of learning can be reasoned biologically [109]: When a baby is learning to recognize objects, it is learning visual patterns. The knowledge of human faces can later help to recognize animals' faces. When training a model for one task, all information helping the model to do the classification task is used, all unneeded information is discarded. The more different tasks there are, the more information from the input samples has to be kept, given that all tasks are independent of each other. Keeping more information is improving a model's ability to generalize [22], because the more information is discarded, the more is a model overfitting onto a data set. Creating training data that is suited for

multitask learning is more costly than creating data for one single-task learning: Using multitask learning means having multiple labels for each training sample.

Multiple authors have shown the power of multitask learning. In the context of drug discovery, a dataset containing 40 million measurements has been used to predict biological targets [98]. The authors have shown that the more targets were given to the model, the better the model performance on all tasks was. In the context of language data, [28] trained a model to do several NLP tasks at once: For a given sentence, they made the network predict part-of-speech tags, chunks, named entity tags, semantic roles, semantically similar words, and the likelihood that the sentence makes sense (grammatically and semantically) using a language model. They stated that the joint learning of all tasks improved the model's overall performance. The authors in [73] put an image into a model and had the model, on the one hand, produce a set of possible image tags and, on the other hand, a full sentence describing the image. They found out that having the model create a set of possible tags for the image improved the sentence generation. They reasoned that creating simple tags is easier and enables the model to quickly learn features to solve that task. Those features then helped the sentence creating part of the model to create a full sentence from those features. For translation tasks, [133, 72] made use of multitask learning by inputting a sentence one language into a model and have the model output that same sentence in a different language. Simultaneously, the model has to create a list of pos tags for the sentence in the input language. It was also described that there are several types of multitask learning, namely one-to-many, many-to-many, and many-to-one. In the recent past, one of the most famous use-cases of multitask learning was the creation of the *BERT* language model [32], whereas BERT stands for *Bidirectional Encoder Representations from Transformers*. This model is a multilayer recurrent model that receives a sentence as input and was trained to solve several tasks at once, like predicting missing words from a sequence or predicting the following sentence.

Multitask learning is used in sections 3.1 and 3.3. Here, models trained with language data are trained to solve several problems at once. For example, a model has to predict masked words from a sequence as well as that word's POS tag.

2.1.5 Unsupervised learning

As mentioned before, unsupervised learning approaches deal with data set which contain no labels at all. Learning without labels can be seen as more natural since when humans or animals gather information about their surroundings with their senses, such inputs are not labeled. Only on very few occasions, a mother could tell her child, "that is a dog." The child then has to connect learned features from the past with the now learned label.[3] Thus, because of lacking labels, it is not the goal to map the input sample onto a predefined representation but onto an undefined representation with reduced dimensionality compared to the input samples' dimen-

[3] Example from Geoffrey Hinton, 1996. Quoted in [43].

sionality. When dealing with unsupervised learning, the task can be described as a *density estimation* [85] or *compression* [123] by reducing redundancies in the input data.

When reducing the dimensionality of data, the data's original size is being decreased to a smaller representation. In the non-machine learning context, there exist different ways to create such representation, depending on the desired use case. A commonly used dimensionality reducing method is hashing, which maps data of any given length or size onto a fixed-length representation. With this, it is desired that two similar data samples in their raw representation result in very different hashed representations. A use case for this would be storing passwords: If two passwords with similar characters had a similar hash representation, it would be possible to find the original password given a password's hash. A different, very technical way of reducing data is compression. Here, the data that is to be compressed is searched for patterns that appear multiple times to store these patterns only once. The looked-for similarity of these patterns is binary: Either two patterns are the same or different. This strictness has to be used since, when extracting the data, it has to be extracted to the same representation as it had before compression. Another way to reduce data representation is by using domain knowledge and defining features with which data shall be looked at. For example, when looking at textual data, one could be interested in the number of occurrences of a specific set of words or word groups. When counting these words, a text or document can then be described by using only these counts. When two documents contain the same number of words one is interested in, their reduced representation would be the same. The question of whether the two documents are similar then is very subjective: They may contain different content, different lengths, or different metadata, but given this one aspect they are looked at, they are similar.

Unsupervised data dimensionality reduction methods all work such that they look for repeating patterns in data or related features, respectively, and combine them into one. PCA [38] compares features by their correlation and groups those features such that the reduced features are as uncorrelated as possible. Dimensionality reduced data samples are similar in their reduced representation when their features, which are most used by the PCA reduction, are similar in their original state. As an example, images could be dimensionality reduced using PCA. Groups of pixels that always have the same color would be correlated and combined into one reduced feature [75]. When two images' pixels have a similar color at these spots, their reduced representation would be similar. The patterns to look for do not need to be the same but just have to be correlated or *similar*. This approach is similar to the previous compression approach, but in this case, it is not possible to reconstruct the data given only the reduced state since the compression is done with a data loss. The used features for dimensionality reduction also do not necessarily be the same set of features humans would be interested in. If there is a set of images with animals, a human probably would say two images are the same if they contain the same animal. On the other hand, if there are substantial repeating patterns on the images, a PCA algorithm could choose this pattern for combining features

like a big blue sky. When then comparing the similarity of images, they would be grouped by the fact whether they contain a big blue sky or not. Technically, this may be the best approach to reduce the data's dimensionality, and semantically this could not be desired.

Training neural networks in an unsupervised manner shall be investigated in detail in 2.2. The concept of unsupervised training will be investigated in detail in chapter 3. Here, unsupervised training is done with different types of data. For example, a model is trained with visual data by having it first encode the data before decoding it again. When considering language data, one model is trained to create a vector representation of sentences. These vector representations are compared, whereas the model has to create the vectors such that two sentences' vectors are as similar as possible if they appear as coherent sentences in the given training corpus.

2.1.6 Weak supervision

Weak supervision [103] (also called *inaccurate supervision* [140, 139], *minimal supervision* [19] or *distant supervision* [79, 4]) is similar to supervised learning as both provide labels for all input samples. The key difference is that these labels are knowingly possibly incorrect or may even be incorrect most of the time. The reason for using incorrect labels is that they may be way cheaper to create compared to ground truth data or that they are used as a heuristic for the lack of real labels. There are similar forms of creating such labels, depending on the use case, but they all have in common that labels may be contradicting.

For learning relationships between entities, [19] have used a text corpus crawled from the web. The authors use a few predefined entities with known relations, e.g., whether company A has acquired company B as a starting point to find new relations in their corpus. They are aware that some of their findings may contradict other findings, e.g., sentences could be saying that a company has acquired another company, but other sentences may be neglecting that statement. Thus, they use an SVM classifier to classify a relation between the two entities using the found statements with the entities appearing. In [4], the authors also try to learn relations between entities but are using topic models. They state that a sentence can contain entities like *Barack Obama* and *United States*, which is a hint that there is a relation between these two entities. Still, they mention that other sentences are likely not to contain these two entities, which weakens their relation. In [79], the authors try to find relations between entities as well, but they are using a database with known relations as a baseline to train a classifier that finds new relations in large text corpora.

A different type of incorrect labels can be the usage of labels created by crowdsourcing. Here, humans create labels, but since they cannot be fully trusted, labels for the same input sample can be given several times and may be contradictive [137, 96].

In this thesis, weak supervision is used for some of the unsupervised learning experiments as well as for one supervised learning experiment. When training the

models for word or sentence embeddings, words or sentences are taken randomly taken from the corpus to create training pairs (see sections 3.1 and 3.3). Some of these pairs may be misleading since, for example, a word pair that is randomly chosen as being non-coherent may be appearing coherently somewhere else in the corpus. In the case of supervised learning, section 4.3.4 is describing an experiment where a model has to predict the success of treatment for a disease given a set of images. In some of the cases, the images may give the impression that the treatment has been successful, although the label says otherwise, which has to do with external factors, which again are not present in the data.

2.1.7 Reinforcement learning

Reinforcement learning [126] deals with problems where an actor or agent has to decide on actions in states to complete a task. A teacher or critic cannot give feedback on each actor's choice but can only give feedback when a specific state has been reached. It is the actor's task to find a series of actions that result in the teacher's best feedback. It is essential to know that such best series of actions is not known beforehand. A popular use case for reinforcement learning is having a program learning to play chess: For a human, it is obvious when a game of chess has been won or lost, but it is not known which action in any state is the best choice. Reinforcement learning could be seen as a special case of semi-supervised learning since, for some states, there exists a label, but for most states, there is no label. Given the environment, some tasks could also be defined as weakly supervised tasks since, in some environment states, the same action could lead to different, contradicting outcomes since not the full state is given to the model, but some states are kept hidden.

For learning a series of actions that lead to a successful endstate, an algorithm gives values to each state's actions. The actions' values depend on the state's value, which can be used by choosing that particular action. The only fixed values are values given by the teachers. All other actions' values are derived from experience. In order to not force an algorithm to keep track of every visited state, which can be impossible for environments with infinitely many or practically infinitely many states, neural networks have shown to be able to solve this problem ([80, 121, 83, 67] among others). Depending on the context, reinforcement learning model can be initialized with knowledge learned from tasks from the same domain, which is why transfer learning can be used in the context of reinforcement learning.

2.1.8 Inexact supervision

For some learning tasks, there may exist a data set with input samples and corresponding labels, but the labels may be *inexact* [140, 139]. An example could be a set of images with labels whether a specific object can be found on an image, whereas for a given task, the specific location of such an object could be needed. However, the labels only contain a rough information about where the object to be

found is in the image. For textual data, there could be a dataset with sentences and sentiment information as label, whereas a sentiment information for specific words or word groups would have been more accurate.

2.2 Non-supervised neural network architectures

When being given a large labeled data set, it is relatively easy to train a neural network such that it learns to classify the data. However, for most use-cases, such data sets do not exist. There is often not much data for a given task, and if there is, there are hardly any labels. This section tackles these problems by describing architectures that have been used for non-supervised tasks. All these architectures have in common that they need to solve the critical problem, that non-supervised training of neural networks may at first sight not be possible: The usage of gradient descent creates the need of a target value for each input sample in order to calculate an error and a resulting gradient. However, when using *artificial* labels, neural networks can be trained unsupervised by having them solve a task that has the side effect of, for example, reducing the input data's dimensionality. Nearly all described architectures follow the *encoder-decoder paradigm* [101] by encoding input samples onto a vector representation and then decoding it.

2.2.1 Restricted Boltzman Machines

The breakthrough [35] for training neural networks came in 2006 when a strategy was developed that enabled the training of deep networks [50, 49, 36]. Here, a Restricted Boltzman Machine (RBM) was used, which is a small network that consists of an input layer (with so-called *visible units*), hidden layer (with so-called *hidden units*), and one weight matrix. When training, input samples are mapped onto a hidden representation, and the network has to reconstruct them using the very same weights:

$$P(h_j = 1|v) = \sigma\left(b_j + \sum_{i=1}^{m} w_{i,j} v_i\right) \tag{2.1}$$

$$P(v_i = 1|h) = \sigma\left(a_i + \sum_{j=1}^{n} w_{i,j} h_j\right) \tag{2.2}$$

When training an RBM, input data is passed through the network three times:

1. The input is encoded $P(h|v)$
2. The input is reconstructed using its hidden represenation: $P(v'|h)$
3. The reconstructed input is encoded: $P(h'|v')$

Steps 2 and 3 could be repeated several times (so-called *Gibbs-sampling*), but in practice, one step more than often is enough. Using the variables v, h, v', h', the gradients can be calculated using *constrastive divergence*, of which detailed explanation can be found in [21]. Using this approach, the network learns to compress the input samples such that the reconstruction loss is minimized, although that can only be measured indirectly.

When one layer has been trained sufficiently, the trained layer's weights are fixed, and the next layer is trained to create a more abstract representation of the input data. This is done by first passing an input sample to the layer with the already trained weights and then using this hidden representation as the next layer's visible representation. After having trained several layers that way, the network can be *fine-tuned* by being trained in a supervised manner for a classification task. For visual data, images are represented by a single vector, whereas each value in the vector represents one pixel. Training the RBM results in the model learning which pixels' appearances are correlating. When dealing with textual data, the input vector can, for example, represent a document, whereas each value presents a word being used in that document. When training, the RBM learns about co-occurring words in a document. An exemplary use case for that approach is the hashing of documents [113], which was done by representing documents by tokens used in them and having an RBM learn to reduce the document's dimensionality. Using the approach, documents can be compared using their reduced vector representation.

RBMs have the disadvantages of using symmetric weights, which limits the model's capability to learn patterns. Also, since they are designed as probabilistic models, they are not well defined for dealing with non-binary input.

2.2.2 Autoencoders

Autoencoders are similar to the previously described Restricted Boltzman Machines in that sense that they map data onto a reduced vector representation and then reconstruct the input data using that reduced representation [9]. Still, there are several key differences:

- An autoencoder uses different weights matrices for decoding and encoding, compared to one symmetric weight matrix of an RBM
- For calculating the training error and gradients, the reconstruction error is used
- Backpropagation is used instead of contrastive divergence
- Non-binary values can be used without any problems
- Only two steps have to be taken instead of three

These advantages have lead to autoencoders being favored over RBMs. Also, they have been enhanced such that they are forced to learn more abstract patterns

in the input data [129]: The input samples can be corrupted (*noising*), but the network has to reconstruct the uncorrupted data (*denoising*). That way, instead of just compressing and decompressing the given data, the network is forced to learn what patterns are noise and what patterns belong to the input samples.

Autoencoders can be used for the same use cases as RBMs can, but they are easier to train given their advantages. Except for being used for the pre-training paradigm, they are also used to have data clustered ([44, 6, 136], among others). [41] has created a supervised use-case for this architecture: Here, a neural network was trained to denoise medical images. During training, the authors used clean images to which they added noise. The network had to reconstruct the clean version of the images. The same concept is used in section 3.2.

2.2.3 Word2Vec

Another famous example of non-supervised neural network architecture is Word2Vec [78]. It comprises an architecture and training procedure which are very similar to an autoencoder. It is a network architecture that maps single words onto vector representation. The network's architecture consists of two layers, of which the input layer has as many nodes as words in a dictionary and as many output nodes as the number of features of the words' vector representation. The second layer is the reverse of the first layer. During training, the network is given data in the form of sentences, of which a word is chosen as focus word along with its context, which is the focus word's surrounding words in the given sentence. The network has to predict (reconstruct) the context words given the focus word (so-called *Skip gram*) or predict (reconstruct) the focus word given the context words (so-called *CBOW*). In both variants, the focus word's or the context's vector representation has to be such that the reconstruction is as close to the target as possible. The network has to learn that some words appear more often in a focus word's context than other words so that two words with two very different contexts have to have a different vector representation compared to two words with a very similar context.

When there is a phrase *to drink coffee in a café*, *coffee* could be the center word. Its vector representation is used to predict the probability of other words appearing in its context. The representation has to be such that words like *drink* and *café* get a high probability, and other words, which are unlikely to appear in the context of coffee, get a low probability. Another word like *tea* also has words like *drink* and *café* in its context and thus will receive a similar vector representation like *coffee* since they have to predict a similar context. Given their vector representation, the words *coffee* and *tea* will be similar.

Training Word2Vec consists of two-step, of which the first one is very cheap computationally and the second one is very expensive. When getting a word's vector representation, one just has to look at its corresponding entry in the weight matrix. To get the vector representation of a context, one has to look up the corresponding word's vector representations and then calculate the sum. For the prediction, the dot product of the vector and the output-weight matrix has to be cal-

culated in order to get the probabilities for all words in the dictionary. The bigger the dictionary, the more expensive that calculation is. To speed up this step, the dot product is not created for all words from the dictionary but for the desired targets and some randomly chosen samples, which are seen as negative targets.

When given a context, not the whole context is reconstructed but only a focus word. Apart from this, another key difference is when comparing Word2Vec to autoencoders: The targets may be contradictory. Focus words and their context are chosen randomly from input sentences. One focus word likely appears in different contexts. Also, since negative samples are chosen randomly, a pair that previously has been used as a positive sample can now be used as a negative sample and thus revert the previously done weight change. These contradictive training samples slow down the training process. Still, when considering a given context combined with a positive target and a randomly chosen negative target, it is very unlikely that the two targets are switched for the same context. More than enough for the given context and positive target, different negative targets are samples. Thus, in the long run, the impact of contradictive samples can be neglected.

Autoencoders and Word2Vec are both trained in an unsupervised manner, but autoencoders have consistent labels, whereas Word2Vec has inconsistent labels. Thus, Word2Vec should be classified as a weakly-supervised paradigm. An external implementation of Word2Vec is used in section 3.1.

2.2.4 Sequence to sequence

The previously used examples for neural networks all have all assumed that data is static, as it is the case for images or the previously described word contexts that have not considered word orders. However, a lot of data is sequential, meaning the order in which input values are fed into a network plays a crucial role in the data's meaning. When looking at words used in a sentence, for example, a lot more meaning can be taken from a sentence when the correct order of its words is known. Also, when translating from one language into another, one might be interested in translating full sentences instead of single words.

For use cases that receive a sequence of input samples as input and have a different sequence as output, sequence to sequence has been proven to be a suited architecture [26, 125, 72, 59]. For some tasks, using a single recurrent layer as input and output may be sufficient. However, a sequence to sequence architecture has the advantage of being able to map input onto outputs when both have different lengths.

A sequence to sequence architecture contains two recurrent layers. The first layer receives the input data, has no output layer, and learns a representation of the whole sequence at its last internal state. This representation is then given into a different layer by initializing the internal state with the last internal state of the previous recurrent layer. The layer then produces an output sequence typically until a predefined stop token has been produced. The number of layers used can vary. The lower bound of used layers is one since the two described steps, i.e.,

consuming the input sequence and producing the output sequence, can be done theoretically by the same recurrent layer. As is the case with other types of neural network architectures, there is no upper bound for the number of layers.

When being trained in a supervised manner, sequence to sequence has been used to translate sentences from one language to sentences from another language [125]. Here, a recurrent layer received a sentence token-wise. Next, a different recurrent layer created a sequence of tokens from another language using the previous layer's last internal state. The same architecture was used by [26], although here, the more neutral name *encoder-decoder* was used to describe the architecture. In [72], the sequence to sequence architecture has been used together with the multitask learning paradigm. Here, an input layer received a sentence as a sequence of tokens. Next, the recurrent layer's last state was used to initialize to different recurrent layers, of which the first one had to create a sentence in a different language. The second one had to create a sequence of pos tags of the sentence in the target language. It was shown that having added the second task improved the performance of the first task. For unsupervised training with weak labels, sequence to sequence was used to create a vector representing the semantic meaning of a vector [59]. The authors have given a sentence token-wise into a recurrent layer and used the recurrent layer's last state to initialize to different recurrent layers, which had the task to generate the input sentence's preceding and succeeding sentence from the training corpus. That way, the network had to learn which patterns in the input sentences lead to patterns in the output sentences. The architecture was named *Skip thought*, and the sentence vector representations have been used to find similar sentences by comparing sentences using their vector representation.

Sequence to sequence is used in section 3.3 the same way as it was used by [125, 26].

2.2.5 Siamese neural networks

The previously described unsupervised architectures all received data as input, mapped it into a representation with reduced dimensionality (encoding), and then reconstructed the input data (decoding). As a result of this, RBMs consistent of three steps since the reconstructed input data was encoded again. Autoencoders contain an encoding step and a decoding step. The described *Skip thought* architecture did not reconstruct input data but had the network generate preceding and succeeding sentences but contained dedicated encoding and decoding steps. For most use cases, decoding data is expensive. In the case of image data, decoding data means creating an image of the same size as the input image, which then is compared to the input image. This means, given images of the typical size of 256×256, 65536 pixels values for a greyscale image would need to be created and compared with the input image. For a colored image, that number would need to be calculated by three. For textual data, decoding data is expensive because the probability of tokens from a dictionary must be predicted. The larger the dictionary, the larger the number of weights to be trained. For sequential data with a

large dictionary, the weights for only the last step can take up the biggest share of the entire network's number of weights.

To circumvent the need for decoding the input data, *siamese neural networks* have been proven useful. This type of architecture takes input samples and encodes them to a vector representation without needing to decode the data. This is because when training siamese networks, input data is trained in pairs. These pairs are then either *positive* or *negative*. When positive pairs are given into the network, their encoded vector representations have to be *similar*. When a negative pair is given into the network, their encoded network representations have to be *unsimilar*. What accounts for a positive or negative pair depends on the use case and shall be discussed by presenting different examples of use cases of siamese architectures. When training the model using pairs, four data samples have to be encoded for getting loss values for a positive and a negative pair. Instead of handling data in pairs, it is also possible to treat data in triplets [24, 115, 107]. Oftentimes, the three contained data samples are referred to as *anchor*, *positive* and *negative*. Hereby, the *positive* and *negative* samples do get their label in relation to the *anchor* sample. Three samples have to be encoded to create positive and negative loss values when calculating the model's loss. Additionally, it is possible to limit the loss such that the similarity of the positive pair only should be bigger compared to the similarity of the negative pair:

$$\text{loss}(a, p, n) = \max(0, sim(a, n) - sim(a, p) + \epsilon) \tag{2.3}$$

Using that equation for loss calculation, the loss is 0 if the similarity of the pair (a, p) is higher compared to the similarity of the pair (a, n). When thinking of mapping input samples into a vector space, the loss is only greater than 0 if the anchor sample is closer to the negative sample compared to the positive sample. Without using the max function, the model would be trained to move the positive pair as close to each other as possible. At the same time, the negative pair is moved as far away from each other as possible. When using a similarity function like euclidian distance, a negative pair's samples would need to be placed in opposite corners of the vector space to have a similarity score of 0. This may seem implausible since there still may be shared features even if the pair is negative.

Using the $max(0, \bullet)$ approach seems more plausible since the criteria for choosing positive and negative pairs often are vague. For example, in the case of training image similarity, two images may, on the one hand, contain the same desired object and are thus treated as a positive pair. On the other hand, they could be totally different in all other aspects. When wanting to pair data, it may be enough to train the model only such that these two image's vector representations are closer to each other than a vector representation of a random different image. The downside using the $max(0, \bullet)$ approach is that in many cases, the loss is 0, which results in having no gradients for that training pair and, thus, in slower training. Also, since the positive and negative training pairs are often created randomly, each sample is paired with different other samples and thus is *pushed* into several different di-

rections in the vector space of the same time, which cancels some aspects out and hinders extreme values.

When counting the number of de- and encoding steps when comparing siamese models to autoencoders, one might argue that siamese models also need to steps, the same as autoencoders, since two input samples have to be encoded when training. The key difference between these two architectures is that an autoencoder's decoding part is needed for training but not needed for the actual encoding for data. Thus, a part of the network is trained that only plays a role during training. For siamese models, all parts that are trained will be used after training for encoding data.

The earliest usage of a siamese architecture seems to be by [13]: In their project, the authors wanted to calculate the similarity between signatures. The signatures have been given into the neural network by a set of different predefined features. A positive pair contained two signatures from the same person; negative signatures were two signatures from two different persons. The last layer's output was linear, and the vector representation of two signatures was compared using cosine similarity. A positive pair needed to have a cosine similarity of $+1$, a negative pair needed to have a cosine similarity of -1. [27] created another siamese approach: Here, the authors created a convolutional neural network with a linear output. During training, the network was trained to give face images from the same person a cosine similarity of 1. When the images came from two different persons, the cosine similarity should be -1. Another siamese architecture for text was created by [87]. Job descriptions were given character-wise into a neural network to create a vector representation. The different descriptions had been labeled beforehand, such that the network was given data in pairs that depended on those labels.

When considering learning paradigms, siamese architectures fall under unsupervised learning since they reduce data dimensionality, and the learned representation is not defined beforehand. In the provided examples by other authors, they have been trained using well-defined labels. Throughout this thesis, siamese neural networks shall be trained using weak labels (see sections 3.1 and 3.3). It shall also be shown how this approach can be used for transfer learning by pre-training a siamese network and then using it for a different task.

2.3 Efficiency

The success of things like new training strategies, hyperparameter settings, or activation functions is mostly measured in accuracy. For example, at many big conferences, most articles presented their contribution by improving accuracy, or some related measure [116]. The study's authors form the term *Red AI* and mention that typically a model's performance is increased by using more data, more layers, or more training time. Each of these three aspects makes a model's creation more expensive: Using more training data implies that more data has to be stored. The

dataset *Common Crawl 2019*[4] contains text data with the size of 220TB. Using more layers means the gradients during training are smaller, which again means the model has to be trained longer, except when more expensive hardware is used.

The authors also define the term *Green AI* and suggest that when measuring a model's cost, not only the time to do inference on one sample is measured, but also the time and cost needed to create the model [116]. In these measures, they include carbon emission, electrical usage, elapsed time, and FLOPs. All these factors are hard to measure and hard to compare: The emission of carbons and electrical usage, for example, highly depends on the used hardware and can be lowered by using efficient modern hardware, which on the other hand is more expensive than older hardware.

In the context of this thesis, two metrics shall be used to compare models' efficiencies. The first metric to use is time. All trained models in this thesis have been trained using the same data, hardware, and time. Only the type of model differs. For example, when training word vectors, one model is trained using an encoder-decoder approach while the other one only uses an encoder. Section 3.1 investigates the results, giving an indication about which architecture type learns faster.

The second metric used to determine efficiency is FLOPs. FLOP stands for *floating point operation* and defines the number of mathematical computations [47]. For example, if a vector has 128 features and is mapped onto a layer with 64 features, there needs to be a weight matrix with shape $(128, 64)$ and since the dot product is used, there are $128 \times 64 = 8192$ FLOPs needed for this one layer for one sample.[5] Different deep-learning libraries like *tensorflow* and *pytorch* offer so-called profilers to measure FLOPs or similar metrics. However, the results of those profilers are seldomly exactly the same since the internal implementation of a layer may differ. Also, when using recurrent layers for sequential data, the number of FLOPs needed to compute an output depends on the length of the sequence. As a result, the number of needed FLOPs to train a recurrent model could be lowered by using shorter sequences. These inaccuracies result in the numbers of FLOPs being mentioned throughout this thesis being hardly comparable with numbers mention in other articles. Thus, this thesis does not compare models using their exact number of FLOPs needed during training but looks at what kind of layers architectures contain, how costly these layers are and which other architecture type eliminates costly layers.

[4] `https://commoncrawl.org/2019/07`, accessed 2020-23-12.

[5] In practice, such a layer typically also has a bias which is added to the result of such dot product. However, modern hardware is optimized to do these two operations in a single operation.

Chapter 3

Siamese training

The siamese architecture was introduced as a method of how to pretrain neural networks (2.2.5). The following chapter describes several pratical experiments in which the approach of training neural networks using that architecture is used together with using weak labels (see section 2.1.6). Since, both, *weak supervision* and siamese neural networks falls under the category of unsupervised learning, all experiments are compared with other architectural approaches which do not use the siamese architecture approach. It is then examined how hyperparameter settings as well as auxiliary training targets influence the result of the training.

To show that the siamese training approach is applicable to different kinds of data, three different types are used. The shown exemplary input data types are presented in the following order:

static data is used in the form of word embeddings. Words presented to the network together with their context in order to learn semantic representations.

visual data is used in the form of image sequences. Images themselves can be seen as two dimensional data, whereas the two dimensions are width and height of the image. When each positions contains three values to represent the color value of the corresponding pixel, the images are three dimensional. In the presented examples the images are two dimensional but have a time dimensionality.

textual data is used in the form of sentences. Sentences are represented as a sequence of tokens and are, similar to word embeddings, to the neural network in their context, whereas context on a sentence level shall be defined as neighbouring sentences or, on a broader level, sentences from the same document.

3.1 Word embeddings

Word embeddings are the results of a process in which words from a dictionary are mapped onto a numerical representation that makes it possible to calculate a similarity between words. The aspects with which two words are considered similar depend on the chosen method of creating the numerical representations. Word embedding approaches are built on meta-data derivable from text data, like which words are appearing in the same contexts ([61, 48, 78]), other approaches built on ontologies like the Wikipedia article link structure [97]. In all cases, it is assumed that words themselves have no meaning. Their meaning depends on words in a context. Unknown words, for example, have no meaning to a reader. Only when that unknown word is used in different contexts and or referred to as a physical object, it gains meaning. Word embeddings make use of this concept. Machine learning approaches do not have prior knowledge of language or general knowledge about the world. Machine learning approaches are bound to only experience things in the form of numbers or indexes, respectively. This section first describes the linguistic theory behind word embeddings before then presenting evaluating different approaches to train word embeddings with respect to distance functions and model architectures.

3.1.1 Structural linguistics

Data contains information. That information is represented by different symbols which themselves do not contain meaning. When considering spoken language, language consists of a sequence of sounds, whereas the sounds only make sense when combined using the target language's rules. The same rule applies when language is being represented by text. Textual data consists of character sequences. The characters are needed to build words, which, when being combined correctly, form sentences in the target language. Although not mentioned in linguistical contexts, the same pattern applies to visual data: Images are represented by pixel, whereas a single pixel does not have meaning. However, a group of pixels can create patterns like edges and areas, whereas the combination can form complex objects.

The idea of linguistical concepts being represented by smaller things which on their own do not have meaning is called *structural linguistics* [48]. Here, utterances are being viewed on different levels [118] which can be (among others):

phonemes , which are sounds used to form morphemes

morphemes , which are one or more phonemes, which are used to create words. The English word *cats* consists of the morphemes *cat* and *-s*, whereas the first morpheme is also a valid word, whereas the second one is not.

lexical categories , which (also called part-of-speech tags) are categories which can be assigned to words to determine their role in a sentence. Lexical categories are (among others) nouns, verbs, and adjectives.

noun phrases contain a (pro-)noun as the sentence's subject.

On each of these different levels, the different units are in two kind of relations [114]: *syntagmatic relations* and *paradigmatic relations*. When looking at the two relations on a word level, two words are in a paradigmatic relation if they appear in a similar context. In the two sentences *The dog likes to eat food* and *The cat likes to eat food*, the words *dog* and *cat* are surrounded by the very same words. Thus they appear in the same context and share a paradigmatic relation. On the other hand, words share a syntagmatic relation when they appear in the same context. Thus, every word shares a syntagmatic relation with all other words from the sentence it appears in.

The appearance of two words in the same sentence can also be called *co-occurrence* ([48, 12], among others). As a consequence of that definition, given enough data, any two words can share a syntagmatic relation. To circumvent the problem, the definition is extended such that two words are *significant co-occurrences* if their joint appearance is not random, which again is defined by the probability of two words appearing together is above a predefined threshold [62].

3.1.2 Model types

Co-occurrences

In linguistics, co-occurrences ([61, 48], among others) refer to the joint occurrence of two lexical units (typically words) in a corpus, document, or sentence. It is assumed that there exists a dependence between these two terms if their joint occurrence happens conspicuously often together. When looking at the two sentences *The weather is sunny today* and *The weather is cloudy today*, the words *weather* and *today* appear together, from which an assumption can be drawn that they in some way related. In practice, co-occurrences are found by counting words and calculating their relation using, for example, the dice coefficient. Some authors use the term *collocation* when two words are occurrence directly adjacently ([10], among others). When considering the word pairs *strong coffee* and *powerful coffee*, *strong* and *powerful* do not co-occur but both have the same collocation. When two words appear in the same context but do not co-occur, this can be a strong hint for two words being synonyms.

Co-occurrences are calculated by using a matrix. Each row and column in that matrix represents one type from a dictionary. The values in the matrix represent the number of co-occurrences of the two words, depending on the specific row and column. Here, co-occurring can be defined as two types appearing in the same window, the same sentences, paragraph, document, or any other way how to split the used corpus into different parts. After having created the co-occurrence matrix, a similarity measure like the dice-coefficient can be used to calculate the similarity of two types: Given the words that appear alongside a given term, if two terms are appearing in similar contexts, their dice-coefficient will be high. If two types appear in different contexts, their dice-coefficient will be low.

Word2Vec

The use case of Word2Vec (see section 2.2.3) is to use the resulting word vectors to calculate similarities between words. When looking at the Word2Vec architecture, two problems stand out. The first one is that the architecture consists of two matrixes, of which only the first matrix will be used after training. The second matrix is only necessary for the training process. This problem can be solved by using the same matrix for both steps, i.e., encoding and decoding. The second problem is that the network is trained to predict a context (or one word given a context), but when using the model, the actual prediction is not used but only the network's internal state. When training the network, the fact that semantically similar words receive similar vector representations is a pleasant side effect, but the network is not explicitly trained for this. The resulting problem is that the network is not trained to compare two vectors with a specific computational method like euclidian or cosine distance but still, cosine distance is typically used when comparing two-word vectors.

When considering computation costs, the encoding step is very cheap since the word's vector only has to be taken from the weight matrix without any other computation needed. The expensive part for training the model lies in predicting a word's context: Here, the probability for each word has to be calculated by first using the dot product involving the word vector and the weight matrix before then calculating the probability for each word using the softmax activation. The number of needed FLOPs for training depends on the dictionary size D and vector size V. A dot product has to be calculated for decoding, so the number of FLOPs is $D \times V$. Using only a random subset for the decoding steps replaces the dot product at the cost of not training all weights at the same time.

Siamese architecture

The siamese architecture (see section 2.2.5) has been described as an encoder architecture with no decoder. Input pairs are mapped onto a vector representation and then compared using a predefined similarity measure. When using a siamese architecture for word vector representations, the training procedure is very similar to the one being used by Word2Vec: Training data is used such that words are taken from sentences together with their context, i.e., neighboring words. Two words are given into the network, and their vector representation is trained such that the similarity is 1 (or that the distance is 0) given the predefined similarity measure. The network has to be trained with negative pairs to consider the trivial case where the network assigns the same vector representation to every input sample, which would always result in a vector similarity of 1, to which the network has to give a vector similarity of 0. Negative pairs can be any two randomly chosen words from the dictionary. By chance, two words that appear together often could be chosen as a negative pair that way, but given the fact that they are often appearing together and thus often a positive pair, choosing them as negative pair does no harm but only

slows down the training.

When comparing the siamese architecture to the Word2Vec architecture, the two mentioned problems of Word2Vec are solved: Word2Vec uses a decoding matrix in training, which will not be used after training, while the siamese architecture does not need a decoding matrix. Also, the vector representations are trained to be similar using a specific similarity measure. Thus two word vectors being similar is not a pleasant side effect but the actual training target. Computation costs and the number of FLOPs during training are also significantly lower since neither a dot product nor a probability has to be calculated.

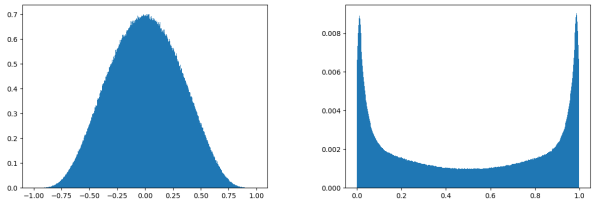
3.1.3 Distance functions

When training a siamese model, the network creates a vector representation of two input samples. That input samples are then compared. If the pair is positive, the distance between the two vector representations has to be small. When the pair is negative, the distance has to be high. Deciding for a specific distance function leads to different results in the vector representations and impacts the model's further use. This section describes four types of distance functions. In some contexts, the term similarity functions is used. Here, all distance functions are defined, such that a positive pair receives a distance value of 0 and a negative pair a distance value of 1. The similarity value can be calculated using the formula $similarity = 1 - distance$. When training neural networks, it can be suggested to limit the model's output. Thus, when using cosine distance, the vectors are activated using *tanh* to limit the feature range to $(-1, 1)$. For all the other distance functions, the vectors are activated using *sigmoid* to limit the feature range to $(0, 1)$. Without these limitations, values could become very large, making training unstable.

Cosine distance. When calculating the distance of two vectors, cosine distance is often used, for example in the original article describing Word2Vec [78]. Cosine distance takes two vectors as input and calculates the angle between them:

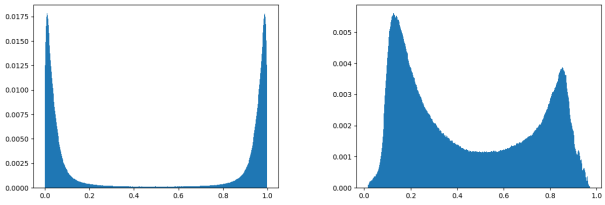
$$\cos(A, B) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{3.1}$$

When two vectors are placed in a vector space, their distance is not calculated by their distance from the origin but only from their direction. When considering a two-dimensional space, when there is a vector $(1, 1)$ and a vector $(2, 2)$, the second vector is farther away from the origin than the first vector. However, when traveling from the origin to both data points, both points lie on the same line, which creates an angle of $0°$ between them. Thus their cosine distance is 0. When there are two vectors $(1, 1)$ and $(-1, -1)$, their distance from the origin is the same. To reach them from the origin, one has to draw lines from the origin in the



(a) Model trained with cosine distance

(b) Model trained with euclidian distance



(c) Model trained with Jaccard distance

(d) Model trained with weighted Jaccard distance

Figure 3.1: Average feature value of two siamese models trained with different distance functions.

opposite direction. Thus, the angle between those lines is $180°$, which results in the maximum cosine distance of 1. When training a siamese model with the cosine distance, each feature's absolute value does not play a significant role. It is much more important whether a value is below or above 0. Since neural networks are typically initialized using Gaussian distribution such that each weight is around 0, the weights hardly have to be changed during training, which results in the model converging much faster. The final distribution of exemplary features can be found in figure 3.1a. As the downside of using cosine distance compared to the other distance functions is that calculating cosine distance is much slower.

Euclidian distance. When considering a vector space, euclidian distance calculated the squared distance of two points:

$$eucl(A, B) = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2} \tag{3.2}$$

Contrary to cosine distance, the two points' relation to the vector space's origin is not considered. When beginning training, each feature's value starts at 0.5 after sigmoid activation. Thus, all input samples are placed very close to each other in the vector space. To accord for negative data pairs which shall receive a distance value of 1, features are changed such that they are either close to 0 or 1. Since the

weights have to be changed more than the needed changed when using cosine distance, training takes more time. On the other hand, calculating euclidian distance is much faster. An exemplary feature distribution of a model trained with euclidian distance can be found in figure 3.1b.

Jaccard distance. The Jaccard distance (also called Jaccard index or Intersection over Union (IoU)) calculates the distance or similarity between two sets. Here, the intersection of the two sets is put in relation to the two sets' union. If the intersection is the same as the union, the two sets are equal. On the other hand, the smaller the intersection is in relation to the union, the more distant the two sets are.

$$jaccard(A, B) = 1 - (\frac{AB}{2\sum A} + \frac{AB}{2\sum B}) \tag{3.3}$$

When dealing with numerical vectors being bound to values between 0 and 1, the intersection is the product of these two vectors. If the two vectors are $(1, 1)$ and $(1, 0)$, their intersection is $(1, 0)$. The relation of the intersection to the first vector can be calculated by dividing the sum of the intersection by the sum of the first vector, which is $1/2 = 0.5$. For the second vector, this formula equals 1. The average of the two results has to be calculated to get the Jaccard similarity, which is 0.75. If these two vectors were to be made more similar, the feature's values which are not 1 for both vector have to be changed: When changing the second feature of the first vector to 0 or the second feature of the second vector to 1, the two vector's Jaccard similarity would equal to 1. This means only features with a value of 1 have an impact on two vector's Jaccard similarity.

Intuitively spoken, two vectors have high Jaccard similarity if the same features are *present*. When training a model using Jaccard distance, the neural network must learn which features are shared for a given input pair. An advantage compared to euclidian and cosine distance is that two vectors easily can be made distant: When a negative training pairs two vector's *present* features partially overlap, only these overlapping features need to be changed by setting them to 0 for one of the two vectors. The rest of the *present* features can be changed without affecting that distance. Simultaneously, when using cosine or euclidian distance, any change in one input sample's vector representation affects its relation to all other samples since each feature will have the same impact on the distance to other vectors.

A downside of using Jaccard index lies in calculating the product of two vectors: For example, if two vectors have the value 0.5 for one feature, the product will be 0.25. This again leads to a similarity smaller than 1, although the feature's two values are the same. This means that features' values have to be close to either 0 or 1. This effect can be seen in figure 3.1c, showing exemplary feature values of word embeddings.

Weighted Jaccard distance. To overcome the disadvantage of values having to be either 0 or 1 when using Jaccard index, an extension of that approach can be

used:

$$weighted\_jaccard(A, B) = 1 - (\frac{\sum_i \min(A_i, B_i)}{\sum_i \max(A_i, B_i)}) \tag{3.4}$$

Here, instead of using the product of two vectors, each feature's value is considered using the minimum and maximum values. If two vectors are $(1, 0.5)$, their Jaccard similarity would not equal 1 since the vectors' second feature does not equal either 0 or 1. On the other hand, the weighted Jaccard index accounts for the values being equal while at the same time weighing it lower than the first feature. When using Jaccard similarity, features either have to be 0 or 1, meaning features are binary. When using weighted Jaccard similarity, features can be any value between 0 or 1. This difference can be seen in figure 3.1d.

3.1.4 Word embedding evaluation

After having explained three approaches to calculate word similarity, the presented approaches are to be evaluated. The problem when evaluating models that have been trained in an unsupervised manner to assign vector representation to words is that there are no correct or incorrect vector representations. Thus, the vector representations have to be evaluated for plausibility: The evaluation is done qualitatively by looking at exemplary words and similar words and quantitatively using lists of predefined word pairs.

There are three basic types of models that are to be compared:

- Word2Vec
- Siamese architecture
- Co-occurrences
- GloVe

The first two types are neural network-based and are trained on a list of ten million English sentences taken from the Wortschatz project at Leipzig University[1]. The sentences contain no specific content, but they are a subset of a larger dataset. Each sentence was transformed into a POS tag sequence from that larger dataset, which was then used to find typical and non-typical sentence structures [86]. The mentioned subset, which has been used for training, contains sentences that contain typical POS tag sequences. The Word2Vec model was trained for about 48 hours using a library called Gensim[2] and contains 120,000 words.

Of the siamese architectures, four have been trained, which differ only in the type of activation and distance function used as described in section 3.1.3. Both models have been trained for twelve hours.

[1] `https://corpora.uni-leipzig.de/`, [40]

[2] `https://radimrehurek.com/gensim/`

The fifth type of model is a set of pre-calculated co-occurrences by the aforementioned Wortschatz project based on different types of data sources, like news and Wikipedia articles. The data contains similarity values for pairs calculated using cosine and the dice-coefficient. If a pair is not in the given data set, its similarity is unknown.

The sixth type of models are word vectors trained using *fasttext*[3]. The authors mention that there are many different words with minor word variations for some languages, as is the case for German. To overcome this problem, they describe an approach for which there is not one word vector for one word, but instead, words are described by character tri-grams, which themselves get a vector representation. A word's vector representation then is the sum of the vector representations of its character tri-grams.

Lastly, the GloVe [4] model is a set of word vectors which are created from singular value decomposition and are based on a co-occurrences matrix. The GloVe model contains 2,196,017 words. The authors mention that similar words can be found using the cosine or euclidian distance.

Similar words

For evaluating similar word lists, reference words are chosen as a pre-chosen type. The top 20 words have been calculated for each type of model. The lists are to be evaluated whether the found similar words are of the expected type. The chosen types are:

- stop word
- high frequent verb
- verb in past tense
- domain-specific noun

Stop words. Stop words are used to build sentences and contain no semantic meaning themselves. They rather connect words with semantic meaning. Having no semantic meaning, it is expected that similar words also have no semantic meaning, thus are stop words themselves. As an exemplary stop word, the word *an* was chosen. The results in table 3.1 are interesting because there are not only exclusively other stop words. Moreover, the words seem to be rather non-sense. The given definition of stop words contains the aspect that a word has no semantic meaning. When training is just beginning for a neural network, the network does not know anything about the words. If then, during training, words are shown in different contexts, their vector representation is moved to a designated place in the vector space. If a word is not being presented in many different contexts, but for

[3] `https://fasttext.cc/`, [11]

[4] Global Vectors, `https://nlp.stanford.edu/projects/glove/`, [91]

Rank	Siamese cosine	euclidian	Jaccard	weighted Jaccard
1	gratification	a	idolater	afterthought
2	another	gratification	adopter	eyesore
3	a	another	acko	outdoorsman
4	the	warrants	aider	understatement
5	any	the	appendectomy	epiphany
6	warrants	mech	ressam	ultimatum
7	gyros	continuous	aesthete	phoblacht
8	this	beholder	glazyev	errant
9	nesters	forensic	arrestable	electrician
10	beholder	chaos	oxymoron	undisclosed

Table 3.1: Similar words of the word *an* for models trained with the siamese approach.

Rank	Siamese cosine	euclidian	Jaccard	weighted Jaccard
1	another	a	a	that
2	a	so	this	the
3	the	the	and	this
4	any	that	in	a
5	this	and	which	so
6	it	this	as	it
7	in	something	that	of
8	that	as	it	in
9	as	it	for	only
10	for	which	being	another

Table 3.2: Similar words of the word *an* for models trained with the siamese approach with non-frequent words filtered.

	W2V		Crawl	
Rank	cosine	euclidian	cosine	euclidian
1	another	a	it.An	An
2	a	so	them.An	another
3	the	the	.An	so
4	any	that	-an	the
5	this	and	-An	that
6	it	this	–an	and
7	in	something	withan	a
8	that	as	time.An	this
9	as	it	ofan	something
10	for	which	here.An	as

Table 3.3: Similar words of the word *an* for neural network based models with non-frequent words filtered for the Word2Vec model.

	Co-occurrences		GloVe	
Rank	cosine	dice	cosine	euclidian
1	a	a	another	another
2	another	another	a	one
3	the	the	is	is
4	one	one	one	a
5	its	its	the	as
6	his	his	as	the
7	some	some	which	which
8	any	any	that	it
9	plenty	plenty	with	be
10	more	more	it	only

Table 3.4: Similar words of the word *an* for co-occurrences based models.

Rank	Siamese cosine	euclidian	Jaccard	weighted Jaccard
1	hear	hear	know	know
2	forgive	forgive	do	think
3	convince	kill	anything	do
4	kill	teach	let	feel
5	remind	inspire	hear	what
6	teach	recognize	think	understand
7	criticize	explain	something	hear
8	inspire	telling	remember	mean
9	inform	remind	imagine	remember
10	marry	frighten	anybody	want

Table 3.5: Similar words of the word *tell* for models trained with the siamese approach.

Rank	W2V cosine	W2V euclidian	Crawl cosine	Crawl euclidian
1	hear	hear	telling	telling
2	forgive	forgive	Tell	say
3	convince	kill	tell.	told
4	kill	teach	tells	know
5	remind	inspire	Telling	ask
6	teach	recognize	told	do
7	criticize	explain	tell.The	tells
8	inspire	telling	re-tell	believe
9	inform	remind	say	see
10	marry	frighten	retell	story

Table 3.6: Similar words of the word *tell* for neural network based models.

example, only one context, its vector representation is very close to the words it co-occurred with. Thus, a word like *outdoorsman* may have only occurred once together with the word *an*. The results in table 3.2 show similarities of the same model, whereas non-frequent words have been filtered and are therefore not listed. The lists now show only stop words as similar words. The Word2Vec model trained with the gensim framework suffered from the same problem. Its result list is shown with filtered words (see table 3.3.

The result lists for the other model types show other stop words. The only exception is when the crawl model with cosine distance. As was mentioned before, the crawl model's word vectors are created from character tri-grams. Thus, a different word with similar character tri-grams will get a similar vector representation.

	Co-occurrences		GloVe	
Rank	cosine	dice	cosine	euclidian
1	ask	ask	know	know
2	know	know	telling	telling
3	see	see	ask	ask
4	let	let	say	say
5	give	give	what	why
6	remember	remember	why	what
7	do	do	let	let
8	get	get	me	sure
9	hear	hear	want	want
10	understand	understand	how	how

Table 3.7: Similar words of the word *tell* for co-occurrences based models.

Rank	cosine	euclidian	W2V	
1	cat	cat	cat	cat
2	puppy	puppy	neighbor	neighbor
3	kitte	pig	wallet	wallet
4	toy	kitten	horse	horse
5	rat	horse	baby	laptop
6	pet	monkey	car	toy
7	pig	mouse	laptop	puppy
8	horse	bunny	toy	baby
9	mouse	bird	nose	nose
10	bug	hound	pig	bird

Table 3.8: Similar words of the word *dog* for neural network based model types.

High frequent verb. Tables 3.5, 3.6 and 3.7 contain lists of similar words for the high frequent verb *tell*. All lists except the lists for the GloVe model contain words that play a similar role in a sentence, i.e., the word tell could in some sentences easily be replaced with a verb like *ask*.[5] It shall be noted that the similar words of the siamese models are closer to other high-frequency words regarding conversations compared to the similar words of the Word2Vec model. The similar words of the GloVe model do not seem to be different verbs, but words that are likely to appear in a sentence alongside the verb *tell*. For the crawl model with cosine as distance function, there are again words with similar character tri-grams at the top of the list.

Frequent noun. Tables 3.8, 3.9 and 3.10 show similar words for the noun *dog*. As with the similar word lists of the verb *tell*, the similar words of the Word2Vec

[5]Exemplary sentences could be *Let me tell you.* and *Let me ask you.*

	W2V		Crawl	
Rank	cosine	euclidian	cosine	euclidian
1	cat	cat	dogs	dogs
2	neighbor	neighbor	puppy	puppy
3	wallet	wallet	Dog	Dog
4	horse	horse	doggie	pup
5	baby	laptop	canine	doggie
6	car	toy	pup	pet
7	laptop	puppy	dog.	canine
8	toy	baby	doggy	doggy
9	nose	nose	dachshund	cat
10	pig	bird	terrier	poodle

Table 3.9: Similar words of the word *dog* based on Word2Vec models.

	Co-occurrences		GloVe	
Rank	cosine	dice	cosine	euclidian
1	dogs	dogs	dogs	dogs
2	sled	cat	puppy	puppy
3	cat	pet	cat	pet
4	pet	sled	pet	cat
5	bird	animal	pup	puppies
6	barks	bird	canine	canine
7	treats	pets	puppies	pup
8	cats	horse	cats	cats
9	mushers	girl	kitten	terrier
10	animal	animals	terrier	kitten

Table 3.10: Similar words of the word *dog* for co-occurrence based model types.

	Siamese			
Rank	cosine	euclidian	Jaccard	weighted Jaccard
1	stole	acquired	had	dislocated
2	sold	spotted	worked	slept
3	acquired	stole	lived	stole
4	purchased	rented	started	caught
5	spotted	dug	played	saw
6	visited	ate	was	ran
7	rented	poured	became	walked
8	rebuilt	drank	met	paroled
9	ate	visited	gave	thrown
10	drove	drove	shot	fainted

Table 3.11: Similar words of the word *bought* for models trained with the siamese approach.

	W2V		Crawl	
Rank	cosine	euclidian	cosine	euclidian
1	stole	acquired	purchased	purchased
2	sold	spotted	sold	sold
3	acquired	stole	puchased	buy
4	purchased	rented	baught	got
5	spotted	dug	Bought	buying
6	visited	ate	bougth	puchased
7	rented	poured	purchsed.	purchsed
8	rebuilt	drank	buy	purchase
9	ate	visited	splurged	went
10	drove	drove	purchaced	used

Table 3.12: Similar words of the word *bought* based on Word2Vec models.

model seem to be of a wider variety, as can been seen by words like *laptop* and *wallet*, which are not related to the domain of animal words. The other models all have other animals or words closely related to dogs (*sled*) as similar words. The only exception for this are the two siamese models having been trained using the Jaccard distance. Their similar words contain lists that are not words from the animal domain but are words that are probably used alongside the word *dog* like, for example, *lovely* or *little*.

Verb in past tense. Tables 3.11, 3.12 and 3.13 show similar words for the high frequent verb *bought*. All models show different verbs which are also in past tense. The only exception are the word lists of the crawl model, which have a high share of words with similar character tri-grams.

	Co-occurrences		GloVe	
Rank	cosine	dice	cosine	euclidian
1	purchased	purchas	purchased	purchased
2	buy	buy	sold	sold
3	sold	sold	buying	got
4	buying	buying	buy	buying
5	acquired	acquired	sell	picked
6	sell	sell	got	sell
7	owns	owns	selling	wanted
8	paid	paid	sells	selling
9	purchase	purchase	picked	buy
10	invested	invested	buys	decided

Table 3.13: Similar words of the word *bought* for co-occurrences based models.

	Siamese			
Rank	cosine	euclidian	Jaccard	weighted Jaccard
1	iframes	iframes	online	iframes
2	iframe	geolocation	watch	password
3	trackback	iframe	address	username
4	retargeting	password	visit	app
5	reprint	bookmark	send	delete
6	newsnow	embed	welcome	publish
7	spamming	url	register	customize
8	arvixe	username	find	login
9	https	html	share	spam
10	plugin	cultist	eat	subscription

Table 3.14: Similar words of the word *browser* for models trained with the siamese approach.

Domain specific noun. Tables 3.14, 3.15 and 3.16 show similar words for the domain-specific word *browser*. This word was chosen because it is linked to a single domain. The lists of similar words all contain words from that same domain solely, but the lists based on neural network models, i.e., the siamese models and the Word2Vec model, contain words in a wider variety. This is coherent with the previous lists, where also a wide variety of words from the same context was shown.

All model types show plausible similar words for the chosen reference words. However, it has to be mentioned that the models have been trained for different amounts of time: The siamese models have only been trained for twelve hours and show similar results as the Word2Vec model, which has been trained about four times longer. It was explained that the siamese model only contains an encoder

	W2V		Crawl	
Rank	cosine	euclidian	cosine	euclidian
1	upgrade	inbox	browsers	browsers
2	inbox	upgrade	broswer	Browser
3	password	username	web-browser	web-brows
4	subscription	calendars	Browser	broswer
5	smartphone	organize	Firefox	Firefox
6	buy	horizons	browers	browers
7	activate	hips	webbrowser	FireFox
8	username	homepage	browser.	browser.
9	calendars	doorstep	firefox	webbrowser
10	bodyweight	fingertips	FireFox	firefox

Table 3.15: Similar words of the word *browser* based on Word2Vec and co-occurrences similarities.

	Co-occurrences		GloVe	
Rank	cosine	dice	cosine	euclidian
1	browsers	browsers	browsers	browsers
2	chrome	chrome	firefox	firefox
3	firefox	firefox	browser	ie
4	os	os	ie	toolbar
5	safari	web	toolbar	web
6	web	safari	javascript	webpage
7	web	web	mozilla	browsing
8	browsing	browsing	netscape	user
9	desktop	desktop	web	mozilla
10	mozilla	explorer	user	webpages

Table 3.16: Similar words of the word *browser* for co-occurrences based models.

part, whereas the Word2Vec model contains both an encoder and a decoder part. Two aspects explain the additional needed training time: First, the additional parameters lead to additional computations when doing a feed-forward step and when doing backpropagation. Additionally, the parameters used as word embeddings are the parameters from the model's encoder part. When training, these gradients depend on the gradients of the decoder part of the model. When using backpropagation, the error is passed through the network using its weights. The error gets multiplied by the weights, and since weights are typically small values centered around zero, the errors get smaller with each additional layer. When comparing a Word2Vec model with a siamese model, there is one additional layer between the encoder part of the Word2Vec model and the point where gradients are calculated initially.

It was also shown that the Word2Vec model's similar word lists often contain words that seem to not belong to the same domain as the word with which to compare with, whereas the other models had fewer non-fitting words. The reason for this may be that the Word2Vec model's word vectors have been compared using either euclidian or cosine distance. Both distances gave plausible results, but, as mentioned before, the model was not trained to give word vectors that have to be compared using a specific similarity function. It just turns out that these two functions give somewhat plausible results. It shall be noted that the siamese models have been trained to create word vectors for which a specific distance function has to be used in order to compare these vectors. Using other functions would create a list with words that do not make sense, and thus these lists are not shown here.

It can be concluded that Word2Vec is an impressive tool that leads to useful word vectors, but it seems that the resulting lists of similar lists are a bit of a nice coincidence. It shall also be noted that the shown results could be different when using a model that has been trained longer and on more data. However, when comparing Word2Vec with the siamese approach, the siamese approach can get the same results with far fewer resources.

When comparing the siamese models with the co-occurrence-based models, they show similar results. A significant downside of the co-occurrence-based models is that each pair's similarities must be calculated in advance. With an increasing number of words, the number of pairs is growing exponentially, as is the needed amount of memory to save that similarity pairs.

Predefined word lists

In the previous chapter, lists of similar words have been evaluated in a qualitative matter by picking exemplary reference words and looking at similar word lists for different models. When looking at lists of similar words, the binary decision, whether a word is similar or not, differs from person to person. When given the reference word *dog*, some people would argue that *kennel* is a word related to it because it is from the same domain, while others would argue that even *cat* is not similar to *dog* since it is a different animal. To overcome this problem, this section

evaluates the previously used models quantitatively by using similar word lists that have been created by a group of people. When several people decide whether two words are similar, they will come to different conclusions. Using the average value, words with a higher average can be considered *more similar* than words with a lower average.

The presented word embedding models use different similarity functions to calculate the similarity score between words. When looking at similar words and their similarity score, looking at the actual score value is not very useful. For example, when working with neural networks with many features for the word embeddings, many of those features could be very similar or even the same for all the words because the network did not *need* these features. When calculating the similarity scores, these scores will never be zero because some values are always the same. Similarly, considering cosine or euclidian similarity, a similarity score of zero would mean that two vectors are exactly the opposite. It is hard two imagine two words having exactly opposite features. Two words like *black* and *white* may be opposites, but they still have in common that they are both colors. Thus, it is not possible to say for similar words which actual score could be expected.

When considering predefined word lists with similarity scores, it has to be noted that these lists contain word pairs with a specific range of similarity scores. The models which are to be used for this evaluation give different similarity scores. The following steps have been done to make the scores comparable: For all word pairs, the model's similarity score has been calculated and stored together with the predefined list score. These score pairs are used to create an average factor. That factor is used to convert the predefined list's similarity score to the models' score. For example, when the word pair *black* and *white* has a similarity score of 5/5 in the predefined list, and the euclidian similarity is 0.95, the resulting factor will be 0.19. This also respects the fact that the used models only give the maximum similarity score when the same word is paired with itself, and otherwise, the score will always be lower than one. Using such a factor can determine the maximum similarity score for word pairs, where the two words are not the same. When it has to be evaluated how close the model's prediction is for a specific word pair, the used factor shall be the average factor of all known pairs except for the pair that has to be evaluated. The evaluation results are the average absolute distance of the model's prediction to the predefined similarity. Since the co-occurrence-based models can only give similarity scores when a score for that pair has been pre-calculated, missing pairs are skipped.

Three datasets will be used to evaluate the used models:

WordSim353 The first dataset is the *WordSim353* [1] dataset. The dataset consists of 353 word pairs, which have been annotated by either their relatedness or their similarity on a scale from 0 to 10. The list of similar pairs mostly contain words that have something in common, e.g. *train* and *car* are both vehicles or *physics* and *chemistry* both are sciences. The list of related pairs mostly consists of two words which both belong to the same domain. Exemplary

	Siamese			
	cosine	euclidian	Jaccard	weighted Jaccard
ws353, similarity	0.044	0.064	0.059	0.047
ws353, relatedness	0.038	0.054	0.047	0.045
rg65	0.096	0.121	0.111	0.103

Table 3.17: Average error on the datasets ws353 and rg65 for models trained with the siamese approach.

	W2V		Crawl	
	cosine	euclidian	cosine	euclidian
ws353, similarity	0.066	0.068	0.191	0.19
ws353, relatedness	0.045	0.049	0.087	0.099
rg65	0.112	0.112	0.305	0.359

Table 3.18: Average error on the datasets ws353 and rg65 based on Word2Vec similarities.

highly related pairs are *computer* and *keyword*, *country* and *citizen* or *day* and *dawn*. Pairs which have received low relatedness scores are for example *drink* and *ear* or *rooster* and *voyage*.

RG65 The second dataset is called *RG65* [108] and contains 65 word pairs. The pair similarity was defined on a scale from 0 to 4. An exemplary high similarity pair is *midday* and *noon*, an exemplary low similarity pair is *fruit* and *furnace*.

MEN The third used dataset is called MEN [15] and contains 3000 pairs. The pairs are rated by their similarity in a range from 0 to 50. In contrast to the other two used datasets, all words are annotated by their part-of-speech tag. An exemplary pair of high similarity is *sun* and *sunlight*, of which both are annotated as nouns. A pair with a high similarity score and differing part-of-speech tags is *flame* and *burn*.

For all datasets, the score range will be normalized to values between 0 and 1 before splitting the existing pairs into *close pairs* and *distant pairs*. Close pairs are pairs with a similarity score bigger than 0.5. Distant pairs are pairs with a similarity score smaller than 0.5.

The results in tables 3.17, 3.18 and 3.19 show the average error for the chosen models on the ws353 and rg65 datasets. The neural network-based models perform better than the co-occurrence-based models, while the siamese models perform best. All models perform better on the similarity task compared to the relatedness task. For both ws353 tasks, similarity, and relatedness, the scores are in the same range, i.e., a similarity pair with a high score like *football* and *soccer* has the same value as a relatedness pair with a high score like *psychology* and *Freud*. Thus, the

	Co-occurrences		GloVe	
	cosine	dice	cosine	euclidian
ws353, similarity	0.111	0.113	0.224	0.223
ws353, relatedness	0.109	0.113	0.105	0.103
rg65	0.322	0.346	0.368	0.372

Table 3.19: Average error on the datasets ws353 and rg65 dataset for co-occurrence based models.

	Siamese		
	euclidian (48h)	Jaccard (600)	Jaccard (900)
ws353, similarity	0.052	0.051	0.046
ws353, relatedness	0.043	0.046	0.045
rg65	0.101	0.104	0.101

Table 3.20: Average error on the datasets ws353 and rg65 dataset for co-occurrence based models.

	Siamese			
	cosine	euclidian	Jaccard	weighted Jaccard
all pairs	0.045	0.071	0.057	0.052
only nouns	0.049	0.079	0.061	0.056
only verbs	0.024	0.033	0.027	0.027
only adjectives	0.024	0.049	0.028	0.023
nouns with rest	0.047	0.073	0.059	0.054
verbs with rest	0.036	0.052	0.048	0.044
adjectives with rest	0.024	0.049	0.028	0.023

Table 3.21: Average error on MEN dataset for models trained with the siamese approach.

	W2V		Cawl	
	cosine	euclidian	cosine	euclidian
all pairs	0.061	0.063	0.162	0.178
only nouns	0.066	0.067	0.182	0.203
only verbs	0.03	0.031	0.042	0.052
only adjectives	0.035	0.038	0.218	0.277
nouns with rest	0.063	0.064	0.166	0.182
verbs with rest	0.047	0.049	0.082	0.086
adjectives with rest	0.035	0.038	0.218	0.277

Table 3.22: Average error on MEN dataset for Word2Vec models.

	Co-occurrences		GloVe	
	cosine	dice	cosine	euclidian
all pairs	0.147	0.151	0.218	0.215
only nouns	0.169	0.173	0.258	0.254
only verbs	0.062	0.065	0.051	0.076
only adjectives	0.1	0.106	0.265	0.259
nouns with rest	0.16	0.163	0.231	0.228
verbs with rest	0.123	0.127	0.091	0.091
adjectives with rest	0.1	0.106	0.265	0.259

Table 3.23: Average error on MEN dataset for co-occurrence based models.

relatedness pairs are more distant in their meaning, which results in lower similarity scores created by the models. The previously described factor may even out the difference in ranges, but the range of semantic distance of the relatedness pairs varies more than the similarity pairs. The fact that all models perform better on the relatedness score shows that it is easier to say that two words belong to the same domain and thus are related compared to saying two words are closely related and thus similar.

When looking at lists of similar words in section 3.1.4, it becomes evident that the Word2Vec model's lists often contain words as being similar, which seemed to not belong in that lists. That effect may also be why the Word2Vec model performs worse than the siamese models.

In tables 3.21, 3.22 and 3.23, the evaluation results using the MEN dataset can be found. In general, the results are similar to the results of the other two evaluations, meaning that the neural network-based models perform better than the co-occurrence-based ones, while the siamese models perform better than the Word2Vec model.

When looking at the siamese models' results, it is evident that their performance differs regarding the distance function. The model trained with the cosine function always had the lowest average error, while the model trained with the euclidian distance function always performed worse. The reason for this may be because it is faster to train a model with cosine distance, as was explained in section 3.1.3. To further prove this point, a siamese model with euclidian distance was trained for 48 instead of 12 hours. The evaluation results for that model can be found in table 3.20, showing that the average error went down significantly. Another noticeable pattern is that the weighted Jaccard distance model is constantly performing better than the model with non-weighted Jaccard distance. The problem for this may be that when using non-weighted Jaccard distance, features are close to being binary (as shown in 3.1.3). Since the number of features for all models is the same, the fact that the features are binary decreases that model's expressiveness. Two additional models have been trained with 600 and 1200 features, respectively, to prove this point. The evaluation results of these models can

be found in table 3.20, showing that the downside of storing information binary can be evened out with more available features.

To summarize the effect of different distance functions, the experiments show that cosine distance seems to be the best choice when wanting to get the best results, although its execution time is higher compared to the other functions.[6] When speed is an issue, weighted Jaccard's performance is close to cosine distance while being significantly faster. Non-weighted Jaccard distance performs worse, although increasing the number of features solves this problem. Since binary values can be stored way more efficiently than float values, non-weighted Jaccard is a good choice when memory usage is an issue. On the other hand, euclidian distance performed worse on all evaluation experiments, which can only be solved by training longer. When comparing the siamese models' results to the other models, it is evident that they perform better. This demonstrates that they are explicitly trained to be used with a given distance function results in better results than models that are not explicitly trained for any distance function.

Transfer learning

The result of unsupervised learning tasks can be used to map data onto a reduced vector representation and before comparing input samples with each other, as has been done in the previous two sections 3.1.4 and 3.1.4. However, as was mentioned in section 2.1.2, trained models in an unsupervised manner can also be used as pretrained models on another task. This section makes use of pretrained word embeddings and transfers them onto different tasks. The evaluation pipeline used is called *SentEval*[7]. SentEval offers a standardized evaluation of word and sentence embeddings on different predefined tasks. When using this pipeline, either pre-calculated word embeddings can be used or, when wanting to use sentence embeddings, a custom function has to be defined with which to vectorize given sentences. The created vectors are then given into a simple multilayer neural network before creating an output for one of the tasks. The tasks used in the thesis are:

MR contains each 5331 positive and negative movie reviews [131].

CR contains 2407 positive and 1368 negative product reviews [53].

SUBJ contains each 5000 subjective and objective movie reviews and plot summaries [90].

MPQA contains words or word groups taken from news articles that have been classified for either being positive or negative [132].

[6] The cosine distance has been calculated using the *cosine_similarity* function of the *sklearn* library. The calculation of the similarity value of two vectors is significantly slower than calculating the other three distance functions.

[7] `https://github.com/facebookresearch/SentEval`, [29]

	Siamese			
	cosine	euclidian	Jaccard	weighted Jaccard
MR	66.37	66.27	65.24	69.14
CR	73.99	73.46	74.36	75.31
SUBJ	84.59	84.99	84.63	86.81
MPQA	77.38	78.66	78.96	81.78
SST2	66.78	67.38	67.0	71.33
SST5	34.07	34.57	33.12	36.02
TREC	64.6	68.0	73.4	79.4
MRPC	69.62	70.38	69.97	69.51
SICKE	73.8	76.5	72.1	74.67

Table 3.24: Test accuracy on the senteval tasks for the models trained with the siamese approach.

	W2V	Crawl	GloVe
MR	63.46	75.27	77.42
CR	72.03	77.51	80.77
SUBJ	80.55	89.89	91.56
MPQA	73.39	86.09	87.69
SST2	62.99	77.76	81.11
SST5	31.27	39.86	44.34
TREC	77.2	80.8	84.2
MRPC	68.52	71.88	73.1
SICKE	74.47	78.95	78.77

Table 3.25: Test accuracy on the senteval tasks.

SST2/SST5 are datasets containing sentences with either binary or fine-grained sentiment information [124].

TREC contains 5000 questions of different categories [66].

MRPC contains sentence pairs that are labeled as either being paraphrased or not paraphrased [33].

SICKEntailment contains sentence pairs which are labeled with a similarity score[8].

Table 3.24 and 3.25 shows the results for several tasks from the SentEval framework. When looking at the results, it becomes evident that the siamese approach performs better on all tasks than the Word2Vec model, having been trained using the *gensim* framework but worse to the Crawl and GloVe word vector models.

Simultaneously, when looking only at the four siamese models' results, the model trained with euclidian distance performs better on all tasks than the model

[8] https://wiki.cimec.unitn.it/tiki-index.php?page=CLIC

having been trained with cosine distance. This may be related to the fact that the evaluation tasks deal with sentences and thus with sequences of word vectors. These vector sequences are merged using each feature's average value. Creating the average value of these vectors has a different impact on the two model types since their features have different characteristics (see figure 3.1). When using cosine distance to compare two vectors, the angle of two vectors is calculated. The angle's origin is the value 0. The features of the model having been trained using cosine distances all lie around 0. When a feature is changed from $+0.001$ to -0.001, the absolute change is minimal, but this change can have a huge impact when calculating an angle. Thus, when training the model using cosine distance, the feature values do not necessarily need to become large. Instead, assigning a small but precise value can lead to good results. However, when these precisely trained values are averaged, information can get lost. On the other hand, when the model is trained using euclidian distance, the features have to be as much distant from each other as possible to account for negative examples. Thus, the features are close to 0 or 1. When creating the average value of a sequence of features of which, for example, a lot have the value 1 and some 0, the average value may still be high, which means less information has been lost.

Training variations

Next, siamese models are trained using different training parameters to investigate different settings' impact on evaluation results.

Selected data It can be assumed that when applying a trained model to a task like the movie review task (MR), lots of word vectors are never used. Thus when pretraining the model, lots of trained patterns are never needed. Instead, the time spent on learning unneeded patterns could be spent learning patterns that are actually needed for a specific task. The models are trained on the same data but only with word pairs that appear in the tasks. For example, for the MR task, a model will be trained on the same dataset as before, but when choosing word pairs, only words are chosen that appear in the MR task's data. Thus, the model does not know about words that do not appear in the task's data.

Table 3.26 shows the evaluation results of the additional experiments. There has been a model trained for each specific task and distance function. Interestingly, training the models in filtered data leads to no conclusive results. Looking at this the other way around, adding more data to these tasks did not impact the results either. This result is contrary to the common belief that adding more data will always lead to better results. When generalizing this statement, adding more data to a machine learning experiment could possibly lead to worse results. The additional data can contain patterns that harm a model's performance when applied to a transfer task.

On the other hand, there may be patterns in data that are so common, even with small datasets, they are learned, while adding more data will not affect a

	Siamese			
	cosine	euclidian	Jaccard	weighted Jaccard
MR	$+3.84$	$+1.17$	-0.3	-0.43
CR	$+1.56$	$+0.63$	-0.4	$+0.74$
SUBJ	$+3.11$	-1.03	-4.77	-0.34
MPQA	-3.65	$+0.43$	-4.7	$+0.14$
SST2	-1.0	$+0.11$	$+1.98$	$+0.72$
SST5	$+3.21$	$+0.23$	$+2.94$	$+0.18$
TREC	-1.23	-3.56	-7.31	-4.4
MRPC	-4.58	-1.31	-2.73	-2.05
SICKE	$+1.01$	-0.45	-5.08	$+0.0$

Table 3.26: Test accuracy on the senteval tasks for the models trained with the siamese approach with selected data. The results are presented in relation to the results in table 3.24.

model's performance. When considering the SentEval tasks, the customer review task (CR) is about discriminating sentences by their positivity. This can be done by looking out for specific words like *good* or *nice*. When considering word embeddings, a model only has to give these words a similar vector representation to get good results. Contrary to this, the MRPC task dealt with analyzing sentence pairs and deciding whether they are paraphrased or not. This task requires way more knowledge since sentence pairs could be about any topic. The fact that all models performed better when trained with more training data further proves that point.

Triplet loss Table 3.27 shows the evaluation results of experiments with the same architectures as the models in table 3.24 but are trained with triplet loss with calculating the loss using as follows (see also section 2.2.5:

$$\text{loss}(a, p, n) = \max(0, sim(a, n) - sim(a, p) + \epsilon) \tag{3.5}$$

The results in table 3.27 show that using $max(0, \cdot)$, the models perform worse compared to not using that approach. When not using $max(0, \cdot)$, the models are trained to assign extreme similarity values to training pairs. Since pairs are created randomly, there are hardly predicted labels with extreme values, except if the shown pair is a well-suited representative for a positive or negative pair. Since the models have to predict extreme labels, meaning 1 for positive pairs and 0 for negative pairs, the loss and the resulting gradients are high, resulting in bigger weight changes. When using $max(0, \cdot)$, the loss value is 0 if the pair (a, p) is more similar than the pair (a, n). Since the average loss value is assumed to be lower compared to not using $max(0, \cdot)$, training is significantly slowed down. Also, the models trained with euclidian and cosine distance perform way better than the models trained with Jaccard index. When having to assign a similarity value of 0 to a negative pair with euclidian or cosine distance, this results in placing samples

	Siamese			
	cosine	euclidian	Jaccard	weighted Jaccard
MR	-3.27	-5.06	-5.5	-8.88
CR	-1.67	-9.7	-10.6	-11.55
SUBJ	-2.87	-7.26	-8.99	-11.07
MPQA	-2.2	-7.2	-6.99	-11.82
SST2	-1.27	-8.4	-7.69	-17.51
SST5	-2.03	-5.57	-1.67	-10.82
TREC	$+1.2$	$+0.2$	-6.2	-17.01
MRPC	-0.43	-3.31	-1.51	-3.02
SICKE	$+0.22$	-5.16	-7.22	-17.98

Table 3.27: Test accuracy on the senteval tasks for the models trained with the siamese approach and triplet loss using $max(0, \cdot)$. The results are presented in relation to the results in table 3.24.

into opposite spaces in the vector space. When using Jaccard index, on the other hand, the overlapping features are considered when calculating similarity. If there are no overlapping features, the similarity is 0, but a sample's features can still be changed without affecting a negative pair's similarity. Since it is possible when using Jaccard index to assign a similarity score of 0 to negative pairs, the loss value will be 0 more often compared to using euclidian or cosine distance, resulting in even lower average gradients and thus slower training.

Additional training targets In section 3.2, it is shown that a siamese model is only learning features that are necessary for the auxiliary training task. Patterns in the input data that are not needed are not represented in the encoded vector representation. On the other hand: The more complex the auxiliary training task is, meaning if many features from the input data are needed to pair positive correctly and negatives pairs, the richer the vector representation gets. Adding a different target to the model instead of pairing input samples is another way to enrich vector representations. Table 3.28 shows the test accuracy of several models on the SentEval tasks in comparison to the base models in table 3.24. These new models have been trained to pair positive and negative pairs correctly and have been given the additional task to predict an input token's POS tag given that token's vector representation. Thus, the POS tag information needs to be represented in the token's vector representation. For the model using cosine distance for calculating the similarity of two vectors, the results have improved. For all the other distance functions, the results are worse. This shows that using different similarity functions has a different impact on using additional training targets. When using euclidian, Jaccard similarity, and weighted Jaccard similarity, the similarity depends on the vectors' absolute values. Thus, when the model is trained with an additional training task, the absolute vector values negatively affect both training targets. On the

	Siamese			
	cosine	euclidian	Jaccard	weighted Jaccard
MR	$+1.75$	-1.78	-5.46	-3.77
CR	$+0.45$	-0.77	-7.16	-0.61
SUBJ	$+1.19$	-0.32	-4.88	-1.07
MPQA	$+3.42$	-3.39	-5.74	-5.3
SST2	$+2.69$	-0.77	-7.25	-5.27
SST5	$+2.4$	-2.62	-3.03	-2.9
TREC	$+14.0$	$+4.8$	-1.6	-0.2
MRPC	$+0.07$	-1.22	$+0.93$	$+2.08$
SICKE	$+1.48$	-5.18	-3.44	-0.23

Table 3.28: Test accuracy on the senteval tasks for the models trained with the siamese approach with predicting an input token's pos tag as additional task. The results are presented in relation to the results in table 3.24.

other hand, cosine similarity is indifferent to absolute value changes since only the angle between two vectors affects the similarity. Thus, the model can store more information than the other models.

Number of positive/negative examples All models have been trained in the previously presented evaluations by showing the same number of positive and negative examples. If a model was to be only trained with positive examples, the model could learn to map every input sample onto the same vector representation. To prevent this, negative examples have to be shown. Table 3.29 shows the impact of showing a different number of negative pairs for each positive pair. The numbers represent the average test accuracy of all SentEval tasks for each model, depending on the number of negative pairs shown during training and the used distance function. The results show that the number of shown negative examples does not significantly impact the transfer tasks' performance. This further shows that training the model with weak labels can lead to good results.

When thinking of the training as moving elements in a vector space, showing positive pairs means moving two elements closer together while showing negative pairs moving them away from each other. Figure 3.2 shows an exemplary vector space with two possible training patterns. In the left part, the blue input sample is paired with several other samples, whereas all pairs are being labeled as negative. The blue sample has to be moved to the left to increase its distance to the other samples. In the right half, the blue sample is paired with one other sample, whereas that pair is labeled as positive. Here, the blue sample is also moved to the left. This shows that even with no or hardly any positive pairs, the model can learn useful representations. In word embeddings, a negative pair could mean choosing two random words from two random sentences.

In a fictitious experiment, there are four different words: *the*, *a*, *football* and

Negative examples for each positive example	cosine	euclidian	Jaccard	weighted Jaccard
0	30.92	31.21	29.43	34.48
0.1	67.99	67.13	61.08	66.72
0.5	70.55	68.84	62.13	69.23
1	67.91	68.91	68.75	71.55
2	71.7	69.08	65.33	69.23
10	69.73	68.48	60.13	69.03
inf	58.58	61.13	55.09	67.99

Table 3.29: Average test accuracy on the senteval tasks for models trained with N negative pairs for each positive pair. Choosing infinite negative pairs for each positive pair means is equal to only chosing negative pairs.

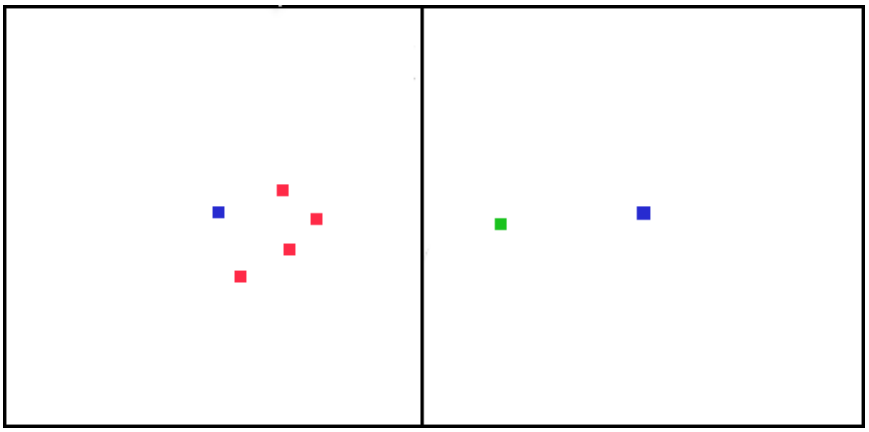


Figure 3.2: Test accuracy on the transfer learning task for the four pretrained models and one not-pretrained model.

player. The first two words appear with a very high frequency in that corpus, while the latter two appear with low frequency. The vector representations contain three features. The used similarity function is Jaccard index. The two words *the* and *a* are chosen as negative pairs with the highest probability. As a result, the model's optimal solution is to assign one of the three features for each of the two words. The other two words will then both be assigned to the third feature. This means that even if the words *football* and *player* are not explicitly labeled as a positive pair, the fact that they are chosen as a negative pair with lower frequency compared to all other possible pairs in the fictitious corpus makes them an implicit positive pair.

Conclusion

At the end of section 3.2, four statements regarding pre-training neural networks for visual data have been made. These four statements can be transferred onto

training word embeddings:

- Using an autoencoder when training word embeddings is expensive because the decoder often needs to map a small vector representation onto a large dictionary, which at the same time is sparse. Training with the siamese approach can remove that expensive decoding step, resulting in faster training because fewer weights have to be trained, showing the advantage of a slim model (see research question 3 in section 1.2). This was shown in section 3.1.4, where the word lists based on the Word2Vec model and the siamese models contain roughly the same words, although the Word2Vec model was trained four times longer.

- The basic Word2Vec is trained with an encoder-decoder architecture, of which the output of the encoder is used as token representation, although that layer has not explicitly been trained to be used as such. On the other hand, if an encoder's output is explicitly trained to be paired with other outputs, the created list of similar pairs contains less seemingly unfit entries.

- It was shown that the models trained with cosine similarity train faster, especially compared to euclidian distance (see section 3.1.4), which shows the impact of hyper-parameter decisions. The reason is that cosine distance (as weighted Jaccard similarity) does not rely on absolute values. Instead, only the angle of vectors is important. In the case of euclidian distance, the model has to assign extreme values to all features to get a similarity value of 0.

- Jaccard index assigns features such that a feature value of 1 represents a feature being present. At the same time, a feature value of 0 represents a feature being absent. To receive a high similarity score, the feature values have to be as close as possible to 1, resulting in the features being binary. Weighted Jaccard similarity, on the other hand, enables the model to assign any value between 0 and 1 to a feature. Having binary features is an advantage when wanting to store data as compact as possible. If this is not an issue, the binary features restrict the model's expressiveness and consequently its performance on transfer learning tasks (see section 3.1.4).

- The previous two points refer to research question 1 stated in section 1.2 and show that the choice of the similarity function is a critical choice to make when training a siamese model.

- Using more data when pre-training a model does not necessarily improve the model's result on a transfer learning task (see section 3.1.4). In fact, it is more important that the data used for pre-training contains features that can be used on the transfer learning tasks. A neural network learns about patterns in relation to the frequency with which patterns appear. If a feature that is critical for a transfer learning task appears infrequently in the dataset used

for pre-training, adding more data with new features can result in a model not learning that critical feature.

- In order to enrich that vector representation for word embeddings, a model has been trained that needed to predict a token's POS tag given that token's vector representation. The results in table 3.28 show that these additional training tasks increase the model's performance on transfer tasks.
- Even if there are hardly any true positive pairs compared to the number of negative pairs, the model can still learn to create useful vector representations.

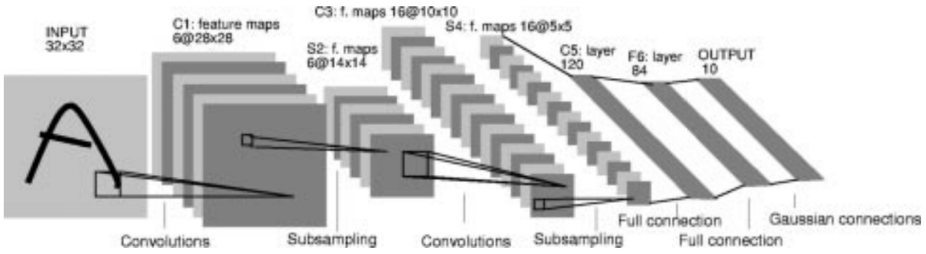


Figure 3.3: Convolutional kernels for feature extraction on visual data. Image taken from [65].

3.2 Visual data

Next to textual data, visual data is the other most available type of data. Since cameras are ubiquitously available, it is very easy to create images of everyday situations. Artificial intelligence models can be used for these photographs for various purposes, such as classifying what kind of object can be seen on an image [46, 60] or classifying and locating objects on images [105, 106]. Other non-photographic images may be scans of handwritten documents that are to be transformed into full digital documents. Thus a model has to recognize the layout of a scan and the text on it [84]. In other contexts, medical devices can create images of specific parts of the human body, which then should be analyzed whether diseases can be found [57, 68, 88, 17].

When working with textual data, texts contain sequences of words, whereas the combination of words creates meaning. Images are similar in the sense that they consist of areas of single pixels, the combination of which creates visual objects. When neural networks process that type of data, convolutional neural networks (CNNs) are used for this [64, 65]. A convolutional layer consists of several so-called kernels. A kernel is a matrix with height and width corresponding to a number of pixels that are processed (see figure 3.3). These kernels react to different visual features represented by pixels in that area: The weights of some kernel may lead to a high activation if the pixels form a horizontal stripe, another kernel may react to vertical stripes while another kernel may react to edges. The next convolutional layer's kernels then can process the created features of found lines and edges, combining those and creating more abstract visual features [138]. When working with faces, the first layer may find lines and edges, the second layer square, circles, and rectangles, the third eyes, noses, and ears, and the fourth layer a whole face.

The difficulty with textual data is that a word can be used in several contexts and have slightly different meanings each time. On the other hand, a word's presence is a piece of binary information: Either a word is used or not used. Visual data, on the other hand, is way more complex. When looking at different images of the very same object, it may look different each time: The perspective, lighting, background, and other factors can differ each time. Also, if the object is not unique

but belongs to a class of objects, like if the object is a car or a dog, that object may look very different on each image but still be of the same class. Thus, to detect a visual object, its abstract visual features have to be learned. For example, a dog is not defined by particular pixels being present but by seeing four legs, ears, a snout, and a tail. To have a model learn to identify a dog on an image robustly, it needs to get training images with a variety of the mentioned features.

Although there is a variety of visual data available, as is the case with text, not all of it is labeled. Additionally, if a model has to learn a particular task, there may be only very few available images suited for training. In the context of neural networks, image classifying tasks have been among the first for which pre-training has been done. This section investigates different pre-training approaches for models dealing with visual data and introduces a siamese pre-training approach for visual data.

3.2.1 Pre-training approaches

Autoencoders

Autoencoders have been created that receive an image as input, reduce its dimensionality, and reconstruct the image (see section 2.2.2). That way, the model has to learn what visual features exist in order to be able to fully reconstruct an image. This is approach can be seen as compression: An image consists of a fixed number of pixels. For several images, there exist specific patterns that are repeated. For example, when looking at the MNIST data[9], which contains small images of handwritten digits, all these digits are created by similar patterns being stripes. Many digits share the same type of stripe. Thus, the model can reduce the input image's dimensionality by transforming the pattern being represented by individual features to being represented by features in the model's layers. When reconstructing the image, the autoencoder's decoding part can *paint* that stripe because it knows which part of the image is represented by that single feature.

As an extension of plain autoencoders, denoising autoencoders have been introduced [129]. Here, noise is added to the input image before being given into the encoder. The decoder has to reconstruct the input image but without the noise. This means that the model cannot merely compress the input image but has to understand what the original image looks like. A practical use case for this is the denoising of images ([41], among others). Here, a model is trained with a set of *clean* images. During training, noise is added to the images, and the model has to reconstruct the clean image. When used in production, the model can remove noise from images because it has learned which of the image's patterns are wanted and which have to be removed.

Depending on the size of an image, training an autoencoder can be costly computation-wise. The usage of a convolutional kernel is quite cheap. For example, using a 32 kernels of size 3×3 on a black-white image requires $32\times3\times3 = 288$

[9] `http://yann.lecun.com/exdb/mnist/`

operations. Modern hardware is optimized to simultaneously apply these operations to the whole image, so executing a single convolutional layer is relatively cheap. On the other hand, when calculating the reconstruction loss, the difference between each input and output pixel has to be calculated. For an image of 512×512 pixels, this results in 262144 operations. Comparing these numbers shows the cost of training a model containing a decoder.

Transfer learning

A different way of pre-training large models consisting of convolutional layers is transfer learning (see section 2.1.2). Here, a model is trained on a dataset with many classes and has to classify the images. Given a huge variety of classes, the model has to learn about many different visual features. When transferring the model to another task, its last layer - being the classification layer - is removed and exchanged by a new classification layer used to classify other classes. When training the model with the new set of images, it may find visual patterns in the new training images similar to those seen in the previous training phase. Other patterns that have been seen before may not occur in the new training data. However, given the huge variety of classes, it is likely that a new classifying layer can make use of the found features by the pretrained model. Exemplary model among others for this use case are *VGG16* [122] and *ResNet* [45].

Using autoencoders for pre-training has the disadvantage that the model consists of an encoder-decoder structure. As is the case with decoding of words (see section 3.1) and sentences (see section 3.3), the decoding part of a model, on the one hand, is expensive. On the other hand, weights are trained, which are not used after pre-training. Pretrained models may be suitable if the transfer learning use case contains similar input data compared to the data the model has been trained with. If the number of shared features of the two datasets is too large, such a model cannot be used. If a custom model is to be pretrained like the previously mentioned *VGG16* model, this can only be done if there is class information.

Siamese approach

In this section, Siamese pre-training is presented to circumvent the mentioned disadvantages of the existing pre-training methods regarding models handling visual data. Since only an encoder is needed and no decoder, roughly half of the weights are used compared to an encoder-decoder structure like an autoencoder. As is the case when dealing with textual data, images have to be used in positive and negative pairs. Pairing can be done using meta-information regarding the training dataset. If there is class information, a positive pair can be two images of the same class. If there is no class information, but the images are stored in a specific order (similar to sentences in a document), coherent images can be used as a positive pair (see section 4.3.4). If there is no suited meta-information, and every image has been viewed as unique, random subparts of the image can be used as positive pairs,

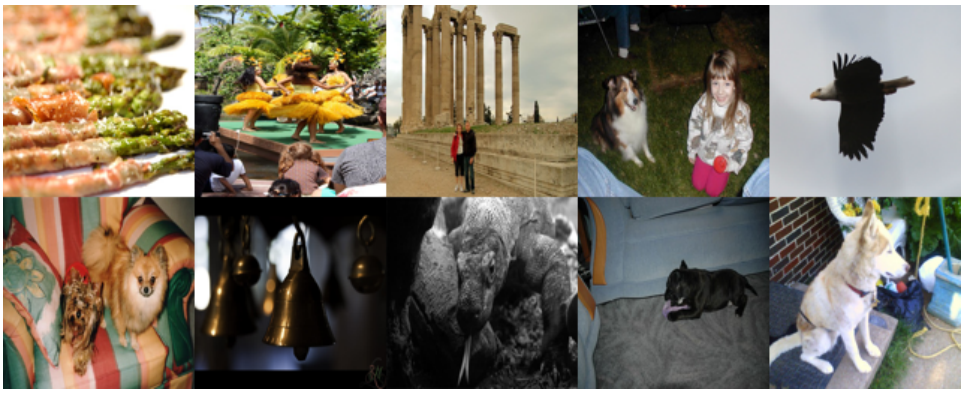


Figure 3.4: Ten exemplary images from the imagenet dataset.

while random subparts from different images are used as negative pairs.

3.2.2 Model evaluation

The presented training approaches have been evaluated. For this, the Imagenet dataset[10] was used. That dataset contains 1000 classes with 1.28 million images. For the following evaluation, 200 random classes have been used with roughly $130,000$ images, which have been split into training and test images using a random subset of 80% of each class for training and the remaining 20% for testing. The 200 classes have been split into two sets, of which the first set has been used for pre-training and the second set for classification. The images are of different sizes and have been reshaped to 256×256 pixels for training. Ten exemplary images from that dataset can be found in figure 3.4. All models have been trained using the same encoder structure (see figure 3.6) to ensure that the training approach leads to different results. The encoder has been created by using skip-connections [46] (see figure 3.5).

Pretraining

When training autoencoders on image data, the training process can easily be visualized. The model is trained to recreate input images while passing them through a bottleneck. Over the cause of training, the difference between the input images and the reconstructed images gets smaller and smaller. Exemplary reconstructed images showing the model's training progress can be found in figure 3.7. It becomes evident that in the beginning, the model is only reconstructing very broad features like an average color of an area and roughly what shapes in an image are present. For example, in figure 3.7a the leftmost image contains a dog. In the reconstruction, shapes of the face are recognizable, but all reconstructions lack color information. Over time, the reconstructed images gain more and more details. Still,

[10] `http://image-net.org/`, [110]

[h!]

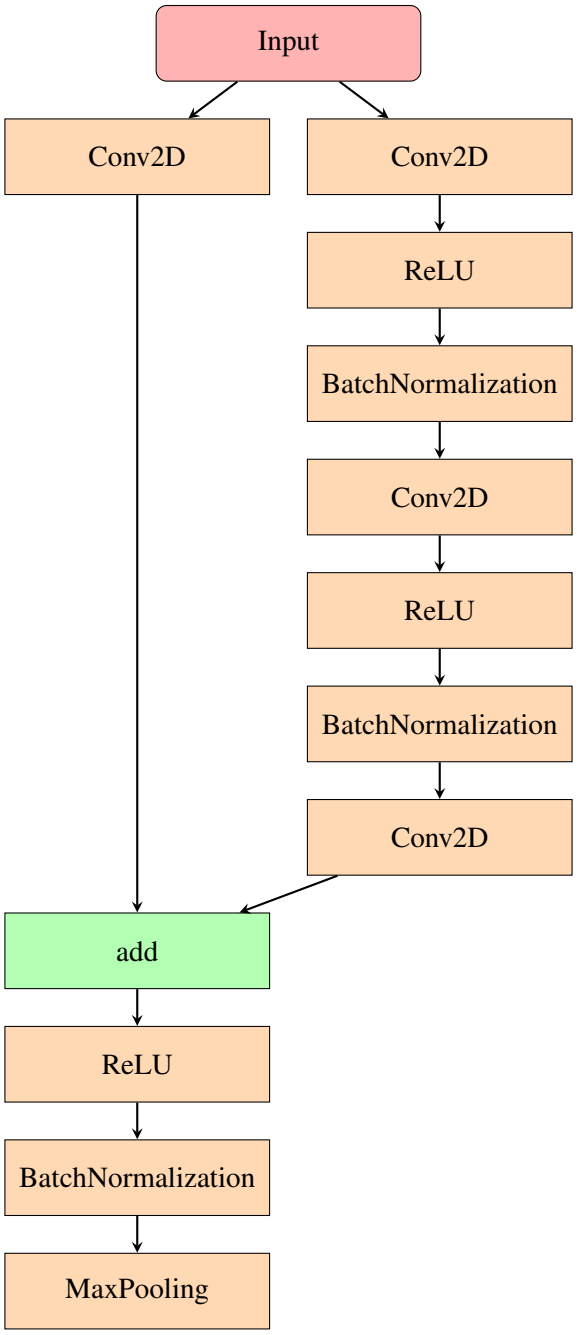


Figure 3.5: A skip-connection block [46].

[h!]

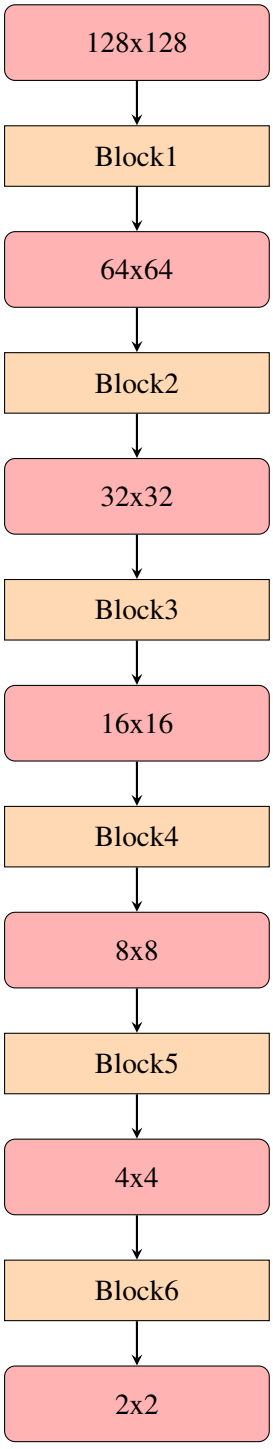
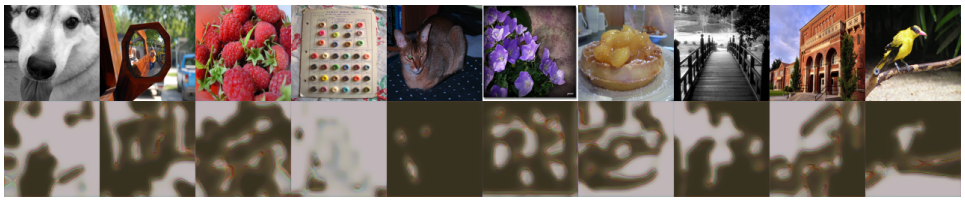
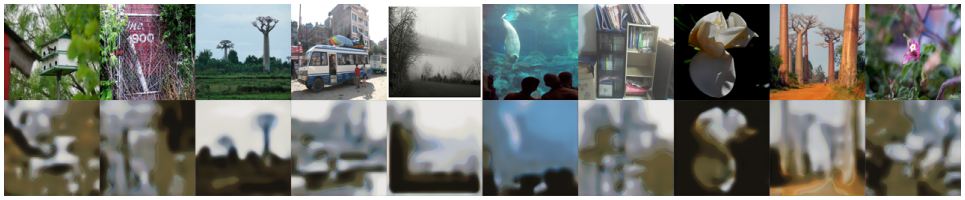


Figure 3.6: The used encoder structure using blocks with skip-connection blocks.

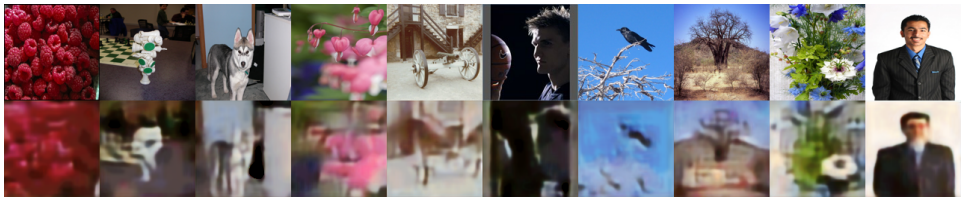


(a) Progress shortly after one epoch.



(b) Progress shortly after two epochs.

(c) Progress shortly after three epochs.



(d) Progress shortly after ten epochs.

Figure 3.7: Autoencoder training progress by visualizing exemplary reconstruction of input images.

the images are never reconstructed perfectly. This is because the model is forced to pass the image's information through a bottleneck. The smaller that bottleneck is, the less detailed the reconstructed images become because most of the details have to be discarded. Instead, only an average representation of the images can be learned. On the other hand, the bigger the bottleneck is, the more details can be kept. An extreme example would be a bottleneck, the same size as the number of pixels. In that case, no information would be discarded.

Contrary to autoencoders, the siamese models' training progress is hard to visualize. What instead can be done is visualizing the learned features after the models have been trained. A second model is trained to do that: The siamese model only contains an encoder part. The second model gets that pretrained encoder and adds a new decoder that reconstructs the images. The weights of the encoder are fixed during training. That way, the decoder is forced to reconstruct the input images using only the siamese model's vector representation. Exemplary reconstructed images can be found in figure 3.9. The reconstructed images of the basic siamese model (see figure 3.9a), which had to pair images during training by their class label, are more blurry compared to the reconstructed images of the autoencoder. This shows that the different training target leads to different features being learned: The model is trained to pair vector representation of images from the same class. Thus, the model does not need to contain as many features of the input image in its vector representation. If one of the classes has a very distinguishable visual feature, the vector representation only needs to contain that feature.

The other siamese model did not receive the full input image but received two subparts of the same image as positive pair. Figure 3.8 shows an exemplary positive pair. The model receives images of the size 32×32 pixels and halves height and weight till it reaches 2×2. Thus the model halves the images' sizes four times. If the model receives an image of the size 128×128, the images are reduced to 8×8. If that model is to be used as a classifier, additional blocks (see figure 3.5) would need to be added and trained. Thus, the model contains a smaller encoder than the autoencoder or the siamese model working with full images, but when transferring it to another task, the encoder has to be enhanced to make up for the missing blocks. The reconstructions of that siamese model can be found in figure 3.9b. The reconstructions are more detailed compared to the reconstructions of the other siamese model. This has to do with two facts: On the other hand, the encoder shrinks the input images fewer times, which means the bottleneck is bigger. On the other hand, the encoder had to learn other features. The siamese model trained on classes only had to learn distinguishable features from the whole class, while the siamese model trained on image subparts had to learn visual features from the images themselves. If a face is shown on an image, the two subparts may cover that face, so features representing faces must be learned.

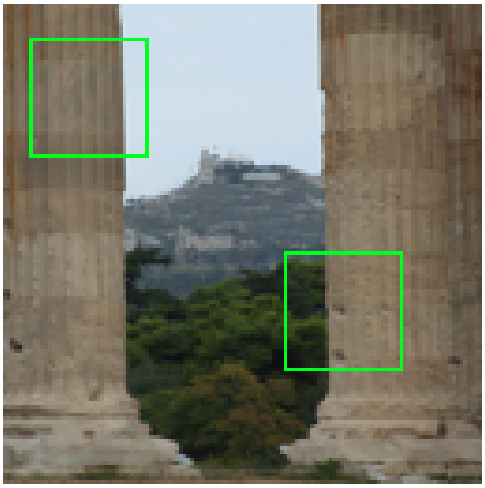
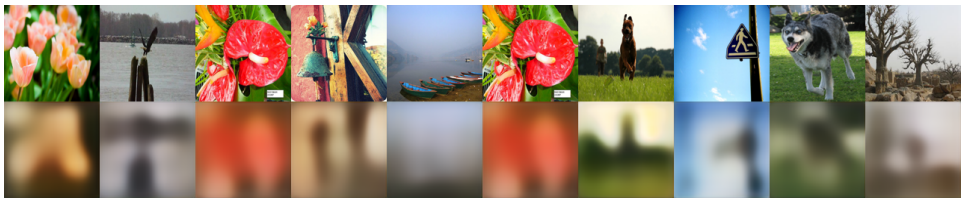
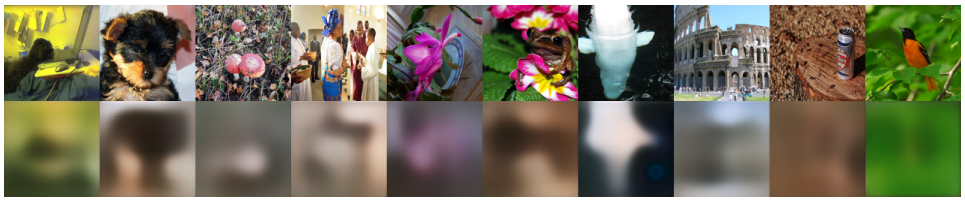


Figure 3.8: Two random subparts from the same image which are used as positive pairs for siamese training.



(a) Reconstructed images of an siamese encoder trained on class information.



(b) Reconstructed images of an siamese encoder that received subparts of an image during training.

Figure 3.9: Image reconstructions of input images using the vector representations created by two different siamese models.

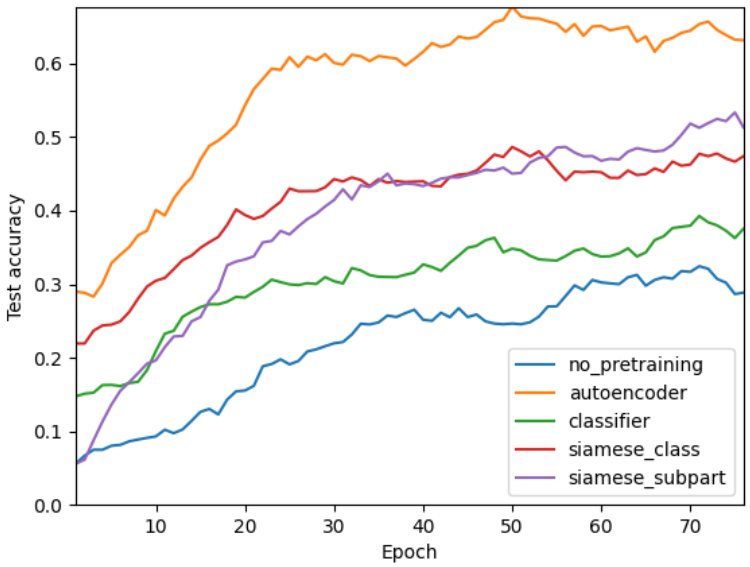


Figure 3.10: Test accuracy on the transfer learning task for the four pretrained models and one not-pretrained model.

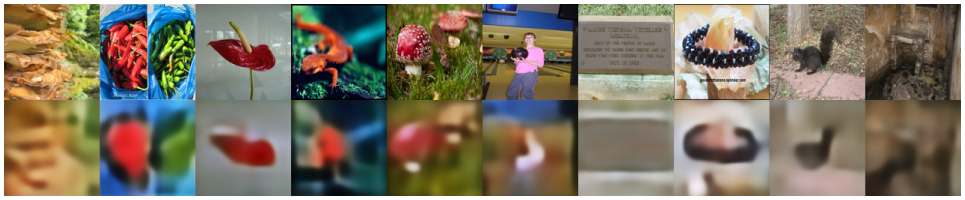
Transfer learning

This section presents the transfer learning task results with the pretrained models and a non-pretrained model. Different classes from the Imagenet data were used for the classifying task compared to pre-training. The results can be found in figure 3.10, which shows that the models perform differently on the transfer learning task.

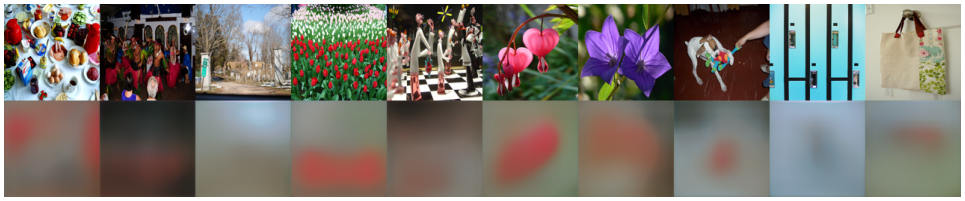
The worst performing model is the model that has not been pretrained. This is expected because its weights were initialized randomly, and the model does not know about any visual features. Over time the test accuracy is rising, but the progress is much slower compared to other models.

The second worst performing model is the model that has been pretrained by classifying 100 classes and for which an untrained one replaced the last layer. All other layers are used with their already trained weights, thus the layers can recognize visual features. The still bad performance compared to the other models may result from the layers learning features that are very focused on the classes used in the pre-training phase. To further prove this point, the encoded vector representation of input images has been used to reconstruct that very input images (see figure 3.11). The reconstructed images show that the models which have been explicitly trained to reconstruct the images, i.e., the autoencoder, can recreate the images with more details than the models trained to classify the input images. The reconstructed images' quality does not differ significantly when using the same input images used during training or images that have not been used during training. For

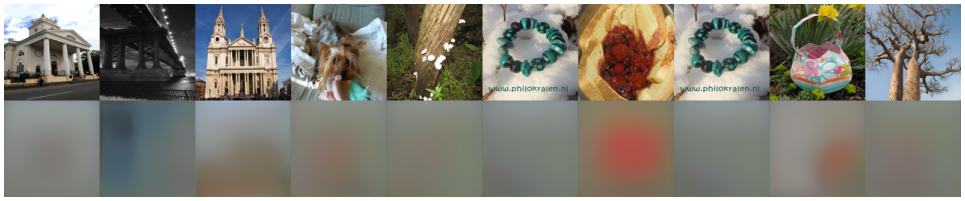
(a) Reconstructed images of an autoencoder using input images from the training dataset.



(b) Reconstructed images of an autoencoder using input images from the transfer dataset.



(c) Reconstructed images of an decoder using input images from the training dataset which have been encoded by a model trained to classify images.



(d) Reconstructed images of an decoder using input images from the transfer dataset which have been encoded by a model trained to classify images.

Figure 3.11: Reconstructed images using decoders after encoding input images. The encoders are either explicitly trained such that the decoder can reconstruct the images (i.e. autoencoder) or such that a classifier can correctly classify the input images.

the classifying model, the reconstructed images contain significantly fewer details compared to the autoencoder's images. The most noticeable feature of the reconstructed images is that color areas very roughly copy the structure of the input image. This shows that the encoder is looking for specific patterns. For example, given a flower as an input image, the encoder covers the information that a red shape is surrounded by a green shape (see figure 3.11c, 6th image). When using the trained encoder on images that have not been used during training, the reconstruction show even fewer features, showing that there are hardly any features found that the classifier can use.

The classifier initialized with the siamese models' weights performs significantly better than the model initialized with the weights from the classifying model. The fact that a decoder can create more features from an image's vector representation created by a siamese model (see figure 3.9) compared to an encoder trained to classify images (see figures 3.11c and 3.11d) leads to better performance on a transfer learning task since more features can be used on the new dataset. The model initialized with weights from the encoder of an autoencoder is performing the best, while its reconstructions also do compare the most features compared to the other models (see figures 3.11a and 3.11b).

Learned features

To further show that the autoencoder keeps more features than the siamese model trained on whole images, an experiment with a toy-dataset was executed. The dataset contains images with a size of 64×64. Each image contains one stripe, while each stripe has two properties: The stripe is either horizontal or vertical and has one of three colors, red, green, or blue. The siamese model is trained with positive and negative pairs. Two images are a positive pair when their stripes have the same orientation, while the color information is ignored. After training the siamese model, a decoder is trained to reconstruct the input images using the siamese model's vector representations of the input images.

Figure 3.10 show the results of the reconstruction of the two models. The middle row contains the reconstruction of the autoencoder. The autoencoder can reconstruct the images using both properties of the stripe, namely the orientation and the color. The bottom row contains the reconstruction of a decoder that had to use the vector representation of a siamese model. The siamese model was trained to pair images by the orientation of the stripes. The decoder was able to reconstruct the orientation but could not reconstruct the color. This shows that the images' vector representation does not contain any color information.

This experiment with the toy-dataset explains why the encoder in the transfer learning task with the Imagenet data performs better than the siamese models trained on class information: When reconstructing full images, the autoencoder has to keep as much information as possible. The siamese model can discard much information. If it was only trained on two classes of images that contain either cars or people, the model only has to look out for wheels or faces. Other features are

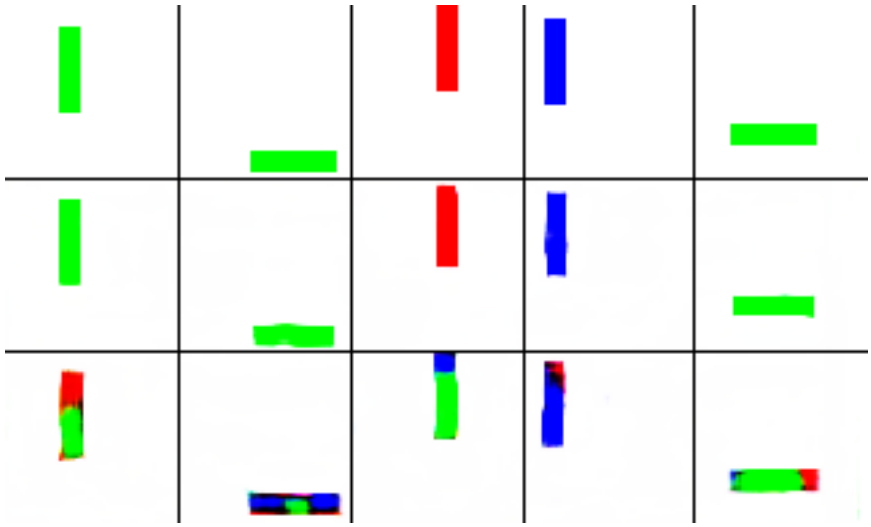


Figure 3.12: Reconstruction of the toy-dataset. The upper row contains the original images, the middle row the construction of the autoencoder, the bottom row the reconstruction of a decor using a siamese model's vector representation of the input images.

not learned because it is not necessary to learn those features.

Conclusion

When deciding how to pre-train a model when dealing with visual data, the following conclusions can be drawn:

- The decoder part of an autoencoder may be expensive but given small images, that cost can be neglected. The bigger the images are, the less inclined one should be to use an autoencoder.
- If the ratio of images per class compared to the number of classes is high, e.g., if there are only two classes, a siamese model should not be trained using that class information because lots of information can be lost. Instead, a siamese model can be trained using subparts of the available images.
- If the ratio of images per class compared to the number of classes is low, meaning there are lots of classes, a siamese model using that class information can be used since the model has to consider many visual features from the input images.
- A model trained with the siamese approach has to create vector representations using the information of positive and negative pairs. If there is one critical feature that is sufficient to distinguish those positive and negative

pairs, other features will not be learned (see figure 3.10). This refers to research question 2 from section 1.2, showing that the learned encodings may not have enough features for transfer learning tasks.

3.3 Sentence embeddings

The models shown in the previous section 3.1 have been used to calculate similarities between words. That kind of data was described as one-dimensional data. This section covers two-dimensional data in the form of sentences. A model has to find common patterns in the given training data when doing unsupervised pre-training. In the context of training word embeddings, these patterns are words that appear in a similar context because they carry a similar meaning. A similar exemplary meaning is shared by the two words *red* and *blue*, which both are colors and thus are likely to appear in similar, if not the same, contexts. Sentences contain sequences of words. These words gain additional meaning when being combined. For example, the word *merry* describes something positive, the word *christmas* describes a Christian holiday. The combination of both, *merry christmas*, not only describes a happy Christian holiday but has additional cultural context as being a phrase being used at a specific time of the year.

At the same time, due to being made of several words, sentences can have several statements. The sentence *I am hungry* can be spoken while stressing each of the three words. If the first one is stressed, the focus is put on the person being hungry. If the second word is stressed, the focus is on the verb and its tense. If the third one is stressed, the focus is on the actual feeling. For the given example, it is difficult to say what other sentences this sentence is related to since there are many dimensions of similarity. The sentence *I am not hungry* may be an opposite statement, but it contains the same features of person, time, and feeling. The sentence *I was hungry* may also be considered opposite because the mentioned feeling is not being felt right now but has been overcome, i.e., the sentences *I was hungry* and *I am not hungry* describe the same state with different words, which is that a person was hungry in the past but is not hungry anymore.

When thinking about the similarity of two words, the main factor probably is the similarity of the contexts the two words in question appear in. When thinking about the similarity of two sentences, the answer is way more subjective since a sentence contains way more information. One person may be focussing on the grammatical structure of sentences, like the tense of words. A second person may be interested in actual words used, while a third person may only be interested in the topic. Thus, it is hardly possible to give one good list of similar sentences. Moreover, a model being trained to create vector representations of sentences has to gather various aspects in a sentence so that, given a use case, the desired aspects can be used while other aspects can be discarded.

This section describes how to train models with a siamese approach to get a vector representation for sentences. Similar to the previous section 3.1 with word embeddings, these sentence embeddings are first analyzed on a qualitative by looking at lists of similar sentences before then using the pretrained models for transfer tasks.

3.3.1 Model types

This section describes several approaches to how to create sentence embeddings, of which the siamese approach will be described in most detail.

Word embeddings

Since sentences are created using words, the baseline for creating sentence embedding shall be created by using the average vector representation of all words in a given sentence. For this, any model that can create a vector representation for single words can be used to create a sentence embedding. For short and simple sentences, it can be assumed that just adding the vector representation of words equals to merging the meanings of the words. On the other hand, the longer a sentence is, the more likely some vectors cancel each other out.

Sentence2Vec

When using word vectors that have not been trained for being used in sentences, the vectors may have unwanted properties. For example, the meaning of a sentence is probably not defined by the vector representation of stop words but by verbs and nouns. When the word vectors are summed, stop words could be over-represented in the sentence's vector representation. To account for this, the approach *Sentence2Vec* [63] trains word vectors such that for a given sentence, the vector representation of a subset of words is summed. Next, a classifier has to predict the next subset's succeeding word. This approach is very similar to the CBOW approach used by Word2Vec, where the word in the middle of a context has to be predicted.

SkipThoughts

While the two previously mentioned approaches are focussing on words in only one sentence and also do not consider the order of such words, *SkipThoughts* [59] uses a recurrent layer that receives a sequence of word vectors. The last state of the recurrent layer is used to initialize a second recurrent layer, which has to generate a sequence of words. The sentence that is to be generated is the sentence's succeeding sentence used as an input sentence. That way, the first recurrent layer has to collect information from the input sentence such that the second recurrent layer can create a likely succeeding sentence. The result is that the vector representations of sentences are similar if their succeeding sentences are similar. The first recurrent layer can be seen as an encoding layer, while the second recurrent layer can be seen as a decoding layer. This creates the same problem of all encoder-decoder approaches, which is that the decoder may only be used during training. When generating sequences of words, the problem becomes even more significant since the decoding step involves predicting the likelihood of words appearing in succeeding sentences, including the likelihood for a whole dictionary.

The number of needed FLOPs for training the SkipThought architecture consists of three factors: The recurrent encoder, the current decoder, and the prediction of words from a dictionary. The computational cost depends on the length of the sequences, the size of the recurrent layers, and the size of the dictionary.

QuickThoughts

In contrast to *SkipThoughts*, *QuickThoughts* [71] consists of an encoder part but has no decoder part. Instead, multiple sentences are being encoded and then given into a classifier. The sentences are taken from a corpus, and two sentences are coherent, meaning in the corpus, one sentence is appearing after the other. The rest of the sentences are random sentences. The classifier then has to decide which sentences are coherent. Since the same encoder is used for all sentences, sentences with similar words are likely to share the same vector representation. If among multiple vector representations of sentences, two are similar, the classifier can learn that these two sentences are the coherent pair.

On the other hand, the vector representation of two coherent sentences does not necessarily need to be similar. If one sentence's vector representation's features contain information about the color word *red* being used in a sentence, and if another sentence contains the word *blue*, these two features could be represented way differently. The classifier then has to learn that these two features are related while at the same time being different represented differently in the vector representation. Thus, when comparing these sentence vectors using a distance function like cosine, the vectors may be dissimilar. Simultaneously, this is not a problem for the classifier because it has learned that aspect of the encoder.

The QuickThought architecture requires no recurrent decoding layer as well as no prediction onto a dictionary. As a result, the number of FLOPs needed for training is significantly lower compared to the SkipThought approach.

BERT

In the recent past, the BERT model [32] has been shown to be an architecture that can successfully be transferred onto many different tasks. The architecture consists of several bidirectional recurrent layers that have been extended with a self-attention mechanism called *transformer* [128]. When considering all models investigated in this thesis, the BERT model needs the highest number of FLOPs during training due to the relatively high number of layers and the fact that each layer contains multiple weight matrixes. During training, a sequence of tokens is given into the model, while the model has to predict the same tokens at its output layer. The critical constraint is that some of the input tokens are masked. Thus, the model does not see some of the sequence's tokens but still has to predict them. This section investigates why this approach leads to excellent performance on transfer learning tasks. For all experiments including the BERT model, the framework

HuggingFace[11] [134] is used. Of the different BERT model types that framework offers, only the BERT model from Google is used.

Siamese architecture

The approach of training a model to create vector representations of sentences is very close to the previously mentioned *QuickThoughts* approach. It also contains an encoder while at the same time having no additional classifier. Instead, the encoder's output is used as the input sentence's vector representation and is given into a pre-defined distance function to calculate the distance to another given input sentence. This results in a similar but a little lower number of FLOPs needed during training compared to the QuickThought approach. As described previously, the siamese model has to be trained with positive and negative pairs. If the model would be trained only using positives pairs, meaning all input pairs should receive a distance of 0, the model could converge to a trivial solution by giving the same vector representation to each input sentence.

3.3.2 Training parameters

When training a neural network such that it provides embeddings for sentences, there are several ways to train that model, both regarding architecture and data. This section looks at different aspects and discusses how they influence a sentence's vector representations. The discussion will focus on the siamese training approach, while each discussed aspect can be applied to some of the other model types.

Sentence pairs

As mentioned in section 2.2.5, the siamese training approach includes the usage of data pairs. A pair consists of two input samples from the dataset and is either *positive* or *negative*. When training word embeddings, a positive pair consists of two words from the same context, while the context is defined by a window size. This means, if the window size is five, all words that appear together with not more than five words between them are considered a positive pair.

When dealing with sentences, sentences contain sequences of words and carry more information than a single word. Many options can be considered as a positive pair or context, respectively. The narrowest definition of a context, which is typically used, defines two directly coherent sentences as a positive pair. This seems quite intuitive: A sequence of sentences is typically used to describe a bigger picture. Thus, two coherent sentences are most likely referring to the same abstract topic, leading to the two sentences sharing similar words and expressions. If they are shown to the siamese model with the positive label, the model can learn that the words or word groups from these two sentences are related, meaning they will get similar vector representations. On the other hand, since sentences describe a

[11] `https://github.com/huggingface/transformers/`

topic, using a whole document as context can also seem feasible: If there is a news article about a football match, specific words are reoccurring in the whole article. On the other hand, if the document is a book, there may be too many topics such that a model can hardly identify whether a pair is positive or negative.

Negative examples are used to prevent the model from assigning all input samples the same vector representation. Deciding from which context to take other sentences significantly impacts the resulting vector representations. The easiest and typically chosen approach is to use random input samples as negative examples. Using this approach, a sentence from a news article about football might be paired with articles about politics. The model can then learn that these used words or word groups do not belong together and assign different vector representations. A smaller context could be a sentence pair with several sentences in between. This may be suited for documents with varying topics. If the corpus only contains documents dealing with one topic, this context size may not be a good choice since the negative pairs may always contradict the positive pairs. On the other hand, if, for example, the beginning of a document is written in a different style compared to the end of a document, this aspect could be learned when using negative pairs from the same document.

In any case, given that negative examples are chosen randomly, and their scope overlaps the scope of positive pairs, it could happen that a positive pair might also be chosen as a negative pair. This slows down the model's training, but in the long run, this is a minor problem: The scope of positive pairs could cover sentences from the same document, and the scope of negative pairs could cover the whole corpus. A negative pair might consist of sentences from the same document. Simultaneously, it is much more likely that the same pair is chosen as a positive pair, which cancels the previously mentioned negative effect.

Sentence length

When training a neural network, training is done in batches. Each batch contains a pre-defined number of training samples. When training a model using the siamese approach, there is a fixed number of sentences used as sentence one and a fixed number used as sentence two. The network then creates the vector representations for sentences one and sentence two, calculates the pairwise distance, and then changes the weights with respect to whether the pair was positive or negative. The data for each sentence group has to be converted into fixed-length matrixes. When considering token sequences, the matrixes consist of three dimensions: input sample, time step, token index. When a group of sentences is converted into one matrix, the sentences have different lengths, while the matrix's time step dimension has a fixed size. This means if there are sentences that are shorter than other sentences, their reserved space in the input matrix will not be used. The bigger the length differences of sentences are, the more space will be empty. In the worst case, one sentence of a batch could be very long with, for example, 50 tokens, while the rest of the sentences are all very short. In that case, nearly all of the matrix will be

empty.

To circumvent the problem, there are two approaches that both have their drawbacks. The first is to make sure when creating an input matrix that all used sentences have a similar length or even the same length. The problem with this solution is that some sentences may be over-represented: Sentence lengths are not distributed evenly, but depending on the language and corpus, most sentence lengths are distributed around an average length. For languages like German or English, one could think of setting a minimum length of sentences since each sentence should at least contain one subject and a verb, while one-word sentences mostly are imperative statements like *Stop!*. At the same time, sentences, in theory, could be infinitely long. Also, the longer a sentence is, the smaller is the number of sentences with precisely the same number of tokens. Thus, when pairing words by their token length and choosing a sentence with, for example, 75 tokens, there may be no other sentence with exactly that length. At the same time, memory usage grows with each additional time step.

The other solution of dealing with different sentence lengths is to use a random subsequent of a sentence. An exemplary length could be five tokens. This means for each sentence pair, a random coherent subpart of five tokens is chosen. This way, it is very likely that no parts of the input matrix are left empty, ensuring faster training. At the same time, it increases the noise in the training data. For example, there could be a sentence pair like *The sun is shining today* and *It is a very nice day*. If the first five words are chosen from the second sentence, it is basically not possible to decide whether these two sentences are coherent. Training sentences in pairs falls under weak supervision because of their weak labels (see 2.1.6). Using random subsets of sentences makes the labels even weaker.

Recurrent layers

When dealing with sequential data, using recurrent layers is an obvious choice. Recurrent layers have two inputs per time-step, which are the input data (or value from a previous layer) and data from the previous time step. This way, the internal state of a recurrent node is adjusted at each time step of the sequence, no matter its length.[12] When the model's task is to create similar vector representations for similar input sentences, a recurrent layer's internal state can learn features like what words or word groups have been seen. The last internal state of the recurrent layer can then be used as the sentence's vector representation.

Several recurrent layers, optionally combined with bi-directional layers, increase the level of abstraction of the sentence's vector representation. At the same, recurrent layers are resource-heavy, especially when working with long sequences. A different approach than using multiple recurrent layers is extending the model by adding one additional layer after the recurrent layer. That layer receives the

[12]Recurrent layers suffer from the problem of vanishing gradients the longer a sequence is. Complex node structures tackle that problem by using, for example, LSTMs [51], the details of which shall not be discussed here.

recurrent layer's last internal state and maps that state onto a new vector. When only using one recurrent layer, the problem of using the last internal state of the recurrent layer is that the last time-step's input will be passed once through the network.

Using the last internal state of the recurrent layer forces the recurrent layer to memorize all of the sentence's features in its internal state. An alternative approach to using the last internal state is using the average or sum of the internal state with regard to all time steps. This way, the sentence's features have not to be carried over all time steps. Instead, the recurrent layer can output certain features at specific time steps, and the need to carry all features over the whole sequence is lowered. As layer type, LSTMs are more fitted for this approach compared to GRUs [26]: GRUs are a simpler form of LSTMs, which do not have a dedicated output gate. Thus they output their internal state at each time step. LSTMs, on the other hand, have an output gate, which is used to decide which information is presented to other nodes or layers at which time step. If a layer offers its internal state at each time step, creating an average value of that internal state may result in many features being canceled out. When only dedicated values are presented, this effect can be lowered.

If the use case to which the trained model should be applied contains a fixed sentence length, an alternative to using recurrent layers are convolutional layers. Convolutional layers contain so-called kernels that are applied to input data. The kernels are a weight matrix that covers a size that is typically smaller than the input data. If an input sample consists of five time-steps and the kernel length covers three time-steps, the kernel is used three times: For tokens 1 to 3, 2 to 4, and 3 to 5. The result of this one convolutional layer is a new sequence of length three, which again can be given into a new convolutional layer to shorten the sequence. The fact that a kernel is used multiple times on the input data with the same weights ensures fast execution. Also, if a kernel covers a word group, the word group's position in the sequence does not affect the kernel.

Model output

The most significant impact of how a sentence is represented by a vector is caused by deciding how the model should create that single vector. There are two types of creating that vector representing, of which the simplest is to use static word embedding. Each token can be transformed into a vector representation, while all vectors are stored in a two-dimensional matrix with shape sizes consisting of sequence length and feature size. To transform this matrix into a vector, the matrix is reduced at the first dimension, typically using the mean or sum function. Some aspects of a word's features may be canceled out with other words' features using that approach. For example, if one word has the value -0.1 for a feature and a different word has the value $+0.1$ for the same feature, these two values get canceled out, and information gets lost. On the other hand, when several words have the same value for the same feature, this can significantly impact the resulting

sentence vector. While being simple, reducing a matrix to a single vector using the average or sum does not consider the words' order.

To account for the order of words, using recurrent layers is an obvious choice. When using these types of layers, there are two possible ways to create a vector representation of a sentence. One of these two options is to use the last internal state of the (last) recurrent layer. That state contains information on the whole sequence since the whole input sequence has been given into the network at that point in the network. What exact information the network is storing at that state depends on what auxiliary task the network has to solve. The *SkipThought* approach trains the network such that the following layer has to create the sequence of the following sentence from the training corpus. Thus, the previously mentioned last internal state must contain enough information such that the next layers can create that succeeding sentence. The *QuickThought* approach uses the recurrent layer's last internal state of multiple input sentences, which are given into a following layer, which has to classify which input sentences are coherent. Thus, the sentences' vector representations have to contain information that the following classifying layer has enough information to decide which sentences belong together. Given the corpus, very few aspects of an input sentence may be needed to decide which sentences belong together. If there is a corpus of news articles and only one article contains words regarding football, it may enough to only watch out for specific keywords. Thus, these sentences are expected to receive a similar vector representation even if they consist of different sentence structures. The mentioned problem also applies to the siamese training approach, which is an improvement of the *QuickThought* approach by removing the classifying part and instead directly using the recurrent layer's last internal state for calculating vector similarity using a pre-defined similarity function.

The BERT model uses a different approach regarding the last layer. Here, not the last state of the last recurrent layer is used. Instead, the model is predicting a sequence with the same length. Contrary to the *SkipThought* approach, it is not using a sequence-to-sequence approach with one layer's last internal state as a bottleneck. The model is trained such that it is given tokens from a dictionary and has to predict the very same tokens. Additionally, some tokens from the input sequence are replaced by a mask token, but the model has to predict the original token. If no input tokens would be masked, the model could simply pass the input information straight to the output without using recurrent weights. With masked tokens, the model has to learn which exact tokens have been replaced by the mask token and is forced to use that token's context. A different advantage of predicting tokens is that the recurrent layers are not forced to remember the sequence as a whole in their hidden state. If a token from the beginning of a sentence needs to be predicted, many surrounding input tokens may hold enough information to make a good prediction.

Apart from learning to predict masked tokens, the BERT model is also trained on a second task. A second sentence is given into the model. As is the case with the *QuickThought* or siamese approach, that second sentence is either a following

coherent sentence from the training corpus or a random sentence. A so-called class token is added at the beginning of the sequence. The last recurrent layer's output of both sentences at time-step 0 is given into an extra layer dedicated to deciding whether the two shown sentences are a positive or negative training sample.

When predicting a sequence of tokens as the BERT model does, the output can be transformed into a single vector by, for example, taking each feature's average value over the sequence. This approach is the same as when using a simple word embedding model. However, when using word embeddings without a recurrent layer, the vectors at each time-step are independent. When using a recurrent layer, the resulting word embeddings depend on their context. This leads to homonyms getting different word vectors given their context. It can be derived which of the several meanings of a homonym is referred to. Alternatively, the last recurrent layer's output at time-step 0 can be used as a sentence vector since that vector is trained similarly to the *QuickThought* approach.

The token sequence output can be trained in a siamese manner, meaning the decoder part can be removed. The BERT model uses the last recurrent layer to predict tokens from a dictionary. Instead of using such an expensive prediction, the output can be changed such that the last recurrent layer's output at each time-step is compared to the vector representation of the target token. This approach is similar to the siamese word embeddings approach described in section 3.1.2. All vectors of the last recurrent layer are compared to a static vector representation of the input sequence as well as with static vector representations of a random sentence. As is the case with the static siamese word embeddings, the recurrent vectors have to be similar to the same sentence's static vectors while being dissimilar to vectors of the random sentence.

3.3.3 Sentence embedding evaluation

As has been done when evaluating word embeddings (see section 3.1.4), sentence embeddings shall be evaluated qualitatively and quantitatively. The sentence embeddings will be used to find sentences with similar vector representations to see what training parameters result in what kind of similar sentences. After that, the sentence embeddings will be used on a set of transfer-learning tasks.

The models for this evaluation are trained on a corpus of news articles from the year 2016. The dataset has been gathered from the Wortschatz project at Leipzig University.[13] The dataset contains roughly 56,000 news articles with about 2,630,000 different sentences. To reduce the memory size needed, if a chosen sentence from the dataset contains more than 15 tokens, a random subpart of 15 coherent tokens will be used.

Word embeddings The models being evaluated are related to the mentioned model types in section 3.3.1. The base model for sentence embeddings will be created by using the average vector representation of a sentence's words using the siamese

[13] `https://corpora.uni-leipzig.de/`, [40]

model with cosine distance function to make sure all models are trained on the same dataset. The *Sentence2Vec* approach will not be considered in evaluation since that approach is very close to the usage of average word vectors.

SkipThoughts A model with the *SkipThought* approach has been trained with coherent sentences from the mentioned news articles dataset as training pairs. Since sequence generation requires many resources, a smaller dictionary of the most frequent 20,000 words has been used.

QuickThoughts A model with the *QuickThought* approach has been trained with three input sentences, of which two are coherent sentences from the same news article and the other one is a random sentence. The three sentences' vector representations had been concatenated to a single vector, which was then given into a classifier that had to decide whether the second or third shown input sentence is the random sentence.

BERT The mentioned BERT architecture (see section 2.1.4) is trained to predict missing tokens from a sequence. At the same time, the model is trained to predict whether a pair of two sentences is coherent or not. The model's output consists of word embeddings for the input sentence's tokens plus an additional vector, which is given into the coherence classifier during training. When using the BERT model, the token embedding sequence will be used and the additional output.

Siamese The base siamese model has been trained with one recurrent layer with tanh as recurrent activation function and cosine distance function. The sentence pairs have been chosen the same way as for the *QuickThought* approach. Apart from the base model, there have also been other siamese models trained in order to investigate the effects of the training parameters mentioned in section 3.3.2.

Similar sentences

In order to get a list of similar sentences, a long list of sentences is chosen. The Wortschatz project at Leipzig University offers a list of *typical* sentences, whereby *typical* is defined by a sentence having a POS tag structure that is frequently used [86]. The dataset contains roughly 50,000,000 sentences.

An exemplary list of similar sentences can be found in table 3.30. The used base sentence is listed with Rank 0, while all similar sentences are listed with Rank 1 to 10, while the sentence listed with Rank 1 has the highest similarity score. As can be seen from this list, most similar sentences are basically the same since they differ only in a few words. Other experiments with different base sentences or model types have shown that similar sentences consist of mostly other sentences with only one or two words changed. When wanting to compare different models, there is not much to learn from these lists. Thus instead of using all available

Rank	Sentence
0	But criminal investigations are different.
1	But friendlies are different.
2	But ideologues are different.
3	But refineries are different.
4	But kokanee are different.
5	But diaries are different.
6	But reptiles are different.
7	But SUVs are different.
8	But painkillers are different.
9	But nonprofits are different.
10	But many taxis are different.

Table 3.30: Similar sentences for the base sentence *But criminal investigations are different* using a model with word embeddings based of 50,000,000 different sentences.

sentences, 1,000,000 sentences are chosen. The fewer sentences there are, the more significant the variations of similar sentences are. This gives further insight into what effects different model types have on sentence similarity.

After transforming all these sentences into their vector representation, one of these sentences is chosen, and its similarity to all other sentences is calculated. Since lists of similar sentences need more space than lists of similar words, there are fewer examples shown compared to section 3.1.4. The shown examples are chosen such that the most distinguishing features of a model type can be exemplarily presented. The results are shown as tables, whereas the first sentence with rank 0 is the base sentence, and all the other sentences are the sentences that have been paired with the base sentence. Similarity values are not shown since the actual value is less important than the order of sentences. Also, different distance functions have different value ranges, so it is hard to compare the absolute similarity values directly. For example, for a vector pair, to have a similarity value of 0 using cosine distance, the vectors would need to point in exactly opposite directions, which is unlikely to happen. On the other hand, when using Jaccard similarity, it is much more likely that there is a similarity value of 0 or close to 0 since this means that there are hardly any overlapping features.

Word embeddings The first model, which is evaluated, is a model that maps words onto their vector representation. The model has been trained using the siamese approach with cosine as distance function. Using cosine has the advantage that sentences with different lengths can easily be compared. When, for example, creating the sum of a sequence of word vectors, cosine distance does not consider the absolute value of the result but only the vector's direction. Thus, when two sentences are compared to which one is short and one is long, they are similar if

Rank	Sentence
0	She took on the new form in her book.
1	She illustrated her first book in 1990.
2	She chronicles this movement in her new book.
3	Viriginia Cymbal enters her name in a drawing.
4	She maintains a studio in her home.
5	Gabehart reviews her work in her studio.
6	She announced her candidacy in a YouTube video.
7	Her number is in the phone book.
8	She is working on her first book.
9	She left notes in the guest book.
10	She wrote it in the dust on her furniture.

Table 3.31: Similar sentences for the base sentence *She took on the new form in her book* using a model with word embeddings based on 1,000,000 different sentences.

their vector point in similar directions.

Table 3.31 shows a list of similar sentences for a given base sentence. The base sentence contains several words or word groups, which can also be found in the similar sentences. For example, all sentences contain either the word *she* or *her*. Also, all sentences contain *in her* or something similar like *in the* as well as the word *book*. Since not all sentences contain the word *book* but all contain either *she* or *her*, this shows that the impact of words from the same word group on the similarity is bigger compared to the impact of words like *took* or *form*, which do not belong together.

To further prove this point, table 3.32 shows a list of similar sentences for a base sentence with only high-frequent words with the word *expensive* as an exception. The other sentences all contain the word group *it's not* or a slight variation, while no sentence contains the word *expensive* or even a variation. It shall also be noted that the sentence vector does not contain information about word order, which again shows the immense impact of high frequent words or word groups. Table 3.33, on the other hand, contains a base sentence with few high-frequent words but words representing a specific topic. The other sentences all contain words of the same topic. The last example further shows the impact when dealing with a sum of word vectors: The word vectors all point in a specific direction, and when using the sum of these vectors, some features may be canceled out while shared features stay.

Table 3.34 contains a list of similar sentences created by a model that has been trained using euclidian distance. Since this sentence embedding also does not take word order into account, it is a coincidence that eight out of ten sentences start with the same word. On the other hand, it shall be noted that the other sentences all contain the word *she*, *her*, and a preposition. At the same time, two sentences seem to not belong to that list. This phenomenon also occurred in other word lists,

Rank	Sentence
0	No, it's not expensive.
1	No, it's not fun.
2	No, it's not Bigfoot.
3	No, it's not Interlachen.
4	No, it's not forever.
5	Obviously, it's not appropriate.
6	So, no, it's not clear.
7	No, it's factual.
8	No, there's not enough.
9	It's not really there either.
10	It's not arbitrary, either.

Table 3.32: Similar sentences for the base sentence *No, it's not expensive* using a model with word embeddings.

Rank	Sentence
0	But criminal investigations are different.
1	These are serious charges.
2	Defendants are no different.
3	These claims are serious.
4	These criminal charges are pending.
5	They are strictly minor criticisms.
6	These discrimination charges are false.
7	But the situations are completely unrelated.
8	These are not isolated incidents.
9	These charges are merely accusations.
10	The charges are serious.

Table 3.33: Similar sentences for the base sentence *But criminal investigations are differente* using a model with word embeddings.

Rank	Sentence
0	She took on the new form in her book.
1	She pivoted her head and took in the clientele.
2	She chronicles this movement in her new book.
3	Turpel-Lafond took the new position in 2006.
4	The wedding took place on Blennerhassett Island in Parkersburg.
5	She spent her childhood on the Front Range.
6	She wrote it in the dust on her furniture.
7	She spent her childhood in the Glentana area.
8	She spent her days on the couch.
9	She did her thesis on the growth of trout.
10	She still keeps the funeral program in her wallet.

Table 3.34: Similar sentences for the base sentence *She took on the new form in her book.* using a model with word embeddings which has been trained using euclidian distance

which are not shown here. A reason for this may be that it is easier to train a model using cosine distance than euclidian distance, as mentioned in section 3.1.4. Also, when using euclidian distance, it is crucial to use the average of word vectors and not the sum. If the sum is used, the sentence vector of a long sentence can have very big values compared to the vector of a short sentence. On the one hand, it is difficult to compare sentences with different lengths. On the other hand, the value range of sentence vectors is not limited to a fixed range.

SkipThoughts Table 3.35 shows a list of similar sentences for the *SkipThought* approach. The list consists of sentences containing parts of the base sentence, although the variation of these sentences seems bigger than the sentences in table 3.31. Training a model using the *SkipThought* approach is more difficult than using the siamese approach since generating sequences is always computationally expensive. When giving a sequence of tokens into a model, this step is rather cheap since, from the input matrix, which maps from a dictionary to a vector representation, only one row representing one token has to be taken. When generating a sequence of tokens, as is the case with the *SkipThought* approach, each dictionary entry receives a probability of being the token at the current position. The bigger a dictionary is, the more expensive that last step is.

In the original article [59], the authors mention having tackled this problem by training the model using a dictionary containing 20,000 entries. Out-of-vocabulary (OOV) words have to be replaced by a single token. The downside now is that critical distinguishing words are now merged into one word: A sentence like *The OOV is very nice.* can have many possible preceding sentences. These sentences from different contexts will get the same vector representation since they have the same succeeding sentence. After training the model with a reduced dictionary, the

Rank	Sentence
0	She took on the new form in her book.
1	Serenity Homecare is a new business in the ArkLaMiss.
2	She set her purse on the kitchen counter.
3	“She lost her pulse in the helicopter,” Mehta said.
4	Needleman probes in the book.
5	Tess Baldonado took the loss in the circle for Holbrook.
6	She found her son in the street after the gunfight.
7	She was last seen in Seine Bight on Sunday.
8	She nodded her head in the affirmative.
9	She could not show her face in the seventh grade.
10	Amanda Kasbohm earned the win in the cage on Saturday.

Table 3.35: Similar sentences for the base sentence *She took on the new form in her book* using a model trained with the *SkipThought* approach.

authors have used a larger pretrained word embedding matrix to use with the model. Thus, when creating sentence vectors after training, the model did not suffer from the OOV problem.

It can be assumed that the model's performance gets better the larger the used dictionary during training is. The basic approach is similar to the *Word2Vec* approach, which also had an encoder-decoder architecture and predicted entities that appeared in the same context. Also, the sentence vectors have not been trained for being used with an explicit distance function. It can be assumed that lists of similar sentences would be better if that would have been the case.

QuickThoughts Tables 3.36 and 3.37 show lists of similar sentences based a model trained with the *QuickThought* approach. The similar sentences all start with the same word as the two base sentences. Of the first list, eight of ten sentences have a verb in past tense as the second word, and the two other sentences have a verb in past tense in the second position. The most similar sentence for both lists even has the same word as the last word. When comparing these two lists with the lists from the *SkipThough* approach, the lists seem a little better since more aspects of the two base sentences can found in the similar sentences.

This may have to do with the fact that training a model using the *QuickThought* is relatively easy. The same encoder part is reused several times, and there is no decoder. These two things result in *quick* training. A downside of this approach is that the last step of the architecture contains a classifier, which receives several vector representations and has to find the coherent sentence pair. The problem is that the sentence representation is not explicitly trained for being used as a similarity measure. For example, when a sentence vector contains 512 features, one feature could be that a color word has been mentioned. At the same time, a different feature could represent the very same thing. The classifier can now learn that

Rank	Sentence
0	She took on the new form in her book.
1	She created 12 pieces for the book.
2	She married Joseph E. Buchner on July 14, 1951 in Ely.
3	She was born Feb. 21, 1951, in New Brockton.
4	She also used to teach the hawaiian hula.
5	She married Kenneth Inman on Nov. 18, 1967 in Durant.
6	She asks no money for her efforts.
7	She was born August 6, 1962 in New Albany.
8	She married John Shearer in 1953 in Pampa.
9	She also offers a new fashion jewelry line.
10	She was born in New Mexico.

Table 3.36: Similar sentences for the base sentence *She took on the new form in her book* using a model trained with the *QuickThought* approach.

Rank	Sentence
0	But criminal investigations are different.
1	But Thursday's scenario was different.
2	But this product is different.
3	But factoids are not facts.
4	But the counties are already appealing.
5	But the prosecutor wants more.
6	But the involvement of faculty is essential.
7	But Marcotullio is no spring chicken.
8	But Pat Skala was completely flabbergasted.
9	But Moravia can't run.
10	But Japan immediately struck back.

Table 3.37: Similar sentences for the base sentence *But criminal investigations are different* using a model trained with the *QuickThought* approach.

Rank	Sentence
0	She took on the new form in her book.
1	She chronicles this movement in her new book.
2	Forcum explained the method to his work.
3	We used in her first book.
4	Saru Jayaraman has used this model in her work.
5	He wrote much of the book in Capanna.
6	That resonant depth is missing from the new book.
7	She was delicate with the form of her music.
8	This attitude is captured throughout her new book.
9	I addressed it as a form of writing.
10	Savarkar composed his book on Hindutva in 1924.

Table 3.38: Similar sentences for the base sentence *She took on the new form in her book* using a pretrained BERT model.

these two features represent the same thing and can classify these two sentences as coherent. However when comparing these two sentences' vector representations, they may differ since the used features two represent color words being used are not the same. When using such a vector for transfer learning, this problem can be neglected since a succeeding layer can learn that these two features represent the same thing.

BERT Tables 3.38 and 3.39 show lists of similar sentences having used a BERT model. The BERT model offers two outputs. One is a sequence of token vectors and is a class vector. In this case, the sequence of token vectors has been averaged. The sentence vectors have been compared using cosine distance, as has been done with the last three model types. What is different from the previous models is that these similar sentences contain not only the same words from the base sentences but also whole word groups. The first sentence contains the word group *in her book*, while the other sentences contain word groups like *in her new book*, *from the new book* and *her new book*. For the second list, the base sentence and all similar sentences end with *are different*.

BERT offers an innovative tokenizer. Instead of working with full words, the BERT tokenizer splits words into single parts. A German word like *hunde* does not exist in the BERT dictionary. Instead, the word is split into *hund* and *##e*. This means that the base form is used together with a suffix. This enables the model to not learn about specific words but also learn about relations between words: When the word *hunde* is trained, the word *hund* is trained as well.

The BERT model uses several bidirectional recurrent layers and returns one vector representation for each token. Contrary to a sequence of word vectors created by the Word2Vec model, the token vectors are dependent on the other tokens. Thus, the token *mean* has a different token vector depending on its context, mean-

Rank	Sentence
0	But criminal investigations are different.
1	The situations are different.
2	Our policies are different.
3	Our experiences are different.
4	The facts are different.
5	Other areas are different.
6	Demands on schools are different.
7	All their stories are different.
8	Your approaches are different.
9	Our beliefs are different.
10	All families are different.

Table 3.39: Similar sentences for the base sentence *But criminal investigations are different* using a pretrained BERT model.

ing the model accounts for homonyms.

Siamese models Tables 3.40 and 3.41 show lists of similar sentences by a model having been trained using the siamese approach and cosine as distance function. The model was trained on random subparts of sentences ranging from token length 5 to 15. Also, the sentences were given into the model after reversing them. This leads to information coming from the first tokens of a sentence being weighted higher than information from the end of a sentence. This effect can be seen in the two lists of similar sentences: All similar sentences begin with the same word as the base sentence. Nine of ten sentences from the first list even share the first two words with the base sentence, while the only exceptional sentence has that word in the third position.

The fact that similar sentences are most similar with the beginning of a base sentence or the end if not used in reverse shows that using a recurrent layer can be a limiting factor. At each time step, a recurrent layer receives information from the previous time step as well as the previous layer. The longer a sequence is, the likelier it is that information from earlier time steps diminishes. Using advanced recurrent layers like GRUs [26] or LSTMs [51] partially solve this problem, meaning using a plain recurrent layer would probably lead to much worse results. Table 3.42 contains a list of similar sentences based on a model that returns a matrix of token vectors, which is then reduced using the maximum function. Contrary to the previous model type, the similar sentences do not all begin with the same word. Instead, the similar aspects are more broad, meaning all sentences start with the word *she*, followed by a word in past tense, and the possessive pronoun *her*.

Rank	Sentence
0	She took on the new form in her book.
1	She took no prisoners in those interviews.
2	She took on the challenge.
3	She took herself off medication.
4	She took fourth in the discus.
5	She took down 5.9 rebounds per outing.
6	She took on the role of head chef.
7	She first took on the responsibility in 2002.
8	She took on the role of CEO in 2006.
9	She took it the wrong way.
10	She took on a new challenge at a difficult time.

Table 3.40: Similar sentences for the base sentence *She took on the new form in her book* using a model trained with the siamese approach and cosine as distance function.

Rank	Sentence
0	But criminal investigations are different.
1	But board members are human.
2	But his views are shifting.
3	But her motivations are also altruistic.
4	But daily users are different.
5	But healthy choices are available.
6	But his ideas are big.
7	But critics say that wasn't enough.
8	But his credentials are dubious.
9	But drones are no joke.
10	But its causes are not necessarily different.

Table 3.41: Similar sentences for the base sentence *But criminal investigations are different* using a model trained with the siamese approach and cosine as distance function.

Rank	Sentence
0	She took on the new form in her book.
1	She raced her close finish with first place.
2	She was satisfied with her early-season times.
3	She's elegant in her slight frame.
4	She had this quiet strength about her.
5	She then developed lymphoedema in her legs.
6	She bemoaned the ongoing passing of her profession.
7	She lives in Denver with her husband and son.
8	She's won several awards for her public service.
9	She lost her stamina and her hair.
10	She served her country in Kuwait.

Table 3.42: Similar sentences for the base sentence *She took on the new form in her book* using a model trained with the siamese approach and cosine as distance function.

Transfer learning

As was mentioned in section 2.1.2, models trained in an unsupervised manner can be used on different tasks after havin been trained initially. The word embedding models introduced in section 3.1.2 have been transfered onto different tasks. The tasks are provided by the *SentEval*[14] framework and are described in section 3.1.4. The sentence embedding models introduced in section 3.3.1 are now also used

Table 3.43 shows the test accuracy on the Senteval tasks for the models trained with the siamese approach using the last recurrent layer's last internal state. Table 3.44 shows the test accuracy for the same tasks, but the models trained with the QuickThought and SkipThought approach and a pretrained BERT model. When looking at the results, it becomes evident that there are significant differences between the BERT model and the other models. The BERT model's performance, on average, is a little better than the performance of the siamese word embeddings (see table 3.24). The models from section 3.1.4 create a sequence of word embeddings while not accounting for context and order of words. On the other hand, the BERT model creates a sequence of word embeddings that are context-aware. Thus it is to be expected that the BERT model performs better than the models with static word embeddings.

The other models perform worse than the BERT models, although they are explicitly trained to create sentence embeddings. The average test accuracy of the siamese models (table 3.43) and the QuickThought model are very similar.

[14] `https://github.com/facebookresearch/SentEval`, [29]

	Siamese			
	cosine	euclidian	Jaccard	weighted Jaccard
MR	57.02	56.68	57.02	55.99
CR	65.46	63.15	65.46	64.29
SUBJ	78.04	77.37	78.04	76.55
MPQA	75.6	74.45	75.6	75.5
SST2	57.17	59.58	57.17	57.83
SST5	28.14	26.88	28.14	25.48
TREC	66.0	73.2	73.4	79.4
MRPC	65.91	66.32	66.0	63.2
SICKE	66.67	67.55	66.67	64.79

Table 3.43: Test accuracy on the senteval tasks for the models trained with the siamese aproach, using the last recurrent layer's last internal state.

	QuickThought	SkipThought	BERT (average)	BERT (class)
MR	58.59	61.68	69.81	67.79
CR	65.54	69.62	77.62	75.92
SUBJ	73.46	70.86	93.26	92.39
MPQA	75.5	76.02	80.59	79.31
SST2	61.29	61.94	74.03	70.68
SST5	28.6	35.84	36.06	35.25
TREC	70.8	67.4	83.8	67.0
MRPC	66.32	67.3	71.71	69.39
SICKE	70.49	71.06	76.76	65.31

Table 3.44: Test accuracy on the senteval tasks for the models trained with the QuickThought and SkipThought aproach as well as a BERT model.

3.3.4 Siamese extension

The BERT model performs significantly better on all transfer learning tasks than the other models. This subsection is presenting results of different extensions of the previously used siamese architecture in order to show what architectural aspects lead to the BERT model's better results. Apart from being trained for a longer period of time as well as with more diverse data, the central hypothesis to prove is that the BERT model's output contains richer features than the vector representations of the siamese models.

Sequence generation

As described in section 3.3.2, the siamese and the QuickThought approach have in common that the created sentence vectors are used to predict whether a sentence pair is a positive or negative pair. For some training pairs, it may be enough for the model to look out for specific keywords to know whether it is a possible positive pair for an unknown other sentence. Thus, those sentences have a vector representation with very little information. When dealing with visual data, the same effect was shown in section 3.2.2: Here, a siamese model was trained to create a vector representation for images. The images were paired by their class representation, making it possible for the model to ignore visual features that did not correlate with the class information. For sentences, to prove that looking out for specific keywords is a relatively easy task, a siamese model was trained with a different sampling of negative examples: Instead of choosing any random sentence from the whole corpus, negative pairs were always from the same document. Positive pairs were directly coherent sentences.

In order to prove that sentence vectors from the SkipThought approach do contain more information compared to models from the siamese and QuickThought approach, the basic siamese approach was extended. The base model uses the last recurrent layer's last internal state with a sigmoid activation function. That output is used to compare the positive and negative pairs using weighted Jaccard index. That specific combination of activation and similarity function is used because a high value of a feature means that that feature is *present*: When comparing two vectors, they are multiplied. The higher the sum of the product, the more similar these two vectors are. When dealing with the SkipThought model, the model's vector representations cannot be analyzed the same way since they have not been trained with the same constraint. To make the vector representations of the basic siamese model and the SkipThought model comparable, a hybrid model has been trained. That hybrid model consists of an encoding part like the siamese model, but at the same time, the sentence's vector representation is used to recreate the following sentence in a decoder part, as is the case with the SkipThought approach.

The two models are used to create vector representations of the example sentences, which already have been used in section 3.3.3. For all sentences' vectors, the average value of all features is shown in figure 3.13. The values are sorted from

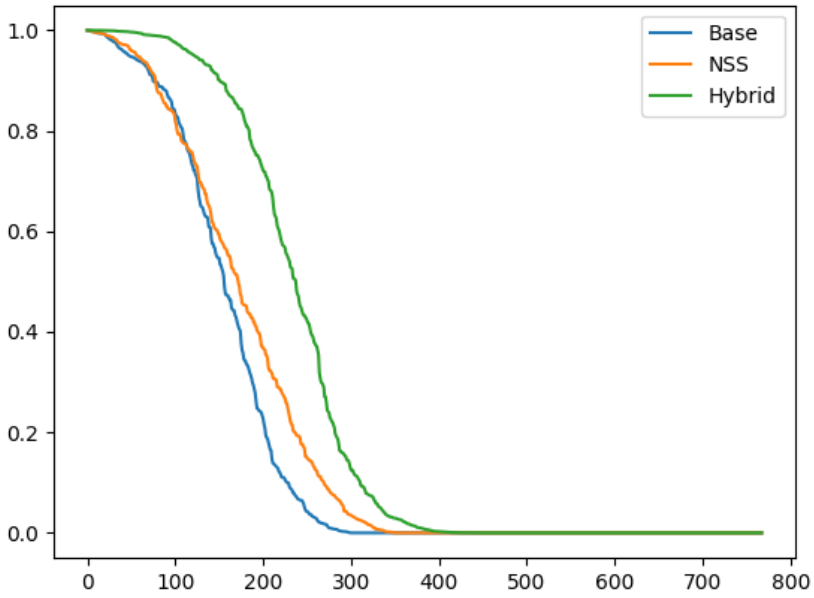


Figure 3.13: Average features activation for a sentence vectors for a basic siamese model, a basic siamese model with negative samples only from the same document (NSD) and a hybrid model of the siamese models and the SkipThought approach.

	Base	NSD	Hybrid
MR	55.99	56.95	60.34
CR	64.29	65.67	69.12
SUBJ	76.55	77.46	75.15
MPQA	75.5	76.17	76.02
SST2	57.83	59.86	62.08
SST5	25.48	30.14	35.21
TREC	79.4	78.2	67.1
MRPC	63.2	68.7	66.9
SICKE	64.79	69.68	70.51

Table 3.45: Test accuracy on the senteval tasks for the basic siamese model, the same model only being trained with negaties pairs from the same document (NSD) as well as the hybrid model of the siamese approach and the SkipThought approach.

highest average to lowest average, meaning that some features have a high value for nearly all sentences while other features are always zero. The image shows that a lot of the available features are never used. It also shows that when using sentences from the same document as negative pairs leads to a few more features being used compared to using random sentences from the whole corpus as negative pairs. This shows that the auxiliary task of distinguishing positive and negative pairs is harder when the negative pairs are taken from the same document. That same model also performs better on nearly all of the SentEval tasks. If that architecture is extended such that the sentences' vector representation has to be used for a second task, recreating another sentence like with the SkipThought approach, the average feature activation is higher, and there are fewer features that are never used. Table 3.45 shows the average test accuracy for the different SentEval tasks. The performance of the extended siamese model is better than the basic model, and the performance is similar to the SkipThought model's performance.

The extension of the basic siamese model and the increased performance leads to two conclusions: First, learning whether two sentences are a positive pair works. When creating list of similar sentences, the lists seem plausible (see section 3.3.3). On the other hand, the created features from a given sentence are less rich than other training approaches. This shows that the auxiliary task is too simple since only a few features are needed. Second, the richness of features in a sentence's vector representation impacts the model's performance on a transfer learning task. If the vector representation only contains features representing that a specific keyword has been found, that vector representation does not contain enough information for a task like getting sentiment information from that same sentence.

Additional training tasks

This extension of the siamese architecture has improved the results on the transfer learning tasks, but on the one hand, the extension has added an expensive decoder part while, on the other hand, still performing worse than the BERT model. When creating a sentence vector representation using the siamese or QuickThought approach, the sentence's vector is used to be compared with another sentence's vector. Here, all words used in a sentence may not be as important as simple keywords. Thus, if a sentence contains a word group that has not been used in training to distinguish positive and negative sentence pairs, that word group is not represented in that sentence's vector representation. On the other hand, the BERT architecture prevents the model from discarding information gained from input tokens since the information of every token is needed in the output layer.

In order to show that this architecture causes richer vector representations, the basic siamese architecture was extended such that another training target was added: Instead of only using the last recurrent layer's last internal state, the model had to create a sequence of vector representations. As is the case with the BERT model, the output sequence's length is the same as the input sequence. Contrary to predicting the input tokens from a dictionary, the output was trained with positive

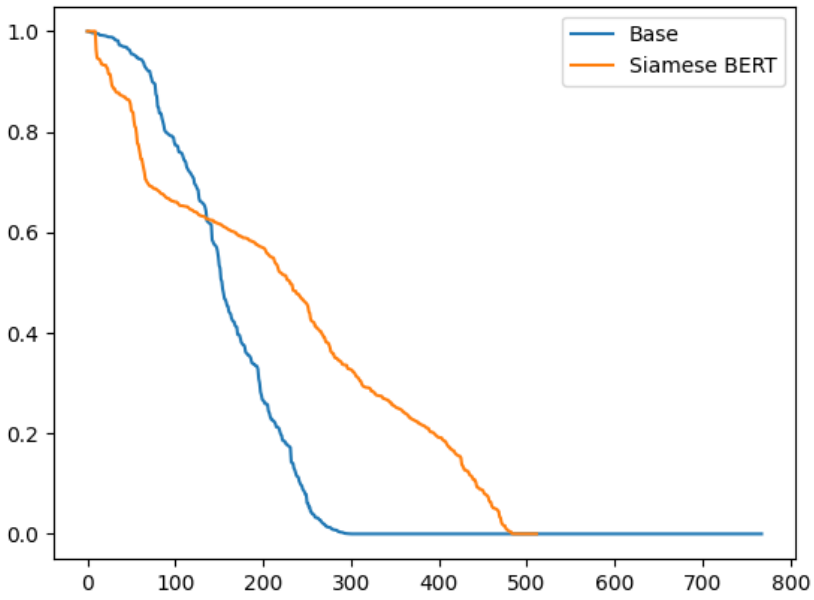


Figure 3.14: Average features activation for a sentence vectors for a basic siamese model and a siamese model trained with the BERT approach.

and negative pairs. That approach is similar to training static word embeddings, as seen in section 3.1.2. Simultaneously, the sequence of output vectors was reduced to a single vector representation, which was used as the sentence's vector representation. As similarity function for the token vectors and the sentence vector, weighted Jaccard index was used. The matrix of token vectors was reduced to a single vector by using each feature's maximum value at each time-step. The positive examples of the token vectors were the same tokens as the input tokens at the same time-step. The negative examples were random tokens from the corpus. The positive example of the sentence vector was the sentence vector of the following sentence in the corpus. The negative example was the sentence vector from a random sentence in the corpus.

Figure 3.14 shows the average feature activation of sentence vectors created by a basic siamese model with the recurrent layer's last internal state used as sentence vector as well as the siamese model with also had to create vector representations for tokens at each time-step. The figure shows that the model trained with an additional training target creates sentence vector representations that contain richer features compared to the model trained on only one task. The original BERT model's feature activation cannot be shown as a comparison because that model's output was not trained to be 0 if a feature is not present and having a value larger than 0 if

	S	S+T	S+T+M	S+T+M+PT	BERT
MR	57.2	63.49	70.9	69.73	69.81
CR	62.94	64.29	75.13	74.44	77.62
SUBJ	72.31	84.38	83.16	87.16	93.26
MPQA	74.9	76.16	77.41	76.25	80.59
SST2	58.87	65.4	64.52	65.86	74.03
SST5	26.56	32.99	33.48	34.38	36.06
TREC	72.8	78.8	77.4	77.1	83.8
MRPC	63.19	66.04	69.26	67.94	71.71
SICKE	67.04	77.37	77.05	75.25	76.76

Table 3.46: Test accuracy on the senteval tasks with models trained with **S**entence pairing, **T**oken pairing, **M**asked input as well as the performance of the original BERT model.

that feature is present. The extended siamese architecture, as well as the BERT architecture, can be compared to a denoising autoencoder (see section 2.2.2, [129]): Here, a model is given input data to which noise has been added. The model has to recreate the original input data and therefore has to learn which input features are correlated and which ones are noise. The positive effect of this has been proven in section 3.2.2: An autoencoder was pre-trained on visual data before using that pretrained weights for initializing a classifier on different data. The autoencoder was performing significantly better than the other pretrained models. Having to recreate the whole input data means that more features have to be learned compared to the use case where a model has to classify input data: Here, it is possible to ignore features that do not correlate with the classification target.

Table 3.46 shows the results of the same model with various additional training tasks. The models have been trained using cosine distance since it was shown that this distance function could cope best with additional training tasks (see section 3.1.4).

S The first model is the basic model that only had to create a sentence vector

T The second model was trained with the additional tasks of creating token embeddings for each time-step. The sequence of token embeddings was summed and used as sentence embedding.

M The third model was trained as the second model but with randomly masked input tokens.

The results show that the more training tasks are used, the better the model's performance is on transfer tasks. This again shows that using more training targets results in a richer vector representation of a sentence. The model trained with three training targets and input masking is even performing only slightly worse than the original BERT model. As an additional training target, another model type was

	S+T+M	S+T+M+PT (cl)	S+T+M+PT (df)	BERT
MR	70.9	69.73	71.45	69.81
CR	75.13	74.44	75.13	77.62
SUBJ	83.16	87.16	88.1	93.26
MPQA	77.41	76.25	78.15	80.59
SST2	64.52	65.86	69.82	74.03
SST5	33.48	34.38	35.93	36.06
TREC	77.4	77.1	79.4	83.8
MRPC	69.26	67.94	69.05	71.71
SICKE	77.05	75.25	76.85	76.76

Table 3.47: Test accuracy on the senteval tasks with models trained with **S**entence pairing, **T**oken pairing, **M**asked input as well as **P**OS-**T**ag prediction. When having to predict a token's POS tag, the vector similarity will either be calculated using a **d**istance **f**unction or a **cl**assifier. When using a classifier, the vector representations of the tokens have been merge using either the **av**erage or **m**inimum/**m**aximum.

trained. That model had to predict a token's POS tag from its vector representation. This, however, creates a new problem: The model's loss function consists partially of the vector similarity of positive and negative pairs and the correct POS tag prediction. The models are trained with a specific distance function to create similar or dissimilar vector representations for positive or negative pairs. Thus, the vectors need to have specific properties to ensure correct similarities. For example, in the case of Jaccard index, a big value represents a feature being present. The loss coming from the prediction of a token's POS tag will also influence a token's vector representation. That additional influence may break the desired property for the chosen vector distance function. To investigate that effect, two model types have been trained: The first one is the model with one bidirectional recurrent layer, the output of which is used as a vector representation. That same vector representation will be given into a classifier to predict the token's POS tag. The second model type is similar to the *QuickThought* approach: The vector similarities will not be calculated using a pre-defined distance function. Instead, two vectors are given into a classifier that has to classify whether the shown vector pair is a positive or negative pair. The same vector representation is also given into another classifier for the POS tag prediction. When creating the vector representation for the whole sentence, the average vector representation from all the sequence's tokens is used. If a recurrent layer was to be used, the same problem presented in sections 3.2 and 3.3.3 would occur: The recurrent layer's last internal state would only contain features needed for calculating the sentence similarity while all other features, for example, features that represent a token's POS tag, would be lost.

The results in table 3.47 show the performance of the models that have been trained with the additional training task of predicting each token's POS tag. The two new models perform slightly better than the model that was not trained with

POS tag prediction for each token. Of the two new models, the one where the vector similarity had been calculated using a dedicated classifier performed on average worse compared to the model, which directly used the token's vector representation. When being trained to create a vector representation that is to be used with a specific distance function, the created vectors need to have specific properties. For example, when using euclidian distance, negative pairs must have maximum values for all features, while vectors being compared with cosine distance can have values close to zero, as long as the vectors point in different directions. When the model is trained with an additional target like predicting a token's POS tag using the token's vector representation, the additional loss may harm the properties needed for the distance function. For example, when using cosine distance, small value changes to predict better the POS tag may significantly impact the similarity of two vectors since small changes in the values can lead to significant changes to the direction the vector points at. To solve this problem, it was argued that giving the vector representations into a classifier may solve that problem because then the vector representation does not have specific properties.

However, the results of table 3.47 show that using additional classifiers does not lead to better results. There are several explanations for this:

- Using one or more additional classifier models slows the learning process since adding more layers to the model lowers the gradients.
- Having a model learn whether two vectors are similar or not is not trivial. If the two vectors to classify are identical, this could be easily solved by calculating the distance between the two. However, a neural network cannot explicitly subtract values and calculate a distance. It has to find patterns in the data. For the executed experiments, the two vectors have been given into the classifier in one case. In another experiment, the euclidian distance of the two vectors was given as additional input. The performance of both variants was the same.
- The classifier has to learn the same information twice. If the vector representation of sentences A and B are given into the classifier, the classifier still has to learn about the pair B A.
- Creating a vector representation that is used for being compared with a specific distance function is an auxiliary task during training. Adding another task during training may break needed properties for the chosen distance function, but this does not harm the performance on transfer learning tasks. However, the performance is harmed when using the vectors for finding similar vectors (see section 4.3).

Conclusion

The presented training extension in the previous section 3.3.4 has shown that a model's performance on the transfer learning task is better the more different aux-

iliary tasks have been used during pre-training. In the end, the model's performance on the SentEval tasks was comparably well compared to the BERT model. However, on more complex tasks, it can be assumed that the BERT model will perform better due to the remaining differences: The BERT model consists of 12 layers of layers, while the presented models only contain one layer. It is also mentioned that it was trained on a corpus of books and Wikipedia data, while the presented models were trained on a set of news articles. Additionally, the BERT model's recurrent layers are extended with a self-attention mechanism [128] instead of using plain bidirectional recurrent layers. The existing siamese architecture has been extended by using more layers and self-attention layers. The self-attention layer usage has given a small performance boost, whereas using more layers has not improved results. Since using more layers results in slower training, it can be assumed that the models have to be trained for a significantly longer period of time to achieve better results: The authors of the BERT model note that their model has been trained for four days and a batch size of 256. The presented multi-target siamese models have been trained for 12 hours using a batch size of 16. Other authors have changed how the BERT model is trained and found that longer training time, longer sequences, and larger batches have improved the model's performance [70]. Thus, it can be assumed that the here presented models also gain from an expansion of hardware resources. However, this section's experiments aimed to show that the auxiliary tasks with which a model is trained, significantly impact that model's output. Also, increased richness in a model's output positively impacts its performance on a transfer learning task.

Given the different experiments executed in this section, several conclusions can be drawn:

- As is the case with images and word embeddings, autoencoders are expensive. When used with sentences, i.e., a sequence of words, they are even more expensive since the processing step onto a vector of the dictionary has to be done several times. The siamese architecture helps to circumvent this expensive step and therefore reduces the training time. However, as was the case with feature-rich images, some patterns of the input data may not be represented in a sequence's feature representation since that feature was not needed for the task of pairing positive and negative pairs.

- To get richer vector representations, the training task can be enhanced by not only pairing sentences but also forcing the model to predict missing tokens from an input sequence: By randomly masking input tokens, the model has to create a rich vector representation for each step of the sequence. Having the model also predict the POS tag for each token of the input sequence further improves transfer learning tasks' performance.

- The previous two points answer research question 1 from section 1.2: The chosen method, how to create positive and negative input pairs, affects the learned features (see also section **??**), which is a hyper-parameter decision.

- If the model's output is to be used directly for calculating vector similarity, the model should be explicitly trained for that task. Using the vector representations for an additional task will harm the task of calculating similarity (see research question 2 from section 1.2).
- One of the experiments in section 3.2 has shown that an input sample's vector representation only contains information that is needed for successfully discriminating positive and negative pairs. In order to enrich that vector representation for word embeddings, a model has been trained that had to predict a token's POS tag given that token's vector representation. The results in table 3.28 show that these additional training tasks improve the model's performance on transfer tasks.

Chapter 4

Case studies

The previous chapters have discussed the concept of pre-training neural networks (see section 2) and presented challenges of pre-training neural networks using the siamese training approach (see section 3). The impacts of different selection of training data or distance functions were shown along with the effect of different training targets. The following chapter describes several case studies in which neural networks are used which have been pre-trained using the learned lessons. The case studies contain visual and textual data. This shows that the concept of siamese pre-training of neural networks using weak labels can be done on several types of data. The trained models' performances will be evaluated by comparing them with commonly available models like BERT for textual data and ResNet for visual data. It will be shown that the custom pre-trained models perform comparably well as the externally pre-trained models, which have pre-trained using significantly more resources. This again will show that using a well-thought pre-training approach is as valuable as using a large amount of resources.

4.1 Text classification using transfer learning

In the previous sections 3.1.4 and 3.3.3, models have been pre-trained on textual data before then being applied onto different tasks. That approach is called transfer learning (see section 2.1.2). Additionally, it was shown that the models trained on large data were, on average, performing roughly the same as the models trained only on the data on which they were evaluated. The only difference lies in the task's complexity: The more complex a task is, the better the model trained on large data is (see section 3.1.4). In section 3.3.4 it was shown that pre-training a model with multiple training targets results in richer vector representations of the input data: When dealing with sentences, instead of only pairing two sentences' vector representations, a model can also be trained to create rich vector representations for each token of a sentence, before then merging those token representations to a single vector representation.

This section makes use of the drawn conclusions from previous experiments. At the same time, it is studied how the merging of multiple sentence representations affects a classification task. The context in which this study was executed is the bots and gender profiling task of PAN 2019 [30, 99, 100, 93]. The rather small dataset provided by *PAN @ CLEF 2019* [30] consists of tweets of different authors [100]. The challenge's task is to categorize authors by whether they are a bot or a human being, and if the latter, whether the author is male or female.[1]

4.1.1 Related Work

Transfer learning [89] is commonly used in the field of computer vision by using the pretrained *ImageNet* [46], which is trained on 1.2 million images, on tasks like image classification [34], object detection [39], image segmentation [31] or others. In the context of textual data, first the decision has to be made whether treating the data as static or sequential. In the first case, existing word vector models like *Word2Vec* [78], *GloVe* [91] or *context2vec* [77] can be used to create a vector representation of a sentence, tweet or document. Typically, this is done by calculating the average of all tokens' vector representations. For example, [69] have created an architecture for question answering using pretrained word vectors using Glove, [25] created a neural network for natural language inference, in which they used pre-trained word vectors, [127] used a pre-trained embedding for creating a neural network to do semantic role labeling. When treating textual data sequentially, the already architecture types like *SkipThougths* [59], *QuickThoughts* [71] or variations of the architecture based on BERT [32, 94].

Approaches for author classification not using neural networks include the approach presented by [7]. Here, the authors extracted features from tweets like the average number of emojis, hashtags or semicolons used or the number of links posted. These features were given into an SVM classifier. [55] created a dictionary with words used only by each of the possible classes of authors. For an unknown

[1] The results from this section have been presented at CLEF 2019 [18].

tweet, the likelihood of belonging to one class can then be calculated based on the words used in that tweet. [95] created a similar approach by normalizing all tweets, e.g., replacing URLs with an *URL* token. N-grams of lengths 3-5 were created, and the ones with the highest tf-idf scores were kept. Tweets were then represented by the n-gram used in them and were classified using an SVM.

4.1.2 Data

For the presented task, tweets of 2880 authors are given as training data. When comparing that amount of authors with the number of users a service like Twitter sees daily, it is incredibly small. With that small amount of data, the problem is that typical tweet patterns or even linguistic patterns may not be presented often enough in the training data that the network can robustly learn to recognize these patterns. Also, some patterns can occur above-average in the training data so that the network could overfit to the training data, which then is disadvantageously when using it on data unseen during the test phase.

Table 4.1 shows a subset of tweets of three different users. The data is very diverse: Some tweets contain short or long sentences; some contain a sentence and a link; others contain direct messages to other users. For every author, there are 100 different tweets. There are 4120 different authors of which 2060 are bots, 1030 are male, and 1030 are female.

4.1.3 Author embeddings

In the previous sections, experiments were executed, creating word- or sentence-embeddings. For the presented task in this section, however, an author embedding has to be created. Previously, words have been described as 1-dimensional data, while text was described as 2-dimension data since text contains sequences of words. When continuing that thought, author embeddings are created from 3-dimensional data since authors' tweets are sets of text. Since the tweets all have a fixed maximum length and the number of tweets is the same for all authors, the tweets could be represented by a three-dimensional variable with shape $(num\_tweets \times maximum\_length \times num_characters)$.

Visual data was introduced as 3-dimensional data, which was processed using convolutional layers. However, convolutional layers cannot be used when dealing with tweets: Convolutional layers' kernels are assuming that neighboring data is related. For example, in an image, neighboring pixels form visual objects. However, when considering a set of tweets, there is no neighboring relation in two different dimensions: The given tweets in the given tasks contain no timestamp. Thus, it is not known in which time-relation the tweets stand. Also, if one was to assume that the given order is related to them being sorted by time, it is unknown whether the tweets are sorted ascending or descending, nor is it known how much time lies between the tweets. The second non-related dimension is the word relation: When using a two-dimensional convolutional layer as with image data, not

Author	Label	Tweet
#1	Bot	I accidentally forgot to graduate from college. - Anne Lamott
#1	Bot	Gluten Free Recipes - Making a Gluten-Free Gravy http://t.co/aEbpDqvyHu
#1	Bot	Read the History of Foreigner http://t.co/Ujp7fjp8vr
#1	Bot	Read How to Use LinkedIn for Your Company http://t.co/OXuGp0OnHS
#1	Bot	I love science fiction. - Pam Grier
#2	Female	@hotelsdotcom Depending what province it has a name. In Ontario it's Simcoe Day, in Manitoba it's Terry Fox Day. It's also not a holiday in all provinces so a different name in each province is kind of cool based on the history of the region!
#2	Female	RT @TheMarilynShow: #ICYMI: But on the bright side, they were definitely well fed!
#2	Female	RT @F55F: Not sure how much to give as a wedding gift this wedding season? @TorontoStar has some guidelines to help you get through: https:...
#2	Female	@jamama_man That was intense!
#2	Female	Goal!!!!!!!!!! #EnglandvsSweden
#3	Male	@ashleeyywatt123 that's actually legit
#3	Male	@vmorra wasssssup? :D
#3	Male	Who's reaching wasaga this weekend?
#3	Male	@ashleeyywatt123 what a random tweet
#3	Male	Ok so change of plans since we did Miami last year we are going to Los Angeles in October instead. Cali here I come!

Table 4.1: Examplary tweets from three different users of the bots and gender profiling task of PAN 2019.

only the neighboring words in a tweet are considered, but also words cross tweets. For example, the first words of tweets would be treated as neighbors. Since the meaning of words is created by what words are following in a sentence or tweet and not by other first words of other tweets, using convolutional layers for this three-dimensional data is not suited.

The tweet data has to be treated hierarchically: There has to be a single vector representation for every tweet. The vector representations of those tweets then have to be merged into a single vector representation. There are multiple approaches possible to do this:

avg Calculating the average value was done for the transfer learning tasks with word embeddings. If the vector representations' features contain high or low values for all tweet encodings, that information can be used. However, when considering two tweets, if one feature contains a high value for one tweet and a low value for the other tweet, the average value would be the same compared to a feature having a mid-ranged value for both tweets. To circumvent that problem, the impact of also using the variance of all features also was investigated.

minmax When creating vector representation while using Jaccard or weighted Jaccard similarity as distance function, the created features gain the property that a high value corresponds to a feature being present while a low value represents a feature being absent. When creating tweets, the corresponding vector representations of those tweets may contain some critical features for one of the classes. If, in such a case, those features have a high value, using the maximum of all features ensures that that high value is being kept.

recurrent The tweets are represented by a sequence of vector representations, the same as text is represented by a sequence of tokens. That fact makes using a recurrent layer to create an author embedding a plausible choice. When calculating the average value of all tweet encodings or using the maximum value, much information is lost. A recurrent layer could consider all features of all tweets to ensure a rich author embedding. However, using a recurrent layer means introducing an additional layer that has to be learned.

For the given tweet data, the label for all authors is known: Half of the tweets are classified as coming from bots, a quarter of the tweets are coming from male authors, and the missing quarter is coming from female authors. If there was data with no known labels for authors, training author embeddings could be done using the siamese training approach: A random subset of an author's tweets is chosen as one entry of a pair. All tweets are encoded and merged using one of the chosen merge functions. For the pair's second entry, either a random subset of different tweets from the same author is chosen to create a positive pair, or a random subset of tweets from a different author is chosen to create a negative pair.

4.1.4 Evaluation

To solve the task of classifying an author's tweets into three different classes, several model types should be investigated:

- The basic siamese model trained to create a vector representation for a given input sequence (see section 3.3.1)
- The extended siamese model is trained to create a vector representation for each token of the input sequence. The created vector representations are reduced to a single vector, which is used as the sequences vector representation (see section 3.3.4)
- The extended siamese model, which also had to predict a token's POS tag (see section 3.3.4)
- The BERT model [32]
- A model that was not pre-trained

The siamese models have been trained with either cosine distance or weighted Jaccard distance. On average, the models with cosine distance performed a little better. Thus only their results are shown. Also, a different kind of model has been trained, which used the vector representations as input values for a classifier. For example, a classifier received two sentences' vector representations and then had to predict whether the presented pair is a positive or negative approach (see section 3.3.1). The created tweet encodings are merged with the average, minmax, or the recurrent method for all model types. In the task's description, the classification task was split into two subtasks: First, it has to be decided whether an author is a bot or human, then if the author is likely human, whether the author is male or female. When the models were trained, they have been trained to predict whether the author is a bot, male or female, i.e., there was no hierarchical decision making. This was chosen because the probability, for example, of an author being female directly influences the probability of that author being a bot. Ten experiments have been executed for all model types and merge modes using 80% of the authors as training data and the rest as test data. For each experiment, the split was done randomly. Also, all models have been trained for one hour.

Table 4.3 shows the average test accuracy of the previously mentioned experiments. The best model correctly classified about 80% of the authors. For the different classes, all classes were predicted with the same accuracy. Thus, no class was particularly hard to predict. The siamese model trained with cosine distance, the tasks called *S+T+M*, and the merge function *minmax* has been used to predict all authors' tweets. These predictions have been used to look at authors with high probability for one class. Looking at those tweets can give insight into what tweet behavior lead to which classification. Also, using the merge function *minmax* has another advantage: Given that the tweet encodings' features are ranged between

	avg	avg+var	minmax	recurrent
no pre-training	0.7055	0.7121	0.7224	0.6490
cosine, S	0.5841	0.5924	0.5930	0.5885
cosine, S+T+M	0.7932	0.7996	0.7909	0.7371
cosine, S+T+M+PT	0.7656	0.7741	0.7605	0.7146
classifier, S	0.6023	0.6171	0.6433	0.6123
classifier, S+T+M	0.7513	0.751	0.7498	0.6595
classifier, S+T+M+PT	0.7185	0.7112	0.7285	0.6208
BERT	0.5992	0.6469	0.6423	0.5871
BERT, 24 hours	0.7815	0.7855	0.7709	0.7552

Table 4.2: Test accuracy on the bot gender profiling task with models trained with **S**entence pairing, **T**oken pairing, **M**asked input as well as **P**OS-**T**ag prediction as well as two other model types: BERT and a model without pre-training.

	avg	avg+var	minmax	recurrent
no pre-training	0.7055	0.7121	0.7224	0.6490
Siamese, S	0.6023	0.6171	0.6433	0.6123
Siamese, S+M	0.7513	0.751	0.7498	0.6595
Siamese, S+M+PT	0.7185	0.7112	0.7285	0.6208
BERT	0.5992	0.6469	0.6423	0.5871
BERT, 24 hours	0.7815	0.7855	0.7709	0.7552

Table 4.3: Test accuracy on the bot gender profiling task with models trained with **S**entence pairing, **T**oken pairing, **M**asked input as well as **P**OS-**T**ag prediction as well as two other model types: BERT and a model without pre-training.

-1 and 1, an encoding with values mostly close to 0 is unlikely to have its values used in the classifier. On the other hand, if there is a tweet of which the encoding's features have a log of very high or very low values, that encoding's values will be used in the classifier. Thus, given all the author's encodings, it is possible to calculate the tweet that has the biggest impact of the result of the *minmax* function.

For the bot class, the author, predicted with the highest bot confidence, had many long tweets with sentences similar to news stories' headlines. Additionally, all tweets end with a link. The bot predicted with the second-highest confidence contains tweets with exclusively buzzwords like *Chances* and *Opportunities*, the tweet with the second most significant impact on the *minmax* advertises a *[...] web-based adult entertainment video on demand network [...]*. Of the male authors, the male author predicted with the highest confidence mostly tweets about soccer-related topics. His tweet with the most impact on the classification mentions that *Daniel Arzani has signed for #CelticFC*, another tweet celebrates *[...] the goal of the weekend [...]*. For the female users with the highest prediction confidence, no particular topics stood out. The only noteworthy aspect is that many tweets tag many other Twitter users.

When not looking at the actual tweets but the performance of the different models given table 4.3, several conclusions can be drawn from these results:

- The model not having been pre-trained performs worse, which again demonstrates the benefit of using pre-trained models, even if they are not pre-trained on data from the same domain.

- The siamese model trained only with the auxiliary task, as to whether two sentences are coherent, performs worse than the models trained on additional tasks. This again confirms the results from sections 3.2.2 and 3.3.4 (see also research question 2 from section 1.2).

- The models trained with the additional task of predicting a tokens POS tag are not performing better compared to the models without that additional task. In section 3.3.4, the very same models were used, but the results were the other way around. This shows that the additional features learned by the POS tag task helped to improve the results for the SentEval tasks. However, the tweets in the bot gender profiling tasks were different in the way that the tweets do not necessarily consist of coherent sentences. Instead, the tweets frequently consist of smilies, emojis, punctuation, unknown words like user names, and grammatically incomplete sentences. Knowledge about POS tags does not help in this context. This further confirms experiments from section 3.1.4: Here, models have been trained on either selected data or the big textual corpus. In some cases, using more data has not led to significant improvement in results, thus showing that the bigger corpus did not lead to features being learned that improve the models' performance on the transfer learning tasks. When dealing with tweets, knowing about grammatically correct sentences does not improve the models' performance.

- The BERT model is performing significantly worse compared to all the other pre-trained models. At first, this is surprising because the BERT model is larger, has been pretrained longer, and on more data. However, the fact that the BERT model consists of significantly more layers is also one of its disadvantages: The more layers a neural network has, the smaller its gradients become. The result is that one hour of training time is too short for the BERT model to adapt to the presented use case. To further show this, it was trained for an additional 23 hours. After having been trained for 24 hours in total, its results were comparable to the results of the other models (see also research question 3 from section 1.2).

- The previous point relates to a statement made in section1.2: Training an autoencoder is expensive because a lot of weights are used in training that are not used for a transfer learning task. In this section, the pre-trained BERT model was used. That model consists of many layers, which lead to very good results in many tasks. However, the other models used in this section achieved comparable results with significantly fewer layers. This relates to the situation with the autoencoder since the BERT model has more layers and thus more weights than are actually needed for some tasks.

- When merging the tweets' encodings to a single encoding by using the features' average values, also adding the features' variance improves the results. This shows that some information is lost when calculating the average. This impact can be prevented by also including the minimum and maximum values. The fact that using minmax as merge mode shows that for this classifying task, for some authors, it is enough to know whether a few specific features are present (i.e., having a very high or low value). Using a recurrent layer as merge mode proved to be the worst of the four options, which shows that having additional parameters learned how to do the merging is slow. In fact, since the pre-trained part of the models, which create the tweet encodings, are further trained, the tweet encodings can be created such that negative aspects of the merge modes are minimized.[2]

[2] It can be assumed that using all four merge modes at once would lead to even better results since each merge mode's disadvantage can be compensated by one of the other merge modes.

4.2 Predicting treatment success of macular edema

When being on the internet, anyone can create data by, for example, creating a post on an internet forum. The meaning of a post on a forum may only be graspable if the topic of that forum is known, i.e., linguistic data is highly variable and highly dependent on context. For machine learning approaches, such problems are challenging, but due to the sheer amount of data, linguistic models can successfully be trained (see models like BERT [32] or the experiments in section 4.1). A use case for linguistical models may be the prediction of a customer's email's sentiment. If the sentiment was incorrectly classified, this might result in an unsatisfied customer.

In contrast to textual data, there hardly exists any medical data. Also, if a machine learning model is used in the medical context, for example, when deciding how to treat a patient, incorrect predictions may do long-term harm to a patient. Given the sparse data and the high demand for model quality, working with medical data must be done more carefully. In the beginning, the data most often is created by expensive and specialized devices. Also, the used data has to be labeled by experts, in this case, being doctors or researchers in the medical field. These two factors play a significant role in the high cost of medical data.

Still, artificial intelligence and neural networks, in particular, are used more and more for supporting doctors in their decisions. Among many existing medical fields, this especially has been the case of treating macular edema [3, 102, 52, 112]. Macular edema is an eye disease that can affect visual acuity. Among the possible causes, the most common diabetic cause of vision loss in different societies is diabetic macular edema [92]. It has been shown that the blindness rate would be reduced by comprehensive screening programs and early treatment of the eyes with effective diagnostic tools [130].

This section describes two created studies for creating models that are dealing with images of macular edema.[3] The first model is used to predict a binary value, representing the model's confidence that something pathological can be found on one of the images. The second model is used to predict the treatment process of patients.

4.2.1 Macular edema dataset

The department of ophthalmology of the university hospital of Leipzig University has created a dataset of Optical Coherence Tomography (OCT) images. The images show a cross-section of an eye's retina. Figure 4.1 shows an exemplary raw image of the macular edema dataset. The image contains two parts: The left part of the image show an eyeball. The dark green lines represent the set of images taken from the retina, whereas the bright green arrow represents the part of the retina shown in the right part of the image. Here, a cross-section of the retina can be seen. This particular retina contains a macular edema. For training the neural

[3] Work building on the results of this section has been presented at MIUA 2019 [17].

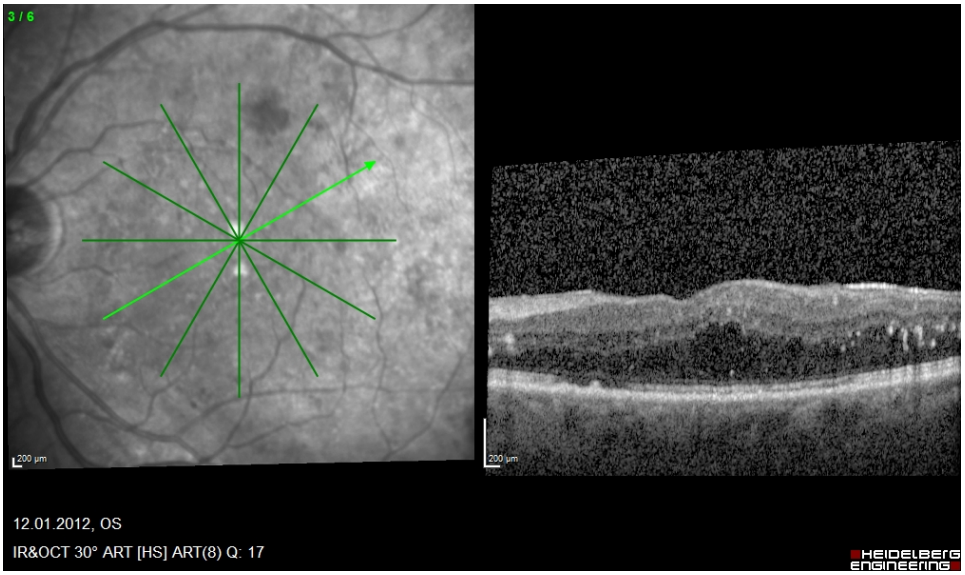
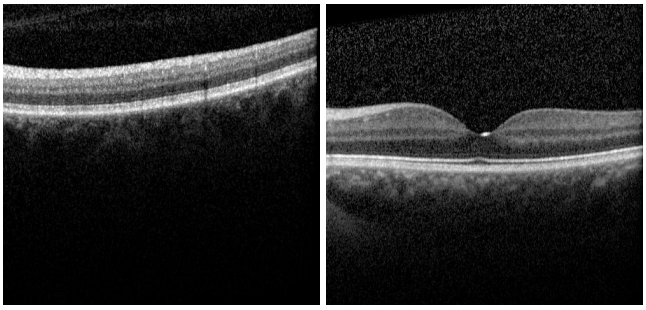
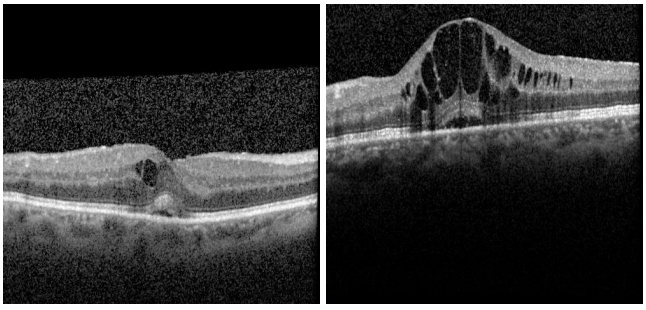


Figure 4.1: An exemplary raw image of the macular edema dataset.



(a) An OCT image of a retina without edema.

(b) An OCT image of the macular without edema.



(c) An OCT image of the macular with visible edema.

(d) An OCT image of the macular with visible edema.

Figure 4.2: Several exemplary OCT images from the macular edema dataset.

networks, the right part of the raw image. Figure 4.2 shows accordingly preprocessed images. The upper two images show an eye's retina without macular edema, whereas the bottom two images contain macular edema. The raw images are all of varying sizes, of which the average width is around 1500 pixels, and the average height is 500 pixels. For the following experiments, all eyes have been resized to $128x128$ pixels.[4]

The dataset consists of images that can be put into two broad categories. The first category contains images with labels containing a retina with or without macular edema. Of these kinds of images, there are 3543 images with signs of macular edema and 1436 images with no signs. The second category contains all images with macular edema. However, for these kinds of images, there exist additional metadata. This metadata includes information like the patient's age, treatment information like specific drugs given, and information regarding the treatment success. The treatment success is measured by a simple test being how many letters on a chart the patient can read. If the number of readable letters increases, this is seen as a sign that the patient's eyesight has improved. The treatment success can also be monitored by measuring specific information regarding the macular edema. However, the patient's subjective impression of eyesight is seen as the critical value to improve. There are 107 images of patients' retina for which there was a treatment success. For patients without treatment success, there are 351 images.

4.2.2 Pretraining

For the two models to be presented, various models have been pretrained to get the most of the existing data. The different approaches that have been used are the same as presented in section 3.2.

Autoencoder The first one is an autoencoder. The autoencoder receives a randomly chosen image and consists of an encoder and a decoder step. The encoder consists of six blocks with skip layers (see figures 3.5 and 3.6) and encodes the input images to a tensor of shape $(2, 2, 256)$. If that encoder were to be used for further tasks, the value tensor would be reshaped to a vector with 1024 values. The decoder receives the encoded tensor, consists of six skip connection blocks, and creates a tensor of shape $(128, 128, 1)$, i.e., of the same shape as the input image.

Siamese The siamese models contain the same encoder part as the previously mentioned autoencoder. However, the encoder's output is flattened to a vector with 1024 features, which is then passed through another feedforward-layer with output shape 512. Depending on the used similarity function, that vector is activated with either sigmoid when using weighted Jaccard index or tanh when using cosine distance. The positive/negative pairs are chosen by the image's class information.

[4]There have been experiments with larger image sizes. However, there was no significant improvement compared to $128x128$ pixels.

Thus, when the two randomly chosen images have the same class, they are considered a positive pair. Otherwise, it is considered a negative pair. A second training target is the reconstruction of the input image using the image's vector representation. That approach is the same as using an autoencoder.

Instead of using a distance function for calculating vector similarity, a classifier can be used for that task as well (see section 3.3.4). In that case, a classifier receives two image encodings and has to predict whether the two images belong to the same class.

ResNet The *ResNet* [45] model was trained on a large image data set and thus have learned to recognize very different visual features. When using it for a transfer learning task, as has been done here, the model's last layer will be removed. That way, the model is only creating an image encoding instead of classifying an input image into one of 1000 different classes. This model was included in the evaluation study to test how well a model that has been pretrained on an entirely different dataset will perform.

4.2.3 Patient classifying

The OCT creates several images of a patient's eye at once (as shown in the left part of figure 4.1). When a doctor examines a patient, the doctor has to look at every taken image to check whether something pathological can be found. If any of the created images contain macular edema signs, the patient has to be treated accordingly. Since the patterns that need to be found by the doctors can be very small, and there are many images, manually checking each image can be very time-consuming.

This section presents the results of a study that has been executed using the macular edema dataset (see section 4.2.1). The dataset contains sets of image of 289 patients' eyes (see table 4.4). The images are split into sets, while each set belongs to one patient. For that set, there is information on whether something pathological can be found on at least one of the images. In that case, that eye has to be classified as having macular edema. Otherwise, the eye has to be classified as non-edema.

The models used for this task contain the pretrained encoders from the previous section 4.2.2. For each eye, a random subset with a maximum of 16 images is chosen. Each of those images is encoded. Next, the different encodings are merged into a single encoding. There are two different methods used to merge the images: Either the encodings are averaged. In that case, the resulting encoding has the same number of features as each single image encoding. In the other case, the encodings are merged using each feature's minimum and maximum value. Here, the resulting encoding has twice the number of features as a single image encoding. The merged encoding is given into a classifier, consisting of three feedforward layers with a single output at the end, representing the eye's class. The models are trained several times with a different set of training and test data each time.

	edema	healthy
num. of patients	200	89
num. of images	3629	1350
avg. num. of images per patient	18.16	15.17
min. num. of images per patient	1	1
max. num. of images per patient	224	206
med. num. of images per patient	8.5	6

Table 4.4: Numbers of image types in the macular edema dataset for the patient classifying task.

	average			minmax		
	edema	healty	weighted	edema	healty	weighted
cosine	0.589	0.553	0.564	0.477	0.539	0.520
cosine, reconstruction	0.750	0.742	0.745	0.695	0.764	0.743
classifier	0.631	0.5	0.541	0.486	0.53	0.516
classifier, reconstruction	0.653	0.446	0.589	0.603	0.606	0.604
autoencoder	0.796	0.46	0.692	0.796	0.361	0.662
Resnet	0.727	0.779	0.763	0.812	0.785	0.793
no pretraining	0.756	0.252	0.601	0.657	0.438	0.589

Table 4.5: Test accuracy for the patient classifying task with different pretrained model types and different methods of merging sets of image encodings. The siamiase model was either trained using cosine distance or using an additional classifier.

For both classes, 80% of the data is used for training, and 20% of the data is used for testing. Each model is trained 20 times with random weight initializations each time. The different images per patient could be seen as sequential data. This would result in the different images encoding being passed through a recurrent neural network, typically done with textual data. It was decided not to use recurrent models in this context: Recurrent layers imply that the data given into those layers depends on the data's order. For the given tasks, the order of images is arbitrary, thus using a recurrent layer does not seem suited.

Table 4.5 shows the results for the models given their pretrained encoder and their method of merging the set of image encodings. As was the case with the experiments in sections 3.3.4, 3.1.4 and 4.1.4, the models trained with multiple training targets performed the best. The Resnet model performed the best among these models since it was pre-trained on a huge image dataset containing 1,000 different classes. However, given that this model was trained on 1,000 classes instead of two and probably longer than one hour, its performance is only slightly better than the next best model. This is similar to the results of sections 3.1.4 where word embedding models pre-trained with smaller data performed about as well as models trained on large data. The model pre-trained to classify an image's vector encoding performed worse than the model trained with a specific distance function.

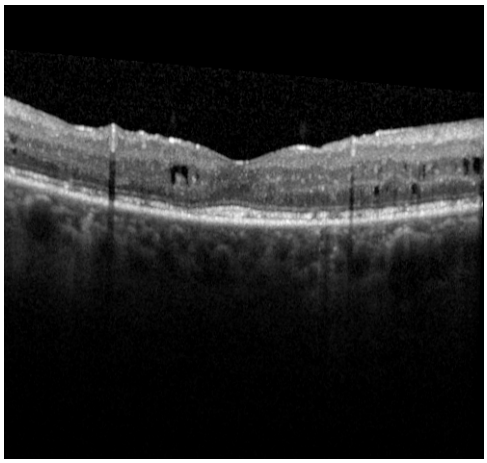


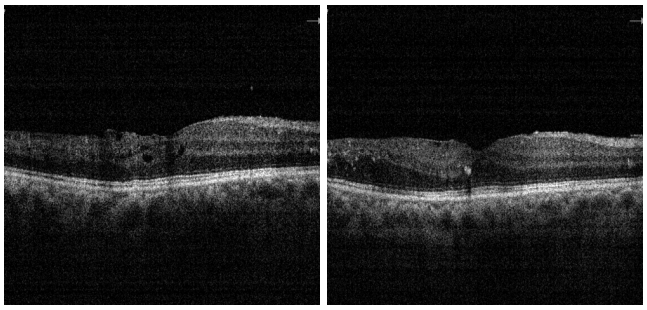
Figure 4.3: An exemplary healthy images with patterns that can also be found in images containing something pathological.

This further confirms the results from the mentioned previous sections. Using an autoencoder for image data may be expensive because of the encoder. However, since the encoder is trained, such that is has to reconstruct as many visual features as possible, leading to learning of many features. This is why it is not surprising that the siamese models pre-trained without the additional target of reconstructing the input image perform worse than the autoencoder. Also, nearly all models performed better on correctly predicting the image sets labeled as having edema. This is not surprising since edema's visual pattern is quite strong: If there is a set of images and one of the images contains a black circle in the middle of the image, all images are classified as edema. On the other hand, if none of the images contain edema, all of the images must be checked thoroughly. Strong evidence for this assumption is the fact that the not-pretrained model classifies the edema images with high accuracy while having trouble with the non-edema images.

When looking at the incorrectly classified image sets, there is another reason why the performance for the healthy images is lower than the performance regarding edema image sets: Apart from the edema images being easier to classify, some of the healthy images contain visual features shared by the edema patient. Figure 4.3 shows an example of those images. Unfortunately, for this image set, there is only a single image in that image set, leading to a miss-classification.

4.2.4 Treatment prediction

The previous section described the task of classifying a patient's eye, whether something pathological could be found. If yes, that eye has to be treated. Of the several possible treatments, one is to give an injection into the edema. If successful, the edema is getting smaller, which results in a better vision for that patient and



(a) OCT image for month 0. (b) OCT image for month 3.

Figure 4.4: Two OCT images for one patient at month 0 and month 3.

thus an increased quality of life. For some patients, though, there is no improvement, and a different kind of treatment has to be tried. Knowing early whether a treatment will lead to no significant improvement is critical because an alternative treatment can be implemented. Thus, given data of a running treatment, the task is to predict the treatment success.

For predicting treatment success, significantly less data is available: Patients have to be monitored over a longer period of time. If a patient is being monitored and, after a certain time moves, loses interest in the study or even dies, the treatment success cannot be known. For these reasons, there exists only data for 20 patients for which the treatment has been successful and 62 for which the treatment has not been successful. For each patient, there exist images from the beginning of the treatment (month 0) and three months (month 3) after the beginning of the treatment. The treatment's success is defined after a period of twelve months. For months 0 and 3, there exist at least two images per patient. Figure 4.4 contains two images of the same patient for month 0 and month 3. Alongside the visual data, there exists meta-data regarding the treatment process. These meta-data includes general information like the patient's age and sex, whether the shown eye is the left or right eye, as well as medical information like what kind of drug has been given alongside the treatment or whether there have been other kinds of surgery done. In total, there are 15 different values of meta-information.

Since there is hardly any data and the visual patterns vary widely between the images, it is evident that a not pre-trained model would easily overfit to the training data. If that model was to be used on test data, it is to be expected that its performance is not good enough to be used in production. Thus, since there exists a large set of images that is of the same type as the data for the task of treatment success prediction, the model will be created with a pretrained encoder. The encoder will be used to create an image encoding for the images from months 0 and 3. The images of the two-time steps are merged separately such that the following layers can learn which change in features over time leads to what kind of treatment success. If all input images were merged into one encoding, the eye's change over time would be lost. Alongside the image encodings for the two time-

	meta-data			no-metadata		
	sucess	no success	weighted	success	no success	weighted
autoencoder	0.682	0.688	0.686	0.658	0.649	0.651
siamese + reconstruction	0.744	0.692	0.704	0.729	0.712	0.716
no pretraining	0.543	0.531	0.533	0.521	0.504	0.508

Table 4.6: Test accuracy for the patient classifying task with different pretrained model types and different methods of merging sets of image encodings.

steps, the meta-data is added. Since the image encoding has 512 features and there are 15 values from the meta-data, the vector given into the classifier contains 1039 features. For these experiments, the image encodings have been merged using both the minmax and the average function. When using both functions, one of the functions' possible disadvantages can be compensated by the other function. Of the different similarity functions, the pretrained models trained with cosine as similarity function have been chosen. Also, that type of model was either trained with the additional target of reconstructing the input image. As was the case with the previous experiment, the following models have been trained several times with a random split of training and test data, whereas 80% of both classes, i.e., treatment success and no treatment success, have been used for training data and 20% have been used for test data.

Table 4.6 shows the results for the task of treatment success prediction for different kinds of models with meta-data either given or not given. The results show that the model that has not been created with a pretrained encoder performs significantly worse than the other models. Since its performance is around 50%, and there are two classes to predict, the model has not learned at all to correctly predict treatment success on the test data. The other models all perform about 70% correct, which shows that given the images, it is possible to predict the treatment success. In the case of additional meta-information, the performance is better for all models. Also, as was the case for the experiments in section 4.4, the model which has been created with the pretrained encoder, which was trained on two targets, performs the best. This again shows that pre-training a model with different targets, even if the labels are weak, improves the model's performance on transfer tasks.

4.2.5 Conclusion

Most importantly, the results in section 4.2.4 show that it is possible to predict treatment success. However, the average prediction accuracy is not sufficient to be used reliably. For all trained models, the average score was calculated for each test image. Figure 4.5 shows the quantity of those scores using ten quantiles, i.e., how many treatments have been predicted correctly in which percentage of the case. The results show that, in some cases, the models can predict the treatment success correctly with high confidence, while for other cases, the models cannot predict

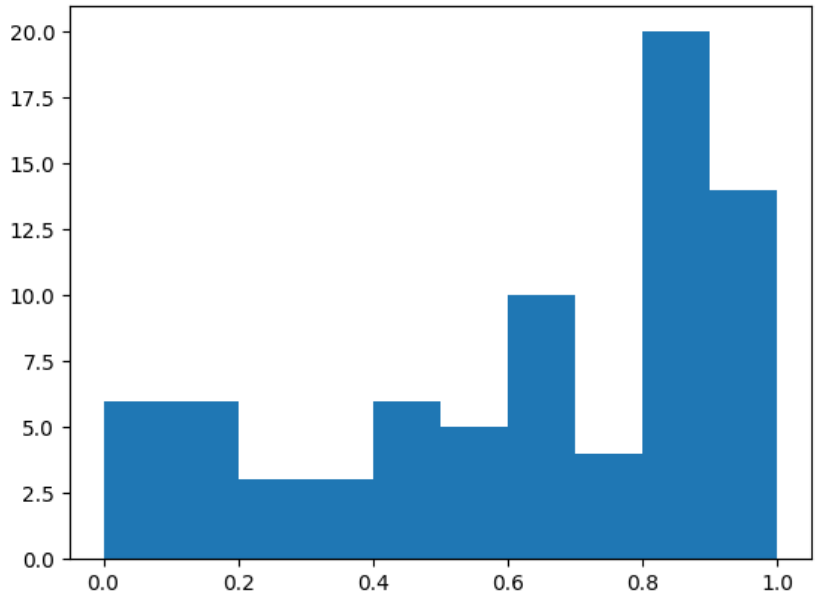


Figure 4.5: Accaracy quantities with wich patient's treatment success has been predicted.

with certainty. Given this data, in future research, the question has to be posed why some cases are predictable while others are not. If validated medically, the presented models could be used for treatment prediction for some of the patients. When looking at research question 2 in section 1.2, the results from this study demonstrate that pre-training a model with multiple training targets, such as input image reconstruction and weak labels by using randomly chosen image pairs, lead to better results compared to no use of pre-training. This further shows that when working on a machine learning task with small amounts of labeled data, the usage of additional non-labeled data delivers better overall results.

4.3 Finding Shakespeare quotes using sentence similarities

Any created text contains patterns that already exist in other text. Such re-use of text may occur knowingly or unknowingly. Texts do not contain independent meaning and are seen as being *intertextual*. Consequently, the study of analyzing texts by re-used patterns and which texts or documents are related is called *intertextuality* [5]. In some cases, those words or phrases can be re-used to communicate a deeper meaning which can only be understood if that other author is known as well [8]. Analyzing whether a text is referencing a different text can be considered being subjective. However, finding re-used sentences or word patterns can give a strong hint about an intertextual reference.

This section covers three executed studies for the detection of Shakespearean intertextuality (see [76]) in post-modern fiction, as Shakespeare's words, topics, characters, and plots are present in some of the most successful writers of the genre like for instance, Neil Gaiman & Terry Pratchett [111]. For the first study, a corpus of 31 books is used together with a list of 109 known quotes.[5] The second study used the same data but added additional training data, which was more suited for the given use case. The third study contained a small corpus of phrases which have been manually annotated with additional information like the type of quote.

4.3.1 Book corpus

The first study presents an NLP-pipeline consisting of two steps: First, a classifier is used to extract sentences in the corpus of post-modern fictional literature that are potential candidates for quoting Shakespeare. Second, these candidate sentences are compared to the actual Shakespearean texts. This comparison is implemented using a sequence vector representation created by a siamese neural network. That model is created using the drawn conclusions from section 3.3.

4.3.1.1 Data

The data for finding reused Shakespeare quotes consists of two parts: The first part includes text created by Shakespeare. They include dramas like *Hamlet*, *Macbeth* and *Othello* as well as other dramas, tragedies and sonnets. In total, there are around 140,000 lines of Shakespeare text. The other part of the data set includes post-modern fictional literature in the form of 31 books. The books and the Shakespeare texts are all parsed to create a unified way of representing text. First, all lines containing capital letters are discarded because those lines often are in the form of *CHAPTER ONE*. Some of the books or Shakespeare texts include line breaks that are not aligned with sentences. Some lines contain the end of one sentence and the next sentence's beginning. Thus, all line breaks are removed to create one long

[5] This study has been executed in corporation with Manuel Burghardt and Johannes Molz [16].

line. That line is split into sentences using the nltk sentence tokenizer [6]. Lastly, punctuation is removed, and all lines are changed to lower case. There may be quotes in modern literature in the middle of the sentence, while the original quote is at the beginning of the sentence, starting with a capital letter. Lowercasing the data makes the lines more similar to each other. Table 4.7 shows the included books along with the number of parsed sentences per book.

Alongside the unlabeled data, there also exists ground truth data [82]. These include found reused Shakespeare quotes from Shakespeare's *Hamlet* in nearly all of the books. An exemplary found quote exists in *Early Riser* by *Jasper Fforde*. One of the lines is *Something rotten in the winter.*. That sentence is referencing Shakespeare's Hamlet: *Something is rotten in the state of Denmark.* Overall, the type of references varies: Some found references do only contain a single word (*Yorrick*, *fishmonger*), others use the whole sentence (*There are more things in heaven and earth*), while others use the original quote but change some words (*to espresso or to latte*). In total, the ground truth contains 109 entries that are seen as being fit for an automated process. Not considered found references include, for example, very long paragraphs or are seen as too abstract. An example of the latter would be a scene where two people are talking, of which one is standing on a balcony. Such a scene would be a reference to Hamlet, but that context is not extractable from a single sentence.

4.3.1.2 Method

To find reused Shakespeare quotes in post-modern fictional data, first, it has to be discussed what exactly should be found. The previous section described existing ground truth data for quotes referencing Shakespeare's Hamlet. The length variety of those quotes is large, i.e., some quotes refer to a single quote while other quotes refer to a whole line or sentence. Thus, it was decided to split the Shakespeare data and the sentences of post-modern fictional literature into n-grams of varying lengths. If there is a very short quote containing only one or two words, any variation will make it unrecognizable. It thus was decided that these kinds of quotes are best to be found by using a predefined list of keywords and then looking them up in another document. For example, the names of characters in Shakespeare's plays are mostly unique. If one of those names is used in another text, it almost always references Shakespeare. In the ground truth data, the longest coherent quotes contain around 13 words. Other references are stretched over whole paragraphs. It was decided not to aim to find those kinds of references because they are mostly very abstract. Thus, the resulting lengths for n-grams are 5, 7, 9, 11, and 13. To reduce the number of n-grams, even numbers are left out since, for example, n-grams of length 6 are included in n-grams of length 7.

[6] `https://www.nltk.org/`

Title	Author	#Sentences
Adams, Douglas	H2G2 The Ultimate Hitchhiker's Guide	21,289
Barnes, Julian	The Noise of Time	3,208
Carter, Angela	Nights At The Circus	5,883
Carter, Angela	Shaking A Leg	15,433
Carter, Angela	Wise Children	5,927
Fforde, Jasper	Early Riser: The new standalone	9,654
Fforde, Jasper	Lost in a Good books	9,258
Fforde, Jasper	One of our Thursdays is Missing	7,640
Fforde, Jasper	Something Rotten	9,201
Fforde, Jasper	The Eyre Affair	8,997
Fforde, Jasper	The Well Of Lost Plots	9,305
Fforde, Jasper	The Woman Who Died a Lot	7,746
Fry, Stephen	Making History	12,608
Fry, Stephen	Moab Is My Washpot	6,951
Fry, Stephen	Paperweight	7,631
Fry, Stephen	The Hippopotamus	7,741
Fry, Stephen	The Liar	8,280
Gaiman, Neil	Anansi Boys	10,778
Gaiman, Neil	Trigger Warning	9,772
Pratchett, Terry	Guards! Guards!	9,649
Pratchett, Terry	Nation	9,084
Pratchett, Terry	Night Watch	11,317
Pratchett, Terry	The Amazing Maurice and His Educated Rodents	7,412
Pratchett, Terry	Wyrd Sisters	8,315
Pratchett, Terry; et al.	The Science of Discworld 2	8,009
Rushdie, Salman	East, West	2,597
Rushdie, Salman	Imaginary Homelands	6,078
Rushdie, Salman	Joseph Anton	12,551
Rushdie, Salman	The Ground Beneath Her Feet	13,006
Rushdie, Salman	The Moor's Last Sigh	8,214
Smith, Zadie	On Beauty	12,631

Table 4.7: Corpus of postmodern fiction novels that will be used to test the automatic approach for text reuse detection.

4.3.1.3 Filtering

When splitting each sentence of the two text sources into n-grams of the shown lengths, the data size grows significantly. Instead of comparing each Shakespeare sentence with each sentence from the post-modern fictional literature data, all n-grams of the Shakespeare sentences have to be compared with all n-grams of all sentences from the post-modern fictional literature data. The number of pairs that have to be compared would be very high, resulting in a very slow process. Simultaneously, by looking at the created n-grams, it was evident that most of them are not quote-worthy. For example, in Adam Douglas's *The Ultimate Hitchhiker's Guide*, there is a 5-gram *at the end of the*. That 5-gram could occur in any sentence by any author, and it is not of further interest. The same can be said about most of the created n-grams.

To split the n-grams into two groups, quote-worthy and not-quote-worthy, a classificator was trained. The classificator received a sequence of tokens and had to predict whether the sequence was taken from the n-grams of the Shakespeare data or the n-grams of the post-modern fictional literature. The n-grams are chosen randomly from the two corpora. There is no weighting when choosing the n-grams. Since the post-modern fictional literature corpus is larger than the Shakespeare corpus, most of the n-grams are chosen from that corpus. This ensures that frequent n-grams are classified as non-Shakespeare, even if they also appear in the Shakespeare corpus. On the other hand, n-grams from the Shakespeare corpus are chosen with less frequency. Given that, and the fact that they contain words that are unlikely to appear in the post-modern corpus, the model is trained to classify these n-grams into the class of Shakespeare n-grams. During training, the n-grams given into the model are randomly masked. This ensures that the model does not memorize the quotes word by word and ensures that quotes with changed words are also classified as a potential Shakespeare quote. The trained classifier assigns an n-gram like *at the end of the* a probability of being from a Shakespeare text of 0.0012. An n-gram like *her too too solid kitchen* a probability of 0.9313, which is a variation of the quote *this too too solid flesh*.

The threshold to be set when filtering the n-grams is critical for finding Shakespeare references. If the threshold is set too high, too many potential references are discarded. To keep as many potential quotes in the data as possible, the threshold is set to a low value of 0.2. N-grams of length 5 were filtered using a higher threshold of 0.7. After looking at the filtered data, it was noticed that n-grams of length 5 contained many non-quoteworthy entries. Also, since the n-grams of length 5 are the shortest, there were significantly more entries than the other lengths. The two threshold values were found by ensuring that many known references are still included in the data. Also, even though the threshold values seem very low, between 80% and 90% of the n-grams are discarded (see table 4.8).

ngram length	number of ngrams	number of ngrams after filtering
5	1,729,519	237,690
7	1,480,019	359,216
9	1,262,438	101,875
11	1,078,270	38,872
13	921,433	5,284
Sum	6,471,679	742,937

Table 4.8: The number of n-grams for each n-gram length before and after filtering for the corpus of postmodern fictional literature.

4.3.1.4 Pairing

The created n-grams are to be paired using their vector representation. Of the different lengths, only n-grams of the same length are paired. The models which are used to create vector representations are the following:

- The basic siamese model trained to create a vector representation for a given input sequence (see section 3.3.1)
- The extended siamese model is trained to create a vector representation for each input sequence's token. The created vector representations are reduced to a single vector, which is used as the sequences vector representation (see section 3.3.4)
- The extended siamese model, which also had to predict a token's POS tag (see section 3.3.4)
- The model trained using the SkipThought approach (see section 3.3.1)
- The model trained using the QuickThought approach (see section 3.3.1)
- The BERT model [32]

4.3.1.5 Evaluation

The presented pipeline in section 4.3.1.2 was executed using a subset of the Shakespeare data as well as the full Shakespeare data. Pairing all n-grams from both corpora would lead to a very high number of pairs. To investigate the different model performances, the quotes from the ground truth data referencing Shakespeare's Hamlet (see section **??**) are used as a subset of the Shakespeare data. This means, instead of using all sentences from the Shakespeare corpus, only sentences are used that contained a known quote. On the one hand, this makes it possible to calculate a precision value for each model: The ranked lists can be annotated by how many pairs are seen as true-positives. Simultaneously, it is known which pairs

ngram length	Shakespeare (full)	Shakespeare (ground truth)
5	276,291	332
7	154,500	300
9	48,276	282
11	6,157	239
13	718	232
Sum	485,942	338

Table 4.9: The number of n-grams for each n-gram length before and after filtering for the Shakespeare corpus.

need to be found, which means a recall can be estimated. After having compared different model types, and estimated suited threshold values for vector similarity, the presented pipeline was used on the full Shakespeare data.

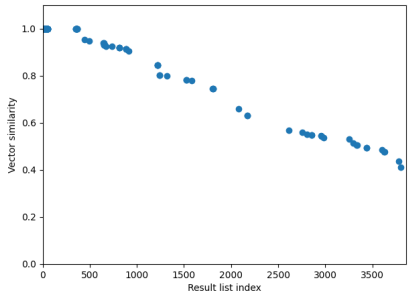
After using the filter presented in section 4.3.1.3, table 4.9 shows the number of n-grams for each n-gram length and corpus.

4.3.1.6 Ground truth data

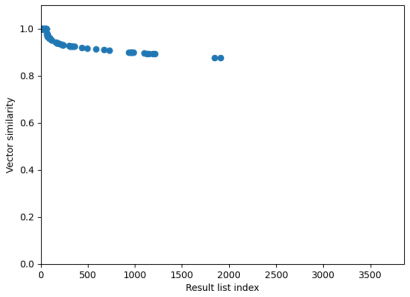
The listed model types in section 4.3.1.4 have been used to create vector representations of all the n-grams from the ground truth subset. These vector representations have been paired with the filtered n-grams presented in section 4.3.1.3. The result lists have been sorted by vector similarity, and each result list has been cut after 2000 entries. The result list of all model types have been merged into a single result list. After removing duplicate entries, that result list contained 3858 entries. The merged result list has been manually annotated: Of the 3858 entries, 119 have been marked as a reference, while 3739 have been marked as not being references. A ranked list for all model types has been created using that annotated list. The average precisions for all model types can be found in table 4.10.

The calculated average precision for the different model types confirms several drawn conclusions from section 3.3. The smaller the number of training tasks during pre-training, the worse the average precision is. The models named *S* and *QuickTought* only had to create a vector representation, which then had been used to determine whether a sentence pair is a positive or negative training example.

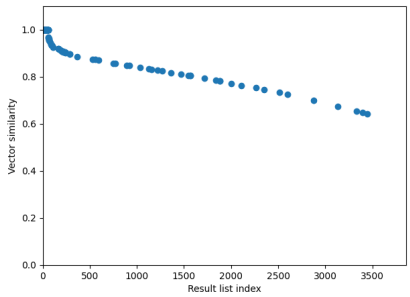
The results in table 4.10 confirm several results from section 3.3. It was shown that a single training task, like pairing two sentences, results in a somewhat usable vector representation. However, these vector representations perform worse compared to other trained model types. It was shown that the worse performance was caused by the encodings not including as many different aspects compared (see sections 3.2.2 and 3.3.3). In the present evaluation task, the model trained with the smallest number of training targets performs worst. The models trained with the *QuickThought* and *SkipThought* approach perform slightly better. The *SkipThought* approach consists of a decoder part used to generate a sentence. Thus, more in-



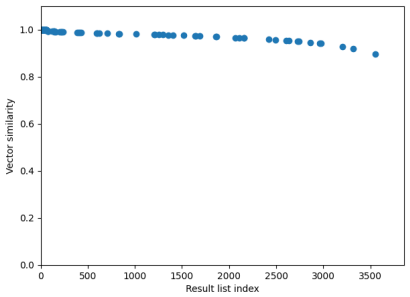
(a) S



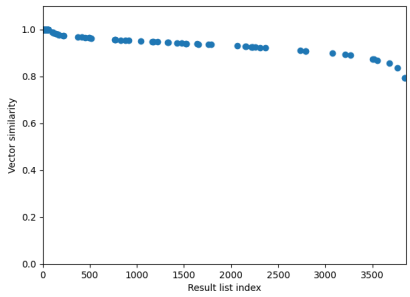
(b) S+T+M



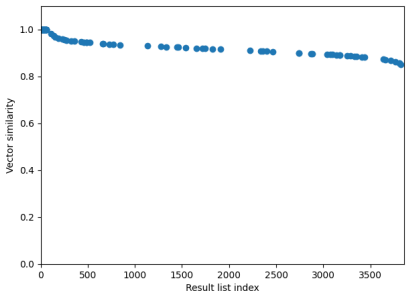
(c) S+T+M+PT



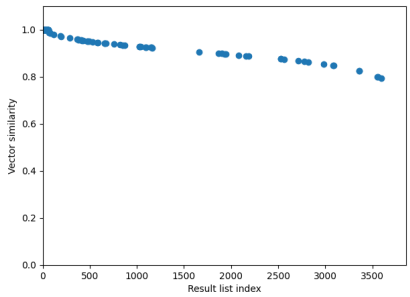
(d) S+T+M (classifier)



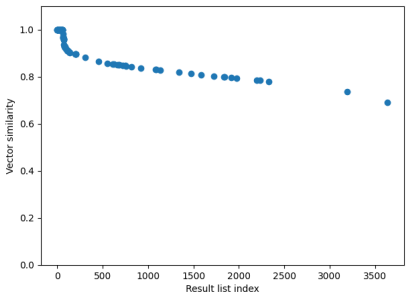
(e) S+T+M+PT (classifier)



(f) SkipThought



(g) QuickThought



(h) BERT

Figure 4.6: Position of the true-positive pairs in the result list of the models shown in table 4.10.

Model	Average precision
S	0.4547
S+T+M	0.7302
S+T+M+PT	0.6601
S	0.5408
S+T+M (classifier)	0.6522
S+T+M+PT (classifier)	0.5746
SkipThought	0.6001
BERT	0.6956

Table 4.10: Average precision on the merged annotated result list with models trained with **S**entence pairing, **T**oken pairing, **M**asked input as well as **P**OS-**T**ag prediction. When having to predict a token's POS tag, the vector similarity will either be calculated using a distance function or a classifier. The vector representations of the tokens have been merged using either the average.

formation from the preceding sentences is needed compared to the *QuickThought* approach, which again leads to better performance on transfer learning tasks (see section 3.3.4) and table 4.10. The models that had to create a vector representation for each token of the input sequence and a token for the whole sentence performed better than the previously mentioned models. The models trained with even one additional training task, being the prediction of each token's POS tag, performed worse (see section 3.3.4). This shows that, when directly using a vector representation for a similarity task, any additional task that breaks the desired vector properties harms the performance on similarity tasks. Additionally, the models trained to create a vector representation using a specific distance function perform better on a task where these vector representations and distance functions are explicitly used. Models like BERT or the siamese models that used a classifier to predict sentence pairs and POS tags perform a little worse since their vector representations have not explicitly been trained to be used with a specific distance function.

Figure 4.6 shows the positions of true-positive pairs in the ranked lists that have been created by the different model types. The X-axis represents a pair's rank, while the Y-axis represents the vector similarity. The fact that some models' similarity score is falling faster than other models is not important since only the relative similarity values are important. That means if one pair is seen as very similar and one pair as very different, it is not important whether the similarity score is 0.1 or 0.5. It is only important how many other pairs are ranked between the two mentioned pairs. The result lists all start with pairs that contain a vector similarity of precisely 1. This is because some pairs contain the same n-grams. For example, the sentence *[What] a piece of work is a man* is used in *Hamlet* as well as in *The Science of Discworld 2* by Terry Pratchett et al. Pairs that are placed on lower ranks on the list are of more interest. Pairs with a vector similarity lower than

1 but still very high are, for example, *The mirror up to nature* paired with *A mirror up to nature* (*The Noise of Time* by Julian Barne) or with *A mirror up to life* (*Wyrd Sisters* by Terry Pratchett). The lower the similarity score is, the more variance there is in the pairs. For the model called *S+T+M*, *to be or not to be that is the* is paired with *to espresso or to latte that is the question* (*Something Rotten* by Jasper Fforde) has received a similarity score of 0.9171. A pair with a lower similarity score is *things in heaven and earth* and *in either earth or heaven* (*Nights At The Circus* by Angela Carter) with a similarity score of 0.8294.

False-positive pairs with a high similarity score are mostly pairs with a high overlap of tokens but are still not considered a reference taken from Shakespeare. For example, the 5-gram *The beauty of the world* is paired with *The end of the world*. These two 5-grams have the same sentence structure, but given the longer contexts the two sentences are in, it is obvious that *The end of the world* is not a reference to Shakespeare. Another example of the same problem is the pair *in my mind s eye* and *in my mind since it*. The true-positive pair with the lowest similarity score contains the n-grams *is rotten in the state* and *something rotton in the state*. The similarity score of that pair is 0.8757.

37 of the ground truth quotes were either not found by the presented approach or ranked with a score so low that they are lost among the other low-ranked false positives like, for example, the references *her father's death* or *O God! God!*. Apart from quotes that have been filtered out in the pre-filtering step of the pipeline, other quotes that were not identified were found to be not suited for the presented approach. Among these quotes, there were ones with only one word like *Yorrick*, for which a simple string-search would be feasible. References like *2b or not 2b* differ too much from the original quote to be found automatically, while other references had typographical errors (*rotton* compared to *rotten*).

4.3.1.7 Full Shakespeare data

The previous section presented an evaluation study on a ground-truth dataset containing found Shakespeare quotes. The study showed that references that contain a large overlap of tokens, as well as quotes that have some changed keywords, can be found. References that are either very short, contain large variations, or are very implicit and therefore require a lot of context knowledge have not been found. Given the different model types, it was also shown that a model that is explicitly trained to create vector representations using a specific distance function produced the result list with the highest average precision. The executed study also helped determine which thresholds to use for the vector similarity. The lowest scored true-positive pair had a vector similarity of 0.8757. Such a threshold does not mean that there are no true-positive pairs to be found with a lower score, but it shows that it is less likely to find true-positive pairs. For the full Shakespeare data, a threshold of 0.8 was used.

The n-grams of the post-modern fictional literature corpus were filtered using a dedicated classifier. That classifier was used to discard n-grams like *at the end of*

ngram length	Shakespeare (full)	Shakespeare (filtered)
5	276,291	44,458
7	154,500	19,552
9	48,276	6,102
11	6,157	861
13	718	161
Sum	485,942	338

Table 4.11: The number of n-grams for each n-gram length before and after filtering for the Shakespeare corpus.

the. Such n-grams probably appear in both corpora but are not of further interest. The study in section 4.3.1.6 only used n-grams from the known ground truth data of the Shakespeare corpus, which made a pre-filtering of n-grams unnecessary since the number of n-grams is relatively small and most of the n-grams are considered quote-worthy. For the full Shakespeare corpus, a pre-filtering of n-grams is necessary since when using the full corpus, many sentences that are not quote-worthy are also included, and thus also many n-grams that need to be discarded. Table 4.11 shows the number of n-grams of the different lengths before and after filtering.

The result list contains $590,369$ entries. Of that list, only the top 5,000 entries were checked. At the top of the list are pairs where both n-grams are identical. For example, the quote *The seeming truth which cunning times put* is used in Shakespeare's *The Merchant of Venice* as well as in *The Moor's Last Sigh* by Salman Rushdie. A different example is *He jests at scars that never felt a wound* used in *Romeo and Juliet* as well as in *The Eyre Affair* by Jasper Fforde. The n-gram *as the end of his* is used in *Love's Labour's Lost* as well as in *Imaginary Homelands* by Salman Rushdie. That pair is a false-positive and made it past the n-gram filtering. Exemplary pairs with a very high similarity score are *there were nothing else to be done* (*Henry IV*) and *there was nothing else to be done* (*Joseph Anton* by Salman Rushie).

A pair with lower vector similarity is the n-gram *I am a most poor woman and* (*Henry VIII*) paired with *I am a poor fellow and i* (*East, West* by Salman Rushdie). Another example is the n-gram *[In] sooth i know not why i am so sad* (*The Merchant of Venice*) paired with *[In] truth i do know why i am so sad* (*Paperweight* by Stephen Fry). The n-gram *and a heart of gold* is paired 19 times with several other variants that contain the words *heart of gold*, all of which appear in The Ultimate Hitchhiker's Guide by *Adam Douglas*.

Of the false-positive pairs, many of them contain the words *of the*. For example, *The beggars of the world* is paired with *The privations of this world*. When training the n-gram filtering classifier, there are n-grams like *The beauty of the world*. Since some of the words are masked during training to ensure variants of quotes are not filtered (see section 4.3.1.3), any quote like *The X of the Y* is seen as a quote with

high confidence. Thus, many variants with those words passed the n-gram filtering along with n-grams containing word groups like *in the*, *by the*, or *of his*.

4.3.2 Asymmetricrical vectorization

When having a results list with several thousand entries, it is not feasible to manually check all entries for true-positive findings. This is especially the case if, at high-ranked positions, many entries contain undesired patterns. In section 4.3.1.6, a result list with around 4,000 annotated n-gram pairs was created. Using these annotations and especially the knowledge about what accounts for a desired pair and especially, since they make the biggest share, what accounts for a negative pair. To do that, a network was created to create vector representations that give lower similarity scores that have been seen as false-negatives. The created annotations are suited to continue training a siamese model since the annotations can be seen as positive and negative data pairs, which are the critical part of training siamese models.

Among the annotated pairs, there are pairs with both entries being *seem to me all the*. If a model is given such a pair as a negative pair, it cannot create a smaller similarity score: The presented siamese architectures use the same weights for all inputs. If the network were trained to give a different vector representation to one of the pair's entries, the other entry's vector representation would change in the same way. To solve that problem, the network's symmetric property has to be broken by training two encoders that share the same architecture but not the same weights. Such a model will be trained in the same way as the symmetric model, only that it is possible to create two different vector representations for the same input. When creating vector representations for the n-grams of the Shakespeare corpus and the post-modern fictional literature corpus, it must be kept in mind to use one branch of the model for each corpus.

The existing model was copied to speed up training, and the weights of the first branch of the model were fixed. That means, if the first branch of the model is used to create vector representations for the n-grams of the Shakespeare corpus, the existing vectors can be kept. For the second branch of the model, the weights have to be changed such that for false-positives, the vector representation is such that the pair will receive a lower score. Of the annotated data, there are 119 positive and 3739 negative pairs. If only those pairs were shown to the asymmetrical model, it would create only high similarity scores for the exact n-grams of that result list. If only the negative pairs would be shown, the second branch could change such that the vector representation will be such that it will not be similar to any other vector representation. It was decided only to use the annotated data's negative pairs for training the second branch. For positive pairs, random coherent sentence pairs from the existing larger corpus of news articles were used.

After training, the model's second branch was used to create a vector representation of the n-grams from the annotated data. With the pairs' new similarity scores, an average precision of 0.9415 was achieved. That high score is not sur-

Model	Average precision
S+T+M (trained asymmetrically)	0.7713
S+T+M (trained symmetrically)	0.7302
BERT	0.6956

Table 4.12: Average precision on the merged annotated result list with an additional model which has been pre-trained with asymmetrically on manually annotated data.

prising given that the model was trained on precisely that data. If it only was trained on positive and negative pairs from that dataset, a score very close to 1 can be expected. Another effect of the two branches is that the similarity scores cannot be directly compared. If there is a pair with two identical n-grams, using two branches leads to that pair receiving a score lower than 1. To achieve a score of 1, the two vector representations would need to be the same. If the second branch of the model is trained with some negative pairs that are identical or near-identical n-grams, this also means that identical or near-identical positive pairs receive a lower score. As a result, the new similarity score shall be called *interest score*.

To evaluate the new vector representations, the list of pairs from section 4.3.1.6 was used. The list was sorted with the new similarity scores before then calculating the average precision. The values presented in table 4.12 show that the new model performs better than the ones previously used. To further analyze the results, the list containing $5,000$ entries from section 4.3.1.7 was used. The vector representations of the n-grams coming from the post-modern fictional literature corpus were recalculated, and the list was re-sorted by the new interest score. For both lists, positional changes of all entries were calculated to see which pairs were ranked higher and which ones were ranked lower. The pair with the entries *and it is mine that* and *and that it is therefore* is ranked 4,090 positions lower with the new score. Other pairs that are moved down in the result list are pairs with entries like *as the sweat of a* and *of the faithful as a* as well as *a fellow of the selfsame* and *is a fellow of the*. The fact that pairs containing words like *of the* are ranked low shows that pairs with those words were frequent negative pairs when training the newly created second branch. The pair with entries *of sun and moon and that the* and *of the sun and the moon and* was moved 200 ranks up in the list. At first, this may seem surprising because the pair contains very similar n-grams, and both the similarity and interest score are around 0.89. However, when considering the similarity score, there are significantly more pairs with a score higher than 0.9, resulting in this particular pair being lower. Many higher ranked pairs are now ranked lower with the interest score. A similar example is the pair with the entries *between my finger and my thumb and* and *my ear between finger and thumb and*, which moved 247 ranks up to rank 141. Also, there is a pair with the entries *at the latter end of* and *at the end of the*. This pair is occurring multiple times since the

Lexia	Shakespeare quote	reference
Hieronimo go by	go, by Saint Hieronimo	go by old Hieronimo
bear a brain	I do bear a brain.	we must bear some brain
carry coals	we will not carry coals	I would bear no coals
hillo ho ho	Illo, ho, ho, my lord	ho, illo ho
host of heaven	death stood gaping wide	host of heaven
mind's eye	In my mind's eye	eye of their mind
planet strike	planets strike	struck with a planet
sea of troubles	sea of troubles	a troubled Sea of passion
the rest is silence	The rest is silence.	The rest is belly
to be or not to be	to be or not to be	to do, or not to do

Table 4.13: An exemplary subset of the *small gold standard quotes* dataset.

second part occurs in several of the used models. These pairs already received a lower score due to the interest score, but this pair is not in the annotated data. If this pair and other unwanted pairs were included in the annotated data, the interest score could be improved even further.

4.3.3 Reference type evaluation

The previously used dataset had the disadvantage that it contained references from 31 books with no guarantee that all references had been found. Thus, if a model were to rank a reference high, which has not been identified as a true-positive yet, it would impact the evaluation metric negatively, although the model acted correctly. To further analyze the impact of different architectures on the results of finding reused Shakespeare quotes, a small dataset of manually annotated references has been created, hereafter called the *small gold standard quotes*[7]. There are 100 different Shakespeare references with 20 unique quotes in this dataset. Table 4.13 contains exemplary lexias, Shakespeare quotes and references. The context in which the references are used is not shown.

The analysis in section 4.3.1.6 is built to handle a large corpus. The presented pipeline includes a step in which n-grams from the Shakespeare corpus are paired with n-grams from the corpus of post-modern fictional literature. Both sets of n-grams have been filtered in a previous step to reduce the number of pairs since, without that filtering, the number of pairs would be too large to get meaningful results in an acceptable amount of time. The presented data in table 4.13 only contains known quotes. Thus, the same approach from section 4.3.1.6 can be used without filtering. Consequently, the results are not influenced by an n-gram classifier or any other arbitrarily set threshold but only depend on the model used to

[7]The dataset has been created by Manuel Burghardt (burghardt@informatik.uni-leipzig.de) by using the WordWeb IDEM portal (https://wordweb-idem.ch/)

	Example	
Type	Original	Reference
Insertion	to be or not to be	to be named or not to be named
Deletion	go, by Saint Hieronimo	go by Hieronimo
Subst.	Illo, ho, ho, my lord	Illo ho , boys
Subst. (semantic)	we will not carry coals	I would bear no coals
Subst. (gram./ortho.)	hell itself should gape	death stood gaping wide
Changed word order	planets strike	struck Corioles like a planet
Repitition	Illo, ho, ho, my lord	ho, by, so, ho, illo ho, illo ho

Table 4.14: Examples for the different reference types with which the quotes from the *small gold standard quotes* dataset have been annotated.

create an n-gram's vector representation. There is additional information about the type of reference for all found references, whereas a reference can be annotated as being of multiple types. The available types are described in table 4.14.

The gold standard dataset is split into two parts. The first part contains the 20 unique Shakespeare quotes, and the second part contains the contexts in which one of the Shakespeare quotes has been used. The contexts have been split into n-grams of length 5, 7, 9, 11, and 13 using a sliding-window approach. After creating vector representations from both sets, each entry from the Shakespeare quotes is paired with each n-gram from the references. For each context, multiple n-grams are paired with the Shakespeare quote, but only the pair with the highest similarity score is kept. The result is 20 different lists with 100 entries each. How the result lists are grouped is equivalent to entering each Shakespeare quote into a search engine and getting a result list of 100 entries. Since it is known which entries belong to which search term, the lists are evaluated using the normalized discounted cumulative gain (nDCG).

Similar to the experiments from sections 3.3.3, 4.1.4 and 4.3.1.6, an externally pre-trained BERT model has been compared with a custom pre-trained model trained with the siamese approach and multiple training targets. All models also have been fine-tuned by further training them with the same training target with which they have been pre-trained but on a dataset containing text a corpus of early modern drama[8]. Training time for all models was limited to six hours. Another approach to make use of the pre-trained BERT model is to add an additional layer, which receives the vector representations of two inputs and has to predict whether the shown pair is a positive or negative pair [9]. The same approach was also implemented using the output of the pre-trained siamese model. When creating the result list, these two models' vectors were not compared using vector similarity but the trained classifier.

[8] https://graphics.cs.wisc.edu/WP/vep/vep-early-modern-drama-collection/
[9] https://keras.io/examples/nlp/semantic_similarity_with_bert/

Model type	nDCG
BERT	0.7524
BERT (fine-tuned)	0.7123
BERT (similarity)	0.3612
BERT (classifer)	0.3547
S+M+T+POS	0.7194
S+M+T+POS (fine-tuned)	0.744
S+M+T+POS (classifier)	0.3451
S+M+T+POS (overlap)	0.7656

Table 4.15: Discounted cumulative gain of different model types on the *small gold standard quotes* dataset.

Table 4.15 shows the results of different models that have been used to create the vector representations. The base BERT model is the best performing BERT model. The fine-tuned version performed slightly worse. With the fine-tuning, the model has been adjusted to words and sentence structures from the same domain, but the output is still not trained to be used as an n-gram's vector representation. Another BERT model was trained with an additional layer on top of the base model. That layer created an output for each time-step, which then has been averaged to create a single vector representation for the whole sequence, similar to the siamese training approach. This model performed significantly worse than the base model, showing that the additional layer needs longer training time since it is trained from scratch. Also, since it is trained on only one task, its vector representations may contain fewer features than the models trained on multiple tasks (see section 3.3.4). The siamese models performed a little worse than the base BERT model, although the fine-tuning had a bigger effect. For both model types, the variant trained to give two vector representations into a dedicated classifier performed worst, which was expected: Similar to the BERT model with the additional layer, the classifier is trained from scratch and has to learn which features indicate a positive pair. Two things slow training down especially: First, when the model is given two identical vectors, the similarity score is not 1 by definition, as is the case for the models trained using vector similarity but has to be learned explicitly. Second, vectors are given into the model as inputs (A, B), while input pair (B, A) would be an unknown pair. The model performing slightly better than the other models is a siamese model, which has been fine-tuned explicitly on giving a high vector similarity to overlapping sentence pairs. During fine-tuning, when selecting positive pairs, half of the positive pairs were coherent sentences from the corpus. The other half were overlapping parts from the very same sentence.

Figure 4.7 shows the frequency of ranks of the true-positive pairs for the BERT model as well as the best-performing model trained with the siamese training approach. The figure shows, similarly to table 4.15, that both models perform com-

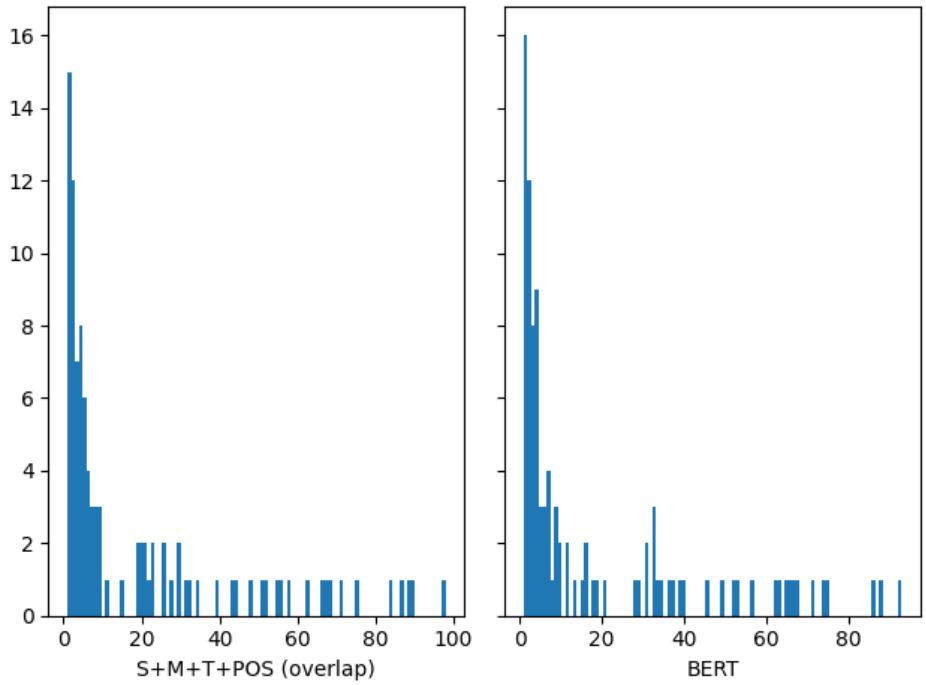


Figure 4.7: The ranking frequency of the true-positive examples of the dataset using two different models.

Type	Count	BERT	S+M+T+POS (overlap)
All	100	0.7524	0.7656
Insertion	30	0.7230	0.7160
Deletion	41	0.7221	0.7327
Subst.	29	0.8873	0.8801
Subst. (semantic)	15	0.5008	0.6965
Subst. (gram./ortho.)	41	0.6088	0.5847
Changed word order	23	0.6532	0.6756
Repitition	11	0.7763	0.7889

Table 4.16: Normalized discounted cumulative gain for different reference types.

parably well. For both model types, most of the true-positive pairs are placed at the beginning of the result lists. For example, for the Shakespeare quote *we will not carry coals*, both models ranked the reference *you must carry no coals* first, with all other true-positives being ranked directly after. Another quote, for which the true-positive were ranked very high by both model types, was the quote *springes to catch woodcocks*. Also, the true-positive pairs for the Shakespeare quote *hell itself should gape* were ranked high. However, the reference *therefore gapeth hell and openeth* was ranked in the middle of the result list for both models, showing that a change in word order has a significant impact on the similarity score. The true-positive pairs for the short quote *planets strike* were ranked low by both models. For example, the BERT model ranked the reference *error and be planet struck* on position 74, showing that short quotes are hard to find in long n-grams. Another quote for which the models performed badly was the quote *All the world's a stage*. One very highly ranked false-positive is the n-gram *all the rest is nothing*. That n-gram has the sentence structure (*All the X is Y*), showing that true-positives for a quote with very common structures are likely to be buried in a list of false-positives.

Table 4.16 shows the normalized DCG for the different reference types. The results show the models perform differently on the different reference types, although both models show similar patterns. The models performed the best on the reference type for which there are simple substitutions in the reference compared to the original quote. An example for this is the quote *to be or not to be* which is referenced as to marry or not to marry.. This result repeats the findings from section 3.3.3: When models are to be used for vector similarity on sentence basis, most models put the highest emphasis on the sentence structure. The fact that references with a changed word order perform significantly worse than average further proves this statement. The models' performances for the references of types insertion and deletion are relatively close because insertion and deletion are related: For each n-gram pair, there is one n-gram shorter than the other one.

4.3.4 Conclusion

The search for reused passages of Shakespeare's texts, at first sight, might look like a standard task for existing text similarity detection approaches. However, finding pairs reused in the sense of an intertextual reference can be rather challenging, as automated searches for these are bound to produce many false positives. Given the different kinds of intertextual references, in order to get an exhaustive result list of possible references, multiple references have to be used. The usage of vector representations was proven to be a suited approach. However, other approaches may be more suitable for finding very short or very abstract references.

This section also presented a multiple-step process to tackle the problem of finding intertextual references at sentence and sub-sentence level. It was found that a critical element is the pre-filtering of potential quote candidate sentences to reduce the number of false positives (as compared to previous studies by [20] and [82]). Without this pre-filtering, the number of pairs that need to be compared would have been extremely high. In the second filtering step, potential pairs were compared by their vector representation using a neural network trained without supervision. A third step was introduced, which builds on annotated n-gram pairs. These annotations are used to train a model that creates a different vector representation such that similarity scores of some pairs with similar or equal structure are low.

This thesis raised the question of how a model's pre-training task affects its performance on a transfer learning task (see section 1.2, research question 1). The BERT model was trained on the masked language model task and the next sentence prediction task. Also, the model contains multiple layers and was trained for four days. Using these time and data resources lead to the model creating outputs with rich features, resulting in good results for transfer tasks like pairing n-grams using their embedding 4.3.1.6. However, fine-tuning that model to this specific proved to be difficult: The last layer's hidden state was not explicitly trained to be used for vector similarity. Fine-tuning the model on different data using the same tasks with which it was trained before does not improve the model's performance on the new task (see section 4.3.3). The siamese model, which has been explicitly trained to create a vector representation for an input sequence, performs worse than the basic BERT model. However, fine-tuning this model to data from the task's domain improved the model's performance. The model performed even better when the training also included positive pairs consisting of two pairs from the same sentence. Since the model is to be used to pair n-grams with token overlap, using this kind of target during training improved the model's performance. The different results of the different model types show, on the one hand, that the BERT model consists of rich features, which, on the other hand, are hard to adjust because of the model's size. The slim siamese models contain a little less rich features, but they are easier to adjust to specific tasks or specific domains (see section 1.2, research question 1). Section 4.3.2 further shows that models result in better performance when they are trained with a task closer to how they will be used: The model was trained with

overlapping sentence pairs, but most of them were selected negative pairs from one of the result lists. The small size of the model made it easy to get these improved results.

Section 4.3.2 showed that training a model, with a selection of false-positives and with non-symmetrical encoders, can lead to significant improvements, which is to be expected since that training task is closest to the way how the model is to be used. Also, the fact that, in section 4.3.3, the siamese models performed comparably well compared to the BERT model shows that a large pre-trained model may not be needed for this case. Furthermore, if a model is fine-tuned with a task that precisely fits the given use case, a smaller model can adjust faster to the new data.

Chapter 5

Conclusion

5.1 Conclusion

Throughout this thesis, it has been shown on multiple occasions that using pre-trained models on transfer learning tasks leads to better results compared to using models that have been trained from scratch:

- Section 4.1 presented a study of classifying tweets of different authors. The authors had to be classified by whether they are a bot, male or female. Given the immense amount of data created on Twitter every day, the amount of labeled data was tiny. The used models to tackle this task had been pre-trained on a set of news articles (see section 3.3). Using that language knowledge learned from those articles, it was possible to classify the author better than a model that had to learn about tweets from scratch (see section 4.1.4).

- When one is to classify images in the field of medicine, the type of model to develop depends on the available data. In an ideal case, each image would have an exact label. However, as was the case for the study in section 4.3.4, the labels were not per image but per set of images. This made it impossible to classify single images. A whole set of images had to be classified at once. Furthermore, since some images had somewhat contradicting labels, the classification results got worse. Additionally, in the field of medicine, the amount of data is small, the amount of labeled data even smaller. This creates a need for pre-trained models. These models are pre-trained on a different set of data and then transferred to another task (see section 2.1.2).

Since there are several ways to pre-train models, several experiments have been executed to evaluate the impact of those choices considering data-selection and architecture types. When pre-training a model on data, the chosen training target is an auxiliary target for the model, meaning that the model most likely is not to be evaluated on that specific training target. The actual interesting part of the model is one of its intermediate states. Since that internal state cannot be trained explicitly,

the process of pre-training neural networks is defined as being unsupervised (see section 2.1.5). When an autoencoder is trained to reconstruct images, the reconstructed images' quality only gives a hint about the model's performance on future tasks. In another example, a model can be trained to create similar vector representations to sentence pairs with similar topics. However, in a transfer learning task, that model will have to solve a different problem but can use the pre-learned knowledge about words and sentence structures.

The first approach presented was Boltzman Machines (see section 2.2.1). Here, a model was trained with contrastive divergence learning to create a hidden representation of input samples. Here, the training target is to minimize the so-called *energy* of the model. That approach had the disadvantages of needing three steps while at the same time only being able to train one layer.

Another typical choice of pre-training target is the reconstruction of data. This is done when dealing with all kinds of data:

- When training word embeddings, the Word2Vec model [78] was trained to reconstruct the context given one word (see section 3.1).
- In the context of image data, the encoding and full decoding of images is called an auto-encoder (see section 3.2).
- In the context of sentences, it is possible to pass a sentence token-wise into a recurrent layer and have a next layer reconstruct that sentence or generate a new one (see section 3.3.1).

These reconstructing models all contain an encoding and a decoding part. Mostly, the encoding part is the part that is re-used and transferred onto other tasks. In those cases, the decoding part is only needed to solve the auxiliary training target. After pre-training, the decoder is discarded, which means that many weights are trained, although they are not needed later on. Also, having more layers in a neural network leads to smaller gradients, and since the decoder is the final part of an autoencoder, the encoder only receives small gradients since there are additional layers between the encoder and the calculation of the model's loss.

Siamese models are models that are pre-trained using an auxiliary target as the previously mentioned models but which only have an encoder part and no decoder part (see section 2.2.5). When training these kinds of models, the auxiliary training target is to create a vector representation for each input sample. The input samples' vector representations are then compared with one another. The goal is to have small distances between vector pairs coming from so-called positive pairs and to have large distances from so-called negative pairs. The selection of positive and negative pairs defines what kind of vector representations are learned. In the context of image data, a positive pair can contain two images from the same class, whereas a negative pair can contain two images of different classes. In that case, the vector representation will contain features such that images from the same class share the same features.

Section 1.2 presented several research questions regarding pre-training that have been investigated in this thesis.

1. What are the effects of different hyper-parameter decisions for the siamese pre-training approach? It has been shown that training a siamese model with positive and negative pairs can create vector representations that can be used for transfer learning tasks. At first, this may seem surprising because compared to autoencoders, the training target contains contradicting information (see section 2.1.6): For example, if pairs are chosen randomly, a pair with very similar features may be chosen as a negative pair. In the case of sentence data, pairs can be created from documents with positive pairs, including sentences from the same document, and negative pairs, including sentences from different documents. In theory, pairs from different documents might contain the same topic but are considered negative. Also, a positive pair from the same document can contain different topics. However, it was shown that these issues do not affect the outcome negatively in the long run. It was shown that it is even possible to train a model using only negative pairs (see section 3.1.4).

Apart from deciding on the training targets and creating pairs, it also has to be decided what kind of distance function to use. It was shown that that choice affects the vector representation significantly. For example, section 3.1.4 showed that models trained with cosine distance trained the fastest. This is because cosine distance calculates the angle between two vectors and is not affected by large values. Thus, the model only has to change the values of features slightly. On the other hand, large values affect euclidian distance, resulting in a need for longer training. When using Jaccard index, it has been shown that the resulting features can be used as binary values. However, that binarization leads to a smaller expressiveness of a model.

It was also shown that models perform better when they have been pre-trained for a specific similarity function. This is the case when dealing with the reduction of sequences of vector representation to a single vector representation. For example, the word embeddings in section 3.1.4 have been averaged to create a single vector representation, although not having been trained for that method. The effects were that the embeddings trained with cosine distance performed poorly: Those vectors are centered around the value 0, whereas the two values -0.1 and 0.1 represent a significant change since they result in the vector pointing in a different direction. Word embeddings trained with other distance functions were not affected by that problem since their vectors depend on absolute values.

2. How does the number and type of pre-training targets affect a model's performance on transfer learning tasks? The effect of training targets was shown in section 3.2.2: Here, images with two features have been created. A line was placed either horizontally or vertically and was either red, green, or blue. The siamese model was trained with pairs being only determined by the bars' direction.

When the image encodings have been given into a decoder such that the decoder reconstructs the original input image, the line's direction could be reconstructed perfectly. However, the decoder could not choose the correct color since that information was not included in the siamese image's vector representation. In the case of learning vector representations for sentences by simply pairing sentences, it may be enough to only lookout for a few keywords in the input sentence to know which sentence pairs are positive or negative. For example, if a training corpus contains news articles about politics and one article about football, the sentence's vector representation for the article dealing with football may contain only one feature. That feature could represent the fact that a word dealing with football was used. If that model were to be transferred to the task of classifying different football articles, its pre-learned features would not be useful.

It was further shown that the richness of features in a vector representation affects the model's performance on transfer learning tasks (see section 3.3.4). The more features of all possible features in a vector representation were used by a siamese model, the better the model was performing on transfer learning tasks. Additional training tasks were added during training to have a model create richer features. In the context of sentences, instead of only creating a vector representation for sentence pairs, the model had to create a vector representation for each sequence's token. That approach resulted in richer vector representations and better performance on transfer learning tasks while then being comparable to the performance of the popular BERT model [32].

When using the models for similarity tasks, models that have been trained to explicitly create vector representations that are then compared using a distance function performed better on such tasks than models where the vector representation was only an internal state. This phenomenon was shown in section 3.1.4 when dealing word similarities as well as section 4.3.1.6 when comparing sentences. For the task of finding similar n-grams, the model that was trained with positive pairs that consisted of overlapping sentence parts performed better than other models when being used for exactly that task.

3. What is the relation between a slim encoder like a small siamese model and a large general-purpose model like BERT? The BERT model is a model that consists of many layers, was trained for several days, and on a very huge dataset. Throughout this thesis, it was shown that the learned features of that model are very rich and lead to good results in many tasks (see sections 3.3.4 and 4.3.3). However, adjusting that model onto specific tasks proved to be difficult because of the model's size. In section 4.1, several models have been fine-tuned for a specific classification task. The slim siamese models achieved good results after one hour of training, while the BERT model only achieved comparable results after having been fine-tuned for 24 hours. Similarly, the BERT model achieved the best results for the evaluation on the small gold standard quotes out-of-the-box (see section 4.3.3). However, several fine-tuning methods did not improve the performance on

that dataset. On the other hand, the small siamese model performed a little worse but could be fine-tuned to perform a little better than the BERT model.

Consequently, it is still encouraged to always try the BERT model's performance on any NLP task. If the task includes a classification problem, meaning a classifier has to be added to the existing layers, the BERT model will probably always perform best because the learned features can be used. However, fine-tuning the model for a similarity task proved to be difficult since the last layer's output was not trained to be used for similarity. Thus, when fine-tuning a pre-trained NLP model requires the inclusion of a specific pre-training task, it is recommended to use a smaller model since it can easier adapt to a specific task.

5.2 Summary and Outlook

Section 1.1 addressed the problem of large models which have been trained using large resources, covering hardware, data, and time. This trend in machine learning is called *Red AI* [116] and creates the problem that researchers without these resources cannot create comparable results. *Green AI*, in contrast, focuses on efficiency. This includes creating comparable results using fewer resources. The positive effect of pre-trained models was shown throughout this thesis: Using a pre-trained model results in faster training time since a model does not need to learn features from scratch. However, a model's performance is affected by how it was pre-trained, including the used pre-training task and data. A central finding is that a relatively small model can compete on different tasks with large pre-trained models given the correct pre-training target. This shows the effectiveness of the siamese pre-training approach as well as that putting thought into a model's architecture can be as valuable as having expensive hardware.

Consequently, researchers are motivated to create their own pre-trained models for data domains, for which there are no existing pre-trained models. In this thesis, the domains of textual and visual data were used as exemplary use cases. Future work has to focus on data from other domains, like audio data, to show further the general applicability of the presented concept in this thesis.

Bibliography

[1] Placing search in context: The concept revisited. ACM Trans. Inf. Syst. 20(1), 116–131 (2002), https://doi.org/10.1145/503104.503110

[2] Agrawala, A.: Learning with a probabilistic teacher. IEEE Transactions on Information Theory 16(4), 373–379 (1970)

[3] Akram, M.U., Tariq, A., Khan, S.A., Javed, M.Y.: Automated detection of exudates and macula for grading of diabetic macular edema. Computer Methods and Programs in Biomedicine 114(2), 141 – 152 (2014), http://www.sciencedirect.com/science/article/pii/S0169260714000224

[4] Alfonseca, E., Filippova, K., Delort, J.Y., Garrido, G.: Pattern learning for relation extraction with a hierarchical topic model. In: Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). pp. 54–59. Association for Computational Linguistics, Jeju Island, Korea (2012)

[5] Allen, G.: Intertextuality. Psychology Press (2000)

[6] Alqahtani, A., Xie, X., Deng, J., Jones, M.W.: A deep convolutional auto-encoder with embedded clustering. In: 2018 IEEE International Conference on Image Processing (ICIP). pp. 4058–4062 (2018)

[7] Bacciu, A., La Morgia, M., Mei, A., Nemmi, E., Neri, V., Stefa, J.: Bot and Gender Detection of Twitter Accounts Using Distortion and LSA. In: Cappellato, L., Ferro, N., Losada, D., Müller, H. (eds.) CLEF 2019 Labs and Workshops, Notebook Papers. CEUR-WS.org (2019), http://ceur-ws.org/Vol-2380/

[8] Bazerman, C.: Intertextuality: How Texts Rely on Other Texts 1, pp. 83–96 (2004)

[9] Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H.: Greedy layer-wise training of deep networks. In: Proceedings of the 19th International Conference on Neural Information Processing Systems. pp. 153–160.

NIPS'06, MIT Press, Cambridge, MA, USA (2006), http://dl.acm.org/citation.cfm?id=2976456.2976476

[10] Biber, D.: Co-occurrence patterns among collocations: A tool for corpus-based lexical knowledge acquisition. Comput. Linguist. 19(3), 531–538 (1993)

[11] Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. CoRR abs/1607.04606 (2016), http://arxiv.org/abs/1607.04606

[12] Bordag, S.: A comparison of co-occurrence and similarity measures as simulations of context. In: In 9th CICLing. pp. 52–63 (2008)

[13] Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., Shah, R.: Signature verification using a "siamese" time delay neural network. In: Proceedings of the 6th International Conference on Neural Information Processing Systems. pp. 737–744. NIPS'93, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1993), http://dl.acm.org/citation.cfm?id=2987189.2987282

[14] Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D.M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., Amodei, D.: Language models are few-shot learners (2020)

[15] Bruni, E., Tran, N.K., Baroni, M.: Multimodal distributional semantics. J. Artif. Int. Res. 49(1), 1–47 (2014)

[16] Bryan, M., Burghardt, M., Molz, J.: A computational expedition into the undiscovered country - evaluating neural networks for the identification of hamlet text reuse. In: CHR (2020)

[17] Bryan, M., Heyer, G., Philipp, N., Rehak, M., Wiedemann, P.: Convolutional attention on images for locating macular edema. In: Zheng, Y., Williams, B.M., Chen, K. (eds.) Medical Image Understanding and Analysis. pp. 391–398. Springer International Publishing, Cham (2020)

[18] Bryan, M., Philipp, J.: Unsupervised pretraining for text classification using siamese transfer learning. In: Cappellato, L., Ferro, N., Losada, D., Müller, H. (eds.) CLEF 2019 Labs and Workshops, Notebook Papers. CEUR-WS.org (2019), http://ceur-ws.org/Vol-2380/

[19] Bunescu, R., Mooney, R.: Learning to extract relations from the web using minimal supervision. In: ACL 2007 - Proceedings of the 45th Annual

Meeting of the Association for Computational Linguistics. pp. 576–583 (2007)

[20] Burghardt, M., Meyer, S., Schmidtbauer, S., Molz, J.: "The Bard meets the Doctor" – Computergestützte Identifikation intertextueller Shakespearebezüge in der Science Fiction-Serie Dr. Who. Book of Abstracts, DHd (2019)

[21] Carreira-Perpinan, M., Hinton, G.: On contrastive divergence learning (2005)

[22] Caruana, R.: Multitask learning. Mach. Learn. 28, 41–75 (1997), http://portal.acm.org/citation.cfm?id=262872&dl=GUIDE&coll=GUIDE&CFID=69344422&CFTOKEN=94303924

[23] Chapelle, O., Schlkopf, B., Zien, A.: Semi-Supervised Learning. The MIT Press, 1st edn. (2010)

[24] Chechik, G., Sharma, V., Shalit, U., Bengio, S.: Large scale online learning of image similarity through ranking. J. Mach. Learn. Res. 11, 1109–1135 (2010)

[25] Chen, Q., Zhu, X., Ling, Z., Wei, S., Jiang, H.: Enhancing and combining sequential and tree LSTM for natural language inference. CoRR abs/1609.06038 (2016), http://arxiv.org/abs/1609.06038

[26] Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder–decoder for statistical machine translation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 1724–1734. Association for Computational Linguistics, Doha, Qatar (2014), https://www.aclweb.org/anthology/D14-1179

[27] Chopra, S., Hadsell, R., Lecun, Y.: Learning a similarity metric discriminatively, with application to face verification. vol. 1, pp. 539– 546 vol. 1 (2005)

[28] Collobert, R., Weston, J.: A unified architecture for natural language processing: Deep neural networks with multitask learning. In: Proceedings of the 25th International Conference on Machine Learning. pp. 160–167. ICML '08, ACM, New York, NY, USA (2008), http://doi.acm.org/10.1145/1390156.1390177

[29] Conneau, A., Kiela, D.: SentEval: An evaluation toolkit for universal sentence representations. In: Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018).

European Language Resources Association (ELRA), Miyazaki, Japan (May 2018), https://www.aclweb.org/anthology/L18-1269

[30] Daelemans, W., Kestemont, M., Manjavancas, E., Potthast, M., Rangel, F., Rosso, P., Specht, G., Stamatatos, E., Stein, B., Tschuggnall, M., Wiegmann, M., Zangerle, E.: Overview of PAN 2019: Author Profiling, Celebrity Profiling, Cross-domain Authorship Attribution and Style Change Detection. In: Crestani, F., Braschler, M., Savoy, J., Rauber, A., Müller, H., Losada, D., Heinatz, G., Cappellato, L., Ferro, N. (eds.) Proceedings of the Tenth International Conference of the CLEF Association (CLEF 2019). Springer (2019)

[31] Dai, J., He, K., Sun, J.: Instance-aware semantic segmentation via multi-task network cascades. CoRR abs/1512.04412 (2015), http://arxiv.org/abs/1512.04412

[32] Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. CoRR abs/1810.04805 (2018), http://arxiv.org/abs/1810.04805

[33] Dolan, B., Quirk, C., Brockett, C.: Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. Proceedings of the 20th International Conference on Computational Linguistics (2004)

[34] Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., Darrell, T.: Decaf: A deep convolutional activation feature for generic visual recognition. In: Xing, E.P., Jebara, T. (eds.) Proceedings of the 31st International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 32, pp. 647–655. PMLR, Bejing, China (2014), http://proceedings.mlr.press/v32/donahue14.html

[35] Erhan, D., Bengio, Y., Courville, A., Manzagol, P.A., Vincent, P., Bengio, S.: Why does unsupervised pre-training help deep learning? J. Mach. Learn. Res. 11, 625–660 (2010), http://dl.acm.org/citation.cfm?id=1756006.1756025

[36] Fischer, A., Igel, C.: An introduction to restricted boltzmann machines. In: in Proceedings of the 17th Iberoamerican Congress on Pattern Recognition, Buenos Aires. pp. 14–36 (2012)

[37] Fralick, S.: Learning to recognize patterns without a teacher. IEEE Trans. Inf. Theor. 13(1), 57–64 (2006), https://doi.org/10.1109/TIT.1967.1053952

[38] F.R.S., K.P.: Liii. on lines and planes of closest fit to systems of points in space. The London, Edinburgh, and Dublin Philosophical Magazine and

Journal of Science 2(11), 559–572 (1901), https://doi.org/10.1080/14786440109462720

[39] Girshick, R.B., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. CoRR abs/1311.2524 (2013), http://arxiv.org/abs/1311.2524

[40] Goldhahn, D., Eckart, T., Quasthoff, U.: Building large monolingual dictionaries at the leipzig corpora collection: From 100 to 200 languages. In: In Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12 (2012)

[41] Gondara, L.: Medical image denoising using convolutional denoising autoencoders. CoRR abs/1608.04667 (2016), http://arxiv.org/abs/1608.04667

[42] Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), http://www.deeplearningbook.org

[43] Gorder, P.: Neural networks show new promise for machine vision. Computing in Science & Engineering 8, 4 – 8 (2006)

[44] Guo, X., Liu, X., Zhu, E., Yin, J.: Deep clustering with convolutional autoencoders. In: Liu, D., Xie, S., Li, Y., Zhao, D., El-Alfy, E.S.M. (eds.) Neural Information Processing. pp. 373–382. Springer International Publishing, Cham (2017)

[45] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR abs/1512.03385 (2015), http://arxiv.org/abs/1512.03385

[46] He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision. pp. 1026–1034 (2015)

[47] Hernandez, D., Brown, T.B.: Measuring the algorithmic efficiency of neural networks (2020)

[48] Heyer, G., Gerhard, Quasthoff, U., Uwe, Wittig, Thomas: Text mining: Wissensrohstoff Text: Konzepte, Algorithmen, Ergebnisse (2006)

[49] Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. Science 313(5786), 504–507 (2006), http://www.ncbi.nlm.nih.gov/sites/entrez?db=pubmed&uid=16873662&cmd=showdetailview&indexed=google

[50] Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural Comput. 18(7), 1527–1554 (2006), http://dx.doi.org/10.1162/neco.2006.18.7.1527

[51] Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation 9, 1735–80 (1997)

[52] Hu, J., Chen, Y., Yi, Z.: Automated segmentation of macular edema in oct using deep neural networks. Medical Image Analysis 55, 216 – 227 (2019), http://www.sciencedirect.com/science/article/pii/S1361841519300386

[53] Hu, M., Liu, B.: Mining and summarizing customer reviews. In: Proceedings of KDD '04, the ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 168–177. ACM Press (2004), https://www.microsoft.com/en-us/research/publication/mining-and-summarizing-customer-reviews/

[54] Huang, G., Liu, Z., Weinberger, K.Q.: Densely connected convolutional networks. CoRR abs/1608.06993 (2016), http://arxiv.org/abs/1608.06993

[55] Ikae, C., Nath, S., Savoy, J.: UniNE at PAN-CLEF 2019: Bots and Gender Task. In: Cappellato, L., Ferro, N., Losada, D., Müller, H. (eds.) CLEF 2019 Labs and Workshops, Notebook Papers. CEUR-WS.org (2019), http://ceur-ws.org/Vol-2380/

[56] Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. CoRR abs/1502.03167 (2015), http://arxiv.org/abs/1502.03167

[57] Jin, Q., Meng, Z., Sun, C., Wei, L., Su, R.: Ra-unet: A hybrid deep attention-aware network to extract liver and tumor in CT scans. CoRR abs/1811.01328 (2018), http://arxiv.org/abs/1811.01328

[58] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization (2014), http://arxiv.org/abs/1412.6980

[59] Kiros, R., Zhu, Y., Salakhutdinov, R., Zemel, R.S., Torralba, A., Urtasun, R., Fidler, S.: Skip-thought vectors. CoRR abs/1506.06726 (2015), http://arxiv.org/abs/1506.06726

[60] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)

[61] Kroeger, P.R.: Analyzing Grammar: An Introduction. Cambridge University Press (2005)

[62] Kulkarni, A., Pedersen, T.: Senseclusters: Unsupervised clustering and labeling of similar contexts. In: Proceedings of the ACL Interactive Poster and Demonstration Sessions. pp. 105–108 (2005)

[63] Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32. p. II–1188–II–1196. ICML'14, JMLR.org (2014)

[64] LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural Comput. 1(4), 541–551 (1989), https://doi.org/10.1162/neco.1989.1.4.541

[65] Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE 86(11), 2278–2324 (1998)

[66] Li, X., Roth, D.: Learning question classifiers. In: Proceedings of the 19th International Conference on Computational Linguistics - Volume 1. p. 1–7. COLING '02, Association for Computational Linguistics, USA (2002), https://doi.org/10.3115/1072228.1072378

[67] Li, Y.: Deep reinforcement learning. CoRR abs/1810.06339 (2018), http://arxiv.org/abs/1810.06339

[68] Lian, S., Luo, Z., Zhong, Z., Lin, X., Su, S., Li, S.: Attention guided u-net for accurate iris segmentation. Journal of Visual Communication and Image Representation 56, 296 – 304 (2018), http://www.sciencedirect.com/science/article/pii/S1047320318302372

[69] Liu, X., Shen, Y., Duh, K., Gao, J.: Stochastic answer networks for machine reading comprehension. CoRR abs/1712.03556 (2017), http://arxiv.org/abs/1712.03556

[70] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V.: Roberta: A robustly optimized BERT pretraining approach. CoRR abs/1907.11692 (2019), http://arxiv.org/abs/1907.11692

[71] Logeswaran, L., Lee, H.: An efficient framework for learning sentence representations. CoRR abs/1803.02893 (2018), http://arxiv.org/abs/1803.02893

[72] Luong, M., Le, Q.V., Sutskever, I., Vinyals, O., Kaiser, L.: Multi-task sequence to sequence learning. In: 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings (2016), http://arxiv.org/abs/1511.06114

[73] Ma, L., Lu, Z., Shang, L., Li, H.: Multimodal convolutional neural networks for matching image and sentence. In: 2015 IEEE International Conference on Computer Vision (ICCV). pp. 2623–2631 (2015)

[74] Marsland, S.: Machine Learning: An Algorithmic Perspective, Second Edition. Chapman & Hall/CRC, 2nd edn. (2014)

[75] Masci, J., Meier, U., Cireşan, D., Schmidhuber, J.: Stacked convolutional auto-encoders for hierarchical feature extraction. In: Proceedings of the 21th International Conference on Artificial Neural Networks - Volume Part I. pp. 52–59. ICANN'11, Springer-Verlag, Berlin, Heidelberg (2011), http://dl.acm.org/citation.cfm?id=2029556.2029563

[76] Maxwell, J., .R.K.: Shakespeare and Quotation. Cambridge University Press (2018)

[77] Melamud, O., Goldberger, J., Dagan, I.: context2vec: Learning generic context embedding with bidirectional LSTM. In: Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning. pp. 51–61. Association for Computational Linguistics, Berlin, Germany (2016), https://www.aclweb.org/anthology/K16-1006

[78] Mikolov, T., Chen, K., Corrado, G.S., Dean, J.: Efficient estimation of word representations in vector space (2013), http://arxiv.org/abs/1301.3781

[79] Mintz, M., Bills, S., Snow, R., Jurafsky, D.: Distant supervision for relation extraction without labeled data. In: Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP. pp. 1003–1011. Association for Computational Linguistics, Suntec, Singapore (2009), https://www.aclweb.org/anthology/P09-1113

[80] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature 518(7540), 529–533 (2015), http://dx.doi.org/10.1038/nature14236

[81] Mohri, M., Rostamizadeh, A., Talwalkar, A.: Foundations of Machine Learning. Adaptive Computation and Machine Learning, MIT Press, Cambridge, MA, 2 edn. (2018)

[82] Molz, J.: A close and distant reading of shakespearean intertextuality - towards a mixed methods approach for literary studies. Universitätsbibliothek der LMU, München (2020)

[83] Moravcík, M., Schmid, M., Burch, N., Lisý, V., Morrill, D., Bard, N., Davis, T., Waugh, K., Johanson, M., Bowling, M.H.: Deepstack: Expert-level artificial intelligence in no-limit poker. CoRR abs/1701.01724 (2017), http://arxiv.org/abs/1701.01724

[84] Muehlberger, G., Seaward, L., Terras, M., Oliveira, S., Bosch, V., Bryan, M., Colutto, S., Déjean, H., Diem, M., Fiel, S., Gatos, B., Grüning, T., Greinoecker, A., Hackl, G., Haukkovaara, V., Heyer, G., Hirvonen, L., Hodel, T., Jokinen, M., Zagoris, K.: Transforming scholarship in the archives through handwritten text recognition: Transkribus as a case study. Journal of Documentation ahead-of-print (2019)

[85] Murphy, K.P.: Machine learning: A probabilistic perspective. MIT Press, Cambridge, Mass. (2013)

[86] Müller, L., Quasthoff, U., Sumalvico, M.: Corpora of typical sentences. In: LREC 2018, Eleventh International Conference on Language Resources and Evaluation. Miyazaki, Japan (2018)

[87] Neculoiu, P., Versteegh, M., Rotaru, M.: Learning text similarity with siamese recurrent networks. In: Rep4NLP@ACL (2016)

[88] Oktay, O., Schlemper, J., Le Folgoc, L., Lee, M., Heinrich, M., Misawa, K., Mori, K., McDonagh, S., Y Hammerla, N., Kainz, B., Glocker, B., Rueckert, D.: Attention U-Net: Learning Where to Look for the Pancreas. arXiv e-prints arXiv:1804.03999 (2018)

[89] Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Trans. on Knowl. and Data Eng. 22(10), 1345–1359 (2010), http://dx.doi.org/10.1109/TKDE.2009.191

[90] Pang, B., Lee, L.: A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. Computing Research Repository - CORR 271-278, 271–278 (2004)

[91] Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: EMNLP. vol. 14, pp. 1532–1543 (2014), https://nlp.stanford.edu/pubs/glove.pdf

[92] Pershing, S., E.E.A.M.B.O.D.K..G.F.J.D.: Cost-effectiveness of treatment of diabetic macular edema. Annals of internal medicine, 160 pp. 18–29 (2014)

[93] Potthast, M., Gollub, T., Wiegmann, M., Stein, B.: TIRA Integrated Research Architecture. In: Ferro, N., Peters, C. (eds.) Information Retrieval Evaluation in a Changing World - Lessons Learned from 20 Years of CLEF. Springer (2019)

[94] Przybyła, P.: Detecting Bot Accounts on Twitter by Measuring Message Predictability. In: Cappellato, L., Ferro, N., Losada, D., Müller, H. (eds.) CLEF 2019 Labs and Workshops, Notebook Papers. CEUR-WS.org (2019), http://ceur-ws.org/Vol-2380/

[95] Puertas, E., Moreno-Sandoval, L., Plaza-del-Arco, F., Alvarado-Valencia, J., Pomares-Quimbaya, A., Ureña-López, L.: Bots and Gender Profiling on Twitter using Sociolinguistic Features. In: Cappellato, L., Ferro, N., Losada, D., Müller, H. (eds.) CLEF 2019 Labs and Workshops, Notebook Papers. CEUR-WS.org (2019), http://ceur-ws.org/Vol-2380/

[96] Quinn, A.J., Bederson, B.B.: Human computation: A survey and taxonomy of a growing field. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. p. 1403–1412. CHI '11, Association for Computing Machinery, New York, NY, USA (2011), https://doi.org/10.1145/1978942.1979148

[97] Qureshi, M.A., Greene, D.: EVE: explainable vector based embedding technique using wikipedia. CoRR abs/1702.06891 (2017), http://arxiv.org/abs/1702.06891

[98] Ramsundar, B., Kearnes, S., Riley, P., Webster, D., Konerding, D., Pande, V.: Massively Multitask Networks for Drug Discovery. arXiv e-prints (2015)

[99] Rangel, F., Franco-Salvador, M., Rosso, P.: A low dimensionality representation for language variety identification. In: Gelbukh, A. (ed.) Computational Linguistics and Intelligent Text Processing. pp. 156–169. Springer International Publishing, Cham (2018)

[100] Rangel, F., Rosso, P.: Overview of the 7th Author Profiling Task at PAN 2019: Bots and Gender Profiling. In: Cappellato, L., Ferro, N., Losada, D., Müller, H. (eds.) CLEF 2019 Labs and Workshops, Notebook Papers. CEUR-WS.org (2019)

[101] Ranzato, M.: Unsupervised Learning of Feature Hierarchies. Ph.D. thesis, New York, NY, USA (2009)

[102] Rasti, R., Rabbani, H., Mehridehnavi, A., Hajizadeh, F.: Macular oct classification using a multi-scale convolutional neural network ensemble. IEEE Transactions on Medical Imaging 37(4), 1024–1034 (2018)

[103] Ratner, A., Bach, S.H., Ehrenberg, H.R., Fries, J.A., Wu, S., Ré, C.: Snorkel: Rapid training data creation with weak supervision. CoRR abs/1711.10160 (2017), http://arxiv.org/abs/1711.10160

[104] Reddi, S.J., Kale, S., Kumar, S.: On the convergence of adam and beyond. CoRR abs/1904.09237 (2019), http://arxiv.org/abs/1904.09237

[105] Redmon, J., Divvala, S.K., Girshick, R.B., Farhadi, A.: You only look once: Unified, real-time object detection. CoRR abs/1506.02640 (2015), http://arxiv.org/abs/1506.02640

[106] Redmon, J., Farhadi, A.: YOLO9000: better, faster, stronger. CoRR abs/1612.08242 (2016), http://arxiv.org/abs/1612.08242

[107] Reimers, N., Gurevych, I.: Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). pp. 3982–3992. Association for Computational Linguistics, Hong Kong, China (2019), https://www.aclweb.org/anthology/D19-1410

[108] Rubenstein, H., Goodenough, J.: Contextual correlates of synonymy. Commun. ACM 8, 627–633 (1965)

[109] Ruder, S.: An overview of multi-task learning in deep neural networks. CoRR abs/1706.05098 (2017), http://arxiv.org/abs/1706.05098

[110] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) 115(3), 211–252 (2015)

[111] Rzyman, A.: The Intertextuality of Terry Pratchett's Discworld as a Major Challenge for the Translator. Cambridge Scholars Publishing (2017)

[112] Sahlsten, J., J.J.K.J.e.a.: Deep learning fundus image analysis for diabetic retinopathy and macular edema grading. Sci Rep 9, 10750 (2019)

[113] Salakhutdinov, R., Hinton, G.: Semantic hashing. Int. J. Approx. Reasoning 50(7), 969–978 (2009), http://dx.doi.org/10.1016/j.ijar.2008.11.006

[114] Saussure, F.D.: Grundfragen der allgemeinen Sprachwissenschaft. de Gruyter (1966)

[115] Schroff, F., Kalenichenko, D., Philbin, J.: Facenet: A unified embedding for face recognition and clustering. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 815–823 (2015)

[116] Schwartz, R., Dodge, J., Smith, N.A., Etzioni, O.: Green AI (2019), http://arxiv.org/abs/1907.10597

[117] Scudder, III, H.: Probability of error of some adaptive pattern-recognition machines. IEEE Trans. Inf. Theor. 11(3), 363–371 (2006), http://dx.doi.org/10.1109/TIT.1965.1053799

[118] Searle, J.R.: Chomsky's revolution in linguistics (1974)

[119] Settles, B.: Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison (2009), http://axon.cs.byu.edu/~martinez/classes/778/Papers/settles.activelearning.pdf

[120] Sharif Razavian, A., Azizpour, H., Sullivan, J., Carlsson, S.: Cnn features off-the-shelf: An astounding baseline for recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops (2014)

[121] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of Go with deep neural networks and tree search. Nature 529(7587), 484–489 (2016)

[122] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015), http://arxiv.org/abs/1409.1556

[123] Snoek, J., Adams, R., Larochelle, H.: On nonparametric guidance for learning autoencoder representations. In: Lawrence, N.D., Girolami, M. (eds.) Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 22, pp. 1073–1080. PMLR, La Palma, Canary Islands (2012), http://proceedings.mlr.press/v22/snoek12.html

[124] Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C.D., Ng, A., Potts, C.: Recursive deep models for semantic compositionality over a sentiment treebank. In: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing. pp. 1631–1642. Association for Computational Linguistics, Seattle, Washington, USA (2013), https://www.aclweb.org/anthology/D13-1170

[125] Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 27, pp. 3104–3112. Curran Associates, Inc. (2014), http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf

[126] Sutton, R.S., Barto, A.G.: Introduction to Reinforcement Learning. MIT Press, Cambridge, MA, USA, 1st edn. (1998)

[127] Tan, Z., Wang, M., Xie, J., Chen, Y., Shi, X.: Deep semantic role labeling with self-attention. CoRR abs/1712.01586 (2017), http://arxiv.org/abs/1712.01586

[128] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L.u., Polosukhin, I.: Attention is all you need. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30, pp. 5998–6008. Curran Associates, Inc. (2017), http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

[129] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.A.: Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. J. Mach. Learn. Res. 11, 3371–3408 (2010), http://dl.acm.org/citation.cfm?id=1756006.1953039

[130] Vos, T., Abajobir, A., Cristiana, A., Abbas, K., Abate, K., Abd-Allah, F., Abdulle, A., Abebo, T., Abera, S., Aboyans, V., Abu-Raddad, L., Ackerman, I., Adamu, A., Adetokunboh, O., Afarideh, M., Afshin, A., Agarwal, S., Aggarwal, R., Agrawal, A., Murray, C.: Global, regional, and national incidence, prevalence, and years lived with disability for 328 diseases and injuries for 195 countries, 1990–2016: a systematic analysis for the global burden of disease study 2016. The Lancet 390, 1211–1259 (2017)

[131] Wang, S.I., Manning, C.D.: Baselines and bigrams: Simple, good sentiment and topic classification. In: Proceedings of the ACL. pp. 90–94 (2012)

[132] Wiebe, J., Wilson, T., Cardie, C.: Annotating expressions of opinions and emotions in language. Language Resources and Evaluation (formerly Computers and the Humanities) 39, 164–210 (2005)

[133] Wolf, L., Hanani, Y., Bar, K., Dershowitz, N.: Joint word2vec networks for bilingual semantic representations. Int. J. Comput. Linguistics Appl. 5, 27–42 (2014)

[134] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Brew, J.: Huggingface's transformers: State-of-the-art natural language processing. ArXiv abs/1910.03771 (2019)

[135] Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. CoRR abs/1708.07747 (2017), http://arxiv.org/abs/1708.07747

[136] Yang, X., Deng, C., Zheng, F., Yan, J., Liu, W.: Deep spectral clustering using dual autoencoder network. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)

[137] Yuen, M., King, I., Leung, K.: A survey of crowdsourcing systems. In: 2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing. pp. 766–773 (2011)

[138] Zeiler, M.D., Fergus, R.: Visualizing and Understanding Convolutional Networks. ArXiv e-prints (2013), https://arxiv.org/abs/1311.2901v3

[139] Zhang, Z.Y., Zhao, P., Jiang, Y., Zhou, Z.H.: Learning from incomplete and inaccurate supervision. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 1017–1025. KDD '19, ACM, New York, NY, USA (2019), http://doi.acm.org/10.1145/3292500.3330902

[140] Zhou, Z.H.: A brief introduction to weakly supervised learning. National Science Review 5(1), 44–53 (2017), https://doi.org/10.1093/nsr/nwx106

[141] Zhou, Z.H., Li, M.: Semi-supervised learning by disagreement. vol. 24, pp. 415–439 (2010), https://doi.org/10.1007/s10115-009-0209-z

[142] Zhuang, J., Tang, T., Ding, Y., Tatikonda, S.C., Dvornek, N., Papademetris, X., Duncan, J.: Adabelief optimizer: Adapting stepsizes by the belief in observed gradients (2020), https://proceedings.neurips.cc/paper/2020/file/d9d4f495e875a2e075a1a4a6e1b9770f-Paper.pdf

Selbstständigkeitserklärung

Hiermit erkläre ich, die vorliegende Dissertation selbständig und ohne unzulässige fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angeführten Quellen und Hilfsmittel benutzt und sämtliche Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, und alle Angaben, die auf mündlichen Auskünften beruhen, als solche kenntlich gemacht. Ebenfalls sind alle von anderen Personen bereitgestellten Materialien oder erbrachten Dienstleistungen als solche gekennzeichnet.

_____
(Ort, Datum)

_____
(Unterschrift)