DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

Transfer and Multitask Learning for Aspect-Based Sentiment Analysis using the Google Transformer Architecture

Felix Schober

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

Transfer and Multitask Learning for Aspect-Based Sentiment Analysis using the Google Transformer Architecture

Transfer- und Multitask-Lernen für aspektbasierte Sentimentanalyse mit der Transformer-Architektur von Google

Author:	Felix Schober
Supervisor:	PD Dr. Georg Groh
Advisor:	Gerhard Hagerer M.Sc.
Submission Date:	15.05.2019

I confirm that this master's thesis in information systems is my own work and I have documented all sources and material used.

Munich, 15.05.2019 Felix Schober

Acknowledgments

Abstract

Contents

1. Introduction

1.1. Motivation

In the recent past researches were able to advance machine learning models for computer vision further and further. These models have reached a state where they outperform humans on certain tasks. Andrej Karpathy, director of AI at Tesla, conducted an experiment at Standford where he trained himself to classify images from ImageNet [31]. After a lot of training he achieved an error score of 6.8% which, at the time, was better than predictions from GoogleLeNet. However, deep neural networks have now reached a level which is significantly below the threshold of 6.8% Karpathy achieved. The best submission for the ImageNet Large Scale Visual Recognition Competition achieve an error rate of only 2.25% [**Hu2018**].

In the past years, researches have shifted their attention to the field of Natural Language Processing (NLP). This trend has been triggered by the development of efficient word embeddings [46]. Word embeddings are powerful mechanisms which are able transform words into meaningful vector representations. This technique allows researches to use deeper and deeper neural networks on tasks like Named-entity recognition (NER), Part-of-Speech (POS) or sentiment analysis. Contrary to computer vision, however, we still have a long way to go until models are able to outperform humans on Natural Language Processing (NLP) tasks.

One of the most useful NLP tasks is sentiment analysis. The goal of sentiment analysis is to predict whether a sequence of words carries a negative, neutral or positive emotion. Automated sentiment analysis can be applied to huge amounts of data to form an opinion landscape or even be used to create sales forecasts based on customer reviews [69].

Aspect Based Sentiment Analysis

As more and more people buy products and services online the significance of online reviews increases. Platforms like amazon encourage customers to write reviews and rate products. However, popular products can receive hundreds or thousands of reviews. It is often not feasible to read all reviews. Yet, reading only several reviews may impose a bias depending on which reviews the customer reads.

To make the decision easier platforms often provide a mean rating. The problem is, that certain aspects may be important for some people but not for others. Consider this review for console game on amazon[1] for instance:

"Decent game, but basically the same exact thing as the normal Mario Kart 8."

This review reduces the overall score for the game even though this aspect might not be relevant for customers who did not play the previous game before.
Another great example are hotel reviews. One guest might care if the room contains a minibar while for another potential guest this aspect does not matter.

Aspect Based Sentiment Analysis (ABSA) is an extension to sentiment analysis. It tries to automate the task of detecting sentiment for a given aspect. Applied on customer reviews this solves a lot of issues for consumers but it has even further implications.

Capturing pure sentiment is often not enough to obtain an accurate sentiment classification. For instance, the sentence

"I like the taste of organic apples but I'm not sure they are worth the extra money for me."

contains a positive sentiment for the aspect "*taste*" and a negative sentiment for "*price*". This is relatively easy (at least for humans). At the same time it is challenging to categorize the overall sentiment of the sentence. As a matter of fact, researched demonstrated that "interleaving" aspect and sentiment leads to a better understanding of the underlying sentence [34].

1.2. Challenges

Sentiment classification or even Aspect Based Sentiment Analysis (ABSA) is one of the most challenging tasks in the field of natural language processing [50]. For once, understanding the subtle differences in language is a challenging task. A text can contain humor, negations or words which have different meanings depending on the context. The following review from amazon[2] demonstrates a positive review for a watertight pouch for a kindle eBook reader.

[1] `https://www.amazon.com/Mario-Kart-8-Deluxe-Nintendo-Switch/dp/B01N1037CV`

[2] `https://www.amazon.co.uk/gp/customer-reviews/REII5J393XCJW/ref=cm_cr_dp_d_rvw_ttl?ie=UTF8&ASIN=B007LJ4PPO`

"Got this for the Mother in-law for bath time, hoping it'd be crap, her kindle would slip out and electrocute her. So far, this bloody thing is staying in one piece. Great for waterproof kindling, crap for murder."

This review is undeniably positive. However most ABSA- or sentiment analysis models would likely label this review as negative[3].
Context is also important. A high price denotes a negative sentiment while a high quality indicates a positive sentiment.

Getting data for sentiment analysis in general and aspect based sentiment analysis in particular is a another aggravating problem. It is very time consuming to annotate text data with aspect based sentiment. As a result, models which are trained on ABSA tasks have to be trained with less data.

1.3. Contributions

This thesis proposes a novel deep learning model for aspect based sentiment analysis. This model is based of two proven architectures for natural language processing. Our proposed model ABSA-Transformer (ABSA-T) uses the Google transformer [78] as well as the idea of aspect heads [67]. This allows the model to predict multiple aspects and sentiments end-to-end.

We also experiment with hyperparameter optimization and the implications of optimizing more than just model parameters.
Furthermore, we explore multitask learning as a method to augment limited training data. Additionally, this thesis also examines the impact of transfer learning on the model performance. For this task we also built a new topic based sentiment analysis dataset using amazon reviews.

1.4. Outline

This thesis is structured into 7 chapters. Chapter 2 outlines previous work done on sentiment analysis and aspect based sentiment analysis.
Chapter 3 takes a look at the main concepts and the theory behind word representations, the chosen architecture, transfer- and multitask learning concepts as well as the theory behind hyperparameter optimization.
Chapter 4 provides information about the chosen model architecture and how we use transfer- and multitask learning to boost performance.

[3]The stanford sentiment treebank online demo assigns a "very negative" label to this review

Chapter 5 explains the experimental setup including data preprocessing, the datasets and how the models were trained and evaluated.

Results are then accumulated and discussed in chapter 6. Lastly, the results are summarized in chapter 7 and advice for future work is given.

2. Related Work

This chapter will outline some of the past approaches to solve the task of Sentiment Analysis and Aspect Based Sentiment Analysis, as well as more recent approaches to give a general overview of the field. While the first section will deal with the traditional task of sentiment extraction, section 2.2 will discuss linking sentiment to specific aspects.

2.1. Sentiment Analysis

Traditionally, sentiment analysis has been approached by carfully engineering features which were handtuned to a specific work task. Most approaches used either rule based systems [56] or lexicons to find the sentiment polarity of a document. Huettner and Subasic use a word lexicon for semantic categiues and a word affection lexicon with 4000 words. Each word in the lexicon is then assigned one category [27].
Das and Chen use a similar approach where they manually compile a word lexicon. They use those words in combination with scoring functions to classify user generated posts on stock-market message boards as *bullish* (positive for the stockmarket) or *bearish* (negative for the stock market) [14].
Hu and Liu improve upon the tedious task of building a lexicon by using only a small ammount of seed adjectives which is grown by the use of wordNet [26].

Lexicon based approaches are often manually build for a specific task like movie reviews [74, 73] and therefore, generally domain dependet. For instance, sentiment in movie reviews is expressed very differently compared to a product or restaurant review. On the other hand, general sentiment lexicons like SentiWordNet [1] are mostly to general to successfully capture domain specific sentiment.

Tourney uses an unsupervised system which classifies the sentiment of reviews as thumbs up or thumbs down. This is done by computing the closest association of adjectives and adverbs in a sentence. They then comopare the number of positive associations (associations to words like *good* or *romantic*) in a sentnece to the amount of negative associations (words like *horrific*) [77].

In 2012 Pang et. al. compare traditional machine learning techniques (Naive Bayes classifier, maximum entropy classification and Support Vector Machines (SVMs)) with human feature engineered baselines on movie reviews [50]. They still use word lists to classify the polarity of a sentence but they conclude that a sentiment lexicon generated by a machine learning algorithm always outperforms manually created lexicons.

Usually, lexicon approaches used to work in combination with scoring functions where the most trivial ones would just sum up the negative and positive words in a document. The algorithm would classify a document as positive if the sum of positive words is positive. However, there are limitations to this technique since sentences are often tree structured and may contain negated sub-trees. Just counting the positive and negative words in the sentence "*This film doesn't care about cleverness, wit or any other kind of intelligent humor*" would yield in a highly positive sentiment since it does not take the negation into account. Socher et al. introduce a "recursive neural tensor network" [71]. "Recursive neural networks" or "tree-structured neural networks" use the output of a forward pass as the input and some additional information as a new forward pass. By treating each word as a node in a tree they are able to input a subtree into the network, get the output and then use this output and the words on the next layer as the new input until the root node is reached. With this technique they are able to capture contrastive sentences where the first half contains negative sentiment and the second half positive sentiment in addition to negated sentiment.

2.2. Aspect Based Sentiment Analysis

Aspect Based Sentiment Analysis (ABSA) takes Sentiment Analysis one step further. Instead of just predicting sentiment for a sentence of document, ABSA predicts sentiment for a specific aspect. This solves a major dilemma for normal sentiment analysis when a sentence or document contains multiple contradictive sentiments. For instance, the sentence "*The food tasted great but the price was too high*" contains two conflicting sentiment instances (Positive: great food - Negative: High Price). A classifier just modelling sentiment anaylsis would have to label this sentence as beeing neutral whereas an ABSA system could classify the aspect *food* as positive and *price* as negative. In addition ABSA is also much more useful for consumers and companies alike as discussed in the previous section.

2.2.1. Datasets

There are several datasets which provide tasks that include ABSA. Most widely used are the SemEval datasets. Task 4 of SemEval-2014 contains 7686 annotated sentences about restaurants and laptops [55].

Task 12 of SemEval-2015 expands the dataset of the previous year with the category hotels [54].
Lastly, Task 5 of SemEval-2016 adds additional subtasks for ABSA and text-level aspect-sentiment extraction as well as two new domains in different languages [54]. In addition they also provide a task where classifiers have to perform out-of-domain ABSA. Unfortunatelly, there were no submissions for this subtask. This is one of the most challenging tasks. Even modern deep learning architectures still struggle to successfully classify outside of the domain they were trained on.

Since annotating ABSA datasets is very expensive in term of annotation costs, even the biggest SemEval datasets only contain less than 10,000 sentences over all splits. In contrast, GermEval-2017 is a shared task dataset which contains more than 25,000 comments [82].

2.2.2. Deep Learning on ABSA

One crucial element for deep neural network training on natural language is the ability to transform a sequence of words from sparse high dimesional vectors to dense sentence embeddings. The majority of recent publications on ABSA use sentence embeddings as their first input layer [64, 72, 36, 67, 39, 85] in combination with Long short-term memory (LSTM) architectures [64, 72, 39].

In most approaches reasearches use a pipeline procedure to perform ABSA. During the first step, the models usually try to predict the aspects of the document. Then, in the second step of the pipeline the polarity is classified [34]. The winner of GermEval-2017 task-C use a pipeline approach [36] as well as three of the top performing approaches of SemEval-2016 [9, 33, 83].

Lakkaraju, Socher and Manning investigate the influence of pipeline approaches against joint aspect and sentiment classification [34]. They argue that combining aspect and sentiment detection gives a model the possibility to capture dependencies between aspect and sentiment more efficiently. They demonstrate that their joint aspect sentiment model using parse trees and Recurrent neural networks (RNNs) outperforms their pipeline baseline. To explain this they give several examples from the beer [43, 40] and camara-reviews dataset they use.

- "[...] delicious beer that's highly drinkable" - (Palate : Positive)
- "high carbonation level, kinda thin" - (Palate : Negative)
- "Display quality ofthe camera is high" - (Display : Positive)

- "This camera is highly expensive" - (Price : Negative)

Each of the four sentences contains the word "*high*" but in each sentence it appears in a different form and may lead to both positive and negative sentiment depending on the aspect. Lakkaraju et. al. state that due to the aspect-sentiment linkage the single sentiment approach was not able to correctly classify those sentences whereas their joint aspect sentiment model predicted correctly.

Ruder et. al. use a hierarchical, bidirectional LSTM for ABSA [64]. Although they also assume a pipeline approach where aspects have to be fed in one after another they propose a interesting architecture. Usually, sentiment analyisis in combination with aspect detection is either done by having multiple heads for each aspect [67] or by using a big softmax layer with every possible combination of aspect and sentiment which runs into issues when classifying multiple aspects at the same time [34]. Ruder et. al. propose a third alternative where the aspect itself is fed into the network as an embedding alongside the actual text. After feeding the sentence embeddings through one bidirectional LSTM the text features and the aspect embedding are combined again. Together they are passed to the last bidirectional LSTM layer and classified in a last output layer. They report competitive results against two non-hierarchical baselines and achieve state of the art for multi-domain datasets.

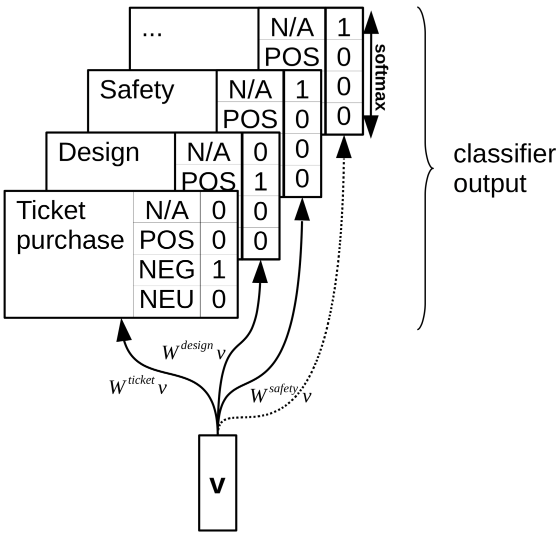


Figure 2.1.: Aspect Heads proposed by Schmitt et al. Source: Schmitt et. al. [67]

In 2018 Schmitt et. al. propose a novel approach to classify aspect and sentiment jointly [67]. Similar to Lakkaraju they also compare joint-approaches and pipeline approaches.

In both instances the joint approaches significantly outperform their corresponding pipeline approaches.
The architecture they use consists of two main components. The first layers are either bi-LSTM or Convolutional Neural Network (CNN).
Sentiment and aspect classification is then performed with aspect heads (Figure 2.1). For each aspect in the dataset one aspect head is constructed. Each aspect head consists of a final softmax layer which produces a four-dimensional vector where three dimensions are polarity labels (negative, neutral, positive) and the last label indicates whether or not the aspect is relevant for this specific sentence.

By using aspect heads this architecture is not only able to classifiy aspect and sentiment together but also cabalble of multilabel classification.
Ruder et. al. tested this architecture on the GermEval-2017 [82]. They report results that significantly outperform any submission on this task for this dataset. Their End-to-end CNN is able to achieve an F1-score of 0.423 and 0.465 on both test sets which is huge increase over the former State-of-the-art (SOTA) of 0.354 and 0.401 reported by Lee et. al. [36].

To the best of our knowledge there are no papers which use a transformer based architecture to perform joint aspect based sentiment analysis. There is one very recent publication which uses a BERT [15] model to perform a pipeline ABSA task where they model those tasks as a sequence labeling problem [84]. Peters et al. use BERT for binary sentiment anaylsis with the Stanford Sentiment Treebank [71] on movie reviews [52] but none of those researches use the transformer directly.

Recently, a new extension to ABSA has emerged which combines ABSA with target-dependent sentiment classification [72]. The task of Target dependent sentiment classification is to infer sentiment based on a sentence and a given target. This is very similar to aspects but in contrast to ABSA the targets are not predefined. A target string resemlbles closely resemlbles a search string which makes this problem even more challenging since a model has to first understand the target string.

3. Theoretical Background

This chapter attends to the theoretical background for the technologies used in this thesis.

3.1. Word Representations

While it is trivial to use images as input for machine learning techniques the same can not be said for language. Images already come in matrix form while words appear as discrete objects. Before language can be used for machine learning, words have to be transformed into a matrix first. A function $W(\text{word}) \rightarrow \mathbb{R}^n \quad \text{word} \in V$ which encodes words from a vocabulary V into a n-dimensional matrix is called a word embedding.

The easiest way to achieve this is a one-hot encoding of a vocabulary. We just take the vocabulary which is a list of N words and number the words sequentially. A word vector is created by getting the index of the word from the vocabulary and setting the value of the word vector at the word-index to one and all other entries to zero.

$$
\begin{aligned}
W(\text{"dog"}) &= [1, 0, 0] \\
W(\text{"cat"}) &= [0, 1, 0] \\
W(\text{"bee"}) &= [0, 0, 1]
\end{aligned}
$$

While this works for small vocabulary sizes, with larger vocabularies one will encounter problems. When having a vocabulary size of $N = 10,000$ this approach will produce $10,000$-dimensional vectors for each word. What makes matters even worse is that those vectors are extremely sparse since they only have one non-zero entry.

3.1.1. Vector Space

In 1992 Schütze (and to some extend Hinton in 1986 [23]) pioneered the idea of an universal word vector space [68]. The general idea is that instead of having sparse, high dimensional word encodings it would be better to have low-dimensional, dense word vectors where semantically similar words appear close together in the continuos word vector space. Unrelated words should be far away from each other. For example, the distance between the words "dog" and "cat" should be low while the distance between "bed" and "rocket" should be high. Figure 3.1 shows a visualization of a word vector space which has been reduced by t-SNE to two dimensions.

To achieve this, Schütze suggested to count co-occurrence of words and their neighbors and construct a co-occurrence matrix [68]. While this makes the word-matrix dense (only 2% of the entries are zero) the resulting matrix is still extremely large. To get more useful vectors dimensionality reduction techniques like Latent Semantic Analysis (LSA) have to be applied.

Of course, this task can also be solved by neural networks. Imagine W to be a matrix $W \in \mathbb{R}^{N \times n}$ where each row corresponds to a word and n is the dimension of the desired word vector equivalent to a lookup table. We initialize the weights of W randomly and use W to transform words into dense vectors so that they look something like this:

$$
\begin{aligned}
W(\text{"dog"}) &= [0.3, -0.8, 0.1, \dots] \\
W(\text{"cat"}) &= [-0.1, 0.0, 0.2, \dots] \\
W(\text{"bee"}) &= [0.3, 0.6, -0.9, \dots]
\end{aligned}
$$

Instead of improving W directly, we use W to train a network on an auxiliary task like predicting the next word in a sequence. Instead of training W supervised we use an unsupervised task and train W indirectly. This is the basic principle how many modern word embeddings are trained and interestingly, this produces fascinating results shown in figure 3.1.

Word vectors even allow for basic vector arithmetic. $W(\text{"King"}) - W(\text{"Man"}) + W(\text{"Woman"}) = W(\text{"Queen"})$ is a famous example for this demonstrated by Mikolov et. al. [48].

Figure 3.1.: Three regions of a word embedding visualized by t-SNE. The left region contains names of countries, the middle region consists of days and the right regions is made up of financial terms. Source: Turian et. al. [76][1]

[1]Full source for image `http://metaoptimize.s3.amazonaws.com/cw-embeddings-ACL2010/embeddings-mostcommon.EMBEDDING_SIZE=50.png`

For many NLP tasks, word representations have become indispensable and one of the factors of success for many systems [38]. The next sections will showcase a few of the most popular embedding techniques.

3.1.2. Word2Vec

Word2Vec by Mikolov et. al. is a very popular method which learns to produce word embeddings unsupervised from raw text [47]. The authors propose two approaches to learn an embedding. Both are very similar and rely on shallow neural networks to generate good embeddings. Moreover, these methods are computationally much more efficient than previous architectures without sacrificing performance. This allows word2vec to be applied on much bigger corpora with billions of words which was not possible before [46].

Skip-gram Model

The training objective for the skip-gram model is to predict the neighborhood context of a given word. For example, given the word w_t and a window size of 2 the objective is to predict the two words before w_t (w_{t-2}, w_{t-1}) as well as the two words after w_t (w_{t+1}, w_{t+2}). The objective can be formalized as maximizing the average log probability [45]:

$$\frac{1}{T}\sum_{t=1}^{T} \sum_{-c\leq j\leq c, j\neq 0} log p(w_{t+j}|w_t) \tag{3.1}$$

T is the number of words in the corpus, c is the context window size and $p(w_{t+j}|w_t)$ is the softmax function. Simply put, skip-gram predicts a context given a source word.

Continuous Bag-of-Words Model (CBOW)

Continuous Bag-of-Words Model (CBOW) is very similar to the skip-gram model. However, instead of predicting context words for a source word, CBOW predicts the next word w_t in a sequence, given the previous words w_{t-j} [46]. In some instances the surrounding words are taken as the context instead of just the previous words. In other words, the objective of CBOW is to predict a word given it's context.

Skip-gram works better for smaller datasets than CBOW since it is possible to generate more training pairs from a sequence. In addition, it is also able to represent rare words better than CBOW. However, CBOW is faster to train than skip-gram and yields slightly better accuracy for more frequent words.

3.1.3. Global Vectors (GloVe)

There are two methods for create word vectors: word co-occurrence counting (section 3.1.1) and the window based skip-gram / CBOW methods (section 3.1.2). Co-occurrence is relatively fast and efficient to collect and train unless the vocabulary is very big. A co-occurrence matrix also captures a lot of global statistical information whereas local window based models only capture the statistics for the window. However, most of the information a co-occurrence matrix provides is about the word similarity and not necessarily about the semantics. In addition, frequent words like "the" or "a" co-occur with a lot of other words. Therefore, they have huge co-occurrence counts with many words without providing any useful information [51].
Local window based models like word2vec have the disadvantage that training is not as efficient and a lot of statistical information is not captured [51].

Global Vectors (GloVe) which was introduced by Pennington et. al. tries to combine the advantages of both model families [51]. GloVe captures the global corpus statistics directly by creating a co-occurence matrix X where the entry X_{ij} denotes how often the word j occurs together with i. The probability that i appears in the context of j is

$$P_{ij} = P(j|i) = \frac{X_{ij}}{X_i} = \frac{X_{ij}}{\sum_k X_{ik}} \quad . \tag{3.2}$$

To put it simply, P_{ij} is just the number of times i and j appeared together divided by the sum of all words that i appeared with [51].
Pennington et. al. demonstrate that the ratio of probabilities P_{ik}/P_{jk} encodes a lot of the meaning that is lost in traditional co-occurence based methods [51].

The authors use F to describe this ratio as

$$F(w_i, w_j, \widetilde{w}_k) \approx \frac{P_{ik}}{P_{jk}} \tag{3.3}$$

w are just word vectors of i, j and k ($\widetilde{w}$) is a context vector. In the end the goal is to get a good candidate for F that fulfills all our requirements. By adding vector differences ($w_i - w_j$) and linear relations (dot-product) F becomes

$$F((w_i - w_j)^T \cdot \widetilde{w}_k) \approx \frac{P_{ik}}{P_{jk}} \quad . \tag{3.4}$$

By replacing the probability ratio with logarithms we arrive at

$$w_i^T \cdot \widetilde{w}_k = log(P_{ik}) = log(X_{ik}) - log(X_i) \quad . \tag{3.5}$$

The weighted least squares objective function is then finally

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \cdot \widetilde{w}_k + b_i + \widetilde{b}_j - log(X_{ij})^2) \tag{3.6}$$

where V is the vocabulary size and b and $\widetilde{b}$ are bias terms that are added.

The last problem is solved by a weighting function $f(X_{ij})$. Co-occurences which are infrequent will contain a lot of noise. Therefore it is important to reduce the influence of noisy words and prevent frequent words like "the" from dominating the model. Therefore, they weight each word to make sure that words are not over or underrepresented.

3.1.4. FastText

FastText is an embedding technique which focuses on efficiency but is still on par with other word embedding techniques [30].

FastText is based on Word2Vec. However, whereas word2vec treats each word as an atomic entity, FastText splits words into several character n-grams. The word "where" consists of the following 5 n-grams for $n = 3$ [8]:

"<wh", "whe", "her", "ere", "re>"

The special characters "<" and ">" are boundary symbols to differentiate prefixes and suffixes from the beginning and ending of words [8].

This notion is very powerful since it provides several advantages over previous models. For one, out of vocabulary words are better recognized since their n-grams are still known. Unknown words can therefore be partly constructed by known n-grams. [8].

Some languages like German which heavily rely on compound words also profit from this technique since those words can also be constructed using known n-grams [8].

Lastly, FastText represents rare words a lot better than word2vec. Even if a word is very rare and only occurs in a sentence a few times in total, there is a high chance that the model consists of n-grams which appear in other, more frequent words [8].

3.2. Google Transformer Architecture

The Google transformer architecture is a novel neural network architecture for automatic machine translation tasks by Vaswani et. al. [78]. Instead of relying on RNN based architectures with recurrence, the transformer uses an attention mechanism to achieve State-of-the-art (SOTA) results on machine translation tasks. Like most of sequence-to-sequence models the transformer also uses the encoder-decoder pattern to translate a sentence from the source to the target language. We will mostly focus on the encoder part of the model since this is the part, we use to perform ABSA classification.

The issue when training many RNN-based architectures is the problem of long-range dependencies. These dependencies may lead to either vanishing or exploding gradient [25]. The dilemma is, that language translations features a lot of long dependencies. For some translations like "I like planes" to "Ich mag Flugzeuge" each token exactly matches the position of the translation. However, in many cases the positioning of words is different. Take the sentence "I think I saw you at the airport *yesterday*". This would translate to "Ich glaube, dass ich dich *gestern* im Flughafen gesehen habe".
When translating this sentence from English to German, a recurrent architecture has to keep the dependency of "saw" of all the way until the end of the German sentence when it has to output the tokens "gesehen habe".

Besides the issue with long-term dependencies, RNN architectures are also almost impossible to parallelize which is especially important for long sequences [78].

The transformer model tries to fix both problems. It is the first transduction model to completely gives up on recurrence or convolutions to compute sequence representations [78].
Instead it relies on attention and specifically multi-head attention. Attention allows a model to focus on relevant information and the idea of multi-head attentions enables parallelization as well as attention heads which focus on their attention on different aspects [78].

3.2.1. Encoder-Decoder Architecture

A high level architecture overview of the transformer is depicted in figure 3.2. As already mentioned the transformer resembles an encoder-decoder architecture. Yet, in contrast to traditional encoder-decoders, the transformer consists of n encoder- and n decoder blocks [78].

The encoder blocks always get the whole input sequence encoded as word vectors. This sequence is then propagated until it reaches the top encoder. From there, the encoder output is passed to all decoder blocks. Furthermore, the decoder blocks also receive the output of the decoder blocks below them. In addition, the first decoder block (in figure 3.2 this block is named "Decoder 1") receives the sequence of tokens the transformer already predicted [78].

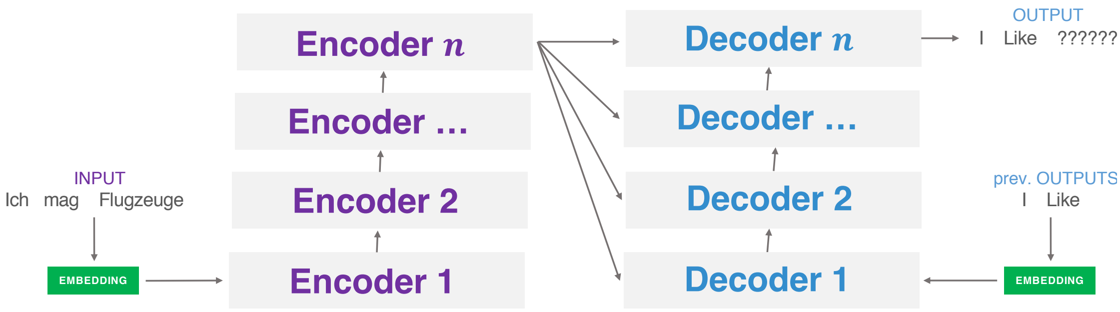


Figure 3.2.: High-level overview of the transformer Encoder-Decoder architecture, translating the sentence "I mag Flugzeuge" from German to English. The model is at the step where it translates the last word "Flugzeug" to plane.

Following the example in figure 3.2 the translation would start by passing the sentence "Ich mag Flugzeuge" through all encoder blocks until it reaches the top block. All 1 to n decoder blocks receive the encoded sentence. The encoded sentence then flows up from the bottom to the top where it outputs the first token "I".
Decoder 1 now gets two inputs:

1. the encoded source sentence it previously also received
2. the previous predicted output which is the token "I"

Decoder 2 to n now get the encoded input (as before) as well as the output of the decoders below. The top encoder will then output "like" as the next token in the sequence.
The step that would follow this one is shown in figure 3.2 where "I like" is already predicted and only the last token is missing.

Figure 3.3 shows a detailed overview of one encoder and one decoder block. This is useful to comprehend the information flow on the decoder side.

3.2.2. Attention Mechanism

The transformer uses attention to focus on the important and relevant information in a sequence. The question what is regarded as important is learned [78].
The specific attention mechanism, the transformer uses is called "Scaled Dot-Product Attention" (Figure 3.4 left). The input for the attention function are three matrices called "Queries" Q', "Keys" K' and "Values" V'. We get these matrices from the input embeddings. We just multiply the embeddings with W^Q, W^K and W^V. These projection

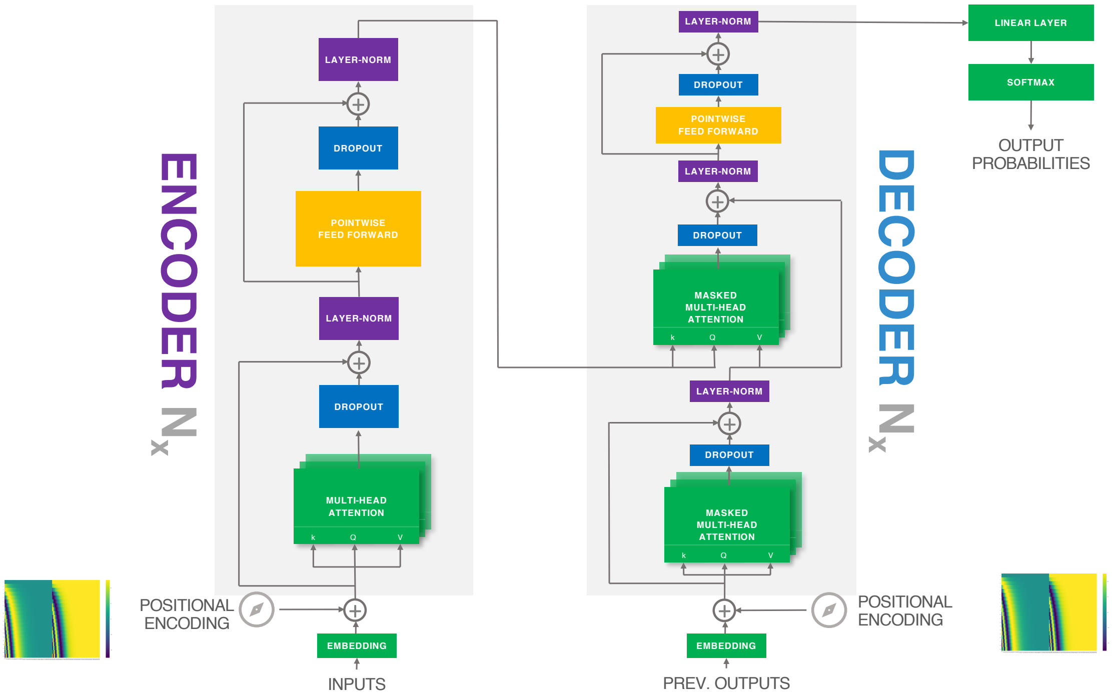


Figure 3.3.: Overview of the whole Transformer architecture.

matrices are learned and project our input to a lower dimension so that we get Q', K', V' [78].

We then take the dot-product of Q' and K' and scale the result by dividing by the square root of d_k where d_k is the dimension of Q' and K'. Next, we can finally apply the attention. From the softmax we get a vector of probabilities which are the attention values. When we apply the dot product on those attentions and the values we scale the values with the attentions [78].

Words which are deemed important by the attention mechanism are multiplied with values near 1 while unimportant words are multiplied with small values and therefore become unimportant for the next layers [78].

One aspect which was not mentioned is the "mask"-step. After the scaling operation we set some values to 0. This is done because every input sequence needs to have the same length which we achieve by adding <pad> tokens to sentences which are not long enough. Those tokens carry no information so we can safely discard them in this step [78].
In the decoder part we need to mask specific parts of the sentence which the transformer

has not predicted yet. This is done to prevent the model from cheating by just replaying the input sequence [78].

The transformer is constructed to use multiple "attention-heads" where each "head" performs its own dot-product attention (Figure 3.4 right). To get different values, each head i has its own set of W_i^Q, W_i^K and W_i^V matrices. The main idea behind the concept of multi head dot-product attention is that each head specializes on some aspect. One head might pay attention to entities while another head looks out for actions [78]. This being said in practice this is not as clear and the distinction is more fuzzy.
The following equation summarizes the process which is described above:

$$\text{Attention}(Q', K', V') = \text{softmax}(\frac{Q' \cdot K'^T}{\sqrt{d_k}}) \cdot V' \tag{3.7}$$

$$\text{AttentionHead}_i(Q, K, V) = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{3.8}$$

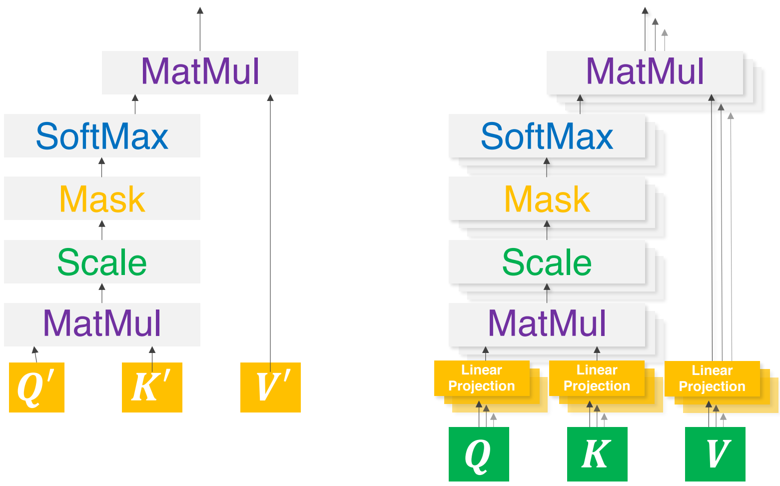


Figure 3.4.: left: single scaled dot-product attention mechanism. right: multiple scaled dot-product attention heads stacked together. Each head receives its representation of Q, K and V by using projection layers.

3.2.3. Positional Encoding

The transformer always gets the complete sequence of words as the input. However, since the transformer has no recurrence it has no memory. This means that it does not know which word is at which position. Yet, this information is very important to understand sentences [78].

This issue is solved by a "positional encoding". This encoding is added to the word embeddings at the bottom of the encoders and decoders. It has the same dimension as the embedding so it can be added element-wise [78].
For the positional encoding the authors use a combination of sine and cosine functions. They also experimented with a learned encoding but this approach did not perform significantly better [78].

3.2.4. Point-wise Layer

After the input is passed through the attention heads, a pair of fully connected layers with nonlinearities are used [78].
The first linear layer scales the input to the inner layer dimensionality of 2048 and the second layer scales the input back to the previous model size of 512 [78].

3.2.5. Adam

Vaswani et. al. propose to use the Adaptive Moment Estimation (Adam) optimizer [32]. Adam is an extension to the popular Stochastic gradient descent (SGD) optimizer. However, instead of having a fixed learning rate like SGD, Adam computes a learning rate for each trainable network parameter. Those learning rates are directly obtained by the decaying mean and uncentered variance of the past gradients [32].

This is different to AdaGrad [17] and RMSProp [24] which also maintain a learning rate for each parameter. However, RMSProp only use the mean and AdaGrad updates parameters based on how frequent the specific parameter is used as a feature.

3.3. Multi-Task Learning

Rich Caruana first introduced Multi-Task Learning (MTL) in 1993. Conventional machine learning approaches break a problem down in smaller tasks and solve one task at a time (e.g., word-by-word POS-tagging [75], word-by-word NER [66] or handwritten image classification [35]). In each of these tasks a classification algorithm solves exactly one task (Assigning a 'part-of-speech' or entity type to a word, or the classification of handwritten digits). Caruana shows that combining multiple related tasks improves model performance [12][11].
In Multi-Task Learning (MTL), multiple related tasks are learned in parallel and share a common representation. Generally speaking every machine learning model which optimizes multiple objectives for a single sample can be considered as Multitask Learning. This includes multi-label classification where one sample can have multiple

labels as well as instances where different sample distributions or datasets are used for different tasks.
MTL is similar to how humans learn. Generally, humans learn new tasks by applying knowledge from previous experiences and activities. For instance, it is easier to learn ice skating when someone previously learned inline skating. This is because all the underlying important aspects of the tasks are very similar.
When tasks are related this also holds true for machine learning. When learning these tasks in parallel model performance is improved compared to learning them individually since the additional knowledge that a related task carries, can be used to improve on the original task [11].
There are four important aspects one can use to determine if MTL can bring performance boosts for a specific objective:

1. Multi Label Task: Multi Label classification task where one sample can have more than one label are almost always inherintly solved using MTL if labels are predicted by one model. Multiple authors show that adding tasks always improves performance compared to a separate model for each task as an alternative [61].

2. Shared low-level features: MTL only makes sense if the tasks share low level features. For instance, image classification and NLP do not share common features. In this case the model would not benefit from MTL because one task can not help to improve the other task. Therefore, it is important to choose tasks that are related to each other [88]. In most cases MTL will work with NLP tasks because they usually share at least some kind of sentence or word embedding as a common layer.

3. Task Data Amount: Several authors have suggested that it is important for the success of MTL training that the amount of data for the tasks is similar. Otherwise the model will mainly optimize for the task with most training samples.

4. Model Size: Finally, the multi-task model needs to have enough parameters to support all tasks [11].

3.3.1. Differentiation against Transfer Learning

Training samples from one task can help improve the other task and vice versa. This is important for the differentiation against transfer learning [57]. In MTL each task is equally important. In transfer learning the source task is only used to improve the target task so the target task is more important than the source task [88]. In addition, Transfer Learning uses a linear training timeline. First, the source task is learned and then after learning is completed this knowledge is applied to boost the learning process of the target task. MTL, in contrast, is learning both tasks jointly together instead of one after the other.

3.3.2. Improvements through Generalization

There are several reasons why the MTL paradigm performs so well. For instance, the generalization error is lower on shared tasks [12]. MTL acts as a regularization method and encourages the model to accept hypothesis that explain more than one task at the same time [63]. The model is forced to develop a representation that fits the data distributions for all tasks. In the end this creates a model that generalizes better because it must attend to different objectives.

3.3.3. Improvements through Data Augmentation

Secondly, Multi-Task Learning increases the number of available data points for training. All tasks share a common representation. While training one task all other tasks are also implicitly trained through the common representation.

Statistical Data Amplification

Each new task also introduces new noise. Traditionally, a model tries to learn by ignoring the noise from its data. However, if the model does not have enough training samples it will overfit because it focuses too much on the noise to explain the data. By introducing additional tasks, new data and therefore new noise is introduced which the model has to try and ignore [63]. This aspect is called *Statistical Data Amplification*[10].

Blocking Data Amplification

Blocking Data Amplification occurs when there is little or no noise. Consider the simple example from Caruana [10] that there are two tasks T and T' and common features F. The first task T is $T = A \wedge F$ and the second task T' is $T' = \neg A \wedge F$. For $A = 0$ only T uses feature F and T' does not and for $A = 1$ it's the other way around. By training on both tasks F is used no matter what value A takes.
Rei makes use of these aspects and proposed a sequence labeling framework which uses a secondary, unsupervised word prediction task to augment other tasks such as NER or chunking. They show that by including the word prediction the auxiliary task performance is improved for all sequence labeling benchmarks they tried [62].
Similarly, Plank et al. show that learning to predict word-frequencies along with POS-tagging also improves the total model performance [53]. They argue that predicting word frequencies helps to learn the differentiation between rare and common words.

3.3.4. Architecture

The most common architecture for multitask learning is shown in figure 3.5. It is called hard parameter sharing and consists of at least one layer which is shared among all tasks. In addition, each task has at least one separate layer. This approach is also the one we used for our model which is described in chapter 4.
The easiest way to compute the loss for a hard parameter sharing MTL architecture is to take the sum of all losses for the individual tasks which is shown in equation ...

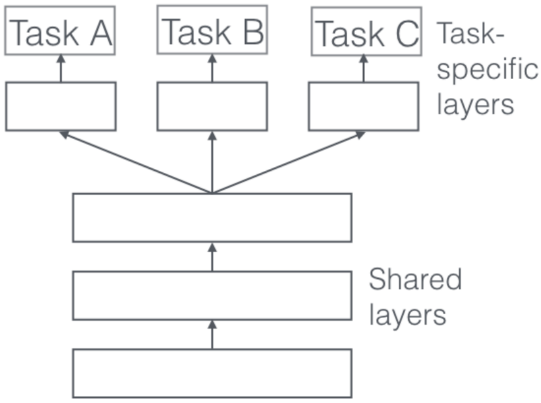


Figure 3.5.: Hard parameter sharing. The first three layers are shared among tasks A, B and C. Each task also has one or more layers. Source: Ruder 2017 [63]

3.4. Transfer Learning

In 1991, Pratt et al. suggested to transfer information encoded in a neural network by reusing the network weights in a new network [58]. They show that even accounting for the training time of the source network they achieved significant speedups when training a target network compared to random weight initialization.
Yosinski et al. provide a more modern definition: First, a base network is trained on a base dataset. Then, the learned features (the knowledge) of the base network is transferred to a second target network which is then trained on the target dataset and task [86]. This process works well if the base and target dataset and tasks are similar.
Goodfellow et al. give a more general definition. They define Transfer Learning (TL) as the transfer of previously learned knowledge from one or multiple sources to a target domain with fewer examples [21].
Figure 3.6 communicates those definitions.

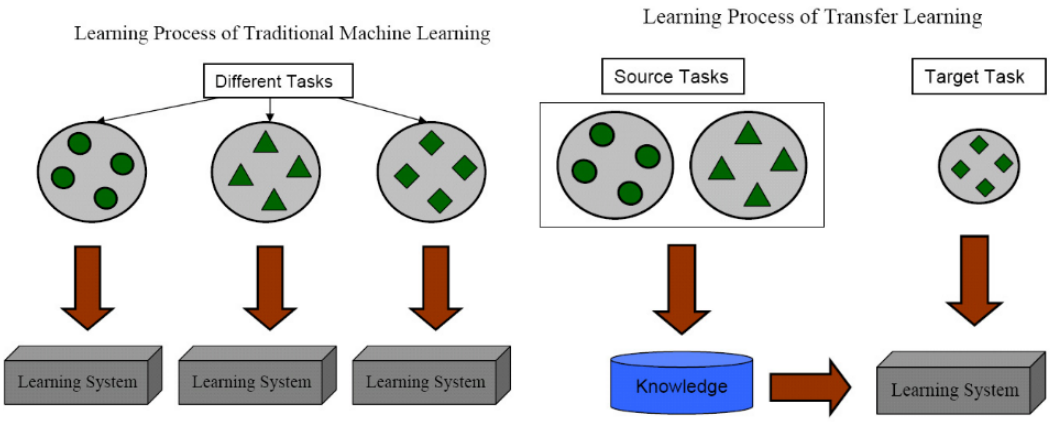


Figure 3.6.: Difference in traditional machine learning were each model uses its own dataset and task. In contrast, in transfer learning a model is first trained on source tasks and part of the features are transformed to the target model to facilitate training. Source: Pan and Yang [49]

In practice it is very expensive to collect or recollect training data for every new domain. Transfer learning makes it possible to transfer knowledge from a larger dataset to a smaller dataset which greatly reduces the labeling effort [7]. When the target dataset has significantly fewer examples than the base dataset studies showed that it is possible to train large networks without overfitting [16][87]. Usually, after the base model has been trained on the large dataset, the first n layers of the base model are copied over as the first n layers of the target model. The remaining layers of the target model are then randomly initialized and trained. The weights of the n layers from the base model can either be *frozen* or *finetuned* along the rest of the target model. If the target dataset has few samples compared to the number of parameters in the first n layers, finetuning can actually result in overfitting which is a reason why the error during target training is often not backpropagated to the first n layers [86].

Pre Training

The most common way to employ transfer learning is pre-training. Pre-training is often used in image recognition were interestingly, the first few layers generally form into the same feature regardless of the domain or task [86]. Consequently, researchers are able to exploit this by taking the first layers from a model which was previously trained on a large dataset like ImageNet [65] and use these weights for their tasks which might have less examples.

This paradigm can also be applied on natural language processing. Understanding what words mean is the fundamental problem every NLP model has to solve. Therefore, it is sensible to use an embedding layer which has been pre-trained on large datasets like common crawl which contain petabytes of information [70]. This pre-trained embedding layer can then be used in a model trained on a much smaller dataset.

3.5. Hyperparameter Optimization

Generally, there are two sets of parameters in machine learning: learned parameters and hyperparameters which are used to configure various aspects of the training process. Learned parameters such as neural network weights are optimized during training whereas hyperparameters are usually defined at the beginning and without a few exceptions (e.g. learning rate) do not change during training.
It has been demonstrated that there are a few hyperparameters which have an enormous impact on the overall model performance but identifying those parameters among a big set of possible candidates is difficult [4]. However, correctly setting these parameters is crucial for achieving a good model performance. Cox and Pinto demonstrated that hyperparameters make the difference between a state of the art model and a model which does not perform better than a random classifier [13]. Therefore, hyperparamter tuning is critical for the model performance.
Hyperparameters are either hand tuned by reviewing similar literature or by the researchers understanding of the underlying architecutre and how he expects certain parameters to influence the architecture. Another way is to semi automatically optimize or fully automatically optimize the search for good hyperparameters.
This section presents three approaches to optimize the hyperparameter space. The first two are naive, semi-automatic approaches where a set of possible parameters is tried and the success is recorded. The researcher then selects the most promising results. The third example, HyperOpt, is an algorithm which treats the hyperparamter search as an optimization problem and identifies critical parameters and tries to find their optimum value for the given model and task [6].

3.5.1. Grid Search

Grid search is a very easy method to search for optimal hyperparameters. When performing a grid search for a parameter a new value is sampled from a predefined parameter subset at a fixed interval. Each trial with the new parameter value is then evaluated on the model. For multiple parameters each distinct parameter value is tested against all other parameter values, therefore creating a *grid* of parameter values to test. This approach is very easy to implement and is trivial to parallelize.

3.5.2. Random Search

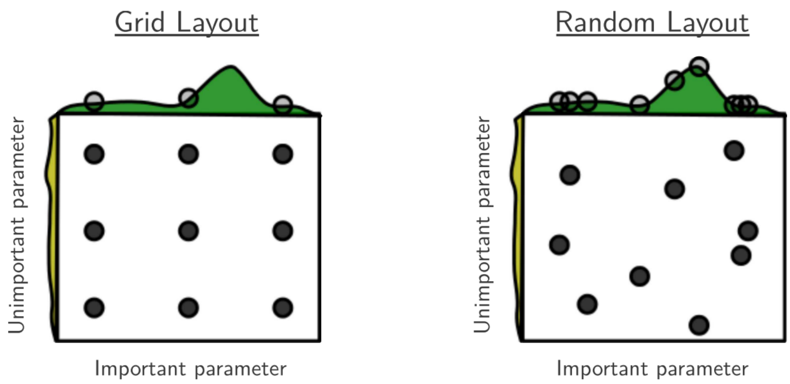


Figure 3.7.: This figure from Bergstra and Bengio demonstrates the advantage of random searches over grid searches in a two-dimensional space. Nine trials are performed to optimize a function $f(x,y) = g(x) + h(y) \approx g(x)$. $g(x)$ shown in green has a bigger impact compared to $h(y)$ shown in yellow on the left. Each gray circle represents a trial. Because of the two-dimensional space, grid search can only test $g(x)$ in three places. Random search tries a different x in every trial and is therefore able to find a value close to the optimum. Source: [4]

Surprisingly, Bergstra and Bengio proofed that randomly choosing hyperparameters is more efficient than performing a grid search [4] for high dimensional search spaces. Instead of defining values in a grid, they randomly sample from the grid space.
The problem with grid search is that by increasing the number of dimensions the number of trials has to increase exponentially to provide the same number of distinct trials for a single parameter [4]. When performing a grid search on a one-dimensional parameter space, three runs on the model have to be performed in order to test three distinct values of the parameter. Optimizing two parameters (shown in figure 3.7) increases the number of runs to 3^2 and optimizing n parameters m times will lead to m^n runs.
Grid search is set up on the assumption that each parameter is equally important. However, it has been shown that not all parameters are equally significant for the model performance [4]. Figure 3.7 demonstrates why this is an advantage for random search over grid search. In this specific example one parameter constitutes more towards model performance than the other. However, grid search can only sample three values for the important parameter and is therefore not able to find the optimum value. According to Bergstra and Bengio this situation is the norm rather than the exception

for grid search [4].

3.5.3. HyperOpt

Hyperopt is an open source[2] hyperparameter optimization package by Bergstra et al. [5]. It treats the hyperparameter search as an optimization problem. Bergstra et al. show that by using Tree of Parzen Estimators (TPEs) and Gaussian processes Hyperopt is able to find hyperparameters that outperform random searches and traditional manual hyperparameter tuning [3].

Challenges

There are certain challenges when treating hyperparameter tuning as an optimization problem. For instance, the parameter search space is often high dimensional and may contain a mix of continuous (e.g. learning rate), discrete (e.g. hidden layer size), boolean (e.g. preprocessing steps [28]) and even conditional variables [6].
For instance the choice of a optimizer or even the machine learning algorithm itself can be seen as a hyperparameter. Each choice then has its own set of parameters which are independent of the other choices. For instance, the Adam optimizer uses certain parameters like β_1 which the SGD optimizer does not use. Hyperopt generates a graph from these conditional parameters and then uses a tree like structure for solving the optimization problem [3].

Another difficulty is the limited "fitness evaluation budget" [3]. This means that for each evaluation of a hyperparameter set the model has to be trained which potentially takes a long time. Therefore, Hyperopt has to cope with less evaluation steps than a normal optimization algorithm.

Tree of Parzen Estimators (TPEs)

The Hyperopt package uses TPEs to sample good hyperparaters from the hyperparameter search space [5]. To model $p(x|y)$ TPE replaces, all distributions in the configuration space by Gaussian mixture equivalents: uniform $\rightarrow$ truncated Gaussian mixture, log-uniform $\rightarrow$ exponentiated trucanted Gaussian mixture and categorical $\rightarrow$ re-weighted categorical [5]. The prior for the calculation - the different observations $\{x^1, ..., x^n, ..., x^k\}$ - is initialized by performing n random runs where the default value for n is 10.

[2]Official repository `https://github.com/hyperopt/hyperopt`

The TPE defines $p(x|y)$ as

$$p(x|y) = \begin{cases} l(x) & \text{if } y < y^* \\ g(x) & \text{if } y \geq y^* \end{cases} \tag{3.9}$$

where $l(x)$ (first case) is a density formed by an observation $\{x^i\}$ where the loss y of $f(x^i) = y$ was less than a threshold y^*. $g(x)$ is the density by using all other remaining observations [5]. y^* is higher than the best observation so that the density $l(x)$ is formed by more than just one observation. $l(x)$ and $g(x)$ model the hyperparameter search space which means that they have to be hierarchical when the search space contains conditional and discrete variables. $l(x)$ and $g(x)$ are then used to optimize the expected improvement and after each iteration the parameter set with the highest expected improvement is chosen for the next iteration which then becomes the next observation x^{k+1} [5].

3.6. Performance Measurements

The following section describes the performance measurement which were used to evaluate the performance of a model.

3.6.1. Precession - Recall - F1 Score

The most commonly used measurement for NLP tasks is the F1-score. This metric has one advantage over accuracy. Accuracy does not take data imbalance into account. Accuracy is just the number of correctly classified samples divided by the total amount of samples. This means a classifier which just predicts the majority class will get a high accuracy.

The combination of precision and recall on the other hand, objectively measures the actual relevance and performance of a classifier for a given class. Precision and recall are defined as:

$$Precision = \frac{T_p}{T_p + F_p} \quad Recall = \frac{T_p}{T_p + F_n}. \tag{3.10}$$

- True positives T_P is the number of correctly classified samples for a class.
- False positives F_P are all samples which the model predicted to be part of the class but are in reality not part of the class. (Type I Error)

- False negatives F_N are all samples that belong to the class but are labeled as not being part of the class (Type II Error)

A high recall means that many samples were matched correctly and a high precision denotes a low number of incorrectly classified samples.

The F1 score is the harmonic mean between the precision and the recall and is given as

$$F_1 = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}} \tag{3.11}$$

The F1 score is scaled from $[0, 1]$ were 0 is a result with no true positive samples. A classifier which achieves a score of 1 classified all samples correctly meaning it has no false positives or false negatives.

Micro- and Macro F1 score

There are two popular methods to aggregate the F1 score. Researches can use the micro- or the macro F1 score. Equation 3.11 remains the same but the way it is aggregated changes.

The micro F1 score is calculated by taking the sum of all true positives, false positives and false negatives. This has the implication that the class with the biggest number of samples contributes most towards the total score. Therefore, descriptions of the micro F1 score often point out that this method takes label imbalance into account.
The side effect of this is, that classifiers that tend to predict the most frequent classes will achieve a high micro F1 score. Classes which only contain a few samples will not count much towards the overall score.

The macro F1 score is calculated by averaging F1 scores for the classes. First, micro F1 scores are calculated at the lowest level. Then, the average F1 score is taken over all classes. This has the effect that each class counts the same towards the overall F1 score. Classes with only a few samples are therefore as important as classes with the majority of samples. This method does therefore not take label imbalance into account.

There is one small exension to the macro F1 score which is the weighted F1 score. This score does also average the micro f1 scores but it uses weights to change the influence a class has over the total score.

Both scores are very valuable to asses the performance of a model. The micro F1 score tells a researcher how well the model is able to replicate the overall data distribtion since models which predict majority classes more often will achieve a higher score.

The macro F1 score is useful to measure how the model treats minority classes. A high micro F1 score and a low macro F1 score imply that the model predicts classes with many samples very well but fails to predict classes with few samples.
Therefore, it is always useful to provide both scores when presenting results.

4. Method

The transformer model has shown great potential on a variety challenging NLP tasks. Originally, the transformer was created to perform machine translation where it outperformed previous models by a huge margin [79]. In fact, at the time of writing, the transformer is the core of the Google translate engine.

OpenAI uses the transformer to perform various NLP tasks [59]. Their Generative Pre-Training (GPT)-model solves classification, entailment, similarity and multiple choice question answering tasks. They use a transformer encoder stack with 12 encoder blocks and task dependent classification heads.
They show that even though the same base transformer model is used for each task they achieve SOTA results for most of the tasks.

One year later, in 2019, OpenAI published GPT-2 which is an extension to GPT [60]. With GPT-2 Radford et. al. are able to achieve revolutionary results generating text with a transformer architecture[1]. Again, they also achieve SOTA results on other NLP task mentioned above.

We propose the ABSA-Transformer (ABSA-T) which builds on the knowledge that the transformer is a powerful architecture useful for a variety of NLP tasks. Radford et. al. already show that the transformer can be used to predict sentiment for a document [59].
However, sentiment analysis is one of the few tasks where GPT-1 is not able to achieve SOTA results. Moreover, GPT-1 only classifies standard sentiment and not aspect-based sentiment.

This will be the first architecture which uses a transformer to perform multi-task aspect-based sentiment analysis. The next section describes the architecture of the ABSA-Transformer (ABSA-T) model. Section 4.4 describes how ABSA-T is used in combination with multi task learning and finally, section 4.5 explains how we used transfer learning to boost the training performance.

[1]OpenAI will not release the trained model for the language generation task as they are afraid that it will be used for malicious intents - Source: OpenAI blog `https://openai.com/blog/better-language-models/`

4.1. General Model Architecture

The following section describes the proposed ABSA-Transformer (ABSA-T) architecture. As the name suggests this design is based on the transformer model [79]. The second model characteristic is influenced by the work of Schmitt et. al. and their concept of separate aspect heads [67].

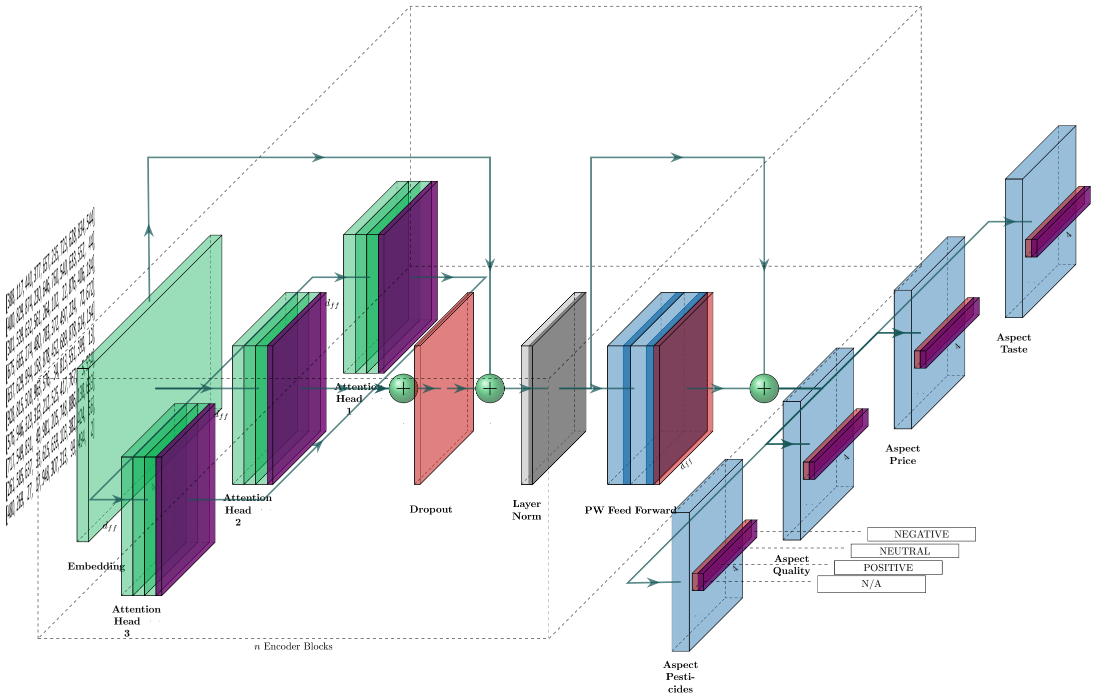


Figure 4.1.: Visualization of an exemplary ABSA-Transformer (ABSA-T) model with one encoder block consisting of three attention heads and four aspect heads. The positional encoding is not visualized in this figure.

Figure 4.1 visualizes a simplified ABSA-T model. This specific instance consists one encoder block which contains 3 attention heads. To the right of the encoder block are four aspect heads. Each aspect head is trained to classify the sentiment for one aspect. In figure 4.1 those aspects are *GMOs*, *Quality*, *Price* and *Taste*. Each aspect has four output classes (visualized in the figure for the "Pesticides"-aspect):

- negative
- neutral

- positive
- not applicable (N/A)

Each aspect head is isolated from the rest which allows the transformer to perform multi-label ABSA. When a document or sentence contains a specific aspect the corresponding aspect head outputs either *negative*, *neutral* or *positive*. If this aspect is not part of the sentence, the output is *not applicable*.
We propose two different versions of aspect heads which are described in section 4.3 in detail.

4.2. Transformer

The original transformer uses undisclosed word embeddings which output a 512-dimensional vector[2]. It is possible that the original transformer does not use pretrained embeddings. We performed experiments with ABSA-T and untrained word embeddings but concluded that pretrained word embeddings outperform untrained word embeddings.
However, the difference was only a few percent and it is conceivable that a transformer with more training data would be able to train its own word embeddings.

Considering the smaller datasets that we use for ABSA, the ABSA-T model uses pretrained embeddings instead of untrained embeddings. Pretrained embeddings for both GloVe and fastText are only available for up to 300 dimensions. As a consequence, the model size of transformer is only 300 instead of 512.

Similar to the vanilla transformer, ABSA-T also uses an Adam optimizer [32] and a special learning rate decay which is called NOAM[3] [79]

$$\text{NOAM:} \quad lr = d_{\text{model}}^{0.5} * min(step\_num^{0.5}, step\_num * warmup\_steps^{-1.5}) \tag{4.1}$$

Contrary to the transformer model, we do not use label smoothing as a regularization technique. Experiments showed that this impacted the F1-score negatively. Instead, ABSA-T uses weight decay with a decay value of $\epsilon_w = 1e - 8$.

[2]They also experiment with 256 and 1024 dimensional vectors

[3]It is not clear what noam stands for or where this learning rate decay schema came from. It is not mentioned or cited as noam but it is referred to as noam by the authors in discussions: `https://github.com/tensorflow/tensor2tensor/issues/280`

4.3. Aspect Heads

The transformer is designed as an encoder-decoder architecture with multiple stacks of encoders and decoders. Therefore, the input of an encoder (or decoder) has the same dimensionality as the output. Consequently, the input of an aspect head has the following dimensionality: $[batch\_size, s, d_{model}]$ where d_{model} is the model size and s is the sequence lenght. In other words, the transformer provides a d_{model} dimensional vector for each word w_i in a sequence.

The aspect heads have the role of transforming this vector to a vector which can be used for sentiment classification. For aspect-based sentiment classification this dimension is $[batch\_size, 4]$.

4.3.1. Linear Mean Head (LM-H)

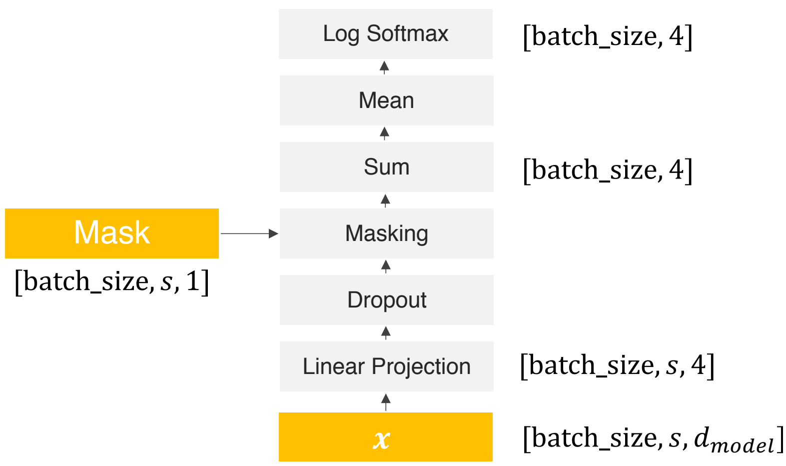


Figure 4.2.: Visualization of the operations of an ABSA Linear Mean Head (LM-H)

The first aspect head design is called Linear Mean Head (LM-H). This head consists of a linear layer which projects the d_{model} dimensional word vector to a 4-dimensional word vector. This reduces the tensor to $[batch\_size, sequence\_lenght, 4]$. The tensor is then summed up along the second dimension and the mean is taken. Finally, a softmax operation calculates the log probabilities.
Similar to the transformer layers, optional masking is applied on the tensor. Predictions for words in the sequence which are only padding tokens are replaced by zeros.

Mean Operation

The mean after the sum acts as a normalization for the prediction values before they pass through the softmax. The log softmax is defined as

$$\sigma(y_i) = log(\frac{e^{y_i}}{\sum_{j=1}^{K} e^{y_j}}) \quad . \tag{4.2}$$

As a consequence, large values for y_{ij} result in very large negative values after the log softmax. This has two effects on the Negative Log Likelihood (NLL)-loss:

1. Long sequences will get a larger loss than shorter sequences. The reason for this is, that the sum-function will sum up all predictions for every word in a sequence. Of course, a longer sentence will have a larger sum. Therefore, the result of the softmax will be more negative which results in a higher loss.

2. It is not possible to compare the loss of Linear Mean Head (LM-H) and Convolutional Head (CNN-H). CNN-H uses a combination of convolutions and max pooling. Therefore, the numerical value of a prediction before the log softmax will usually be lower than for CNN-Hs, since max pooling takes the maximum and not the sum.

Using a mean operation before the log softmax solves both problems.

4.3.2. Convolutional Head (CNN-H)

The Convolutional Head (CNN-H) uses convolutions in combination with max pooling to perform the final prediction. Figure 4.3 visualizes the operations and how the tensor size changes during a forward pass through the head.

First, the tensor is passed through a convolutional layer. The filter size, the number of filters, stride and padding are controlled through hyperparameters. This means the size of subsequent layers depends on these parameters. The output size c_out of the convolutional layer is given as:

$$c_{out} = \frac{s + 2 * P - F}{S} + 1 \tag{4.3}$$

where F is the filter size, S is the stride and P is the padding amount. The max pooling layer reduces the tensor from $[.., d_{d_model}, c_{out}]$ to $[.., d_{model}]$ along the first dimension. Finally, a linear layer projects the output to class predictions and the log softmax scales the values.

Figure 4.4 depicts the convolution and the pooling in more detail.

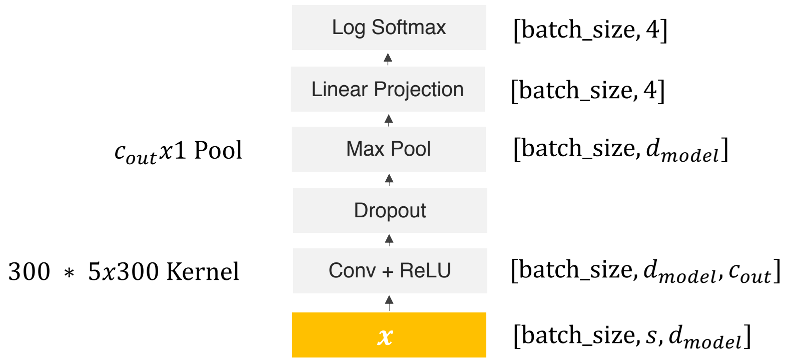


Figure 4.3.: Visualization of the operations of an exemplary ABSA Convolutional Head (CNN-H). This specific head uses a kernel size of 5 and 300 filters.

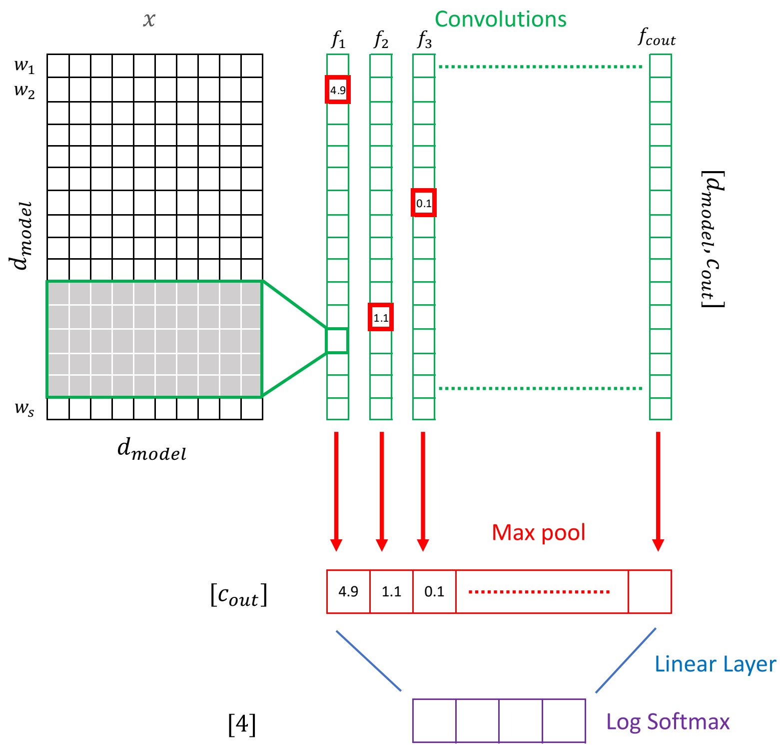


Figure 4.4.: Visualization of the convolutional and max pooling operations of an ABSA-T CNN-H in detail. w_1 to w_s are the words in a sequence with length s. This figure does not include the batch size as an extra dimension.

4.3.3. Weighted Loss

Each aspect head calculates its own loss value using the predictions for the aspect as the targets. To combat data imbalance, we use a weighted NLL loss which is given as

$$\mathcal{L}_{\text{NLL}} = -\frac{1}{n}\sum_{i=1}^{n} w_i * (y_i \cdot log(\hat{y}_i)) \tag{4.4}$$

where n is the number of classes, w_i is the specific class weight, y_i is the ground truth and $\hat{y}_i$ is the prediction. Since we only use the aspect heads for sentiment classification, i will always be either a sentiment or not applicable. Therefore $i \in \{\text{negative}, \text{neutral}, \text{positive}, \text{not applicable}\}$ and $n = 4$.

The class weights are calculated during data loading. The equation for the calculation of a weight scalar for a class weight for class i is given as

$$w_i = 1 - \frac{x_i}{N} \tag{4.5}$$

where x_i is the sum of all of i-s occurrences for the aspect. N is the total number of times an aspect occurred in the dataset.

Equation 4.5 assigns a low numerical value to classes which occur more frequently. As a consequence, the NLL-loss will be lower for frequent classes with low class weights which reduces the influence of those frequent classes.

4.4. Multi-Task Learning

The way the ABSA-Transformer is build, inevitably necessitates multi-task learning since for each aspect-head a separate NLL-loss is computed. For each of the m aspect classes a loss value is calculated. In the end the mean of all losses is taken as shown in equation 4.6:

$$\mathcal{L}_{\text{MultiTask}} = \frac{1}{m}\sum_{j=1}^{m} \mathcal{L}(f(x), y_m) \tag{4.6}$$

where $\mathcal{L}(f(x), y_m)$ is the NLL-loss of the model f with input the x defined in equation 4.4.

4.4.1. Multitask Task Data Augmentation

As described in section 3.3.2 it is also possible to augment the data by using an auxiliary task in addition to the regular classification tasks.
There are three possible auxiliary tasks to consider:

1. Predict additional label from the source data
2. Use an additional dataset B in combination with the source dataset A and predict labels for the classes in A and the classes in B simulatanously. This approach is similar to transfer learning but instead of training the models sequentially, this approach would train them together.
3. Predict additional aspects of the source data which can be trained unsupervised.

This thesis will focus on the first type of auxiliary task where we will try predict an additional label for the source dataset. As the source dataset we choose the GermEval-2017 data since this dataset provides an additional document-wide sentiment label. This label was chosen since the other aspect heads already perform sentiment analysis so this task is very similar on the one hand but can provide additional data points for the training of the model. In addition, the dataset provides a reasonable amount of training data.
Training with the auxiliary sentiment label is performed by adding an additional sentiment head to the model. During training the loss of the auxiliary tasks contributes to the improvement of the transformer base.
However, during evaluation this task is ignored when calculating the F1-score for the model.

The results for this experiment is located in section 6.4.

4.5. Transfer Learning

Besides, multi-task learning we also perform transfer learning to test if the transformer is able to transfer knowledge from one domain to another domain. The transformer model is often used for language modelling and language understanding. These tasks are essential for every NLP task and in general and sentiment analysis specifically. Hence, the transformer base should be domain independent.

It has been already shown that transferring knowledge from the word embedding layer from one domain to another is not only possible but beneficial for the overall performance [86]. Therefore, in addition to the transformer base we will also use pretrained word embeddings.

Due to the nature of the aspect heads we can not transfer knowledge from one domain to another. Not only are the aspect heads highly domain specific but also aspect specific.

We performed experiments to test this theory. To do this we trained the model on a dataset, stopped training and then reshuffled the aspect heads so that they would need to predict different aspects from there on.
This experiment was performed to answer two questions:

- How much predictive power originates from the transformer and how much is produced by the aspect heads in comparison.
- Is it possible to transfer knowledge from aspect heads to a different domain?

In other words, this was a transfer learning experiment on the same dataset.
The results were in line with our theories that the transformer produces the majority of the domain independent knowledge. After the training was stopped and the aspects were reshuffled, the evaluation score dropped back to a level which would usually be achieved after the first or second epoch. Of course, this experiment did not improve upon the F1 score of the model. After a few epochs the evaluation score was back to the levels were training was previously stopped. This being said, the number of iterations it took to reach that score was shorter than on the first run.

4.5.1. Knowledge Transfer from Amazon Reviews

In section 6.5 we will perform experiments to asses the theory about the transferability of the transformer base. We will train a transformer model on the amazon reviews dataset (section 5.2.4) which we collected just for this experiment. This dataset is very large and balanced along the aspects.

After training is completed we take the models embedding and transformer base and exchange the aspect heads with new heads for the training on the target dataset. For the target dataset we use the organic-2019 dataset (section 5.2.3).

Transfer of Knowledge from Word Embeddings

A lot of information is encoded in the word embeddings. In fact it makes up a vast majority of the trainable network parameters (see section 6.3.2). Furthermore, the transformer is conditioned on the word vectors the embedding produces as they shift from the pretrained embeddings to more domain specific embeddings. This can be observed when we freeze the embeddings during training so that they do not change. This significantly impacts the performance negatively.

Unfortunately, this is a threat for the transferability of the embedding layer. When the embeddings are first created, they are initialized with the vocabulary of the dataset.

Especially, for the amazon dataset there is a huge number of infrequent words (see section 5.2.4) that do not necessarily occur in the target dataset. Even when removing the most infrequent tokens a large percentage of the vocabulary from the amazon dataset is not used for the target dataset which makes a large portion of the embedding domain specific.

On the other hand, there are tokens in the target dataset which do not occur in the amazon reviews dataset. For the organic dataset these tokens are highly domain specific. Most of these tokens are chemical compounds and other words like "Glyphosate" which are very important for this dataset.

To solve this dilemma we combine both vocabularies before we start with the training on the amazon dataset. By doing this we ensure that both tasks are able to use the same embedding layer.
Unfortunately, due to computational restraints on the Graphics Processing Unit (GPU) memory we have to further restrict the size of the shared vocabulary. This means that we can not create embeddings for every token in the dataset and some infrequent tokens are replaced with <*UNK*>.

For this reason, the results of the transfer learning experiments can not be directly compared to the best results on the individual datasets. However, we perform a baseline run with the same vocabulary restrictions so that we are able to asses the effect of the transfer learning experiment.

5. Experimental Setup

The following chapter describes the experimental setup for the discussion of results in chapter 6. The first section of the chapter deals with data preprocessing. Section 5.2 lists all datasets used for evaluations of the models and finally section 5.3 provides detail about the training and evaluation process used to generate the results.

5.1. Data Preprocessing

The following section describes the general data preprocessing steps which were taken for all datasets described in section 5.2. Some of the preprocessing steps are specific to certain datasets and will be described there. All data preprocessing steps can be enabled or disabled to evaluate the impact on the performance of these preprocessing steps. Some of those results will be discussed in section 6.1.3 in chapter 6.

5.1.1. Text Cleaning

The main goal of the text cleaning step is

1. Reduce the number of words which are out of vocabulary
2. Keep the vocabulary size as small as possible.

without changing the semantics of the text.
The first step of the data preprocessing pipeline is the removal of all unknown characters which are not UTF-8 compatible. Those characters can occur because of encoding issues or words outside of the target language.

Contraction Expansion

Before we remove any special characters all contractions are expanded with the goal of reducing the vocabulary size and language normalization. Contractions are shortened versions of several words or syllables. In the English language, vowels are often replaced by an apostrophe. Especially in social media and spoken language a lot of contractions are used. *'I'll've'* and *'I will have'* have the same meaning but if they are not expanded they produce a completely different embedding. *'I'll've'* will produce a (300)-dimensional vector (for glove and fasttext) whereas *'I will have'* will be interpreted as 3 300-dimensional vectors.

The contraction expansion is followed by the replacement of Uniform Resource Locators (URLs) with the token '<URL>' and e-mail addresses with the token '<MAIL>'. E-Mails and URLs are always out-of vocabulary and contain very little information that is worth encoding.

In addition any special characters are completely removed. Dashes ('-') are kept because there are compound-words which rely on dashes (e.g. non-organic).

Spell Checking

When writing comments in social media people tend to make spelling mistakes. Unfortunately, each spelling mistake is an out-of vocabulary word which we want to reduce as much as possible.
Therefore, a spell checker is used to prevent these mistakes. The first spell checker[1] which was evaluated relies on the Levenshtein Distance [37] and a dictionary to determine if a word is spelled incorrectly and to make suggestions which word was meant originally. Although, word replacement suggestions are good, the spell checking is slow especially with large dictionaries.

The second spell checker is called Hunspell developed by László Németh[2]. Hunspell is used in a variety of open- and closed sourced projects such as OpenOffice, Google Chrome or macOS. Hunspell also utilizes the Levenshtein Distance in addition to several other measurements. Both spell checkers suffer from false positives (word is incorrectly flagged as negative) as well as incorrect suggestions. Below are examples of Hunspells suggestions for words it did not recognize:

- taste/flavor -> flavorless
- GMOs -> G Mos
- Coca Cola -> Chocolate
- didn -> did

All of the above replacements are very bad because they change the meaning of the entire sentence.

[1] PySpellchecker: `https://pyspellchecker.readthedocs.io/en/latest/`
[2] Hunspell: http://hunspell.github.io/

Nevertheless, in terms of vocabulary size reduction they are clearly outperforming other techniques as table 5.4 demonstrates. Running Hunspell on the Amazon dataset reduces the original vocabulary size of 1.6 Million by over 80% to about 311,000 unique words. In addition, as column *SP + TR-1* shows there are no tokens which only appear once. The reason for this is, that Hunspell always suggests something. Even words like *^_^b4* are replaced by new words even if it would make more sense to delete those words altogether.

Stemming and Lemmatization

Stemming were also briefly explored, however, they did not provide a significant performance improvement.

Stopword Removal

5.1.2. Comment Clipping

The transformer works with different input sequence lengths within one batch. Therefore, it is possible to group similar sequence lengths together and have arbitrary sequence lengths. Unfortunately, in each dataset there is a small percentage of sequences which are longer than other sequences. Due to the limited computational resources a batch of those long sequences does not fit into GPU memory. Therefore, all sentences are either padded or clipped to a fixed length. This is also a requirement for the CNN-based transformer aspect head since CNN-layers need a fixed number of input channels.

5.1.3. Sentence Combination

Some datasets feature sentence annotations instead of comment annotations. In this case important information for the aspect and sentiment classification could be encoded in previous sentences. Refer to figure XX for an example.

Therefore, n previous sentences are prepended to the current sentence where n is a hyper parameter which can be optimized. Similar to the clipping of comment wise annotations described in the previous section, these sentence combinations are also clipped and padded.

The process starts by repeatedly adding sentences to a stack. All $n-1$ sentences which are too long are cut at the front. The n-th sentence is cut in the back instead. This is done so that in the case of $n = 2$
See section 6.1.3 for the evaluation of this preprocessing step.

5.2. Data

This section describes the four datasets which were used for the evaluation of the ABSA-Transformer architecture described previously.
The first dataset - CoNLL-2003 - is used to evaluate just the transformer model without the use of aspect heads. The task of this dataset is word level NER prediction. Since the original transformer model provides predictions on the word level this is good task to evaluate just the transformer part.

GermEval-2017 described in section 5.2.2 is a dataset for aspect-based sentiment analysis and contains over 25,000 review documents from social media.

Organic-2019 is a very recent dataset, also providing an aspect-based sentiment analysis task in the domain of organic food. Whereas, GermEval-2017 contains document level annotations, Organic-2019 contains word level over 10,000 annotated sentences. Organic-2019 is described in section 5.2.3.

Finally, section 5.2.4 describes a new dataset consisting of Amazon reviews which was created with the goal to provide a big dataset as the source for transfer learning. The dataset contains almost 1.2 million reviews with 20 domains spanning the Amazon product catalog.

5.2.1. CoNLL-2003 - Named Entity Recognition

The CoNLL-2003 shared task contains datasets in English and German for Named-entity recognition (NER) [19]. NER describes the task of assigning labels to individual words. The four labels which are used for CoNLL-2003 are *persons*, *locations*, *organizations* and *names* [19]. For example the sentence "Gerry is a researcher at TUM in Munich" would be labeled as "[*PER* Gerry] is a researcher at [*ORG* TUM] in [*LOC* Munich]".

The English data which was used for this research consists of news stories which occurred between August 1996 and August 1997 [19]. The English dataset contains a total of 22,137 sentences with 301,421 tokens and is reasonably balanced in comparison to the datasets described in the next sections. Table 5.1 shows the distribution of the labels and the number of samples for each data split.

5.2.2. GermEval-2017 - Customer Feedback on Deutsche Bahn

GermEval 2017 is a dataset for Aspect-Based Sentiment Analysis on customer feedback about *"Deutsche Bahn"* in German [82]. *"Deutsche Bahn"* is the largest railway operator

	Articles	Sentences	Tokens	LOC	MISC	ORG	PER
Train	946	14,987	203,621	7140	3438	6321	6600
Validation	216	3,466	51,362	1837	922	1341	1842
Test	231	3,684	46,435	1668	702	1661	1617

Table 5.1.: Number of samples and labels for each split in the CoNLL-2003 English NER dataset

in Europe[3]. All data is collected from social media, blogs and Q&A pages over the course of one year from May 2015 till June 2016. Each document is annotated with a relevance flag, a document-level sentiment polarity as well as up to 19 different aspect-sentiment combinations such as atmosphere (*Atmosphäre*) or the experience of buying a ticket (*Ticketkauf*).

GermEval-2017 is a shared dataset for four different tasks:

1. Task-A: Relevance Detection
2. Task-B: General Document Sentiment Classification
3. Task-C: Aspect-Based Sentiment Analysis
4. Task-C: Opinion Target Extraction

This work focuses on Subtask C and results for the aspect-based sentiment analysis are reported in section 6.3.1.

Beating the baseline systems of GermEval is not trivial since the dataset is extremely skewed towards the dominant category 'general' (*Allgemein*). This category makes up 62.2% of all the samples in the dataset. Some categories contain less than 50 samples which is only 2% of the whole data. Almost half of the aspects have less than 1% share of the total amount of samples. There is even one aspect *QR-Code* which has a total of two samples and none in the training split. Table 5.2 provides the detailed breakdown of the number of samples per aspect.
This imbalance is the reason why the GermEval-2017 majority class baseline is extremely strong. In fact, during the GermEval-2017 challenge there was only one other model submission from Lee et al [36] that could outperform the baseline models [82].

In addition, there are some issues with the evaluation metric that the organizers of GermEval-2017 provide. Section 5.3.1 deals with this issue in detail.

[3]Financial Earnings Presentation 2014: `https://ir.deutschebahn.com/fileadmin/Deutsch/2014/Anhaenge/2014_finanzpraesentation_asien_de.pdf`

Aspect	Test-1	Test-2	Train	Val	Total	Ratio
Allgemein	1398	1024	12138	1475	16035	62,16%
Atmosphäre	148	53	1046	139	1386	5,37%
Auslastung & Platzangebot	35	20	251	33	339	1.31%
Barrierefreiheit	9	2	64	17	92	0.36%
Connectivity	36	73	257	23	389	1.51%
DB App & Website	28	18	185	23	254	0.98%
Design	4	2	31	4	41	0.16%
Gastronomisches Angebot	3	3	44	4	54	0.21%
Gepäck	2	6	18	3	29	0.11%
Image	0	3	51	7	61	0.24%
Informationen	58	35	330	34	457	1.77%
Komfort & Ausstattung	24	11	153	21	209	0.81%
QR-Code	1	0	0	1	2	0.01%
Reisen mit Kindern	7	2	44	4	57	0.22%
Service & Kundenbetreuung	63	27	486	49	625	2.24%
Sicherheit	84	42	429	63	618	2.40%
Sonstige Unregelmässigkeiten	224	164	1335	145	1868	7.24%
Ticketkauf	95	48	593	70	806	3.12%
Toiletten	7	4	44	5	60	0.23%
Zugfahrt	241	184	1798	190	2413	9.35%
Total	2467	1721	19,297	2310	25,795	100%

Table 5.2.: Number of samples for each aspect per split in the GermEval-2017 shared task dataset.

5.2.3. Organic-2019 - Organic Comments

This dataset was collected and annotated in the end of 2018 and the beginning of 2019. It contains 1,373 comments and 10,439 annotated sentences from Quora, a social question-and-answer website.

Each sentence is annotated with a domain relevance flag, a sentiment and at least one entity-attribute-sentiment triplet. Out of the 10,439 sentences, 5560 sentences are marked as domain relevant. Out of the relevant sentences 668 contain two or more aspect triplets.

There are 9 possible entities, each entity can have one of 14 attributes and the entity-attribute combination is annotated with a three-class sentiment polarity. In theory this combines to a total of 378 possible triplet combinations and 126 entity-attribute combinations. However, there are only 113 actual entity-attribute combinations and some of these combinations only have a few examples in total which makes this dataset even harder to train than GermEval-2017. The appendix contains two figures which show the distribution of the entities (figure A.2) and the attributes (figureA.3).

Since training on the full number of entities and attributes is very challenging the dataset also provides a coarse-grained version which combines both aspects and entities into a total of 18 bigger sets. The distribution for this dataset version is visualized in figure 5.1.

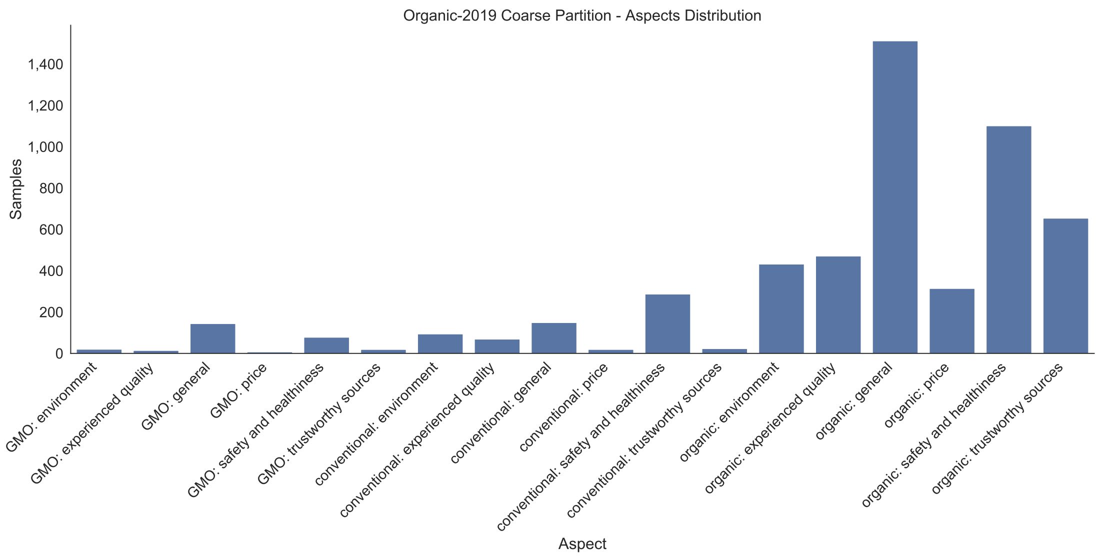


Figure 5.1.: Distribution of the *coarse-grained* aspects in the Organic-2019 dataset

5.2.4. Amazon Reviews Dataset

The Amazon Reviews Dataset consists of over 130 million Amazon product reviews from 1995 until 2015. Therefore, this dataset is one of the richest data sources for sentiment analysis or other related NLP tasks. The raw data is available directly through amazon.[4] The reviews are grouped into 45 product categories such as "Grocery", "Luggage" or "Video Games".

In 2013 McAuley and Leskovec compiled a subset of Amazon reviews [42]. This dataset contains 34,7 million reviews ranging from 1995 till 2013 grouped into 33 categories[5]. The authors also created a "Fine Food" Dataset from Amazon reviews [41] [6]. This dataset consists of 568,454 Amazon reviews from 1995 till 2012. The domain of this specific dataset is related to the organic domain with 273 occurrences of the word 'organic'. Unfortunately, it does not contain predefined aspects so ABSA is not possible without extensive pre-processing to generate aspects out of the reviews.

The datasets created in 2013 contains duplicates so McAuley et. al. generated an improved Amazon Reviews dataset in 2015 without duplicates [44][22]. This iteration of the dataset contains 142.8 million reviews from 1996 till 2014[7]. Due to the size of this dataset the authors provide a smaller dataset which only contains reviews from users who wrote exactly 5 reviews. This 5-core subset features 18 million reviews. The distribution of the domain categories is visualized in figure 5.2. As one can observe the dataset is substantially skewed towards the largest domain 'books' which makes up of 49% of the data.

To combat data imbalance and the sheer size of the dataset we propose a balanced subset of the 5-core dataset with 60000 reviews for each domain aside from *Musical Instruments*, *Amazon Instant Video*, *Automotive* and *Patio, Lawn and Garden*. These categories contain less than 50000 reviews so including them would skew the dataset again. In addition, we also transformed the star-rating to the common negative-neutral-positive rating schema. Similar to Blitzer et. al. we interpret $1-2$ stars as negative, 3 stars as neutral and $4-5$ stars as positive sentiment [7].

To create a balanced dataset not only on domains but also on sentiment we sampled 20000 reviews for each sentiment for each domain. Overall, there are more positive reviews than neutral or negative reviews. Thus, some domains contain less than 20000 reviews per sentiment category. To prevent data imbalance, reviews from the remaining other sentiment categories are sampled so that each domain contains 60000 reviews in

[4] `https://s3.amazonaws.com/amazon-reviews-pds/readme.html`
[5] Available through Stanford `https://snap.stanford.edu/data/web-Amazon.html`
[6] Available through Kaggle `https://www.kaggle.com/snap/amazon-fine-food-reviews`
[7] Available here: `http://jmcauley.ucsd.edu/data/amazon/`

Figure 5.2.: Number of reviews per domain category in the amazon review dataset by McAuley et. al. [44]

sum. This distribution and additional statistics about the dataset are documented in table 5.3.

Token Removal

There are over 145 million words in the dataset. These words combine into a vocabulary size of 1.6 million unique tokens and consequently into a very large embedding layer. (In comparison: the Organic2019 dataset has a vocabulary size of just 11,685.) Two techniques were used to reduce the vocabulary size:

1. Spell checking words
2. Removing rare tokens

The process for the first technique is described in section 5.1.1. Another way to reduce the vocabulary size is by removing tokens, that only occur once or twice. These tokens make up the majority of the vocabulary size but only a small percentage of the overall word count. Table 5.4 shows the proportion of tokens which only occur 1, 2, or 3 times. As demonstrated in the table, infrequent tokens are very rare (all the tokens with one occurrence make up only 0.33% of the whole dataset). Yet, infrequent tokens make up over 74% of the total vocabulary size. Removing all tokens with one occurrence, therefore reduces the vocabulary size by 74% but only 0.33% of information is lost.

Domain Category	helpful mean	Pos. Count	Neu. Count	Neg. Count	stars mean	# words mean	std
Apps for Android	0.22	20000	20000	20000	3.03	47	50
Baby	0.29	17012	17255	17012	3.33	105	106
Beauty	0.32	20000	20000	20000	3.10	90	94
Books	0.43	20000	20000	20000	3.08	176	201
CDs & Vinyl	0.44	20000	20000	20000	3.11	172	168
Cell Phones & Accessories	0.19	20000	20000	20000	3.06	93	138
Clothing Shoes & Jewelry	0.26	20000	20000	20000	3.11	67	70
Digital Music	0.53	47410	6789	5801	4.19	202	190
Electronics	0.43	20000	20000	20000	3.06	122	138
Grocery & Gourmet Food	0.33	28790	17514	13696	3.53	99	97
Health & Personal Care	0.35	20000	20000	20000	3.09	95	126
Home & Kitchen	0.44	20000	20000	20000	3.08	104	110
Kindle Store	0.35	20000	20000	20000	3.07	111	131
Movies & TV	0.39	20000	20000	20000	3.07	184	198
Office Products	0.29	45342	5060	2856	4.35	148	164
Pet Supplies	0.27	26412	15933	17655	3.35	91	96
Sports & Outdoors	0.30	20751	20000	19249	3.14	94	111
Tools & Home Impr.	0.40	39126	10769	10105	3.90	111	134
Toys & Games	0.32	11005	16357	11005	3.70	108	114
Video Games	0.41	20000	20000	20000	3.07	226	267
Total	0.35	506202	349677	337379	3.31	122	151

Table 5.3.: Dataset statistics for the generated Amazon review subset for the domain categories. This table contains mean helpfulness rating; number of positive reviews; number of neutral reviews; number of negative reviews; mean star rating; mean number of words per review; standard deviation of the number of words per review

	Original	SP	SP + TR-1	TR-1	TR-2	TR-3
Word Count	148,129,490	-	0%	0.329%	0.389%	0.414%
Vocabulary Size	1,594,742	80.51%	80.51%	62.97%	74.41%	79.32%

Table 5.4.: Different vocabulary size reduction techniques. This table shows the proportion of tokens that occur only 1, 2 or 3 times in relation to the total word count and the vocabulary size. *SP* is the spell checked dataset; *TR-n* is the token removal technique were *n* is the number times, tokens can occur in the dataset.

Most of these rare tokens are either incorrectly written (*nthis*), are part of structural elements such as headings (*review=======pros*) or are other unidentifiable characters and digits (*^_^b4*).

5.3. Training and Evaluation

5.3.1. Evaluation

The models that are used in this thesis are stochastic models since model parameters are randomly initialized. In addition, samples within the training batches are randomly shuffled. Therefore running the model multiple times leads to different results.

This means that it is necessary to collect model results multiple times. Unfortunately, k-fold cross validation is not possible for three out of the four datasets since the creators of the datasets provide a predefined split and changing the split during k-fold cross validation would prevent comparability with other results.
Therefore, for each dataset-result we repeat the experiment 5-times and report the mean and standard deviation. Iyer and Rhinehart suggest to run an experiment up to a 1000 times to get an optimal result [29]. However, this is not possible for our models due to computational constraints.

All experiments on hyper parameters are performed once with a fixed seed of 42. This should make sure that all experiments on hyper parameters are reproducible. There are however some cudnn functions which are non-deterministic which means that even though a random seed is set the results could differ when running the same model with the same parameters multiple times.

Table 5.5.: Example for GermEval-2017 evaluation. None sentiment is not shown. Document 1 is predicted correctly. Document 2 has a correct prediction for aspect A but an incorrect prediction for the sentiment of aspect B (in bold).

	Gold	**Prediction**
Document 1	A : negative	A : negative
Document 2	A : positive B : **positive**	A : positive B : **negative**

GermEval 2017 - Evaluation

Wojatzki et al. [81] provide an evaluation script for their dataset GermEval-2017. All results from the GermEval 2017 challenge were evaluated using this dataset. Therefore, all results reported in this thesis also use the evaluation script to calculate the f1 score. This is done to be able to compare the results on this datasets to other approaches on this data.
Unfortunately, there are irregularities in the calculation of the micro f1 score. The evaluation script first creates every possible permutation of the combination of aspect and sentiment. If there are just two aspects (Aspect A and Aspect B) and four sentiments (n/a, negative, neutral, positive) this will generate 8 combinations (A-n/a, A-negative, ..., B-positive). This is used as the first input (*aspect_sentiment_combinations*) of the GermEval-2017 evaluation algorithm shown in 1.

In the next step, all gold-labels and predictions are paired together for each document based on the specific aspect-sentiment combination. The example in table **??** will produce the following combinations where the left side represents the gold labels and the right side the predictions. This would be the second input parameter *golds_predictions* for algorithm 1:

1. A:neg - A:neg (Document 1)
2. A:pos - A:pos (Document 2)
3. B:pos - B:n/a (Document 2)
4. B:n/a - B:neg (Document 2)

Using these inputs the algorithm will compute the following results:

- True Positives: 2
- False Positives: 2

- False Negatives: 2
- True Negatives: 26

which results in an f1-score of 0.5. In this example there is one misclassification where instead of predicting a pos. sentiment for aspect B the classifier predicted a neg. sentiment. When looking at the combination B:pos as the *'true class'* the model predicts a negative (NOT pos. sentiment) when in reality this is a positive (pos. sentiment) which is the definition of a *'False Negative'*. When looking at the combination B:neg as the *'true class'* the model predicts a positive (neg. sentiment) when in reality this is a negative (NOT neg. sentiment) which is the definition of a *' False Positive'*.

One could therefore argue that instead of producing two False Positives and two False Negatives the correct evaluation should be one False Positive and one False Negative.

Algorithm 1: GermEval-2017 Evaluation script.

```
Input   : aspect_sentiment_combinations: List of all possible combinations
          between aspects and sentiments including n/a, golds_predictions List of
          all comment wise pairs between gold labels and prediction labels
Output: (tp, fp, tn fn)
tp = 0 fp = 0 tn = 0 fn = 0
foreach (aspect, sentiment) in aspect_sentiment_combinations do
    foreach (gold), (pred) in golds_predictions do
        if gold matches current aspect and sentiment then
            if gold matches prediction then
                tp++
            else
                fn++
            end
        else
            if prediction matches current aspect and sentiment then
                fp++
            else
                tn++
            end
        end
    end
end
return (tp, fp, tn, fn)
```

Table 5.6.: Hardware used for model training

	OS	CPU	RAM	GPU
Desktop	Windows 10 (17134)	Core i5-6500 @ 3.20GHz	16 GB	GTX 1060
Social 5	Ubuntu 16.04.5	Xeon E5-2643 v3 @ 3.40GHz	126 GB	GTX 970
Azure	Ubuntu 16.04.5	Xeon E5-2690 v3 @ 2.60GHz	55 GB	Tesla K80
Google	Ubuntu 16.04.5	Xeon E5-2670 v3 @ 2.60GHz	15 GB	Grid K520

5.3.2. Hardware

Training and evaluation of the models was done on four different machines. One of the servers belongs to the faculty of applied informatics, one is a local desktop machine and the last two are cloud instances. One is an Azure virtual compute instance with 8 Central Processing Unit (CPU) cores and 28 Giga Bytes (GB) of Random Access Memory (RAM) and the other is a Google Cloud GPU compute instance instance with an Intel Xeon E5-2670 processor, 15 GB of RAM and a NVIDIA Grid K520 GPU. See table 5.6 for more details.

5.3.3. Docker

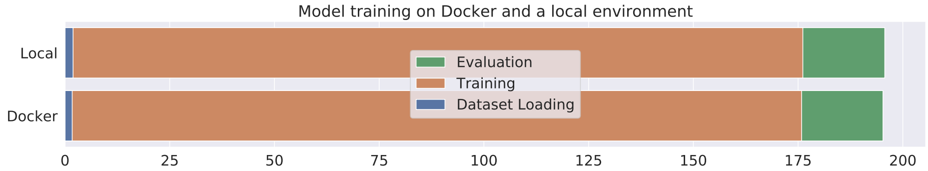


Figure 5.3.: Docker vs. Local environment - Comparision of model training times.

Docker[8] is a framework for container virtualization. Docker containers use the same kernel as the host system but an isolated file system with own system libraries.

Since training was performed on four different environments a Docker image was created which automates the installation of all required frameworks, environments, drivers and versions. An automated build pipeline builds a new image as soon as a new code version is pushed to the repository. Users can install or update an image directly from Docker Hub without rebuilding it every time locally.

The main concern of using Docker for resource intensive task is the loss of performance due to the virtualization overhead. To evaluate this, epoch training time was measured

[8]Docker: `https://www.docker.com`

with and without Docker in a Compute Unified Device Architecture (CUDA) environment. The experiment was performed on Social 5 displayed in table 5.6. For both experiments a complete model was trained for 5 epochs on the Organic2019 dataset. Figure 5.3 visualizes the time each part of the training took. For both environments the mean execution time was around 195 seconds. This means that there is no difference between running a model inside a Docker container or just locally. However, this is only the case when the host is running on a linux environment. On Windows and macOS, Docker has to virtualize part of the linux kernel. Therefore, there is no advantage of running on the exact same kernel as the host system. In addition, At the time of writing, the NVIDIA-runtime[9] is only supported for linux environments.

[9]NVIDIA Docker Runtime: `https://github.com/NVIDIA/nvidia-Docker`

6. Discussion of Results

6.1. Hyper Parameter Optimization

The following presents the results of the hyper parameter optimization on GermEval-2017 Task C and the Organic-2019 Coarse category dataset. We evaluate the performance of Hyperopt compared to a random search. Then, the next sections show how certain parameters impact the model performance.

6.1.1. Hyperopt Evaluation

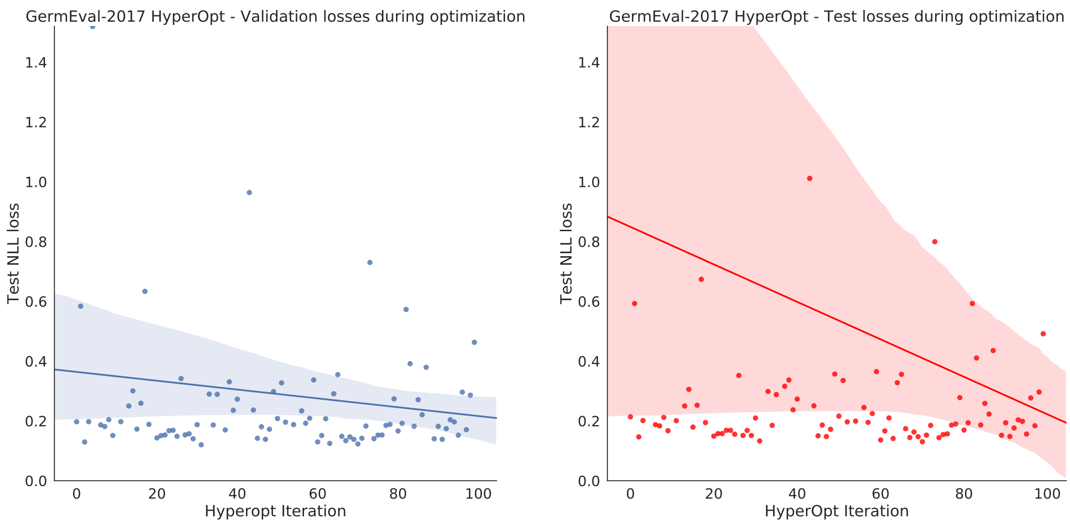


Figure 6.1.: Validation- (blue - left) and test (red - right) losses during 100 Hyperopt iterations on GermEval-2017 dataset

To evaluate Hyperopt three evaluation runs with 100 iterations were performed.

1. GermEval-2017 - TPE on validation loss
2. GermEval-2017 - Random search

3. Organic Coarse Grained - TPE on validation loss

Figure 6.1 visualizes the improvement of the validation- and test losses on the GermEval-2017 dataset after 100 Hyperopt iterations. Is seems as if the regression line is negative in both cases which means that the TPE algorithm Hyperopt uses suggests better results as the time moves on.
Unfortunately, the Ordinary Least Squares (OLS) analysis A.1 and A.2 in the appendix show that the negative correlation is in fact not significant. This implies that the TPE algorithm does not sample parameters from the space which actually improve the loss of the model. This is even more obvious in figure 6.2. This figure shows the development of F1-Score during optimization. While the results on the left for GermEval might look like they improve over the course of the optimization the results[1] for the optimization of the coarse organic dataset clearly show no improvement.

There are several possible explanations which could contribute to this behaviour:

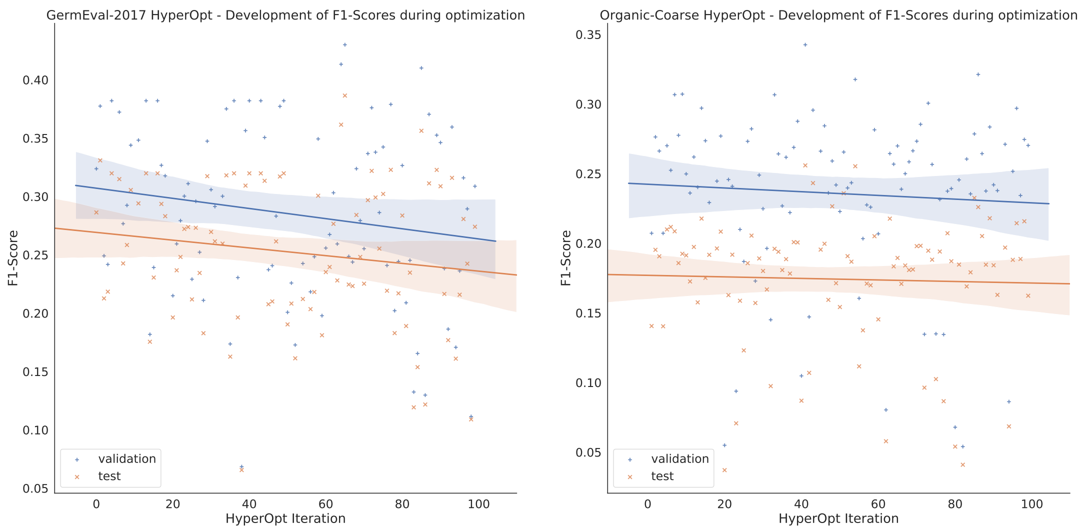


Figure 6.2.: F1 Scores of the hyperparamter optimization of GermEval-2017 (left) and Coarse Organic 2019 datasets (right).

Iterations

First, it could be possible that 100 iterations are not enough to provide a stochastic model which is able to make good predictions in the hyperparameter space. It is worth

[1]The improvement is still not significant (0.340)

GermEval-2017					
	count	mean	std	min	max
Aspect Head					
Warmup Iterations 1 - 10*					
CNN-A	6	0.267644	0.050316	0.212655	0.330922
MLS-A	3	0.294608	0.032134	0.258432	0.319838
TPE Iterations 1 - 100					
CNN-A	67	0.249094	0.066835	0.065565	0.386465
MLS-A	23	0.261533	0.044603	0.181078	0.356296

Table 6.1.: Result of sampling of Aspect Head choices during TPE Hyperopt optimization. Values show micro F1-score achieved by models on the GermEval-2017 dataset. * The 10th iteration failed which is the reason why the warmup does not sum up to 10.

noting that Bergstra et. al. show that hyperopt outperforms random search within 200 trials [6]. However, for most architectures it is not feasible to run an optimization search for much longer than 200 iterations let alone the 1000 iterations they claim as the point where hyperopt converges.

It is also worth mentioning that the Hyperopt module uses a random sampler for the first 10 iterations to get data points to initialize the TPEs. Decreasing this number could yield to better results for computational expensive models since the algorithm is forced to suggest values earlier.

Hyperparamter Search Space

It is possible that the hyperparameter search space which Hyperopt uses to generate new parameters is too large. Table A.3 shows the hyperparameter search space for the optimization of the GermEval-2017 dataset. There are parameters which do not change the outcome by a huge margin and then there are parameters which decide whether or not the model trains at all. However, finding those parameters is a challenging task.

Warmup Phase

TPE supports tree structures for the search space. In the search space used for the optimization there is one tree-like parameter which is the choice of the aspect head architecture. TPE can either choose a Mean Linear Sum Aspect Head (MLS-A) or a

CNN-based Aspect Head (CNN-A). The MLS-A does not have additional parameter nodes, whereas the CNN-A has 4 additional parameters.

MLS-A has a higher mean F1-score of 0.263 compared to CNN-A which achieves 0.250. Despite the higher mean score, TPE only sampled MLS-A 23 times compared to 67 times.
This becomes even more interesting when looking at the TPE warmup phase. During warmup, MLS-A is chosen 3 times and CNN-A is chosen 6 times. This is roughly the same distribution compared to the later TPE iterations.
For greater detail refer to table 6.1. There is also a violinplot which visualizes the impact of the aspect head choice in the appendix as figure A.4.

In contrast, during the warmup phase of the hyperopt run on the coarse organic dataset, Hyperopt sampled CNN-A 2 times and MLS-A 7 times which is exactly the other way around. During the TPE iterations, CNN-A was sampled 14- and MLS-A 71 times. Again, this is the exact opposite of the previous optimization on the GermEval-2017 dataset.

This leads to the following conclusion: During the warmup phase of Hyperopt the search space is randomly sampled. This random sampling distribution is adhered to during the whole TPE suggestion phase. This leads to results which are heavily dependent on the first 10 random iterations.

Comparison with a Random Search

To confirm the findings above a completely random search was performed by Hyperopt on the same number of iterations. The result of this comparison is plotted in figure 6.3. This violinplot visualizes the result of both optimization runs. The light blue density curve on the left shows the F1 scores from the TPE generated models. A wider body on the density curve means that more observations for this particular score value were recorded at this part. The black dots correspond to the actual individual F1-Score observations. The dark blue parts and the white dots correspond to the F1-scores which originated by randomly generated hyperparameters.

6.1.2. Model Parameters

Due to the high dimensionality of the hyperparameter search space, statistical significance tests will not show any significant correlations between the parameter and an improvement of the F1-Score. For each single parameter change, all other parameters will also change and they influence the model as well. While not possible to evaluate

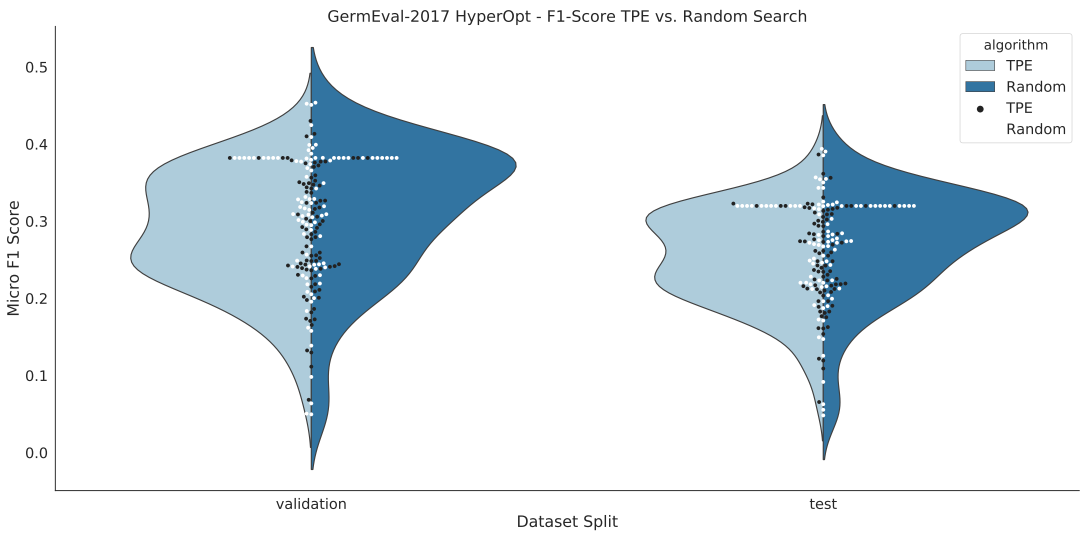


Figure 6.3.: Comparison of HyperOpt TPE algorithm against a classical random search. The results for TPE are light blue on the left, whereas results for the random search are a deep blue on the right.

the results with statistical significance it is possible to derive certain assumptions from the data which will be discussed in the following sections.

Aspect Heads

As discussed in section 6.1.1 it is not entirely possible to favor one or the other aspect head. Both can provide similar results.
Figure 6.4 shows the impact of the two CNN-A parameters 'Kernel Size' and 'Number of Filters'. The number of filters does not seem to impact the result. However, there is a (statistical[2]) significant negative correlation between the kernel size and the model performance.
Smaller filters lead to a performance improvement compared to bigger filters. This result seems to follow the literature. For instance, Schmitt et. al. use filter sizes of 3, 4 and 5 [67].

There is no significant change for the other two parameters 'Kernel Padding' and 'Kernel Stride'. The impact on the F1-Score of both parameters is visualized in figure A.5 in the appendix.

[2]Significant at a p-value of 0.05

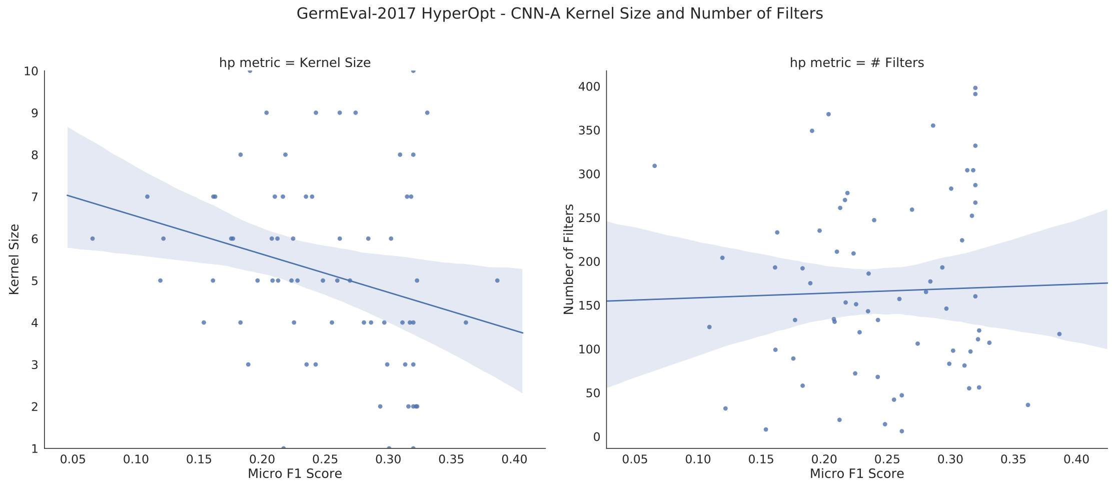


Figure 6.4.: Impact of CNN parameters on micro F1-score. The graph on the left shows the impact of the kernel size on the model performance. The graph on the right depicts the influence of the number of filters on the F1-score.

Point-wise Feed-Forward Layer Size

In the original transformer model the inner Point-wise Feed Forward (PWFC) has a dimensionality of 1024 while the model size has a dimensionality of 512 [80]. This is a 2x increase over the model size. Due to the availability of pretrained Glove or Fasttext embeddings our ABSA-T model only uses a model size of 300. Consequently, the inner Point-wise Feed Forward (PWFC) layer dimensionality should be around 600. However, layer sizes above 300-400 neurons quickly lead to interesting model behavior. After a few training iterations the PWFCs transform every input to the exact same output. In other words, no matter what the model gets as input it always predicts the same output.

The solution for this overfitting behavior is to use a smaller inner PWFC. Values from 100 to 200 neurons lead to the best results.
This is extremely interesting since it completely changes the task of the PWFCs. From a layer with a higher dimensionality than the model to a bottleneck layer with a lower dimensionality. A bigger layer may allow for more complex and expressive features to be learned while a smaller bottleneck layer limits the expressiveness and forces the network to focus on crucial features [61].

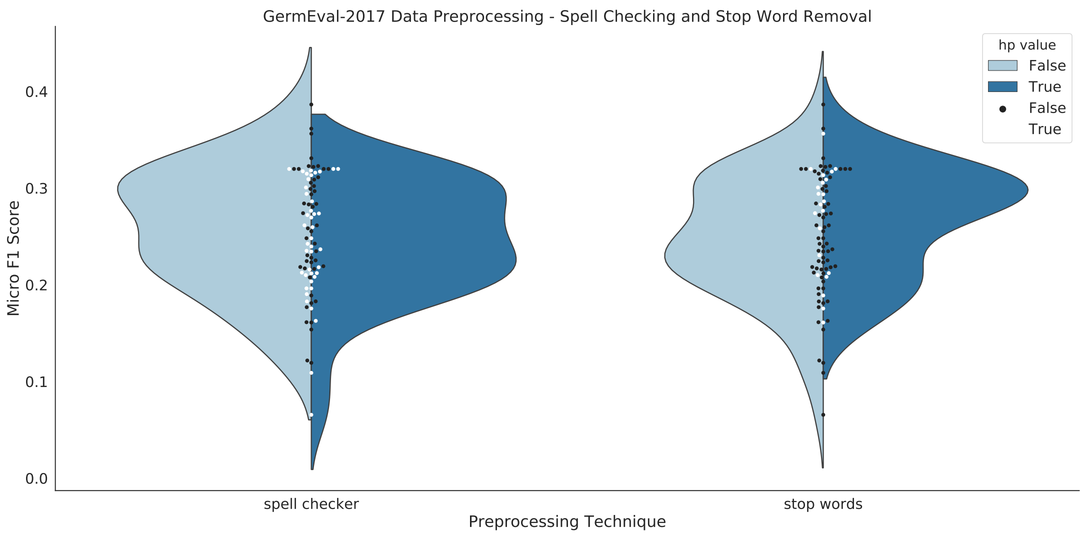


Figure 6.5.: Comparison of preprocessing techniques - Impact of Spell checking and Stop Word Removal on Validation Micro F1-Score

6.1.3. Data Preprocessing

In the following section we discuss the impact of the preprocessing steps and how they affect the overall model performance.

Spell Checking

The left side of figure 6.5 shows the impact of spell checking on the GermEval-2017 dataset. In this instance, spell checking negatively impacted the performance of the classifier. There are a few explanations for this performance.

Social media content contains a lot of special characters and words which are not part of a regular dictionary. However, especially those words might carry the most sentiment. By replacing those words it is possible that a lot of information is lost.

Tweets and forums posts about travel contain a lot of special abbreviations that spell checkers do not recognize. Persons might be talking about the bad performance of the public transport operator Münchner Verkehrsgesellschaft (MVG) but the spell checker replaces 'MVG' with 'mag' which changes the sentence dramatically.

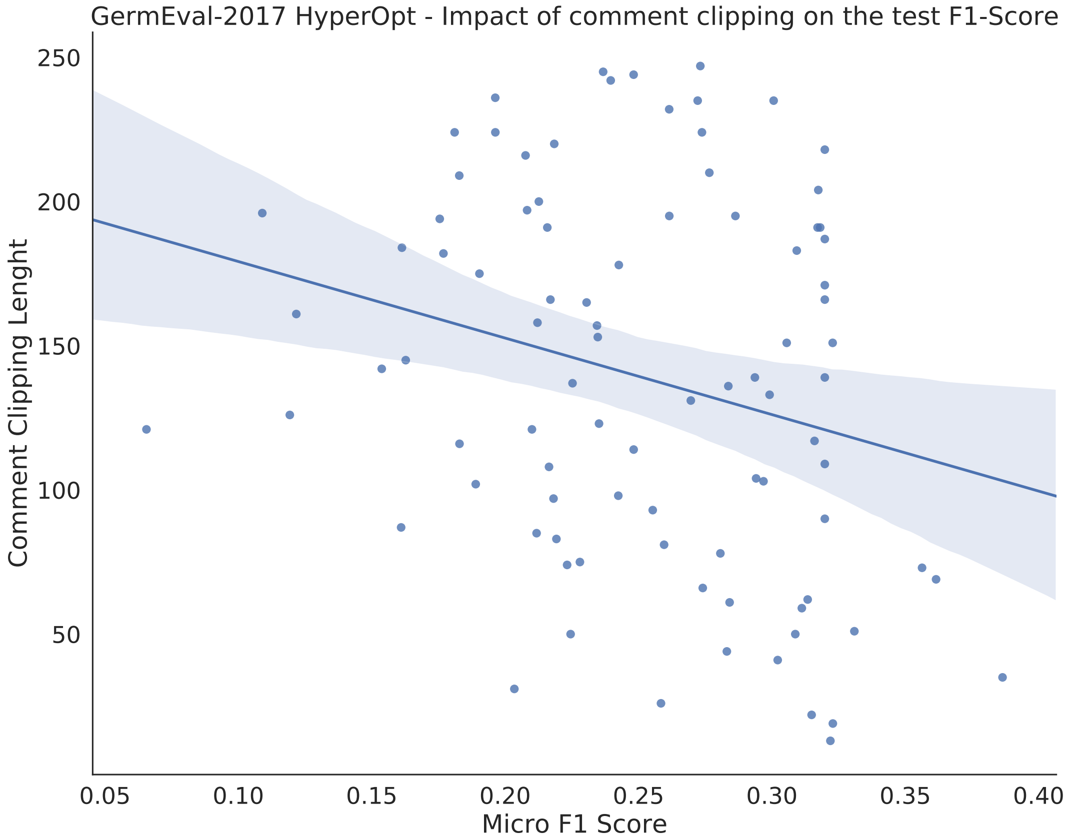


Figure 6.6.: Comparison of preprocessing techniques - Impact of Comment Clipping on Test Micro F1-Score

Stop Word Removal

Stop words are a group of words which are very common in a language but carry little actual information. Examples for stop words are 'the', 'is' or 'what'. The results for stop word removal is shown in figure 6.5 on the right side for GermEval-2017. In this specific instance, removing them improves performance. However, this was not as clear for the organic dataset where removing stop words did not significantly improve the performance.

Comment Clipping

Comment clipping refers to the technique of cutting a sentence or comment after a specific number of tokens. In other words, comments which are too long are shortened and comments which are too short are padded.
Figure 6.6 provides a visualization how comment clipping impacts the model performance on the GermEval-2017 dataset. The regression line shows that a shorter sequence length may improve performance of the overall model[3].

This observation seems to be reasonable since GermEval-2017 contains a mix of very short tweets (140/280 character limit) as well as long newspaper articles. Aligning them to an overall shorter length logical especially considering that most longer newspaper articles include a short summary in the beginning. Cutting those long documents helps the transformer to focus on important information instead of spreading out the attention.

6.2. Results for Named Entity Recognition

The following section contains the final results for the Named-entity recognition (NER) task of the CoNLL-2003 dataset.
Since this task was an auxiliary training task to assess the performance of the transformer part, no hyperparameter tuning was performed.

For this dataset the architecture is slightly different, because CoNLL-2003 has annotations for each word. Since the transformer makes predictions per word there is no need for separate aspect heads.
Therefore, this architecture follows the original transformer and consists of a single linear layer to project the 300-dimensional per-word prediction down to the amount of classes to predict. Finally, a log-softmax is used to provide the log probabilities for the class labels.

For the final evaluations we use transformer model with two encoder blocks, each consisting of two attention heads. We employ a model dropout rate of 0.3 and a pointwise layer size of 300. Furthermore, we use FastText embeddings a batch size of 12 and a weight decay of $1e^{-6}$.

Table 6.2 lists the result of our and other submissions for this tasks in order to assess the fitness of this model. The vanilla transformer model already achieves a very competitive performance. It outperforms every original submission for CoNLL-2013 by a wide

[3]Significant at a p-value of 0.05

Variant	**Macro F1**		**Micro F1**	
	dev	*test*	*dev*	*test*
Random Classifier	0.147	0.187	0.210	0.214
CoNLL-2003 Baseline	-	-	-	0.596
CoNLL-2003 Best Result	-	-	-	0.888
Baevski et. al 2019 (SOTA)	-	-	0.969	**0.935**
Transformer (our)	0.822	0.766	0.939	0.918

CoNLL-2003 - NER

Table 6.2.: Results of models on the shared CoNLL-2003 task for NER. The random classifier acts as a minimal baseline as it will predict completely random classes for each sample. The CoNLL-2003 baseline [19] was provided for the CoNLL-2003 NER competition. The best result at the competition achieved an F1-score of 0.8876 [20]. At the time of writing (April 2019), the best result on CoNLL-2003 NER is achieved by Baevski et. al. [2]

.

margin and is only slightly behind current state of the art [2] even though it was not exclusively created for this purpose.

The normalized confusion matrices in figure 6.7 show the performance of the transformer on the dataset for the individual classes. The class 'LOC' is the most frequent class in the dataset (see table 5.1) aside from class 'O' which is the 'Other' class. Despite this fact, the class performed poorly considering the 'MISC' class which achieved a similar result only has half the training samples.

6.3. Results for Aspect-Based Sentiment Analysis

The following sections outline the results on Aspect Based Sentiment Analysis (ABSA). The first section contains the results for the GermEval-2017 dataset. This dataset uses a special evaluation method. To be able to compare our results we also use the evaluation method that GermEval-2017 provides.
The method GermEval-2017 uses is described in section 5.3.1.

Section 6.3.3 reports results on the Organic-2019 dataset and section 6.3.2 discusses the results on the Amazon Reviews dataset.
Finally, section 6.4 and 6.5 review the results of multitask learning and transfer learning.

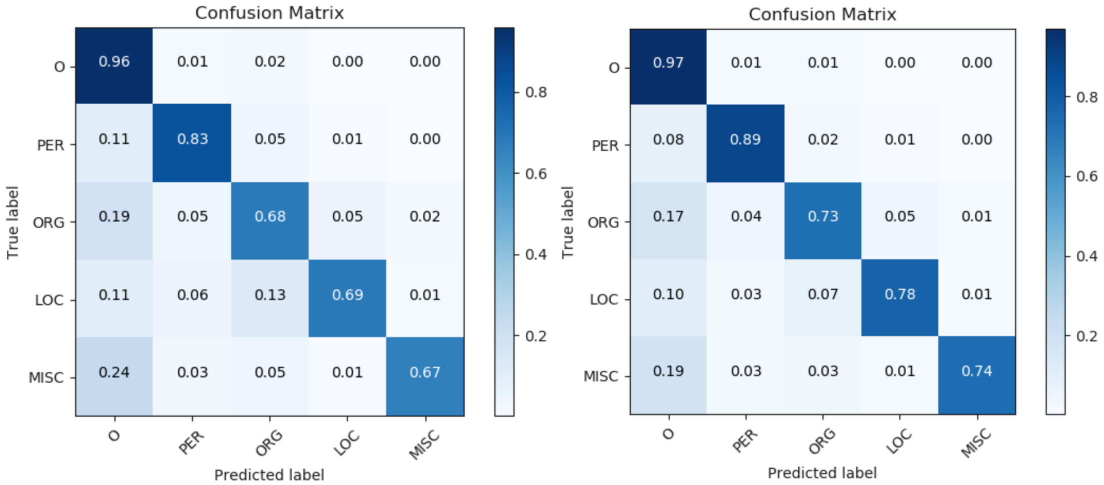


Figure 6.7.: Normalized confusion matrices for the NER task of the CoNLL-2003 dataset. The matrix on the left shows the performance of the model on the test set while the right confusion matrix visualizes the performance on the validation / development set.

6.3.1. Results for GermEval-2017

The GermEval-2017 dataset for Aspect Based Sentiment Analysis (ABSA) is a large dataset for multi label aspect sentiment detection. In addition it is reasonably closely related to the Organic-2019 dataset which is the reason why this dataset was used to evaluate the model. Both datasets use posts form social media. Therefore, the task of detecting aspect-based sentiment is similar. This dataset is also much larger than Sem-Eval-14, -15 or -16.

Furthermore, Schmitt et. al. - one of the inspirations for this thesis - use GermEval to evaluate their model. To compare the base transformer model we used the CoNLL-2003 NER task. To compare the whole architecture against other approaches it makes sense to use GermEval since there are other results which we can use to compare the performance of the ABSA-Transformer (ABSA-T) model.

Table 6.3 reports on the result of the evaluation of our ABSA-T against other GermEval models. The first three results are models created for the GermEval-2017 competition. Wojatzki et. al. provide two baseline systems for the ABSA task [82]. The first one is a majority class baseline which just predicts the class which contains most samples. For this dataset this is the class "Allgemein - Neutral".

Since GermEval is very skewed towards this majority class the baseline is very strong.

Variant	Macro F1		Micro F1	
	synchronic test	*diachronic test*	*synchronic test*	*diachronic test*
	GermEval-2017 competition [82]			
Majority Baseline	-	-	0.315	0.384
SVM Baseline	-	-	0.322	0.389
Best Submission	-	-	0.354	0.401
	Schmitt et. al. 2018 (SOTA) [67]			
Pipeline LSTM + FT	-	-	0.297	0.342
End-to-end LSTM + FT	-	-	0.315	0.384
Pipeline CNN + FT	-	-	0.295	0.342
End-to-end CNN + FT	-	-	**0.423**	**0.465**
	Dugar 2019 [18]			
End-to-end LSTM + FT	0.56	-	0.384	-
	Our Results			
Random Classifier	0.014	0.014	0.018	0.018
ABSA-T + LM-H + FT	0.131	0.111	0.390	0.413
ABSA-T + CNN-H + FT	0.102	-	0.352	-

GermEval-2017 - Task C (ABSA)

Table 6.3.: Evaluation and comparison of models on the shared GermEval-2017 task for ABSA. The random classifier acts as a minimal baseline as it will predict completely random classes for each sample. The organizers of GermEval-2017 provide two baselines [82]. The first, just predicts the majority class which is *"Neutral-Allgemein"*. This baseline already achieves a score of 0.315 on the synchronic test test. The second baseline uses a SVM classifier and achieves strong results very strong results of 0.322 and 0.389. The best submission for the GermEval-2017 challenge was achieved by Lee et. al. [36] with a score of 0.355 and 0.401. Schmitt et. al. achieved SOTA in 2018 with an End-to-end CNN model with custom FastText (FT) embeddings. Their best score is 0.423 on the synchronic- and 0.465 on the diachronic test set. Our ABSA-T model with linear mean heads and FastText embeddings outperforms every submission of the GermEval-2017 competition and is competitive with the end-to-end CNN model of Schmitt et. al.

Refer to figure A.1 for an overview of the aspects and their distributions. The second baseline system is a SVM classifier. At the time of the competition this baseline was very competitive and achieved the second place. Finally, the best submission for GermEval-2017 by Lee et. al. achieves a micro F1 score of 0.354 on the first test set (synchronic test) and 0.401 on the second test set (diachronic test).

The current State-of-the-art (SOTA) is an end-to-end CNN architecture by Schmitt et. al. [67] with custom FastText Embeddings. This architecture achieved a micro F1 score of 0.423 on the synchronic test set and 0.465 on the diachronic test set.

In comparison, our ABSA-T model achieves a micro F1 score of 0.390 and 0.413 on the first and second test sets. This means that our model outperforms even the best submission for the GermEval-2017 competition. In addition, the ABSA-T model with LM-Hs is competitive with the end-to-end CNN created by Schmitt et. al.

Several techniques can be applied to produce even better results for the task. Schmitt et. al. show that pretraining custom FastText embeddings on the target domain improves the model. Dugar also demonstrated that data augmentation can positively impact the model performance [18]. In section 6.5 we also demonstrate, that transfer learning positively impacts model performance. However, finding similar data in German might be challenging.

6.3.2. Results for Amazon Product Reviews

Variant	**Macro F1**		**Micro F1**		
	dev	*test*	*dev*	*test*	Train Duration
Random Classifier	0.031	0.031	0.031	0.031	-
ABSA-T + LM-H + FT + SP	0.461	0.513	0.461	0.513	49h

Amazon Reviews Dataset

Table 6.4.: Results of the ABSA-T model with mean linear aspect heads (LMH) and FastText (FT) embeddings on the custom amazon reviews dataset. Spell checking (SP) is enabled. The model reported in this table uses the full amazon reviews vocabulary. The Train Duration column reports the duration of the training and evaluation process on a Tesla K80 GPU. Training was performed once with a seed of 42.

Due to the large size of the dataset no hyperparameter tuning was performed. Parameters from previous optimizations on the smaller organic dataset were chosen for the evaluation. The final model uses the LM-H architecture with FastText embeddings. Comments are clipped to a fixed length of 100 and spell checking as well as stop word removal is enabled to reduce the vocabulary size. The model consists of one attention head with d_k and d_v of 300 and two encoder blocks. The inner PWFC-layer acts as a bottleneck with a size of 128. Finally, a reduced batch size of 12 was chosen to be able to fit the model into GPU memory which would not have been otherwise possible.

The vocabulary size of the amazon dataset after all reductions consists of $389,371$ unique tokens. This creates an embedding layer which maps the vocabulary size of $389,371$ to a 300-dimensional vector. Unfortunately, a lot of those tokens are not part of the pretrained embedding so the parameters of the embedding layer can not be locked during training. Therefore, the embedding layer has a size of $116,811,300$ trainable parameters. As a result, the first embedding layer makes up over 99% of the overall number of model parameters which is $117,710,780$.

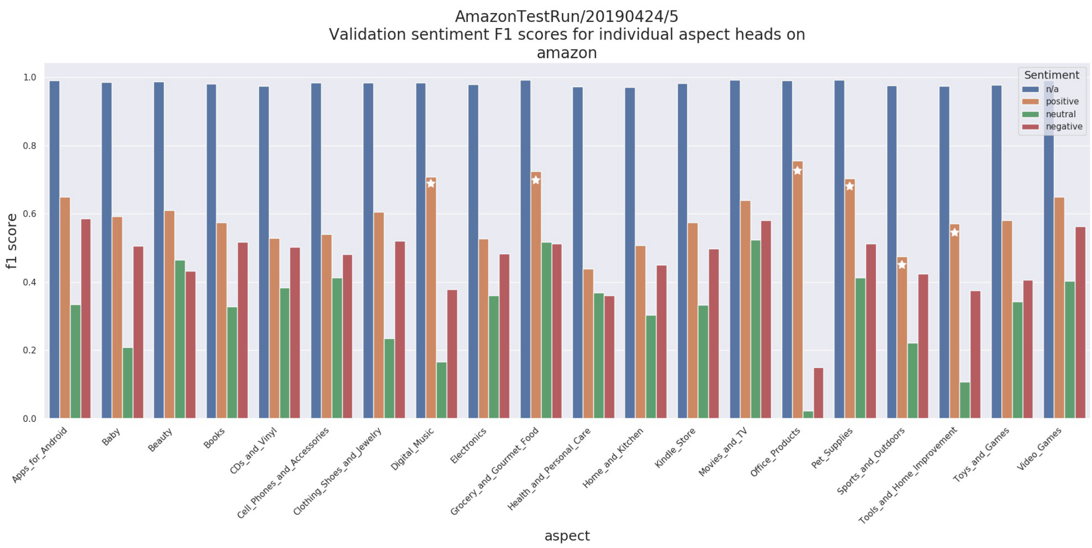


Figure 6.8.: Individual aspect head micro F1 scores achieved on the validation set of the custom amazon reviews dataset. For each aspect head this figure reports the F1 score split by the sentiment class as well as the n/a label. Aspects which have more positive reviews to keep the aspects balanced are annotated with a white star. Those include Digital Music, Grocery and Gourmet Food, Office Products, Sports and Outdoors as well as Tools and Home Improvement

Despite the fact that no hyperparameter tuning was used the model achieved a final micro F1-score of 0.513 which is reported in table 6.4. Since this is a custom dataset it is not possible to compare results with other architectures. However, there are two elements of the result which are interesting:

Data Imbalance

Even though the dataset is fairly balanced the results for the individual aspects are very different. Figure A.6 in the appendix presents the F1 score for the individual aspects. Aspects, which are not balanced across the sentiment dimension (they do not have an equal number of negative, neutral, positive reviews) are marked as red. It is immediately obvious that the top-3 aspects with the highest overall score are red. Furthermore, the only aspect which does not have 60,000 reviews is also the aspect with the highest mean micro f1 score.

The reason for this is the way, the aspects are balanced. If there are not enough reviews for the different sentiment categories the missing reviews are taken from sentiment categories which have enough data. Table 5.3 reports that the "Office Products" category contains more than 15 times more samples than the negative sentiment category. Therefore, the model is able to achieve a good F1 score by predicting a positive sentiment most of the time.

This behavior seems to point to the fact that the ABSA-T architecture is very sensible to data imbalance
It does not come as a surprise that the "n/a" label achieves the highest scores since this label naturally makes up a vast majority of the samples but it is interesting, that the sentiment scores are that different.

Difficulty Classifying Different Sentiment Classes

Figure 6.8 reports the micro F1 score for the individual sentiment classes. It should be noted that even classes which contain exactly the same number of positive, negative and neutral reviews like "Books" for example achieve different results for the sentiment classes. Usually, the positive class achieves the highest score, negative comes second and neutral is generally last. There are two possible explanations for this phenomenon.

It may be possible to explain this behavior by looking at the overall model instead of just individual aspect heads. In total, there are more positive aspects than negative or neutral reviews. Even though the transformer base should focus on general sentence understanding, the model will be skewed towards more positive reviews. In other words, the transformer gives positive reviews more attention than negative reviews.

Therefore, it is easier for the aspect heads to predict a positive sentiment than a negative sentiment, even though the balanced heads do not have any advantage by predicting more positive sentiment.

Another possible explanation might be that it is just easier to predict positive and negative reviews than a neutral sentiment polarity. Customers usually, review a product if they are either very happy or very unhappy. When they like or dislike a product they use phrases which are easier to assign a positive or negative sentiment. However, the reason for giving a product a neutral review are more nuanced than a strong positive or negative emotion. In fact, neutral reviews often contain reasons which are both positive and negative. This differentiation is very challenging for a classifier.

The nature of neutral product reviews are also very different than a neutral sentiment polarity in the GermEval-2017 and the Organic-2019 datasets. In those instances, a neutral sentiment usually means that the aspect was mentioned but neither in a positive or negative way. This is different compared to a review where a customer weights the advantages and disadvantages of product or service.

Hyperparameter Tuning

As already mentioned, we did not perform any hyperparamter tuning on this dataset. This is very noticeable when looking at the reported F1 score for the test and dev splits. Usually, the result for the dev portion of the data is much higher because this part is typically used for hyperparameter optimization. Therefore, researches pick the model with the highest score on the dev split, indirectly picking a model which is overfitted on the validation split to an extent. This demonstrates that it is important to never perform optimization on the test split. However, if no optimization is performed at all, the validation split can be used as additional data for the training.

6.3.3. Results for Organic-2019

Organic-2019 is a dataset on organic food which was labeled by the social computing group at Technische Universität München (TUM) in the beginning of 2019. It contains 10,439 sentences which are annotated with "entities" and "aspects". Each sentence classified as "domain-relevant" receives an entity class. This entity class is further annotated with an attribute class. Sentiment is then annotated for this specific entity-attribute combination.

There are 9 entities and 14 aspects which create 126 possible entity-attribute combinations. Samples exist for 114 out of the 125 possible combinations. This means, a classifier has to predict the probabilities for 456 class labels ($114 * 4$) for each sentence. The appendix contains a list of all entity-attribute combinations at A.1.2.

Classifiers (and humans) struggle with the amount of fine-grained aspects. To provide a easier baseline we created a coarse grained version of the dataset which combines certain entities and attributes which are closely related to a total of 18 different aspects. A complete list of these aspects is located in the appendix at list A.1.2.

Variant	**Num Aspects**	**Micro F1**	
		dev	*test*
Entity-Attributes + Sentiment	114	0.068	0.060
Entities + Sentiment	9	0.204	0.164
Entities 2C + Sentiment	9	0.152	0.147
Attributes + Sentiment	14	0.189	0.139
Coarse + Sentiment	18	0.314	0.255

Organic-2019 - Dataset Partitions

Table 6.5.: Report of results on different data partitions on the Organic-2019 dataset. Entity-Attributes denotes the full fine-grained combinations of entities and attributes. Entities 2C corresponds to the sentence combination technique explained in section 5.1.3

Table 6.5 reports the classifier result on the data partitions. The first result corresponds to a ABSA on the full set of entity-attributes combinations.
The second row combines all attribute classes into their entity pair, so that a classifier only has to predict the entity - sentiment pair. The next row also uses entities as the classification target but in addition the sentence combination technique is applied. For every sentence sample this technique combines $n = 2$ sentences together with the goal of providing context from previous sentences.
Finally, the last two rows report the results on just attribute sentiment analysis and the result on the coarse grained data partition.

We used the same network architecture for each test run regardless of the data split. It consists of two encoder blocks and two aspect heads in each block. It uses a pointwise inner layer size 129 and GloVe embeddings. It is trained with a batch size of 64 and uses a linear mean head. We preprocess the data by removing stop words. The spell checker, however, is turned off.
Furthermore, we apply a dropout of 0.28 for the dropout layers in the transformer and a 0.4 dropout for the output layers. This architecture was discovered by performing

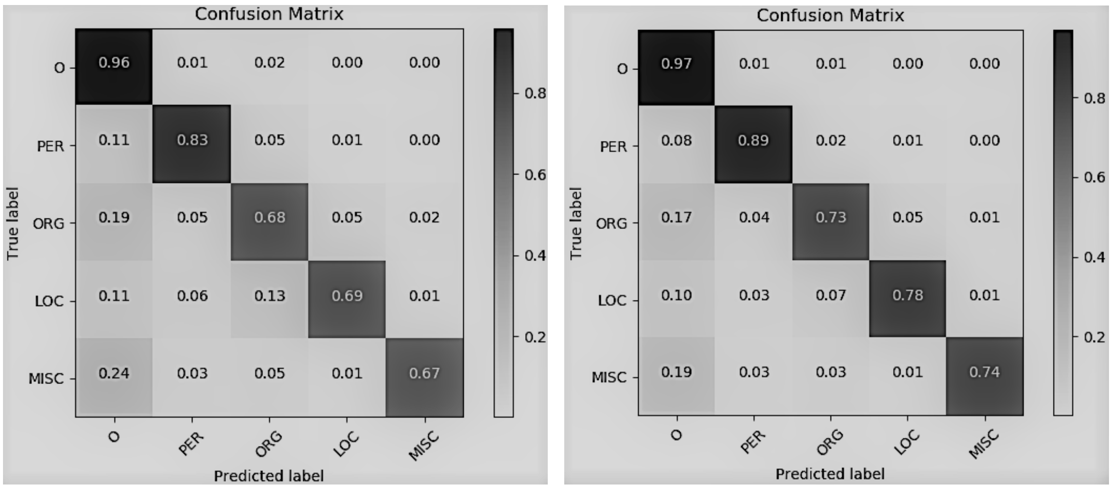
hyperparameter tuning on the dataset for the coarse partition.

This approach has one interesting consequence. When comparing the results of the dev and test splits it is noticeable that the performance on the dev split for the coarse organic partition is significantly better. This is the phenomenon mentioned in section 6.3.2.

However, this is not the case for the other partitions even though the data is exactly the same and the task is very similar. This seems to confirm that by choosing an architecture for a task it is also possible to overfit the model architecture on this specific task.

Results for Organic-2019 Coarse

The following section takes a deeper look into the dataset by utilizing the coarse data partition.



(a) Coarse - TEST

(b) Coarse - DEV

Figure 6.9.: Normalized confusion matrices for the coarse partition on the Organic-2019 dataset. The matrix on the left (a) shows the performance of the model on the test set while the right (b) confusion matrix visualizes the performance on the validation / development set.

Table 6.6 reports on the performance of different architectures on the coarse data partition of the organic dataset. We tested four versions of the model which are the two choices for the aspect head and the embeddings. In addition we used a random classifier as a baseline model.

Contrary to the results on the GermEval dataset, the Convolutional Head (CNN-H) performed significantly better than the Linear Mean Head (LM-H). While the organic dataset contains sentence-wise annotations, GermEval provides document annotations. This main difference might point to the conclusion that the mean is more useful for long documents whereas capturing the exact sequence of words is more important for shorter sequences like single sentences for instance.

Variant	**Macro F1**		**Micro F1**	
	dev	*test*	*dev*	*test*
Random Classifier	0.026	0.033	0.034	0.038
ABSA-T + LM-H + FT	0.089	0.269	0.086	0.200
ABSA-T + LM-H + GL	0.996	0.093	0.259	0.197
ABSA-T + CNN-H + GL	0.113	0.084	0.341	**0.255**
ABSA-T + CNN-H + FT	0.104	**0.098**	0.314	0.233

Organic-2019 - Coarse

Table 6.6.: Evaluation and comparison of Linear Mean Head (LM-H) against Convolutional Head (CNN-H) and two embedding choices FastText (FT) against GloVe (GL) on the Organic-2019 coarse data partition.

When comparing the results on the organic dataset and the GermEval dataset one might wonder why the performance on the coarse organic dataset is worse compared to the GermEval dataset even though both datasets use about the same number of aspects. One might argue that GermEval provides more data and therefore, training is more successful.
However, the number of aspects for the entity and attributes data partition is lower than the number of aspects for GermEval and their performance is even worse.
This points to issues regarding the dataset and a potentially more difficult task.

The total amount of samples in the organic dataset is about half of the GermEval dataset. However, in practice there are far more domain-irrelevant sentences in the organic dataset than in GermEval. Since these sentences do not contribute useful information for ABSA models we can not take advantage of this data. This means the the organic dataset is even smaller than the GermEval dataset when discarding the domain irrelevant sentences.

The version of the organic dataset we used for the evaluation contained a lot of samples with encoding issues. Since GermEval uses document annotations, it is not as problematic when some words cannot be encoded. However, when a sentence only

contains a few words it is very important to get all words to be able to understand the sentence.
It is important to note, however, that future versions of the organic dataset do not contain encoding issues anymore.

Another factor are the aspects themselves. GermEval uses aspects which can be differentiated easily. The aspect of *buying a ticket* ((Ticketkauf)) is very different to the aspect "*train ride*" (Zugfahrt). The organic dataset uses aspects which are hard to differentiate, even for human annotators because some of them are very similar.

Previous experiments show that the ABSA-T model has issues with imbalanced datasets. Especially for the fine-grained version which contains all entity-attribute combinations there are some aspects which are quite common and others where only one sample exists.

Finally, it is possible that the chosen architecture is not the optimal choice for a sentence annotated dataset. The chosen ABSA-T architecture relies on a 1:1 sentence-label relationship. This means that for every sample it needs zero or more unique class labels. This is an issue for sentence annotations because one the one hand we need context from previous sentences while on the other hand we can not classify multiple sentences at the same time.
It is possible to classify documents but only if the document has unique labels. This means the transformer is capable to classify a document with multiple sentences with aspects A, B and C. However, the ABSA-T model cannot classify two sentences at the same time when both sentences are the same aspect.

We tried to overcome this limitation by prepending previous sentences to the current sentence. Hence, only classifying the current sentence and thereby circumventing the limitations. However, as table 6.5 shows this approach actually hurt the performance.

6.4. Impact of Multitask Learning

As explained in section 4.4 we test the effect of Multi-Task Learning (MTL) by predicting an additional sentiment label on the GermEval dataset.
Table 6.7 shows the performance of the transformer with- and without the additional task.
With the additional task the model performance improved slightly. The baseline system achieved an F1 score of 0.390 whereas the MTL-version achieved 0.398. While this is a very small improvement over the baseline this increase in performance was achieved without adding any new data.

Variant	Macro F1		Micro F1	
	dev	*synchronic test*	*dev*	*synchronic test*
ABSA-T + LM-H + FT	0.150	0.117	**0.458**	0.390
ABSA-T + LM-H + FT + MTL	0.135	0.120	0.453	**0.398**

GermEval-2017 Dataset - Multitask

Table 6.7.: Results for Multi-Task Learning (MTL) on GermEval-2017. The MTL model is trained with the additional task of detecting the overall sentiment for a task. This auxiliary task does not count into the calculation of the F1 score.

This specific task has data points for every sample whereas there is usually just one aspect per sample. This means that by adding this task the gradient flow through the transformer base is doubled. Even if a sample has two aspects, we still increase the gradient flow by 33%.

6.5. Impact of Transfer Learning

Finally, we discuss the results of the transfer learning experiments. As discussed in 4.5 we will use the amazon reviews dataset as a big source dataset. This dataset contains 1,193,258 reviews across 20 aspects. In this experiment we try to transfer this knowledge from the amazon reviews domain to the organic dataset domain.

For this experiment we have to use the same architecture for both systems to be able to train on both. Since we want to improve upon our target domain we use the same configuration as for the experiments on the coarse organic data partition. The only difference is, that we limit the vocabulary size to 40,000. This means that infrequent words are replaced by "<UNK>". Therefore, it is not possible to compare the performance reported in table 6.6 directly with these results.
In order to still compare the performance we also trained the same architecture with the same vocabulary restrictions without the knowledge transfer as a baseline.

Table 6.8 reports the results for the transfer learning experiment. The baseline is reported in the first row while the second row shows the result of the transfer learning experiment.

As reported, the model which was pre-trained on the amazon reviews dataset performed significantly better than the baseline. The baseline achieved a micro F1 score of 0.197 while the TL achieved a 35.5% increase in performance.

Variant	Macro F1		Micro F1	
	dev	*test*	*dev*	*test*
ABSA-T + LM-H + FT	0.083	0.084	0.240	0.197
ABSA-T + LM-H + FT + TL	0.084	**0.116**	0.298	**0.269**

Amazon reviews > Organic coarse - Transfer Learning

Table 6.8.: Impact of Transfer Learning (TL) on the model performance. The transformer was first trained for five epochs on the amazon reviews dataset. After this, the aspect heads were exchanged for the new task on the organic-2019 coarse data partition. For this experiment we reduced the combined vocabulary size to the 40,000 most frequent words. Therefore, it is not possible to directly compare the performance of these results to the results reported in table 6.6. To proof wether or not TL can boost performance we also trained a non-TL baseline which uses the same vocabulary as the TL version.

Figure 6.10 visualizes the training process by plotting the F1 score during target training of the baseline and the TL-model. The baseline model started fairly strong up until the fourth epoch where it is overtaken by the transfer learning approach. It seems as if the TL-model is too domain-specific at first and is not able to produce any meaningful data for the transformer heads.

It takes roughly three epochs until the TL-model is able to overcome this issue. The reason for this is the special learning scheduler, that the transformer uses. The learning rate starts very low and increases linearly during a warm-up phase. After the warm-up phase the scheduler reduces the learning rate again.

In the beginning the learning rate is too low to make an impact on the pre-trained weights. However, when the learning rate starts increasing the optimizer is able to change those pretrained weights which put the emphasis on the wrong, domain specific aspects. This is the point where the F1 score drastically improves because the model is able to take full advantage of the pretraining.

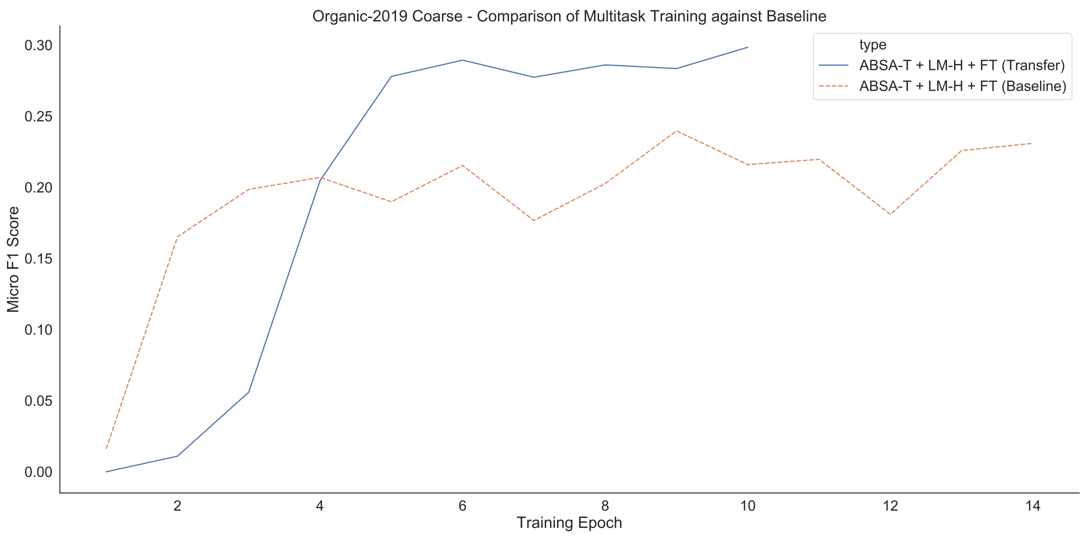


Figure 6.10.: Performance comparison of a model using transfer learning (blue) against a baseline model (orange). The transfer learning model was trained on the amazon reviews dataset as the source dataset. After 5 epochs of training on the amazon dataset normal training on the coarse organic dataset started,

7. Conclusion

During the course of this thesis a novel model architecture was proposed and implemented from scratch. We set out with the objective of testing whether or not a model without any convolutions or long term memory cells can detect and classify aspect based sentiment.

Furthermore, we took a closer look at multitask- and transfer learning. We performed experiments to evaluate how these methods can be used to increase the model performance. Specifically, we used multitask learning to augment our dataset by performing classification on an auxiliary task.

For the transfer learning experiments we created a new dataset out of existing amazon reviews which contains almost 1.2 million samples. We used this dataset as a source dataset and pretrained the transformer base and the word embedding layer. We then used this pretrained network for classification on the coarse organic data partition.

In addition we show that neural network models can be trained inside a virtualized Docker container including CUDA support without any performance decrease.

Lastly, we explored and assessed the performance of an advanced hyperparameter optimization method. Unfortunately, we could not demonstrate a significant improvement of Hyperopts TPE approach, compared to a random search. To make matters worse, we achieved a better hyperparamter configuration using a random search.

We evaluated and benchmarked the architecture and the methods on four datasets. CoNLL-2003 was used for the classic NLP task of named entity recognition. The other three datasets were datasets for ABSA.

The transformer base achieved a very respectable micro F1 score of 0.918 which is within reach of the top-performing results on this dataset. GermEval-2017 Task C was the first ABSA benchmark foor the ABSA-T model. While we did not outperform the current state of the art on this dataset we demonstrated that the ABSA-T is able to outperform previous results on this dataset putting it currently at the second position with a base F1 score of 0.390.

Using multitask learning we were able to improve upon this score and boost it to 0.398. While not a significant improvement this still shows the underused potential of multitask learning.

We did, however, notice a significant improvement using transfer learning. We could improve the baseline result of 0.197 to 0.269. This result points to the conclusion that the transformer model does not reach its full potential without massive amounts of data.

7.1. Future Work

Unfortunately, six months is a very short time period where it is not possible to explore every aspect. We could show that the transformer is not only suited as a pure decoder - encoder but is also useful for classification tasks. There is already a more advanced version which incorporates the transformer called BERT [**Devlin2018a**]. It would be very interesting to directly compare the transformer against the BERT model on the same tasks.

Furthermore, it would be very interesting to further explore the potential of transfer- and multitask learning.

The method we used for multitask learning did slightly improve performance but the model was still constrained by the amount of data in the dataset. One idea would be to use an unsupervised task instead of a supervised task. Previous work used word frequency predictions. However, it is also conceivable to use a clustering task where aspects form clusters and the model has to maximize the distance between unrelated sentences and minimize the distance between related sentences.

Another possible way to improve the performance is to handle data imbalance better. The balanced amazon review dataset shows that data balance is crucial for the model performance. We used a weighted loss function to fight data imbalance on the aspect head level.

At the same time, we did not weight the multitask loss. Weighting this loss function should balance the gradient flows on a more global level so that aspects which occur very often contribute less gradient flow than infrequent aspects.

A. Appendix

A.1. Datasets

A.1.1. GermEval-2017

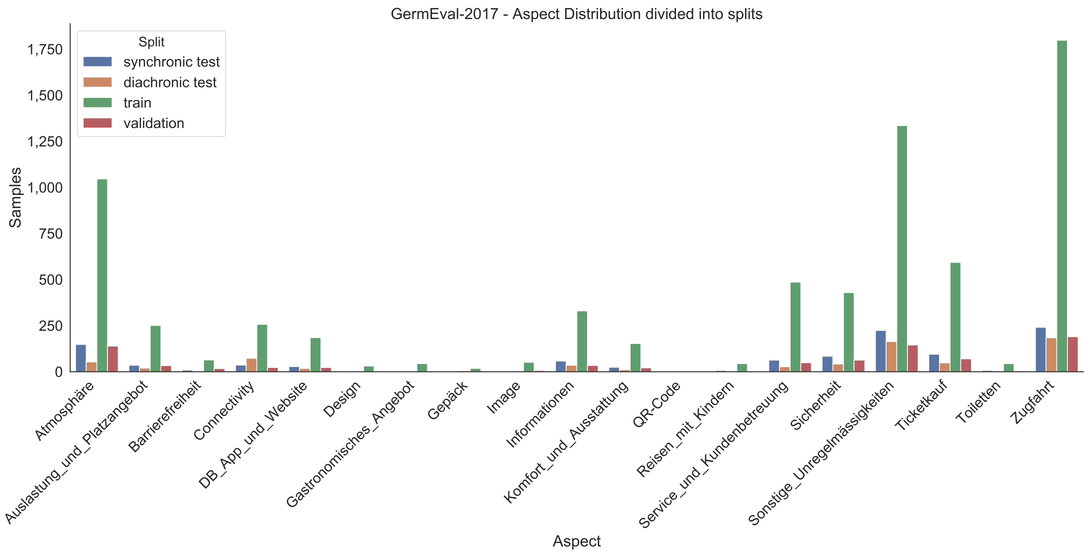


Figure A.1.: Dataset statistics on GermEval-2017 aspects

A.1.2. Organic-2019 Data

Entity - Attribute Combinations

GMOs:Nutritional quality, freshness	GMOs:price
GMOs:chemicals pesticides	GMOs:productivity
GMOs:environment	GMOs:safety
GMOs:general	GMOs:taste
GMOs:healthiness	convent. co.:animal welfare
GMOs:label	convent. co.:availability
GMOs:origin source	convent. co.:chemicals pesticides

convent. co.:environment
convent. co.:general
convent. co.:label
convent. co.:productivity
convent. co.:safety
convent. co.:taste
convent. farming:Nutritional quality, freshness
convent. farming:animal welfare
convent. farming:availability
convent. farming:chemicals pesticides
convent. farming:environment
convent. farming:general
convent. farming:healthiness
convent. farming:label
convent. farming:origin source
convent. farming:price
convent. farming:productivity
convent. farming:safety
convent. farming:taste
convent. general:Nutritional quality, freshness
convent. general:chemicals pesticides
convent. general:environment
convent. general:general
convent. general:healthiness
convent. general:label
convent. general:origin source
convent. general:price
convent. general:productivity
convent. general:safety
convent. products:Nutritional quality, freshness
convent. products:animal welfare
convent. products:availability
convent. products:chemicals pesticides
convent. products:environment
convent. products:general
convent. products:healthiness
convent. products:label
convent. products:local
convent. products:origin source
convent. products:price
convent. products:productivity
convent. products:safety
convent. products:taste
organic co.:Nutritional quality, freshness
organic co.:animal welfare
organic co.:availability
organic co.:chemicals pesticides
organic co.:environment
organic co.:general
organic co.:healthiness
organic co.:label
organic co.:local
organic co.:origin source
organic co.:price
organic co.:productivity
organic co.:safety
organic co.:taste
organic farmers:Nutritional quality, freshness
organic farmers:animal welfare
organic farmers:availability
organic farmers:chemicals pesticides
organic farmers:environment
organic farmers:general
organic farmers:healthiness
organic farmers:label
organic farmers:local
organic farmers:origin source
organic farmers:price
organic farmers:productivity
organic farmers:safety
organic farmers:taste
organic general:Nutritional quality, freshness
organic general:animal welfare
organic general:availability
organic general:chemicals pesticides
organic general:environment
organic general:general
organic general:healthiness
organic general:label
organic general:local
organic general:origin source

organic general:price
organic general:productivity
organic general:safety
organic general:taste
organic products:Nutritional quality, freshness
organic products:animal welfare
organic products:availability
organic products:chemicals pesticides
organic products:environment
organic products:general
organic products:healthiness
organic products:label
organic products:local
organic products:origin source
organic products:price
organic products:productivity
organic products:safety
organic products:taste

Coarse Partition - Combinations

GMO:environment
GMO:experienced quality
GMO:general
GMO:price
GMO:safety and healthiness
GMO:trustworthy sources
conventional:environment
conventional:experienced quality
conventional:general
conventional:price
conventional:safety and healthiness
conventional:trustworthy sources
organic:environment
organic:experienced quality
organic:general
organic:price
organic:safety and healthiness
organic:trustworthy sources

Distribution of Entities and Attributes

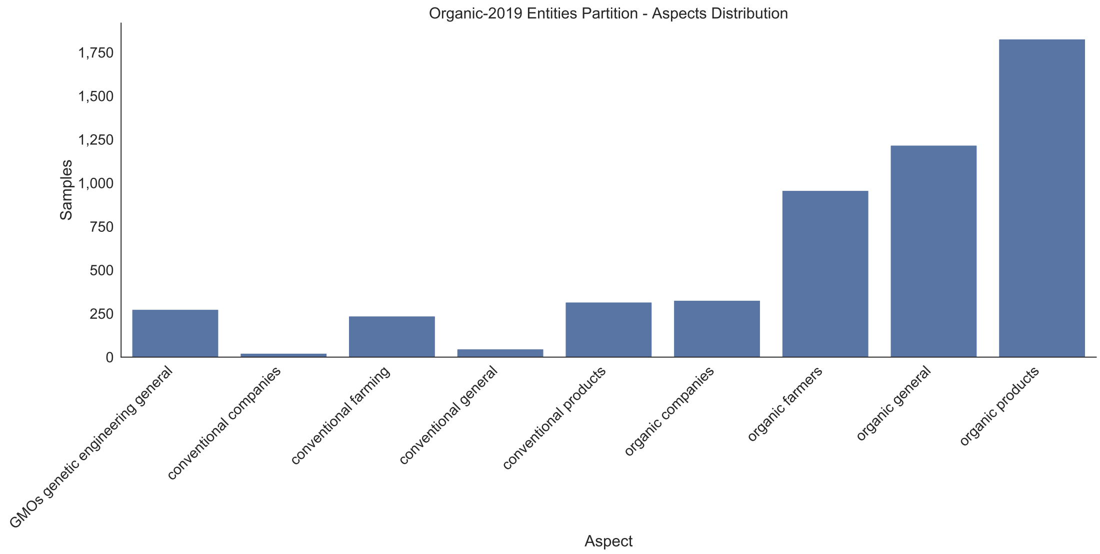


Figure A.2.: Distribution of the aspect *entity* in the Organic-2019 dataset.

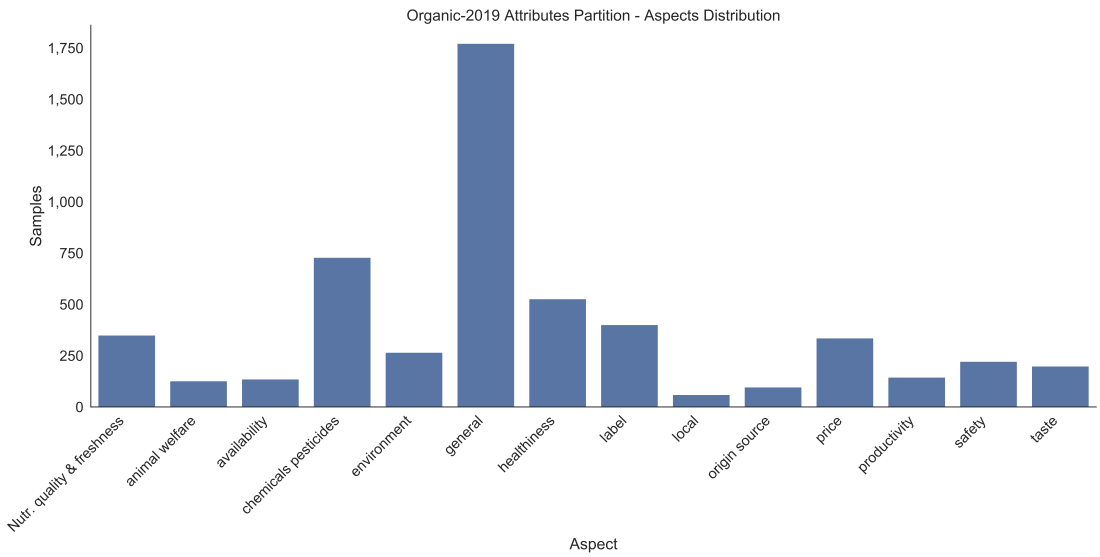


Figure A.3.: Distribution of the aspect *attribute* in the Organic-2019 dataset.

A.2. Optimization

OLS regression for Validation Loss / HyperOpt Iterations			
Dep. Variable:	y	**R-squared:**	0.009
Model:	OLS	**Adj. R-squared:**	-0.002
Method:	Least Squares	**F-statistic:**	0.8307
Date:	Mi, 24 Apr 2019	**Prob (F-statistic):**	0.365
Time:	15:32:34	**Log-Likelihood:**	-51.223
No. Observations:	90	**AIC:**	106.4
Df Residuals:	88	**BIC:**	111.4
Df Model:	1		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	0.3598	0.090	3.980	0.000	0.180	0.539
x1	-0.0016	0.002	-0.911	0.365	-0.005	0.002

Omnibus:	163.475	**Durbin-Watson:**	2.062
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	11634.826
Skew:	6.940	**Prob(JB):**	0.00
Kurtosis:	56.944	**Cond. No.**	102.

Table A.1.: OLS Regression statistics for a regression of HyperOpt TPE optimization losses on the validation set (Dataset:GermEval-2017)

OLS regression for Test Loss / HyperOpt Iterations			
Dep. Variable:	y	**R-squared:**	0.006
Model:	OLS	**Adj. R-squared:**	-0.006
Method:	Least Squares	**F-statistic:**	0.4930
Date:	Mi, 24 Apr 2019	**Prob (F-statistic):**	0.484
Time:	16:22:21	**Log-Likelihood:**	-206.16
No. Observations:	90	**AIC:**	416.3
Df Residuals:	88	**BIC:**	421.3
Df Model:	1		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	0.8356	0.506	1.653	0.102	-0.169	1.840
x1	-0.0069	0.010	-0.702	0.484	-0.026	0.013

Omnibus:	191.171	**Durbin-Watson:**	2.030
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	25929.119
Skew:	9.027	**Prob(JB):**	0.00
Kurtosis:	84.169	**Cond. No.**	102.

Table A.2.: OLS Regression statistics for a regression of HyperOpt TPE optimization losses on the test set (Dataset:GermEval-2017)

Variable	Type	Parameters
Batch Size	QUniform	Interval:[1, 100]
Comment Clipping	QUniform	Interval:[10, 250]
Replace URL Tokens	Bool	[True, False]
Use Stop Words	Bool	[True, False]
Use Spell Checker	Bool	[True, False]
Harmonize Bahn	Bool	[True, False]
Embedding Type	Choice	[Glove, Fasttext]
# Encoder Blocks	QUniform	Interval [1, 8]
# Attention Heads	Choice	[1, 2, 3, 4, 5]
PW Layer Size	QUniform	Interval:[32, 256]
TF Dropout	Uniform	Interval [0, 0.8]
Output Dropout	Uniform	Interval [0, 0.8]
Transformer Bias	Bool	[True, False]
LR Warmup	QUniform	Interval [1000, 9000]
LR Factor	Uniform	Interval [0.01, 4]
Adam $\beta1$	Uniform	Interval [0.7, 0.999]
Adam $\beta2$	Uniform	Interval [0.7, 0.999]
Adam EPS	LogUniform	log(1e-10), log(1)
LR	LogNormal	log(0.01, log(10)
Weight Decay	QUniform	$1e^{[-8,-3]}$
Output Layer	Choice	[CNN*, LinSum]
# CNN Filter*	QUniform	Interval [1, 400]
Kernel Size*	QUniform	Interval [1, 10]
Stride*	QUniform	Interval [1, 10]
Padding*	QUniform	Interval [0, 5]

Table A.3.: Hyperparameter Search space for GermEval-2017. * marks parameters which are only sampled if CNN is chosen as the output layer.

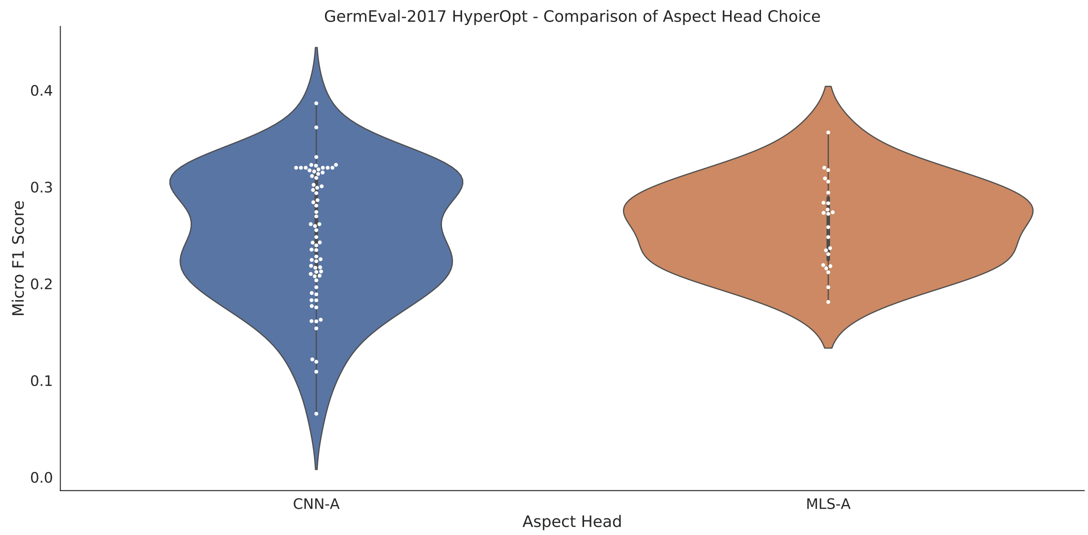


Figure A.4.: Hyperopt - Comparison and impact of aspect head choices and sampling amount

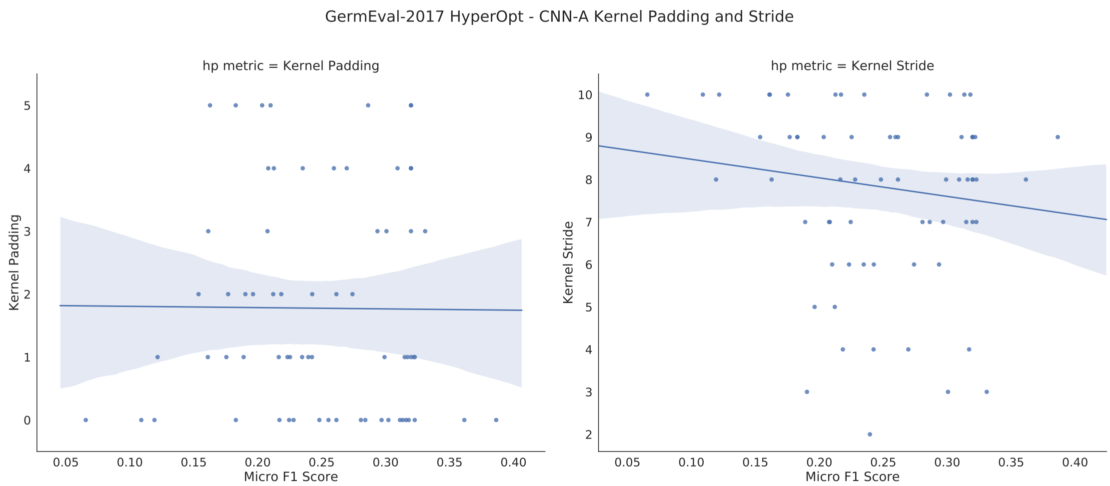
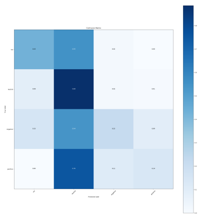
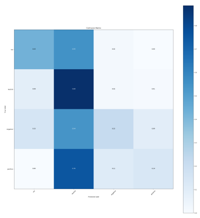


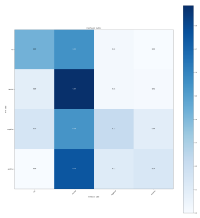
Figure A.5.: Impact of CNN parameters on micro F1-score. The graph on the left shows the impact of the kernel padding on the model performance. The graph on the right depicts the influence of the kernel stride on the F1-score.

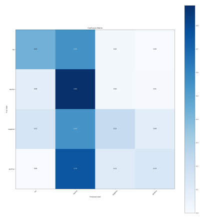
A.3. Results

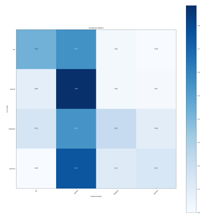
A.3.1. GermEval-2017

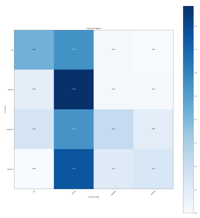

(a) Allgemein

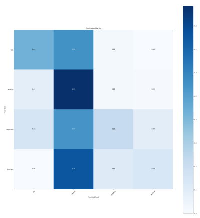

(b) Atmosphäre

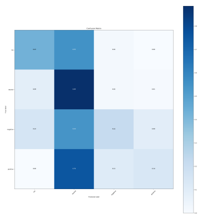

(c) Auslastung + Platz

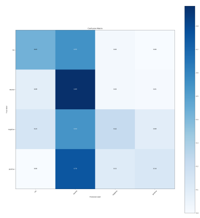

(d) Barrierefreiheit

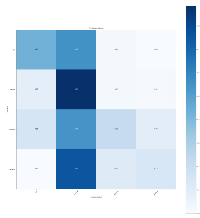

(e) Connectivity

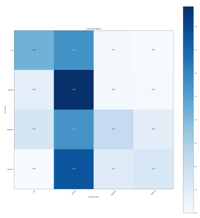

(f) App & Website

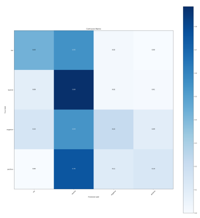

(g) Design

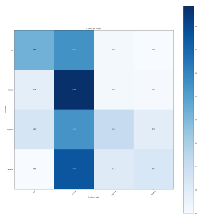

(h) Gastronomie

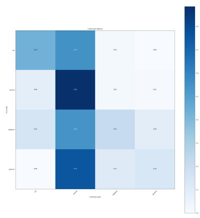

(i) Gepäck

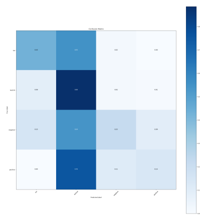

(j) Image

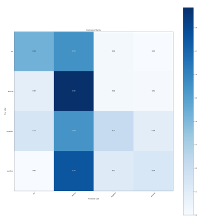

(k) Informationen

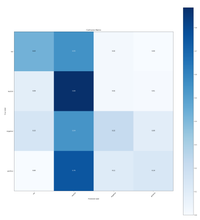

(l) Komfort

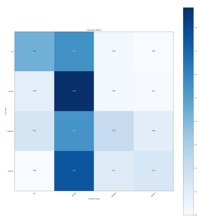

(m) QR-Code

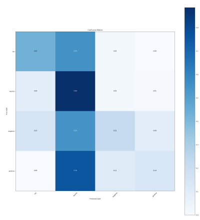

(n) Reise m. Kindern

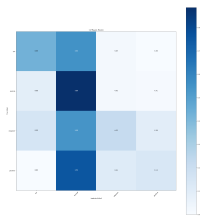

(o) Service & Support


(p) Sicherheit


(q) Unregelmäßigkeiten


(r) Ticketkauf


(s) Toiletten


(t) Zugfahrt

A.3.2. Amazon reviews

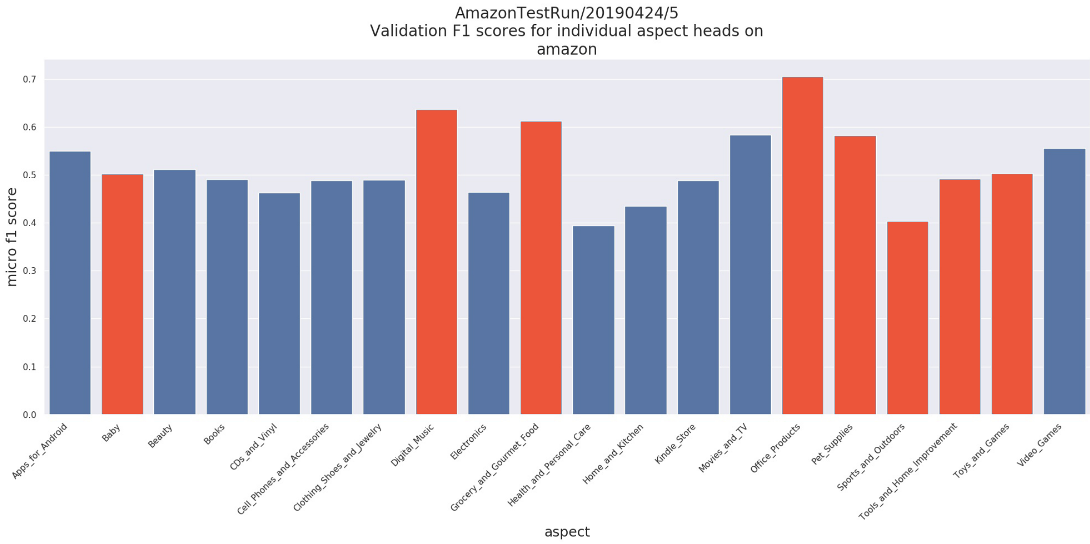


Figure A.6.: Individual aspect head micro F1 scores achieved on the validation set of the custom amazon reviews dataset. This figure reports the overall micro F1 score for the aspect heads without including the metrics for the n/a labels. The classes, marked as red which are not balanced along the sentiment dimension are Baby, Digital Music, Grocery & Gourmet Food, Office Products, Pet Supplies, Sports and Outdoors, Tools and Home Improvement and Toys & Games.

Acronyms

ABSA Aspect Based Sentiment Analysis.

ABSA-T ABSA-Transformer.

ACE Average Cross-Entropy Error.

Adam Adaptive Moment Estimation.

API Application Programming Interface.

BoW Bag of Words.

CBOW Continuous Bag-of-Words Model.

CNN Convolutional Neural Network.

CNN-A CNN-based Aspect Head.

CNN-H Convolutional Head.

CPU Central Processing Unit.

csv Comma Separated Values.

CUDA Compute Unified Device Architecture.

.

GB Giga Bytes.

GloVe Global Vectors.

GPS Global Positioning System.

GPT Generative Pre-Training.

GPU Graphics Processing Unit.

HTML Hypertext Markup Language.

IO Input / Output.

KNN K-nearest Neighbors.

LM-H Linear Mean Head.

LSA Latent Semantic Analysis.

LSTM Long short-term memory.

MLS-A Mean Linear Sum Aspect Head.

MSE Mean Squared Error.

MTL Multi-Task Learning.

MVG Münchner Verkehrsgesellschaft.

NER Named-entity recognition.

NLL Negative Log Likelihood.

NLP Natural Language Processing.

OLS Ordinary Least Squares.

POS Part-of-Speech.

PWFC Point-wise Feed Forward.

RAM Random Access Memory.

RNN Recurrent neural network.

SGD Stochastic gradient descent.

SOTA State-of-the-art.

std Standard Deviation.

SVM Support vector machine.

TPE Tree of Parzen Estimator.

TUM Technische Universität München.

URL Uniform Resource Locator.

.

List of Figures

List of Tables

Bibliography

[1] S. Baccianella, A. Esuli, and F. Sebastiani. "SentiWordNet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining." In: *Proceedings of the International Conference on Language Resources and Evaluation*. Vol. 1. Valletta, Malta, 2010, pp. 2200–2204.

[2] A. Baevski, S. Edunov, Y. Liu, L. Zettlemoyer, and M. Auli. "Cloze-driven Pretraining of Self-attention Networks." In: (2019). arXiv: 1903.07785.

[3] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. "Algorithms for hyper-parameter optimization." In: *NIPS'11 Proceedings of the 24th International Conference on Neural Information Processing Systems*. Granada, Spain: Neural Information Processing Systems, 2011, pp. 2546–2554. ISBN: 9781618395993.

[4] J. Bergstra and Y. Bengio. "Random Search for Hyper-Parameter Optimization." In: *Journal of Machine Learning Research* 13 (2012), pp. 281–305.

[5] J. Bergstra, D. Yamins, and D. D. Cox. "Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms." In: *PROC. OF THE 12th PYTHON IN SCIENCE CONF. (SCIPY*. Vol. 12. 1. Austin, Texas, 2013.

[6] J. Bergstra, D. L. K. Yamins, and D. Cox. "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures." In: *Proceedings of the 30th International Conference on Machine Learning* 28 (2013), pp. 115–123.

[7] J. Blitzer, M. Dredze, and F. Pereira. "Biographies, Bollywood, Boom-boxes and Blenders: Domain Adaptation for Sentiment Classification." In: *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*. Prague, Czech Republic: Association for Computational Linguistics, 2007, pp. 440–447.

[8] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. "Enriching Word Vectors with Subword Information." In: *Transactions of the Association for Computational Linguistics* Volume 5 (2017), pp. 135–146. DOI: 10.1162/tacl_a_00051. arXiv: 1607.04606.

[9] C. Brun, J. Perez, and C. Roux. "XRCE at SemEval-2016 Task 5: Feedbacked Ensemble Modeling on Syntactico-Semantic Knowledge for Aspect Based Sentiment Analysis." In: *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*. San Diego, USA, 2016, pp. 277–281. DOI: 10.18653/v1/s16-1044.

[10] R. Caruana. "Learning Many Related Tasks at the Same Time With Backpropagation." In: *Advances in Neural Information Processing Systems 7*. Ed. by T. G, D. S. Touretzky, and T. K. Leen. MIT Press, 1995, pp. 657–664.

[11] R. Caruana. "Multitask Learning." Ph.D thesis. Carnegie Melon University, 1997.

[12] R. Caruana. "Multitask Learning: A Knowledge-Based Source of Inductive Bias." In: *PROCEEDINGS OF THE TENTH INTERNATIONAL CONFERENCE ON MACHINE LEARNING* (1993), pp. 41–48.

[13] D. Cox and N. Pinto. "Beyond simple features: A large-scale feature search approach to unconstrained face recognition." In: *2011 IEEE International Conference on Automatic Face and Gesture Recognition and Workshops, FG 2011*. Ed. by IEEE. Santa Barbara, CA, USA: IEEE, 2011, pp. 8–15. ISBN: 9781424491407. DOI: 10.1109/FG.2011.5771385.

[14] S. R. Das and M. Y. Chen. "Yahoo! for Amazon: Sentiment Extraction from Small Talk on the Web." In: *Management Science* 53.9 (2007), pp. 1375–1388. ISSN: 0025-1909. DOI: 10.1287/mnsc.1070.0704.

[15] J. Devlin, M.-W. Chang, K. Lee, K. T. Google, and A. I. Language. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. Tech. rep. 2018. arXiv: 1810.04805v1.

[16] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition." In: *Proceedings of the 31st International Conference on Machine Learning*. PMLR, 2013, pp. 647–655. arXiv: 1310.1531v1.

[17] J. Duchi, E. Hazan, and Y. Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." In: *Journal of Machine Learning Research* 12 (2011), pp. 1–40.

[18] S. Dugar. "Aspect-Based Sentiment Analysis Using Deep Neural Networks and Transfer Learning." Master's Thesis in Informatics. TECHNISCHE UNIVERSITÄT MÜNCHEN, 2019.

[19] F. Erik, S. T. Kim, and F. De Meulder. "Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition." In: *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003*. Vol. 4. Edmonton, Canada, 2003, pp. 142–147. DOI: 10.3115/1119176.1119195.

[20] R. Florian, A. Ittycheriah, H. Jing, and T. Zhang. "Named entity recognition through classifier combination." In: *CONLL '03 Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003 - Volume 4*. 2003, pp. 168–171. DOI: 10.3115/1119176.1119201.

[21] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016, p. 785.

[22] R. He and J. McAuley. "Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering." In: (2016). DOI: 10.1145/2872427.2883037. arXiv: 1602.01585.

[23] G. E. Hinton. "Learning Distributed Representations of Concepts." In: *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*. 1986, pp. 1–12.

[24] G. Hinton, N. Srivastava, K. Swersky, and K. Swersky. *Neural Networks for Machine Learning Lecture 6a Overview of mini--batch gradient descent*.

[25] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. "Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies." In: *A Field Guide to Dynamical Recurrent Networks*. Wiley-IEEE Press, 2009. Chap. 20, p. 464. ISBN: 9780470544037. DOI: 10.1109/9780470544037.ch14.

[26] M. Hu and B. Liu. "Mining and summarizing customer reviews." In: *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '04*. Seattle, WA, USA, 2004, pp. 168–177. ISBN: 1581138889. DOI: 10.1145/1014052.1014073.

[27] A. Huettner and P. Subasic. "Fuzzy typing for document management." In: *CL 2000 Companion Volume: Tutorial Abstracts and Demonstration Notes*. 2000, pp. 26–27.

[28] F. Hutter, H. H. Hoos, K. Ca, and S. Be. "ParamILS: An Automatic Algorithm Configuration Framework Kevin Leyton-Brown Thomas StützleSt¨Stützle." In: *Journal of Artificial Intelligence Research* 36 (2009), pp. 267–306.

[29] M. S. Iyer and R. R. Rhinehart. "A method to determine the required number of neural-network training repetitions." In: *IEEE Transactions on Neural Networks* 10.2 (1999), pp. 427–432. ISSN: 10459227. DOI: 10.1109/72.750573.

[30] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. "Bag of Tricks for Efficient Text Classification." In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Valencia, Spain: Association for Computational Linguistics, 2016, pp. 427–431. arXiv: 1607.01759.

[31] A. Karpathy. *What I learned from competing against a ConvNet on ImageNet*. 2014.

[32] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization." In: *International Conference on Learning Representations (ICLR)*. 2014. arXiv: 1412.6980.

[33] A. Kumar, S. Kohail, A. Kumar, A. Ekbal, and C. Biemann. "IIT-TUDA at SemEval-2016 Task 5: Beyond Sentiment Lexicon: Combining Domain Dependency and Distributional Semantics Features for Aspect Based Sentiment Analysis." In: *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*. San Diego, USA, 2016, pp. 1129–1135. DOI: 10.18653/v1/s16-1174.

[34] H. Lakkaraju, R. Socher, and C. Manning. "Aspect Specific Sentiment Analysis using Hierarchical Deep Learning." In: *NIPS Workshop on deep learning and representation learning*. Montreal, 2014.

[35] Y. LeCun; B. Boser, J. S. Denker, R. E. Howard, W. Habbard, and L. D. Jackel. "Handwritten Digit Recognition with a Back-Propagation Network." In: *Advances in Neural Information Processing Systems* (1990), pp. 396–404. ISSN: 1524-4725. DOI: 10.1111/dsu.12130. arXiv: 1004.3732.

[36] J.-U. Lee, S. Eger, J. Daxenberger, and I. Gurevych. "UKP TU-DA at GermEval 2017: Deep Learning for Aspect Based Sentiment Detection." In: *Proceedings of the GSCL GermEval Shared Task on Aspect-based Sentiment in Social Media Customer Feedback*. Berlin, 2017, pp. 22–29.

[37] V. Levenshtein. "Binary Codes Capable of Correcting Deletions, Insertions and Reversals." In: *Insertions and Reversals. Sov* 6 (1966), pp. 707–710.

[38] T. Luong, R. Socher, and C. Manning. "Better word representations with recursive neural networks for morphology." In: *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*. 2013, pp. 104–113. DOI: 10.1007/BF02579642.

[39] Y. Ma, H. Peng, T. Khan, E. Cambria, and A. Hussain. "Sentic LSTM: a Hybrid Network for Targeted Aspect-Based Sentiment Analysis." In: *Cognitive Computation* 10.4 (2018), pp. 639–650. ISSN: 18669964. DOI: 10.1007/s12559-018-9549-x.

[40] J. McAuley and J. Leskovec. "From Amateurs to Connoisseurs: Modeling the Evolution of User Expertise through Online Reviews." In: *WWW '13 Proceedings of the 22nd international conference on World Wide Web*. Rio de Janeiro, Brazil: ACM New York, NY, USA, 2013, pp. 897–908. ISBN: 9781450320351. DOI: 10.1145/2488388.2488466. arXiv: 1303.4402.

[41] J. McAuley and J. Leskovec. "From Amateurs to Connoisseurs: Modeling the Evolution of User Expertise through Online Reviews." In: *WWW '13 Proceedings of the 22Nd International Conference on World Wide Web*. Rio de Janeiro, Brazil: ACM, Mar. 2013, pp. 897–908. DOI: 10.1145/2488388.2488466. arXiv: 1303.4402.

[42] J. McAuley and J. Leskovec. "Hidden Factors and Hidden Topics: Understanding Rating Dimensions with Review Text." In: *RecSys '13 Proceedings of the 7th ACM conference on Recommender systems*. Hong Kong, China: ACM, 2013, pp. 165–172. ISBN: 9781450324090. DOI: 10.1145/2507157.2507163.

[43] J. McAuley, J. Leskovec, and D. Jurafsky. "Learning attitudes and attributes from multi-aspect reviews." In: *Proceedings - IEEE International Conference on Data Mining, ICDM*. 2012, pp. 1020–1025. ISBN: 9780769549057. DOI: 10.1109/ICDM.2012.110.

[44] J. McAuley, C. Targett, Q. Shi, and A. van den Hengel. "Image-based Recommendations on Styles and Substitutes." In: (2015), pp. 1–11. arXiv: 1506.04757.

[45] T. Mikolov, K. Chen, G. Corrado, and J. Dean. "Distributed Representations of Words and Phrases and their Compositionality." In: *NIPS'13 Proceedings of the 26th International Conference on Neural Information Processing Systems*. Lake Tahoe, USA, 2013, pp. 3111–3119. arXiv: 1310.4546v1.

[46] T. Mikolov, K. Chen, G. Corrado, and J. Dean. "Efficient Estimation of Word Representations in Vector Space." In: *Proceedings of the International Conference on Learning Representations (ICLR 2013)*. 2013. arXiv: 1301.3781.

[47] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. "Distributed Representations ofWords and Phrases and their Compositionality." In: *NIPS'13 Proceedings of the 26th International Conference on Neural Information Processing Systems* 2 (Oct. 2013), pp. 3111–3119.

[48] T. Mikolov, W.-T. Yih, and G. Zweig. "Linguistic Regularities in Continuous Space Word Representations." In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Atlanta, USA: Association for Computational Linguistics, 2013, pp. 9–14.

[49] S. J. Pan and Q. Yang. "A survey on transfer learning." In: *IEEE Transactions on Knowledge and Data Engineering*. Vol. 22. 10. Piscataway, NJ, USA: IEEE Educational Activities Department, 2010, pp. 1345–1359. DOI: 10.1109/TKDE.2009.191.

[50] B. Pang, L. Lee, and S. Vaithyanathan. "Thumbs up? Sentiment Classification using Machine Learning Techniques." In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. July. Association for Computational Linguistics, 2012, pp. 79–86. DOI: 10.1515/9783110239171.151.

[51] J. Pennington, R. Socher, and C. Manning. "Glove: Global Vectors for Word Representation." In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162.

[52] M. Peters, S. Ruder, and N. A. Smith. "To Tune or Not to Tune? Adapting Pretrained Representations to Diverse Tasks." In: (2019). arXiv: 1903.05987.

[53] B. Plank, A. Søgaard, and Y. Goldberg. *Multilingual Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Models and Auxiliary Loss*. Tech. rep. arXiv: 1604.05529v3.

[54] M. Pontiki, D. Galanis, and H. Papageorgiou. "SemEval-2015 Task 12 : Aspect Based Sentiment Analysis." In: *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*. Denver, USA: Association for Computational Linguistics, 2015, pp. 486–495. DOI: 10.18653/v1/S15-2082.

[55] M. Pontiki, D. Galanis, J. Pavlopoulos, H. Papageorgiou, I. Androutsopoulos, and S. Manandhar. "SemEval-2014 Task 4: Aspect Based Sentiment Analysis." In: *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*. SemEval. Dublin, Ireland: Association for Computational Linguistics, 2014, pp. 27–35. DOI: 10.3115/v1/s14-2004.

[56] A.-M. Popescu and O. Etzioni. "OPINE: Extracting product features and opinions from reviews." In: *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*. October. Vancouver, Canada, 2005, pp. 339–346.

[57] L. Y. Pratt. "Discriminability-Based Transfer between Neural Networks." In: *Advances in neural information processing systems* (1993), pp. 204–211.

[58] L. Pratt, J. Mostow, and C. Kamm. "Direct Transfer of Learned Information Among Neural Networks." In: *AAAI-91 Proceedings*. AAAI, 1991, pp. 584–589.

[59] A. Radford, N. Karthik, T. Salimans, and I. Sutskever. *Improving Language Understanding by Generative Pre-Training*. Tech. rep. 2018, pp. 1–12. DOI: 10.1093/aob/mcp031. arXiv: 1802.05365.

[60] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. *language model and unsupervised multitask learning*. Tech. rep. 2019.

[61] B. Ramsundar, S. Kearnes, P. Riley, D. Webster, D. Konerding, and V. Pande. "Massively Multitask Networks for Drug Discovery." Feb. 2015.

[62] M. Rei. "Semi-supervised Multitask Learning for Sequence Labeling." In: (Apr. 2017). arXiv: 1704.07156.

[63] S. Ruder. "An Overview of Multi-Task Learning in Deep Neural Networks." In: (June 2017). arXiv: 1706.05098.

[64] S. Ruder, P. Ghaffari, and J. G. Breslin. “A Hierarchical Model of Reviews for Aspect-based Sentiment Analysis.” In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, 2016, pp. 999–1005. arXiv: 1609.02745.

[65] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. “ImageNet Large Scale Visual Recognition Challenge.” In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252. ISSN: 15731405. DOI: 10.1007/s11263-015-0816-y. arXiv: 1409.0575.

[66] E. F. T. K. Sang and F. De Meulder. “Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition.” In: (2003). arXiv: 0306050 [cs].

[67] M. Schmitt, S. Steinheber, K. Schreiber, and B. Roth. “Joint Aspect and Polarity Classification for Aspect-based Sentiment Analysis with End-to-End Neural Networks.” In: (Aug. 2018). arXiv: 1808.09238.

[68] H. Schütze. “Word Space.” In: *Advances in Neural Information Processing Systems 5 (NIPS 1992)*. 1992.

[69] C.-c. C. C.-p. W. F.-y. Shen and Y.-n. Fan. “A sales forecasting model for consumer products based on the influence of online word-of-mouth.” In: *Information Systems and e-Business Management* 13.3 (2015), pp. 445–473. ISSN: 1617-9846. DOI: 10.1007/s10257-014-0265-0.

[70] *So you're ready to get started. – Common Crawl*.

[71] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. “Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank.” In: *PLoS ONE* 8.9 (2013), pp. 1631–1642. ISSN: 19326203. DOI: 10.1371/journal.pone.0073791. arXiv: 1690219.1690245âĂŪ.

[72] D. Tang, B. Qin, X. Feng, and T. Liu. “Effective LSTMs for Target-Dependent Sentiment Classification.” In: *Proceedings of COLING 2016, the 26th international conference on computational linguis- tics: technical papers*. Osaka, Japan, 2016, pp. 3298–3307.

[73] T. T. Thet, J. C. Na, and C. S. Khoo. “Aspect-based sentiment analysis of movie reviews on discussion boards.” In: *Journal of Information Science* 36.6 (2010), pp. 823–848. ISSN: 01655515. DOI: 10.1177/0165551510388123.

[74] R. M. Tong. “An operational system for detecting and tracking opinions in on-line discussion.” In: *Working Notes of the ACM SIGIR 2001 Workshop on Operational Text Classification*. 2001.

[75] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer. "Feature-rich part-of-speech tagging with a cyclic dependency network." In: 2007, pp. 173–180. DOI: 10.3115/1073445.1073478.

[76] J. Turian, L. Ratinov, and Y. Bengio. "Word representations : A simple and general method for semi-supervised learning." In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL-10)*. July. 2010, pp. 384–394.

[77] P. D. Turney. "Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews." In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*. July. Philadelphia, USA, 2007, pp. 417–424. DOI: 10.3115/1073083.1073153. arXiv: 0212032 [cs].

[78] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. "Attention Is All You Need." In: *Advances in Neural Information Processing Systems 30*. Ed. by I. G. Garnett, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Curran Associates, Inc., June 2017, pp. 5998–6008. arXiv: 1706.03762.

[79] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. "Attention Is All You Need." In: (2017). ISSN: 0140-525X. DOI: 10.1017/S0140525X16001837. arXiv: 1706.03762.

[80] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. "Attention Is All You Need." In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., June 2017, pp. 5998–6008. arXiv: 1706.03762.

[81] M. Wojatzki, E. Ruppert, S. Holschneider, T. Zesch, and C. Biemann. "GermEval 2017: Shared Task on Aspect-based Sentiment in Social Media Customer Feedback." In: *Proceedings of the GermEval 2017 – Shared Task on Aspect-based Sentiment in Social Media Customer Feedback*. 2017, pp. 1–12.

[82] M. Wojatzki, E. Ruppert, S. Holschneider, T. Zesch, C. Biemann, and A. CogSci. "GermEval 2017: Shared Task on Aspect-based Sentiment in Social Media Customer Feedback." In: *Proceedings of the GSCL GermEval Shared Task on Aspect-based Sentiment in Social Media Customer Feedback*. Berlin, 2017, pp. 1–12.

[83] D. Xenos, P. Theodorakakos, J. Pavlopoulos, P. Malakasiotis, and I. Androutsopoulos. "AUEB-ABSA at SemEval-2016 Task 5: Ensembles of Classifiers and Embeddings for Aspect Based Sentiment Analysis." In: *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*. San Diego, USA, 2016, pp. 312–317. DOI: 10.18653/v1/s16-1050.

[84] H. Xu, B. Liu, L. Shu, and P. S. Yu. "BERT Post-Training for Review Reading Comprehension and Aspect-based Sentiment Analysis." In: (2019). arXiv: 1904.02232.

[85] W. Xue and T. Li. "Aspect Based Sentiment Analysis with Gated Convolutional Networks." In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, 2018, pp. 2514–2523. arXiv: 1805.07043.

[86] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. "How transferable are features in deep neural networks?" In: *Advances in Neural Information Processing Systems*. Ed. by Z. G. Weinberger, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Vol. 27. Curran Associates, Inc., 2014, pp. 3320–3328. arXiv: 1411.1792.

[87] M. D. Zeiler and R. Fergus. "Visualizing and Understanding Convolutional Networks." In: *Computer Vision – ECCV 2014. ECCV 2014. Lecture Notes in Computer Science*. Springer, Cham, 2014, pp. 818–833. DOI: 10.1007/978-3-319-10590-1_53.

[88] Y. Zhang and Q. Yang. "A Survey on Multi-Task Learning." In: (2017). arXiv: 1707.08114.