Philipp Kogelnik, BSc

Convolutional Neural Networks for Toxicity Classification in Online Comments

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Software Development and Business Management

submitted to

Graz University of Technology

Supervisor

Dipl.-Ing. Dr.techn. Roman Kern

Institute for Interactive Systems and Data Science
Head: Univ.-Prof. Dipl.-Inf. Dr. Stefanie Lindstaedt

Graz, January 2019

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Date	Signature

Abstract

Whether it is a posting spreading hate about a group of people, a comment insulting another person or a status containing obscenities, such types of toxic content have become a common issue for many online platforms. Owners of platforms like blogs, forums or social networks are highly interested in detecting this negative content.

The goal of this thesis is to evaluate the general suitability of convolutional neural networks (CNNs) for classifying toxicity in textual online comments. For this purpose different CNN architectures are developed and their performance is compared to state-of-the-art methods on the data set containing comments from Wikipedia discussion pages. For a better understanding of this type of neural networks this thesis contains three subquestions: a) Which patterns do CNNs learn and which features are important for the classification when being applied to this task? b) Which preprocessing techniques are beneficial to the performance? c) Are CNNs well-suited for comments from sources other than Wikipedia discussion pages?

The evaluation showed a performance similar to other classifiers on the same data set. Moreover, the model showed a comparable performance on a second data set created for this thesis. The best single preprocessing technique in this work improved the F1 score from 0.636 to 0.645 compared to the baseline. An analysis of a trained model revealed that some patterns detected by the convolutional layer are interpretable by humans. The analysis of the influence of words to the prediction highlighted struggles with negations in the text and also revealed a severe bias included in the model.

Contents

List of Figures

1. Introduction

Whether it is a picture including nudity, a comment spreading hate about a group of people or a video showing violence, such types of toxic content have become a common issue for many online platforms. Due to the ever increasing amount of data generated on the internet, this phenomenon has intensified over the years [15][10]. Owners of platforms like blogs, forums or social networks are highly interested in filtering out such content. Leaving it on the platform would create a negative atmosphere and possibly cause a loss of users.

Not removing toxic content is not just a danger for the image of the platform, but can also have legal implications for the company. In Germany a new law has come into force which ensures that social media platforms remove hate speech within 24 hours of notification in straightforward cases [39]. This means content moderation needs to be in place which acts quickly but also precisely.

Even if the content is not forbidden by law it can still have serious effects on users which are not desired by the platforms. According to a survey conducted by DitchTheLabel 17% of the responding teenagers from schools and colleges in the United Kingdom have been bullied online [12]. Just looking at this statistics makes it clear that platforms need to take measures to prevent such situations.

There has been research going on in recent years to automatically detect such content. But up until today this task is still mostly done by human content managers or users reporting inappropriate content which can be seen in the next subsection.

The scope of this work is focused on the classification of toxicity in textual comments on the internet. In general, such content can occur on all kinds of platforms where users can enter text and make it visible to other users. This type of content includes the following sources:

- Tweets on Twitter
- Comments to YouTube videos

- Status message on Facebook
- Chats in video games
- Discussion pages on Wikipedia
- Comments to newspaper articles

As can be seen in this far from complete list, user-generated text content can be found all over the internet.

Early approaches of detecting toxicity used a dictionary with words which are considered as profanity. If such words occur the comment or parts of it got blocked. But there are several problems with this approach. It is still possible to threaten or insult other people without using any kind of bad words.

On the other hand comments might get blocked because they contain one of the black-listed words, but do not use it in a negative sense. The following example is clearly an insult and would be justifiably blocked by such a system:

> You are a Nazi!

But the next comment would be blocked as well although it is just an historic explanation of a term:

> The Nazi Party was a far-right political party in Germany that was active between 1920 and 1945.

As it can be seen in these examples just relying on words itself is not enough in all cases. A possible model needs to be able to capture more complex patterns and take into account the connections between words. Moreover, it needs to have at least some knowledge about sentence structures and grammar.

One way to automatically detect toxicity and not solely relying on bad word dictionaries is applying machine learning models. Such models have the advantage of being capable of making decisions for previously unseen scenarios. They are applied to a range of different tasks like spam detection, object recognition or medical diagnosis.

One type of these models has gained popularity in recent years: artificial neural networks. Their success can be attributed to two main reasons. Firstly, computers have become fast enough to train and apply them on a larger scale. This is especially important as neural networks need large amounts of data to be processed and are

computationally expensive to train. This first reason leads to the second one. Due to suitable hardware have become more affordable, more researchers got access to the computational power needed for their experiments. This led to trying to apply neural networks to a very broad range of topics.

A subtopic of neural networks are so called CNNs. Originally, this type was mostly applied to visual image data where they achieve very good results, especially in image classification tasks [35][56]. Moreover, compared to other image classification algorithms there is less preprocessing needed, which also helped increasing their popularity. In recent years its applications have been extended to other areas as well, like text classification [32] or sequence modeling [3].

This work focuses on applying CNNs for classifying toxic comments and investigating their suitability for this topic.

1.1. Motivation

Although companies like Facebook employ thousands of content managers, they still rely heavily on their users when it comes to reporting inappropriate content. From October to December 2017 about 76 percent of all hate speech removed from the platform has been reported by users first. This percentage dropped to 62 for the period between January and March 2018 [15]. But still, users get to see much inappropriate content before it gets removed. These statistics suggest that content managers need an improvement of their tool set to handle such amounts of data.

Even if there were enough content managers to detect all harmful content from the platforms it is still not the desired solution. Especially for the people doing this job it can result in negative effects to their psyche when looking at such amounts of offensive texts every day. The desired state for this task is an algorithm, which is able to classify most content completely on its own. There should be as little manual work needed as possible.

Another important issue when letting humans decide, which comments are offensive and which are not is their bias. If two persons have a completely different background it is very likely that they have a slightly varying perception of what is acceptable. But not only a different background can lead to different results. If a comment is shown to the same person on different days, their decision might be

different too. Such variation can be caused by a change in the emotional state of the person or by a shift of views and opinions over time. All these factors result in a certain level of disagreement between content managers. The ideal solution for this issue is an algorithm which makes deterministic and comprehensible decisions.

One possible way to provide such a tool is the use of a CNN. Although mostly applied to image data, CNNs have recently been successfully applied to textual data as well [33] [67]. But even though they work in similar areas it does not mean that they work for classifying toxic comments too. A problem which hinders such models from being applied to productive systems is their lack of understandability. A CNN is often referred to as a ”black box”, which means that it is not always intelligible how it derives its decisions. When being applied on a larger scale one should at least understand which features are important for result.

1.2. Contribution

In general, this work tries to answer a single main research questions. However, to support the main question this thesis also contains three different sub questions.

1. Are CNNs suitable for classifying toxic comments?
    a) Which features are important for the classification and which patterns do CNNs learn when being applied to this problem?
    b) Which kind of preprocessing is beneficial to the CNN performance in this area?
    c) Do these models perform equally well when being applied to data from different sources?

For the main question different kinds of CNN models are developed and applied to data. Their performance is then assessed and compared to state-of-the-art machine learning models. To answer this question an architecture needs to be found which shows performance measures which are at least close to the state of the art.

The first sub question is about giving some insights into what a neural network learns about this data. This question is about taking away some parts of the black-box character of a CNN. To give an appropriate answer to this problem, a closer

examination of the trained models is done. The importance of different input features is assessed and the activation of different parts of the network is analyzed.

Sub question number 2 is about applying different preprocessing techniques to the data. For this task different pipelines are developed and compared regarding their performance. The answer to this question shows which techniques benefit the classification results, which take away important information and which ones don not affect the results at all.

For the last sub question the models are applied not just to a single data set, but to different ones. The purpose of this task is to show if a CNN can do toxicity classification on data coming from different sources. A positive answer to this question would make it more suitable for being applied to production systems.

One of the data sets used for the experiments is created as part of this thesis. It consists of comments to YouTube videos which are annotated with toxicity and selected subcategories. To make the annotation process easier a web tool is created where users can collaboratively work on a data set. This tool is supposed to be used to create a multi-lingual toxicity data set. However, the multi-lingual data set is outside the scope of this thesis.

In addition to the answers of the research questions this work tries to give some general recommendations about how to use CNNs in this context.

1.3. Outline

In the following chapter necessary background information about the topics covered in this thesis are given. This includes a theoretical background about toxicity in comments and some of its subcategories. To understand the models in this work this chapter also describes neural networks and some related technical concepts. Additionally, there are also some preprocessing techniques explained which are used in this work.

In chapter 3 related work on this topic is shown. This chapter shows general work about text classification with a CNN and its state of the art. It also gives some intuition about toxicity classification in texts and how it has evolved over time.

1. Introduction

A preliminary project has been done about classifying relations in sentences with a CNN. The details about the project including the architecture and the data used can be found in chapter 4.

Chapter 5 describes the machine learning pipeline used for this work in depth. It explains the developed preprocessing pipelines and all their steps. Moreover, it explains the different developed CNN architectures and their characteristics. Also, detailed information about the applied algorithms is given here.

In chapter 6 the tool developed for annotating the data set is described. This includes all the necessary steps like crawling, the annotation workflow and the resulting data set.

Information about the experiments done in the scope of this work can be found in chapter 7. This chapter starts off with describing the data sets which the models are applied to. Moreover, the implementation details and chosen parameters can be found here.

Chapter 8 shows the information of the experiments and compares the with other state-of-the-art techniques. This part of the work gives some insights about the suitability of the developed models to the specified problem of toxicity classification. Also, it covers the analysis of the learned features to give a better understanding on how the models work.

In chapter 9 the findings of this work are stated and discussed. To make this findings more helpful to the reader, some recommendations are given on how to prepare the data for such a problem and how to apply CNNs to it.

The last chapter gives a short conclusion and additionally, it points out some possible future directions for doing research on this topic.

2. Background

To get a proper understanding of the problem there is some background information necessary. This chapter gives some knowledge about the basic concepts which this work builds up on. Due to the interdisciplinary character of this topic, this chapter includes both technical and non-technical terms and explains them in sufficient depth.

As a first step it explains the problem topic and gives some intuition on what can be considered as toxicity and some related and similar terms. Moreover, it talks about why the detection of toxicity is a non-trivial task. After that the technical background is given and includes the basic concepts of neural networks and some related technologies. To understand the whole pipeline there is a subsection about preprocessing which explains the used steps for this work. The last part of this chapter gives an explanation of the different evaluation possibilities.

2.1. Toxicity

The most challenging issue about toxicity and all its related terms is the lack of a clear and common definition. There are terms which are seen as overlapping by some people but are considered as completely distinct things by others. This leads to a considerably hard and also highly subjective decision-making process. Former US Supreme Court Justice Potter Steward once said in a case that he couldn't define what kind of material is obscene, but he knew it when he saw it [28]. This statement gives a good intuition on how subjective this field is.

The definition for toxicity this work is using is the following one based on the Wikimedia Detox project [63].

> Toxicity describes the likelihood that someone would leave a discussion because of this particular comment.

As one might notice, this definition only talks about discussions and comments. This is due to the focus on textual online comments in this work. To get a general definition for toxicity in conversations, this one would need to be adapted and extended.

This definition describes what toxicity is, but still does not give a clear decision boundary of the term. As already mentioned in the introduction it highly depends on the context on the person who judges the comment. A person who is used to a high amount of toxicity in his internet consumption very likely has a different boundary than a person who almost never experiences such content. That also means that a person can change its boundary over time. This can for example happen after being exposed to more toxicity than before for a longer period of time. Moreover, the decision might also depend on the mood of the person.

The most challenging thing, however, is the fact that some kinds of content only feel toxic to a limited amount of persons. If someone is bullied based on an incident which is only known by a few people, such a comment would not be necessarily seen as bullying from an outsider. This means that not only the comment itself, but also the context of the judging person influences the decision.

Also creating a machine learning algorithm for automatically classifying toxicity can be considered as very difficult. Such an algorithm would still be trained on data labeled by humans and therefore take over their bias [1][2]. However, machine learning algorithms suffer from a couple more problems than just subjective data.

Some examples are given by Hosseini et al [27] who did an analysis of Google's Perspective API. This API takes a textual comment and returns the toxicity score of it.

In table 2.1 two types of shortcomings are shown for this API. The left column gives comments provided as a sample on the Perspective API Website [30]. On the right side are the the attacks developed by Hosseini et al. The scores calculated by the API are shown in parentheses.

The first two sentences show an attack based on misspelled, but still easily recognizable toxic words. An ideal model should still be able to give such comments a high toxicity rating. However, in these two cases the score drops to a significantly

Original Phrase (Toxicity Score)	Modified Phrase (Toxicity Score)
Climate change is happening and it's not changing in our favor. If you think differently you're an **idiot**. (84%)	Climate change is happening and it's not changing in our favor. If you think differently you're an **idiiot**. (20%)
It's **stupid** and wrong (89%)	It's **stuipd** and wrong (17%)
They are liberal idiots who are uneducated. (90%)	They are **not** liberal idiots who are uneducated. (83%)
They are stupid and ignorant with no class (91%)	They are **not** stupid and ignorant with no class (84%)

Table 2.1.: Attacks on Google's Perspective API by Hosseini et al [27]. The first two comments show attacks on the API based on misspelled words. Comments 3 and 4 show false positives because the API does not understand the grammatical structure of the sentences.

lower number which would not be considered as toxic any more. Sentences three and four show a high toxicity rate although they are modified in a way to not be toxic any more. In these two cases the API neglects the semantic meaning of "not" and the toxicity scores only decreases by a few percent. These are two common examples of shortcomings when using machine learning models for this and related tasks and show the complexity of this problem.

This chapter mostly talks about about toxicity in textual content. When considering social media, however, only focusing on this kind of content is not sufficient. Toxicity may occur forms other than text too. Examples for this are images, videos or audio files. In extreme cases the toxicity even consists of combinations of these forms and cannot be detected when focusing only on one of them. To filter out various kinds of toxicity a combined solution which is able to analyze all of these contents would be necessary.

2.1.1. Related Terms

In the section before it is mentioned that toxicity is a very vague term which has no clear boundary to related terms. This section describes the related terms which are beneficial for a better understanding of the topic. It gives an overview of where toxicity starts, where it ends and which sub classifications there are. Moreover, all the terms which occur in later parts of this thesis are explained in necessary detail.

Abusive Language

The term abusive language is a very vague one and used in various ways in literature. In the scope of this thesis it is defined as being meant to insult, threaten or provoke individuals or groups of people. Moreover, it also includes comments which contain profane or offensive language.

Hate Speech

One term which has been discussed frequently in recent years is hate speech. According to Merriam-Webster the term is defined as ”public speech that expresses hate or encourages violence towards a person or group based on something such as race, religion, sex, or sexual orientation” [7]. As the definition already suggests hate speech itself can be further divided into subcategories based on the reason for the hate. Examples for such categories are racism, sexism, homophobia or political hate.

Literature shows that creating an annotated data set consisting of hate speech is a difficult task [51][61]. Annotators tend to disagree in particular cases and so the outcome of such a data set highly depends on the persons involved.

2.2. Neural Networks

In general, this work requires the reader to have a basic understanding of artificial neural networks. However, this section provides a short summary of important

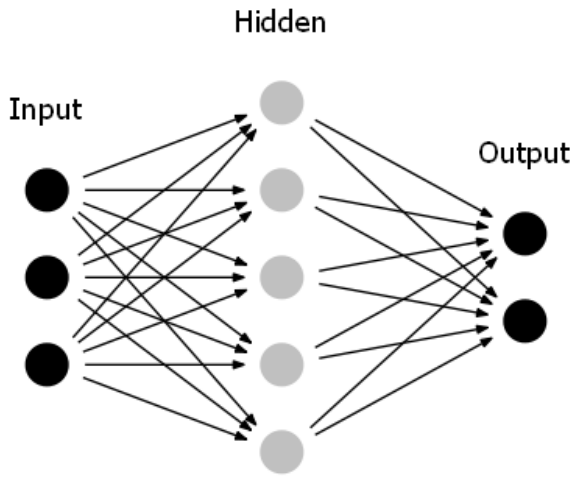


Figure 2.1.: An exemplary architecture of a neural network. It consists of an input layer, an output layer and a hidden layer in between.

concepts of neural networks.

An artificial neural network is a mathematical model inspired by the structure of human neural networks. It consists of neurons and connections between them. Usually, they are organized in multiple layers.

In figure 2.1 an exemplary neural network architecture can be seen. This particular model consists of an input layer with 3 neurons, an hidden layer with 5 neurons and an output layer with 2 neurons. As can be seen on the figure the data flows from the input towards the output layer and there are no connections inside of a layer or against the flow direction. Such an architecture is called feed-forward neural network [55]. Moreover, in this example every neuron in a particular layer is connected to all neurons of the subsequent layer. Such layers are therefore named fully-connected layers.

The characteristics of a neural networks are not only defined by the architecture, but also by its activation functions and the weights of connections. Every connection in the network has its own weight which which is optimized during the training phase. In general, a neuron calculates a weighted sum of all of its inputs which can be seen in equation 2.1. In the equation x_i denotes the input from neuron i and w_{ij}

is the weight for the connection from neuron i to neuron j. At the end a bias term is added to the equation which is denoted by b_j.

$$p_j = \sum_i w_{ij} * x_i + b_j \tag{2.1}$$

Before passing the value on to the next neuron an output function is calculated on the weighted sum. A common function in modern neural networks is the Rectified Linear Unit (ReLU) function which is defined by equation 2.2 [29][43]. This function returns 0 as an output for inputs < 0 and is a linear with slope 1 for inputs > 0.

$$o_j = max(0, p_j) \tag{2.2}$$

During the training stage the weights of the connections are updated to fit the training data. A very common algorithm to do the optimization is backpropagation [36] in combination with stochastic gradient descent. This algorithm calculates the difference between the expected and the actual output in a forward pass through the network. This error is then propagated back through the network and used to calculate the weight updates. These updates are proportional to the negative of the derivative of the error when using stochastic gradient descent. If the error converges the algorithm stops.

2.2.1. Convolutional Neural Networks

A particular type of neural networks are so called Convolutional Neural Networks (CNNs). CNNs gained much popularity after they were successfully applied to image classification, especially by Krizhevsky, Sutskever and Hinton in the ImageNet competition [35]. Since then they have been adopted by other areas and are now used in text classification too.

In general, this type of network makes use of three concepts: receptive fields, shared weights and pooling. In contrast to fully connected layers a neuron in a convolution layer is not connected to all neurons in the previous layer but just to the neighboring ones. These neighbors are called the receptive field of the neuron. In figure 2.2 an example of a receptive field of size 5x5 can be seen. This architecture

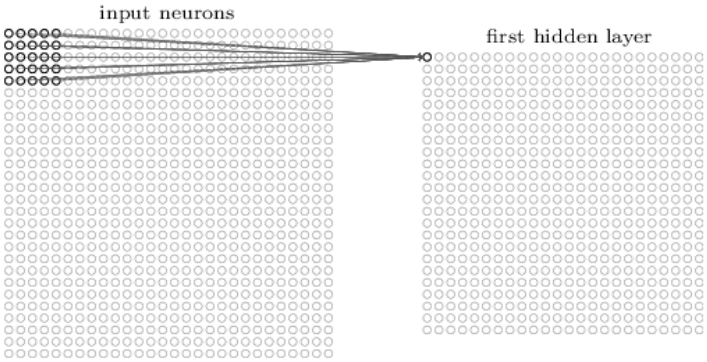


Figure 2.2.: The receptive field of the top left neuron of the convolution layer with a size of 5x5 [45]. The neuron in the hidden layer is only connected to the neurons in its receptive field instead of all neurons in the input layer.

allows for smaller models and make the network faster to train because of fewer connections.

The second concept, shared weights, make Convolutional Neural Networks (CNNs) even more efficient to train. Following this concept all neurons in the convolutional layer share the same set of weights. This set of weights is referred to as kernel or filter. This means that the top left neuron in figure 2.2 has exactly the same weights as the bottom right neuron. The idea behind this is a kernel which learns to detect one feature regardless of the position in the input. Such a feature could be a corner in images or a sequence of words in a sentence. However, a convolutional layer is supposed to detect more than a single feature. This can be achieved by using multiple feature maps. Each of them is supposed to learn a different feature and therefore has its own set of weights. An example of this can be seen in figure 2.3. The hidden layer is a convolutional layer with 3 different feature maps. Each of them comes with a kernel of size 5x5 and this results in a network which is able to detect 3 different features. In reality, however, such a layer consists of many more feature maps than just 3.

The last concept of a CNN is the use of pooling layers after convolutional layers. These layers aggregate the outputs of feature maps in order to provide simpler information to the following layers. A pooling layer takes all input values in a specified window and performs a function to aggregate these values. If the size is set to 2x2 it results in an output with quarter of the size of the input because the

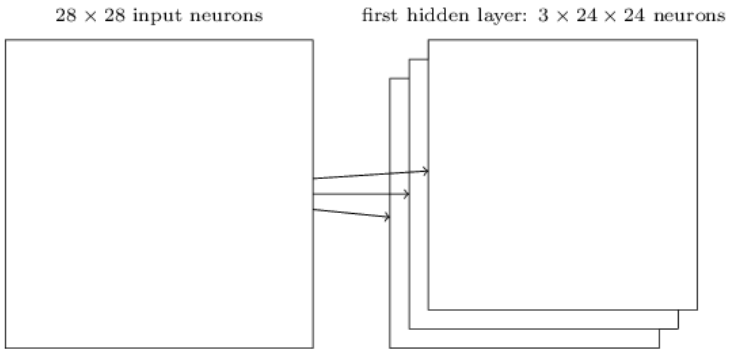


Figure 2.3.: Result of a convolutional layer with 3 kernels of size 5x5 [45]. The weights of the kernel are shared by all neurons in the same feature map.

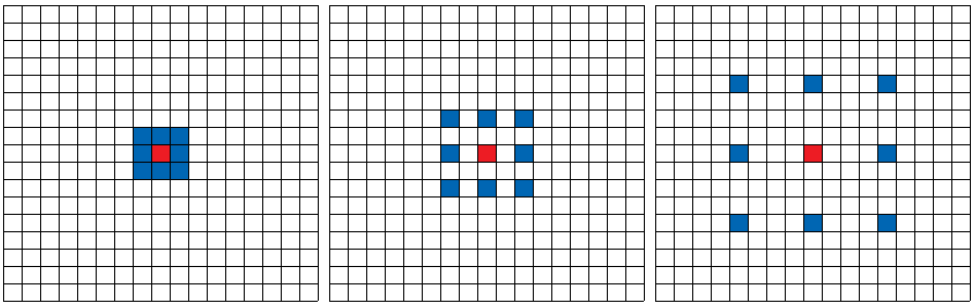
four values in each window are condensed to only a single output value. A common type is max-pooling where the highest value in the window is kept and all other values are discarded. However, a pooling layer is not a necessary part of a CNN. Springenberg et al show a network without them with results of state-of-the-art performance in image classification [54].

Dilated Convolution

Dilated convolutions are a special kind of convolutional layers and have been proposed by Yu and Koltun in 2015, who used this method for image segmentation [64]. Where ordinary convolutional layers consist of a continuous receptive field dilated convolutional layers have gaps in between. This concept allows the receptive field of a neuron to grow exponentially when adding more layers while the number of parameters just grows linearly. A dilation rate of 1 denotes an ordinary convolution with no gaps in between. A dilation rate of 2, however, causes a gap of a single value in the convolutional kernel. In figure 2.4 examples of the behavior of dilation rates 1, 2 and 4 can be found.

2.2.2. Word Embedding

A concept which has gained much popularity in recent years and is now widely used in Natural Language Processing (NLP) is word embedding. When considering



(a) Dilation rate 1 (b) Dilation rate 2 (c) Dilation rate 4

Figure 2.4.: Three examples for the change of the receptive field with different dilation rates. The receptive field consists of the blue and the red pixels. The red pixel also shows the location of the resulting value in the following layer. a) Dilation rate 1 results in an ordinary convolutional layer without gaps. b) Dilation rate 2 shows a gap of size 1 in each direction next to every pixel of the receptive field. c) Similarly dilation rate 4 shows a gap of size 3 in each direction.

a sentence which is split into single words every word is treated as an index in a dictionary. Usually a neural network would get this data in one-hot encoded form, where every row denotes a word in the sentence and every column maps a word in the dictionary. In each row all the values are 0 except for a single 1 which marks the actual word at this position.

This is a simple solution but has some serious shortcomings. First of all, this solution leads to a very large, but also very sparse matrix. When considering a dictionary size of 10000 words and a sentence length of 10 words this would lead to a input matrix with 100000 values. Moreover, out of this 100000 values there are only 10 values which contain a 1 whereas all the other values are set to 0. This results in a density value for this matrix of only 0.01%. The second shortcoming is a lack of contextual information about single words. In a one-hot encoded input it does not make a difference if two words are similar or completely different. They are treated as distinct inputs and do not share any kind of context.

A solution for these two issues can be the utilization of word embeddings. When applying this technique every word in the dictionary is mapped to a real-valued vector of a particular length. A vector length of 100 with a sentence length of 10 words would result in an input matrix with 1000 values. It also solves the sparsity problem of one-hot encoding because such embedding vectors mostly contain non-zero values.

A common paradigm to follow when creating such embeddings is based on a hypothesis of Harris, which states that words which occur in a similar context tend to have a similar meaning [23]. The skip-gram model by Mikolov et al [40] [41] is a technique which follows this paradigm and has been widely used in recent years. This model tries to find a word representation which is good at predicting surrounding words of the current word. This results in vectors which contain a certain amount of semantic information about the word. With a well trained model it is even possible to perform algebraic operations to find similar words. When computing the vector $X = vector(\text{"}biggest\text{"}) - vector(\text{"}big\text{"}) + vector(\text{"}small\text{"})$ the result is similar to the word representation of "$smallest$".

Other notable algorithms for creating word embeddings are GloVe by Pennington, Socher and Manning [49] and fastText by Bojanowski et al [5].

2.2.3. Transfer Learning

In general, many machine learning algorithms work under the assumption that training and test data are drawn from the same distribution. When the distribution changes most models need to be retrained from scratch with data from the new distribution. However, this is not always possible as sometimes the collection of new data is very expensive or time-consuming [47]. In such cases a concept called *transfer learning* can be helpful. It allows knowledge obtained from similar domains to be transferred to the new domain.

An example how transfer learning can benefit a CNN is shown by Zeiler and Fergus in 2013 [65]. They use a model trained on the ImageNet 2012 data set consisting of 1.3M training examples spread over 1000 categories. After that the model is adapted for classification of the Caltech-101 [16] and Caltech-256 [21] data sets. For this they crop the last layer of the ImageNet model and train a new layer fitted on the particular data set. With this method they surpassed state-of-the-art performance at that time in both cases.

2.3. Data Preprocessing

Usually when feeding data into a machine learning model not the raw data set is used. Sometimes it is necessary to transform the data into a shape which is understandable by the model. In other cases it increases performance if unnecessary features are removed, new ones are created or existing ones are modified. All these steps which are performed before actually passing the data to the model are combined in the term data preprocessing. This section gives an overview of preprocessing tasks for textual data used in the scope of this work.

2.3.1. Data Cleaning

Online comments, depending on the source, in some cases contain information which is not helpful to the task which should be performed by the machine learning model. The idea behind this step is to uniform the data and therefore help the model to generalize better to unseen data and not rely on features like misspelled words, multiple punctuation or URLs. The danger when performing these tasks is to remove important information and as a result negatively affect the classification performance. For this reason, these steps need to validated on the actual task instead of just relying on improved results.

Like mention before content like URLs or hash tags on Twitter (*#hashtag*) or mentions of other users on YouTube (*@username*) do not always provide valuable input for doing the classification task. Therefore, such parts might be removed from the raw data. An option to further clean the data is removing punctuation either completely or partially like removing duplicate question marks. On the one hand this step makes the data simpler but on the other hand it might take away an important feature.

In some domains a common task is to remove stop words [52]. This technique deletes common words which do not contribute to the semantic meaning from the text. This includes words like *the*, *a* or *and*. A benefit of this method is the reduction in size of the whole data set.

An step which is not in every case easily done automatically is spelling correction. Misspelled words might be a feature but above all they are differently spelled words

with the exact same meaning as the correct ones. In order to simplify the model and achieve a better generalization such words can be replaced by the correct word.

2.3.2. Tokenization

In section 2.2.2 is it mentioned that every word in the input comment is fed into the neural network as a separate row in the input matrix. The step which divides the comment into its pieces is called tokenization. This task needs rules where to split the text and which parts of it to keep as a token. However, there is no standard for tokenization. The actual choices made for the implementation highly depend on the task domain and the data itself [22]. A very simple tokenizer would only split at white spaces. This has the disadvantage that punctuation would be part of a single token with the word before.

When deciding for a tokenization strategy various things need to be taken into account like treatment of

1. dates
2. numbers
3. acronyms
4. enclitics like in *don't* or *he's*.

Moreover, another question to ask is the treatment of named entities like *New York City*. Syntactically, these are three separate words, but semantically this is a single entity. The detection of such entities, however, would require a certain amount of linguistic processing.

2.4. Evaluation

In order to perform a meaningful evaluation of a neural networks performance it is necessary to chose suitable evaluation measures. As it can be seen in section 7.1 the data sets used in this work contain highly skewed classes. According to He and Garcia [24] overall accuracy does not provide adequate information in the cased of imbalanced data sets. Therefore, other measures need to be used in this work. This section gives an overview of alternatives which are more suitable for the scope of this thesis.

A better measure for dealing with imbalanced data is the F1-score which is defined as the harmonic mean of precision (P) and recall (R). The exact definitions for precision, recall and F1 score can be found in equations 2.3, 2.4 and 2.5 respectively.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{2.3}$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \tag{2.4}$$

$$F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{2.5}$$

Another measure which can be used in such cases is the area under the receiver operating characteristic curve. The Receiver Operation Characteristic (ROC) curve is created by plotting the true positive rate against the false positive rate at multiple classification threshold settings [6]. The true positive rate is also known as recall and, therefore, defined in equation 2.4. The false positive rate, however, is defined as the number of false positives divided by the number of all negative examples in the data set which can be seen in equation 2.6. In order to get a single value instead of a curve for better comparison against other models the Area Under the Curve (AOC) is calculated.

$$False\ Positive\ Rate = \frac{False\ Positives}{True\ Negatives + False\ Positives} \tag{2.6}$$

A third measure which is suitable for skewed classes if the Matthews Correlation Coefficient (MCC). In contrast to the other two, this measure gives a value between -1 and +1. +1 denotes a perfect prediction, 0 is as good as a random prediction and -1 denotes total disagreement. The coefficient is defined in equation 2.7. Because of space limitations this equation uses abbreviations, where TP are true positives, FP are false positives, TN are true negatives and FN are false negatives. A shortcoming of the aforementioned F1 score is the lack of influence of true negatives to the score. MCC, however, overcomes this by giving equal focus on true positives and true negatives. If negative examples are equally important this may be the preferable measure.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.7}$$

However, in their raw form these three measures evaluate the performance of binary classification. To use these concepts in multilabel classification scenarios, like the experiments in this thesis, the measures need to be aggregated to represent the performance for the whole data set. Two common strategies for aggregating them are micro- and macro-averaging. Macro-averaging denotes the arithmetic mean of the measure for each label. This means every label contributes with the same importance to the overall value, regardless of its size. Micro-averaging, on the other hand, sums up the contingency matrices of each label and then calculates the measure. This strategy gives an equal importance to every sample in the data set. Therefore, larger classes have a higher influence to the overall value [18].

3. Related Work

This chapter is intended to give an overview of related work on this topic. Due to the focus of this thesis on both toxicity and CNNs this chapter is divided into two sections. The first one gives an overview of the work done on toxicity classification and similar tasks. To get an intuition about the use of CNNs in text classification tasks the second section focuses on this area.

3.1. Toxicity Classification

The lack of clear definitions of toxicity and similar tasks leads to research on very similar tasks, but using a different terminology [62]. Moreover, there is no standard data set for this area. So most of the work done so far uses a different evaluation set, which makes it even harder to compare their performances. In order to not hold back important work for this area this section covers not only toxicity but also work done in related tasks. This is especially important when considering the application of transfer learning in this work. Table 3.1 gives and overview of the related work mentioned in this section.

Early work in this domain used to maintain a blacklist of profane words. If a comment contains a word on this list it is classified as profane. An example is the work done by Sood, Antin and Churchill [53] who used public blacklists. Moreover, they also took into account profane words which are in a certain Levenshtein distance [37] of words in the comment. However, this approach has a low precision because some words on the blacklist are not profane in every context. Additionally, it also lacks recall as the list does not contain all known bad words and misspelled words might be ignored if the Levenshtein distance is too high.

A different approach are the opinion mining, sentiment analysis and sentiment classification tasks. These concepts are about extracting or classifying the opinion

Year	Authors	Algorithm	Features
2012	Sood, Antin, Churchill [53]	Unsupervised	Words, blacklist
2015	Pandarachalil et al [48]	Unsupervised	Word n-grams, polarity scores
2002	Turney [59]	Unsupervised	2-word phrases
2014	Hutto, Gilbert [19]	Unsupervised	sentiment scores
2014	dos Santos, Gatti [14]	CNN	Word embeddings, char embeddings
2004	Mullen, Collier [42]	SVM	polarity scores, emotive scores, topic, syntactic features
2016	Nobata et al [46]	Logistic regression	Char n-grams, linguistic features, syntactic features, word embeddings
2012	Warner and Hirschberg [60]	SVM	Word n-grams, POS tags
2017	Del Vigna et al [11]	SVM	Char-, word-, lemma n-grams, syntactic features (POS tags, ...), lexicon features (polarity score, ...)
2017	Del Vigna et al [11]	LSTM	Word embeddings
2018	Bhattarai [4]	LSTM, CNN	Word embeddings
2018	Li [38]	LSTM, GRU	Word embeddings

Table 3.1.: All related work presented in this section in the order of appearance in the text. Contained in the list is work on sentiment analysis, offensiveness classification, hate speech classification and toxicity classification. The table shows which algorithm is applied by the authors and which features are used for solving their task.

of a user towards a certain entity. While this is not exactly the same domain as the problem covered in this thesis there are arguably overlapping features which can be used both to detect toxicity and extract negative opinions. Work on these tasks has been done in supervised and unsupervised ways. Unsupervised approaches [48][59][19] use a similar method as the profanity classification in the previous paragraph. They make use of a sentiment lexicon which includes terms and their positivity/negativity score. Supervised methods [14][42] on the other hand need an annotated corpus to train the model on.

For detecting abusive language Nobata et al [46] propose a model which takes into account character n-grams, linguistic features, syntactic features and word embeddings. Linguistic features include information like number of capitalized letters, number of blacklisted words or the number of punctuations. Syntactic features on the other hand make use of word relations inside a sentence or Part-of-Speech (POS) tags. Chen et al [8] use a similar approach for classifying offensive language in social media. They extract the features and feed them into a Support Vector Machine (SVM) classifier.

In the area of hate speech detection there has been much research going on in recent years. Warner and Hirschberg [60] use an SVM fed with n-grams of words and POS tags. Del Vigna et al [11] compare the performance of SVM classifiers and Long Short Term Memory (LSTM) networks.

In the domain of toxicity classification itself not much work has been done so far. Bhattarai [4] shows a comparison of a bidirectional LSTM and a CNN. The results show a better performance when using LSTM. Li [38] applies two different types of recurrent neural networks, LSTM and Gated Recurrent Units (GRU), to this task. The results of this work show a slightly better performance for the model using a LSTM network.

The work by Dixon et al [13] on the other hand is not primarily about the classifier itself, but about measuring and mitigating unintended bias in toxicity classification. Such a bias can emerge from a data set where a word mostly occurs in a toxic context even though it has non-toxic meanings as well. Comments where this word exists get a higher toxicity score because of this unintended bias. Obviously, this can also happen the other way around with a word in the majority of cases occurs in a non-toxic cases but can mean something toxic too.

A common point which can be observed in all mentioned neural network based approaches is the application of word embeddings. Apparently this approach has a positive effect on the classification performance for toxicity and related tasks.

3.2. CNNs for Text Classification

As mentioned in the chapters before CNNs have originally been used for computer vision tasks. In recent years, however, they have been shown to deliver a good performance in text classification for various domains too. A notable work in this direction by Kim [32] uses them for classifying sentences. It uses a single 1D convolution layer but with different filter sizes which act like sliding word windows of different lengths. This architecture is evaluated on 7 data sets where is surpassed the state-of-the-art performance back then in 4 cases.

A very similar architecture is used for other classification tasks too. Georgakopoulos et al [17] use this architecture for classifying toxicity in comments. Nguyen and Grishman [44] apply this setup to the classification of relations between entities in a sentence.

All aforementioned networks make use of words as input features. A different approach for feeding texts into CNNs is to treat every character as a single token. Such a model is proposed by Zhang et al in 2015 [68]. This has the advantage that abnormal character combinations such as misspellings might be learned in the training process. An important choice in this architecture is whether to distinguish between upper-case and lower-case characters. They have observed that it usually gives worse results when this distinction is made.

None of the networks above use more than a single convolutional layer. In image classification, however, it is common to use multiple of those, often more than 10. Szegedy et al [56] use 22 layers for the ImageNet 2014 competition. For the ImageNet 2015 classification task He et al [25] integrate 152 layers into their model. An approach to go deeper for text classification too is proposed by Conneau et al in 2017 [9]. Their models incorporate between 9 and 49 convolutional layers and are fed with character level input. Johnson and Zhang [31] do a comparison of shallow word-level and deep character-level CNNs. Their results state that the shallow word-level model achieves a better performance in all of their experiments.

Moreover, it also computes much faster than the character model. However, the deep character-level uses more parameters and therefore consumes more storage space. This is not a drawback in all cases but might be considered in cases where storage is limited.

In addition to pure CNNs there is also work about hybrid models combining convolutional layers with other concepts. Zhou et al [69] propose a model which uses a convolution layer to extract higher-level features and then apply a LSTM layer on top of it.

4. Preliminary Work

In chapter 3.2 the relation classification task is briefly mentioned. As a preliminary project before starting with this thesis a CNN was developed which is able to classify relations.

For the experiments the data set from SemEval 2010 task 8 is used [26]. This data set contains a total of 8,000 training and 2,717 testing sentences. Each of them contains two marked entities *e1* and *e2*. The relation between them is labeled with one of 9 classes or *other*, if it is not a relevant relation for this task. Moreover, the direction of the relation is provided in the data set.

> The <e1>director</e1> has finished his new <e2>film</e2>.

This is an exemplary sentence in the data set. It is labeled as *Product-Producer(e2, e1)* which means that this is a Product-Producer relation. Film (*e2*) denotes the product while director (*e1*) is the producer.

The architecture for the project can be seen in 4.1. The network is fed with a matrix which includes the word index, the relative position to both entities and the POS tag for each word in the sentence, all of these values one-hot encoded. The first layer of the network transforms all of these values to dense vector representations. For the word index pre-trained word2vec embeddings are used whereas the vectors for positions and POS tags are initialized randomly.

As a next step the embedded input is fed into a single 1D convolution layer where three different filter sizes are used (3, 4 and 5). This approach acts like a sliding window with different lengths over a sentence. Convolution outputs of each of the filter sizes are pooled and then concatenated to a single vector. A fully-connected layer with softmax activation function serves as the output layer of the network.

The experiments done for this project mostly focus on the evaluation of different pooling strategies. The main approach used is global max pooling where only the

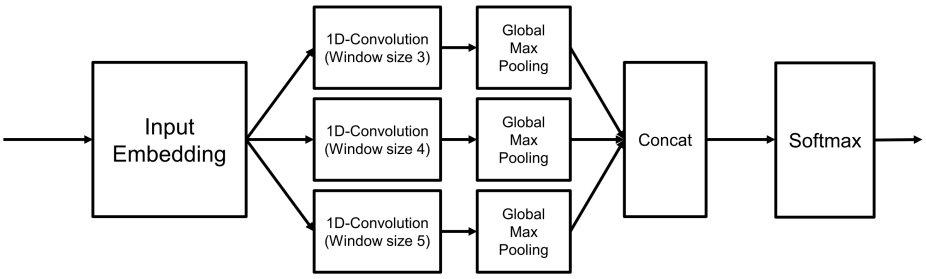


Figure 4.1.: Architecture for the relation classification task using CNNs. The network is fed with a matrix which represents a single word per row. Each row consists of the word index in the dictionary, the relative position to both entities and the POS tag. The embedding layer then transforms all of these features to embeddings. The convolution layer is divided in three different parts. Each of those applies filters of a different size to the embedded input. After applying max-pooling the values are concatenated and a softmax layer calculates the output of the network.

highest activation value of each filter map is kept. Similarly, 2-max and 3-max pooling keep the highest 2 and 3 values of every filter map, respectively. A slightly more complex approach is called piecewise max pooling [66]. This concept divides the output of the convolution layer into k different regions. For each region the pooling layer keeps the highest value. So instead of keeping a single value like global max pooling k values are kept. In this project every sentence is divided into 3 regions. One from the beginning to the first entity, one between the two entities and one from the second entity to the end of the sentence.

The last experiment uses global max pooling but uses a different sliding window approach which is named *E1-W-W-W-E2* approach in this project. It only uses a single filter size with length 3 for the convolutional layer. In contrast to the other models, every receptive field gets padded with the two entities. Considering the example sentence of the data set above the first window in the sentence would include the words *the, girl, ran*. When the *E1-W-W-W-E2* approach is applied the first entity is added to the beginning and the second entity to the end of the window. This results in a sliding window *girl, the, girl, ran, family*. The idea behind this concept is a stronger emphasis on the two entities.

All of these experiments are performed 10 times, and the results can be found in table 4.1. For every experiment the minimum, maximum and average value are taken. Global max pooling is the best-performing model. 2-max pooling and 3-max

	Global Max Pooling	3-max Pooling	2-max Pooling	Piecewise Max Pooling	E1-W-W-W-E2
AVG	83.3	81.4	82.3	83.0	82.1
MAX	83.5	81.7	82.8	83.5	82.6
MIN	83.1	81.1	81.9	82.5	81.5

Table 4.1.: F1 scores of all performed relation classification experiments. The first four experiments are related to different pooling strategies. Global max pooling only keeps the highest activation per feature map while 2-max pooling and 3-max pooling keep the highest two and three values, respectively. Piecewise max pooling divides every sentence into three different regions and keeps the highest value for each of them. E1-W-W-W-E2 relates to an experiment with a different sliding window approach where each window is padded with both entities. This method uses global max pooling.

pooling perform gradually worse. This suggests that each filter map learns to detect a filter which usually only occurs once in a sentence. Therefore, keeping more than one activation does not provide any additional value to the network. Piecewise max pooling performs slightly worse than global max pooling which suggests that the additional complexity added with this method is not helpful for this task. The low performance of the E1-W-W-W-E2 approach might be related to only using a single filter size instead of multiple like in the other models.

5. Method

Based on the theoretical concepts presented so far this chapter aims to give information on how they are applied in this work. It is divided into two different parts. The first part focuses on the preprocessing pipeline used for the experiments. It divides them into steps which are performed for all experiments in this work and the ones which are then evaluated for their contribution to a better performance. The second part gives information about the neural network architectures. These architectures are examined for their suitability to the task of toxicity detection.

5.1. Preprocessing Pipeline

To feed the comments into the networks a certain amount of preprocessing is necessary. This step is important for providing appropriate data to the network which will then be used for the classification. However, preprocessing involves the danger of removing meaningful information. Losing too much of this data might hurt the classification performance in the end.

For this reason, this section is divided into two subsections. The first one presents baseline preprocessing steps. These steps are performed in every experiment and include very basic techniques. The second subsection includes all the preprocessing steps which will later be evaluated on their contribution to the classification performance.

5.1.1. Baseline Techniques

All the preprocessing steps shown in this section are applied in every experiment of this work. They are also meant to serve as a baseline for all experiments on the evaluation of preprocessing steps.

The first technique applied is transforming all letters in the comments to lowercase letters. The idea behind this step is to increase the generalization ability of the classifier. This effect might be even stronger for texts in online discussions because they usually contain improper casing. After lowercasing all letters misspellings based on wrong casing can not occur any more.

The second preprocessing step which is applied is the tokenization step. This is a necessary step for feeding texts in the networks of this work. Tokenization is done so that every word is treated as a separate token. Every special character is treated as its own token. Exceptions of this are emoticons which are preserved and the whole emoticon is seen as a single token. An example can be seen in the following comment.

> You, sir, are my hero. Any chance you remember what page that's on? :-)

After tokenizing it these comment will be generated:

> ['You', ',', 'sir', ',', 'are', 'my', 'hero', '.', 'Any', 'chance', 'you', 'remember', 'what', 'page', "that's", 'on', '?', ':-)']

5.1.2. Evaluated Techniques

In contrast to the techniques in the previous section, the ones in this section are not applied to all experiments. The preprocessing steps presented here are evaluated on their benefit to the performance of the toxicity classification task. All techniques here are later applied independently from each other and evaluated as such.

The first technique which is evaluated is reducing the length of multiple characters. In this step every character which occurs more than 3 times in a row are reduced to a length of 3. A comment *This is waaaaayyyyyy too much for you!!!!!!* gets shortened to *This is waaayyy too much for you!!!*. As it can be seen there are less exclamation marks at the end and the word *way* is spelled with less *l* and *a*. This step is intended to serve two purposes. Firstly, it gives less emphasis on the number of special characters in a row. It does not matter if the comment contains 10 sequential exclamation marks or only three, both are treated the same way. Secondly, it increases the generalization ability. People sometimes multiply characters in a word to give more focus to it. While this potentially helpful feature is still retained, the variations are treated as

the same token independently of the number of character extensions. The number of retained characters is chosen to be 3 to provide a good balance between keeping the increased emphasis and increasing the generalization ability. If setting this parameter to 1 or 2, the transformed version loses the emphasis on the word always (when set to 1) or in some cases (when set to 2 and there are two consecutive letters in the correct word). Increasing this parameter to 4 or even higher would lead to a lower generalization effect. In this case *noooo* and *nooo* would be treated as two different tokens.

Another technique which is applied in the experiments is the removal of special characters. The intuition behind this is to create a highly simplified model. This preprocessing step very likely removes too much information from the comments and therefore effects in a worse classification performance. On the other hand, such a model shows the performance when only using plain words without any kind of additional information. If the results are still reasonably good, this technique might be used to speed of the training process and shrink the model size. As a weaker version only special characters which are not in *[.,!?]* are removed. Applying this weaker filter retains a minimum of sentence structure. This version is also evaluated as part of this work.

A preprocessing technique which focuses on transforming words is called lemmatization. The goal is to get the base form of a word. *going* and *went* would both get transformed to the base word *go*. This way the classifier becomes more robust, especially when being fed with a previously unseen form of a known base word. Assuming the training data only contains the two forms *going* and *went*. When using lemmatization the classification pipeline knows how to deal with the word *gone* too because all of them are transformed to their base form *go*. In this work the original tokens are replaced with their lemmas. A closely related technique is stemming, which has the same goal as lemmatization. However, the base form after stemming does not necessarily result in an actual word. Stemming is also evaluated as part of the preprocessing experiments.

The last technique which is examined in this work is taking a dictionary and replacing all tokens which do not occur in it with a special token. If this dictionary is created from a very large corpus this step is intended to remove misspellings and very rare slang words. The introduction of this special token indicates a potentially misspelled word and might be useful as a feature for the classifier. On the other

hand, there is also the danger that this removes important information of a comment and leads to a worse performance.

All of the techniques presented in these sections are applied before feeding the data into a CNN model. The results of these experiments can be found in chapter 8.2.

5.2. CNN Architectures

One goal of this thesis is to get an idea of how well CNNs perform on the toxicity classification task. To answer this question, experiments on four different architectures are done. This section presents the developed architectures and describes them in necessary detail.

The first model can be seen as a simple baseline architecture which all other architectures are compared to. It consists of an embedding layer in the beginning where the word indices are transformed to vectors of size n. The exact values for all hyperparameters are given in chapter 7.2. This embedding layer is followed by spatial dropout [58]. The purpose of dropout is to have a regularization effect by zeroing out random weights during training. However, regular dropout does not take into account the location of the weights being dropped. Spatial dropout overcomes this shortcoming and drops out certain areas of weights. In this architecture a 1D version of spatial dropout is used which zeros out whole word vectors randomly.

The output of the embedding layer is then fed into a convolutional layer. Instead of using 2-dimensional convolutions which is used in most work with visual image data the 1-dimensional version is used here. The meaning of a token is represented by a whole row vector. Therefore, 2-dimensional convolutions with a filter size less than the length of row vectors would only perceive a partial meaning of a token. 1-dimensional convolutions, on the other hand, contain filter maps which spread over whole rows and their size only defines how many tokens are seen in a single filter. In the end, they act as a sliding window of a certain size which moves over all tokens in a comment.

The results of the convolutional layer are then fed into a pooling layer. In the preliminary project shown in chapter 4 several pooling strategies have been evaluated in the context of relation classification. In this architecture the best performing

pooling strategy of the preliminary project is taken - global max pooling. As explained in 4, this type of pooling only takes the highest activation of each feature map.

After that the pooled output of the convolutional layer is passed to a fully connected layer. The result of this layer is then fed into an output layer. The output layer contains a neuron for every category in the data set and its result is calculated using sigmoid activation.

This is an overview of the baseline architecture. However, there is no detailed information given about filter map sizes, number of hidden units, activation functions and dropout rates. As already mentioned, the information about which hyperparameters are used for which experiments is presented in chapter 7.2. This also holds for the following architectures presented in this chapter.

5.2.1. Singlelayer CNN with Multiple Filter Sizes

The first architecture which is compared to the baseline architecture is still a singlelayer approach. In contrast to the baseline, this architecture uses multiple filter sizes. The intuition behind this architecture is learning from patterns in token n-grams of multiple different lengths. Other than that it still shares many concepts with the baseline model.

The model is build in a way that three convolutional layers operate in parallel. All of them get fed with the output of the embedding layer and their result is then pooled with global max pooling. To combine the different computational paths again the pooled output is concatenated to a single vector.

The overall architecture can be found in figure 5.1. As it can be seen from the description, this is a similar model to the one used in the preliminary work in chapter 4.

5.2.2. Multilayer CNN

The first two architectures shown in this chapter both consist of a single convolutional layer. This one, however, comprises of a second convolutional layer on top of the first one. In general, mutlilayer CNNs are not commonly used in text

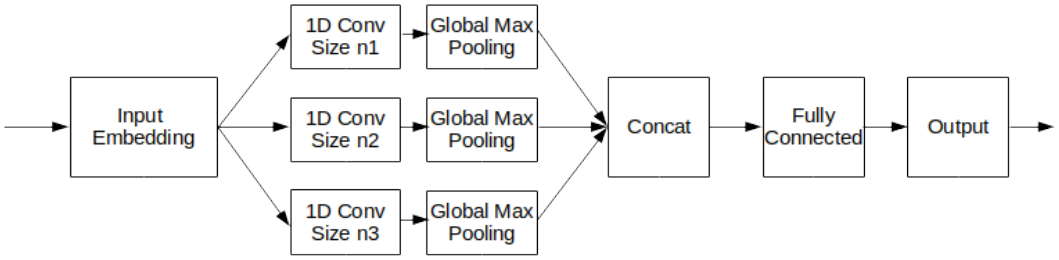


Figure 5.1.: Singlelayer CNN architecture using multiple filter sizes. The architecture is similar to the baseline architecture. The input is fed into an embedding layer which transforms all tokens to vectors. After that a convolutional layer is applied. Instead of using a single size for all filter maps this one uses three different ones. The pooled and concatenated output of these three convolutions is then passed to a fully connected layer. The output of the network is then calculated by a fully connected layer using the sigmoid activation function.

classification tasks. Their main field of application are tasks on images, especially in image classification [35][56][25]. A few studies have been done on using them for text classification. However, most of them use character level instead of word level features [9][31]. This work tries to use them with word level tokens as inputs to the network.

The intuition behind using multiple layers is the ability to learn higher level features. The first convolutional layers learns patterns from token n-grams, where n is the size of the receptive field. When applying a second one this one tries to learn patterns from the low-level patterns detected in the first layer. In theory, such an architecture is able to model more complex structures in comments.

Like the previous architecture, this one is again derived from the baseline model and the foundation if these two is still very similar. The main difference is the second convolutional layer immediately after the first one. The size of the filter maps does not necessarily have the same size as the one of the lower layer. This parameter determines the spread over which higher level features are able to detect patterns. The calculation of the total receptive field size of the second convolutional layer can be found in equation 5.1.

$$r_{out} = r_{in} + (k - 1) \tag{5.1}$$

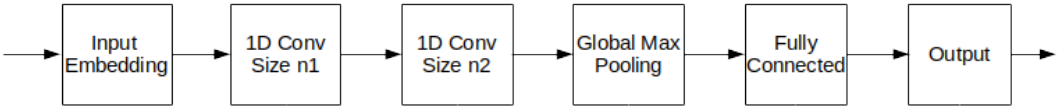


Figure 5.2.: Multilayer CNN architecture. The foundation is again similar to the baseline architecture. The input is fed into an embedding layer which transforms all tokens to vectors. After that a convolutional layer is applied. On top of its output a second convolutional layer is added which learns higher level patterns. Pooling is only applied after the second convolutional layer. The pooled output is then passed to a fully connected layer. In the end, the output of the network is calculated by a fully connected layer using the sigmoid activation function.

In this equation r_{in} is the receptive field of the layer below. k denotes the filter size in the current convolutional layer. This is a simplified version for computing the receptive field size in 1D convolutions.

There is no pooling layer involved between the two convolutions. This has the effect that the second layer learns its patterns from the raw output of the first layer. Pooling is finally done on the output of the second convolutional layer. As in the previous architectures this one also applies global max pooling to only keep the highest activation of a feature map. The higher layers follow the same concepts as in the previous architectures. There is a fully connected hidden layer and another fully connected layer for computing the output of the network. An overview of the architecture can be found in figure 5.2.

5.2.3. Multilayer Dilated CNN

The last architecture which is evaluated as part of this thesis is a multilayer architecture with dilated convolutions. Similarly to the regular multilayer CNN the dilated version of it is also not commonly applied to tasks on textual data. It finds its use for example in image segmentation [64]. In this work the goal is to find out if this architecture is suitable for toxicity classification too. A detailed explanation of dilated convolutions can be found in chapter 2.2.1.

In general, dilated convolutions are a way to increase the size of the receptive field in multilayer CNNs. Compared to a regular convolutional layer this can be achieved by using fewer parameters which need to be optimized. The original authors take convolutions of size 3×3. The dilation rates start with 1 in the first layer and are

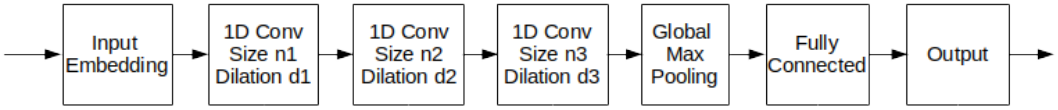


Figure 5.3.: Overview of the architecture which uses dilated convolutions. Like in the other architectures an embedding layer transforms the tokens into vectors. These embeddings are passed into three consecutive convolutional layers with different dilation rates and filter sizes. Dilation rates > 1 increase the size of the receptive field without increasing the number of parameters to learn. The output of the last convolutional layer is pooled and passed to a fully connected layer. As a final step a fully connected layer using the sigmoid function computes the output of the network.

multiplied by two for the subsequent layer. When applying their approach to 1D convolutions the receptive field size for this setting can be calculated as stated in 5.2. In this equation i denotes the index of the convolutional layer. It can be seen that the receptive field grows exponentially, but with a linear increase of parameters by adding new layers with the same filter size.

$$r_i = 2^{i+2} - 1 \tag{5.2}$$

In the architecture of this work three convolutional layers are used. Assuming the dilation rates and filter map sizes of the original authors (dilation rates 1, 2, 4…; filter size 3) are applied to this network this would result in a receptive field size of 15 in the third layer. When applying regular convolutions with a dilation rate of 1 the receptive field would end up with a size of 7. However, both of them use the same number of parameters. The actual hyperparameters (dilation rates, filter sizes, …) which are used for the experiments in this work can be found in chapter 7.2.

An overview of the architecture is given in figure 5.3. It can be seen that between the convolutional layers there is no pooling operation applied. This is consistent to the regular multilayer CNN in the previous section.

5.3. Transfer Learning for Small Data Sets

One of the goals of this thesis is the evaluation of CNNs on a second data set. This is done to verify the results of the experiments on the main data sets. However, in

section 7.1 it can be seen that there is a large difference in the number of comments between these two. To see if the knowledge of the model trained on the large data set can be helpful for training on a small comment set for a similar task, transfer learning is applied.

In this experiment, the singlelayer CNN architecture with multiple filter sizes is used for classification. Details about this architecture can be found in section 5.2.1 earlier in this chapter. This architecture is first trained on the large main data set. As explained in section 2.2.3, the output layer of the model is then cropped and replaced by a new one. The number of neurons in the newly created output layer matches the number of labels in the smaller data set.

After that, the model is trained like in all other experiments. This way, the weights obtained from training on the larger data set are used for initializing the model which is then trained on the smaller comment set. Other approaches freeze the weights transferred from other models and only train the newly added layers. In this experiment, however, all layers are trained with the same settings as in the training phase on the main data set. Therefore, there are no frozen weights in this approach. A detailed explanation of the training process can be found in section 7.3.

6. Tagging Tool

One of the tasks in this thesis is the creation of a data set for toxicity, abusive language, hate speech and some related terms. This is, however, a mostly manual work and therefore very time consuming. In order to minimize the duration for creating this and future data sets a web based tagging tool is created.

The goal of this tool is that a user can upload video metadata crawled from YouTube and their corresponding comments and replies to the system. Additionally, a tagging interface is provided where the user can see the video and a single comment of it. On this interface it is possible to assign labels to this particular comment.

As a prerequisite for the labeling task, a crawling script is developed to load the needed data from YouTube. It is possible to specify a search term and the number of videos wanted in the data set. The script then queries the YouTube API and writes the videos to a file. After inspecting if all videos are suitable a second script extracts all the comments to these videos from the API. The information stored about comments includes their id, an optional id of a parent comment, the author, the comment itself and a language code created by the script. The language code is calculated by the crawling script but is not used by the tagging tool. This information is intended to help filtering out comments in other language before uploading the files. To create a new data set in the system the two generated files are uploaded and a title and a language are specified.

When the data set is finally present in the system users have the possibility to label the comments in it. For this task two modes are available. The first one provides the user with comments which have not been labeled before. This is the main mode if a user is working on a newly created data set.

However, in some cases it is preferred that multiple people label the same comment. This is necessary if the inter-rater agreement should be measured or a more robust data set with fewer mistakes in it should be created. For these cases the second mode provides a user with comments already labeled by other people.

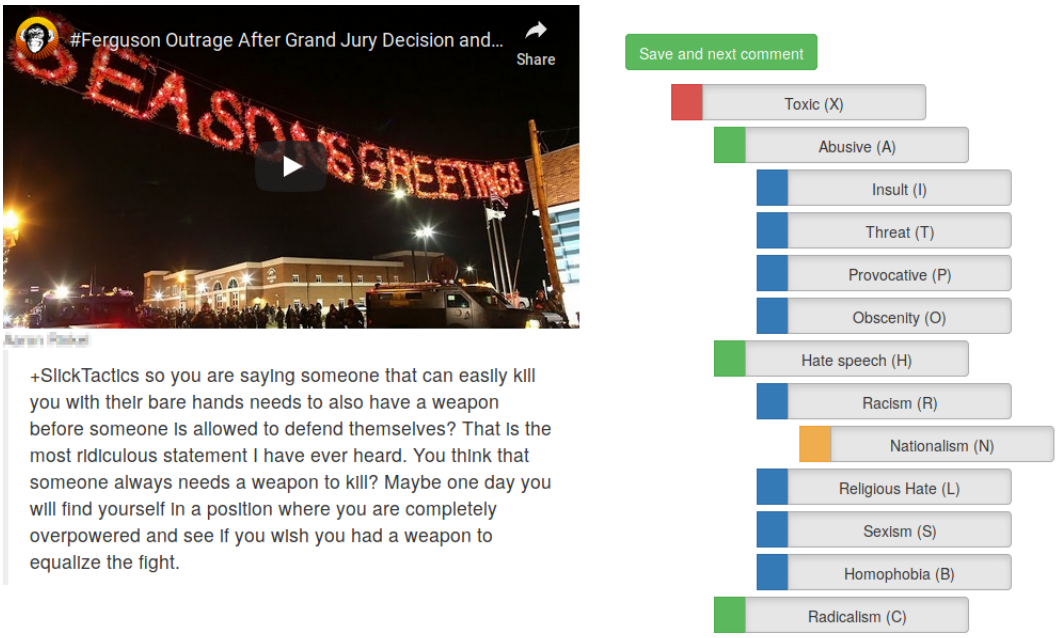


Figure 6.1.: User interface of the tagging tool. A tree with buttons for all labels is provided. If a user selects a label, all of its parent labels are automatically selected, too.

A strong emphasis is put on a fast navigation on the tagging interface. This is especially important because of the number of comments which need to be tagged to get a decently sized data set. The actual tagging interface can be seen on figure 6.1. The buttons on the screenshot show all the different labels available. To create a consistent data set the parent label is automatically checked if one of its children is selected. If a parent is unchecked, on the other hand, all of its children are automatically unselected. To further speed up the tagging process keyboard navigation is provided. Pressing the letter which is specified in parentheses on the button checks or unchecks this label. Moreover, the user is able to skip a comment for later tagging.

Supplementary, a classification script is provided which acts as a simple baseline. This script should give an intuition of the performance gain when tagging more comments.

7. Experimental Setup

This section gives the details about the actual setup of all experiments. Special emphasis is been put on the data sets the models are trained and evaluated on. A topic of high importance for the performance of a neural network is the choice of the network's hyperparameters. The settings of the hyperparameters are listed in section 7.2. Moreover, it also explains the process of finding good values for this problem domain. The last section in this chapter explains the training process for the models.

7.1. Data Sets

Although toxicity and related domains like hate speech or abusive language are widespread phenomenons on the internet there are no standard data set for these problems. This section shows the two data sets which are used for the experiments in this work. Both are taken from different domains are therefore show varying characteristics. These characteristics and the differences and similarities of the data sets are indicated in this section.

7.1.1. Kaggle Toxic Comment Classification

The first data set used for the experiments is part of a Kaggle competition called "Toxic Comment Classification" by Jigsaw. [1] The data has been collected by Thain, Dixon and Wulczyn [57]. It contains comments collected from Wikipedia talk pages where each comment is labeled by 10 crowd workers.

[1] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/

	Train	Test
Total number of comments	159,571	63,978
No Label	143,346	57,735
Toxic	15,294	6,090
Severe Toxic	1,595	367
Obscene	8,449	3,691
Threat	478	211
Insult	7,877	3,427
Identity Hate	1,405	712

Table 7.1.: The size of train and test set of the data taken from the Kaggle "Toxic Comment Classification" competition. It can be seen that the data contains highly imbalanced categories. Roughly 10% of the data is labeled as toxic and all the subcategories are even smaller. The smallest class in the training data, threats, only make up for 0.3%.

The main category in this data set is whether a comment is toxic or not. However, it also contains 5 subclassifications for a further distinction of different reasons why people consider it as toxic. Therefore, there are 6 labels in total in this data set: *toxic*, *severe toxic*, *obscene*, *threat*, *insult* and *identity hate*. *Severe toxic* denotes a stronger form of toxicity whereas the other 4 categories show the kind of toxicity present in the comment. *Identity hate* contains comments which insult a person or a group of people because of traits like skin color, religion, sexual orientation or similar. This content highly correlates with the hate speech definition given in 2.1.1.

The data is divided into two sets, the train and the test set. The actual sizes and the number of comments per label in both sets can be found in table 7.1 A point which is made visible by the class sizes is the highly imbalanced characteristic of this data set. The relative sizes vary from 10% of the training data (*toxic*) to a low of 0.3% of the training data (*threats*).

After closer examination of the numbers one might notice that the sum of comments without a label and toxic comments is lower than the number of total comments. This states that not all of the categories are subsets of the *toxic* label. An investigation of the data resulted in the fact that only *severe toxic* is a subset of *toxic*. A comment labeled as one of the other four categories is not necessarily labeled as *toxic* too. So this data set treats these categories as highly overlapping but not as a subset of

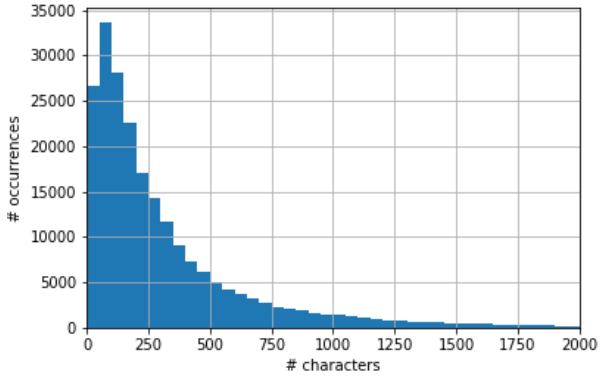


Figure 7.1.: This plot shows the number of characters per comment in the overall data set. The median over all comments has a length of 203 characters.

toxicity.

To provide a better comparison ability, this section gives some information about the shapes of the comments in the data set. In figure 7.1 the number of character in the comments can be seen. This gives some intuition about the actual length of comments on Wikipedia talk pages. The median length is 203 characters whereas the maximum is 5000. This suggests that this data set contains comments only a few words long as well as ones where multiple sentences are included in it. Another interesting measure to compare to textual corpora is the punctuation rate in it. This measure denotes how much punctuation there is in a comment per character. The mean punctuation rate in a comment in this data set is 0.051. Similarly, the mean rate of uppercase characters in a comment is 0.052. A comparison of the two data sets can be found in table 7.3.

7.1.2. YouToxic English

In contrast to the Kaggle data set in chapter 7.1.1 the YouToxic data set is created as part of this thesis. The goal of using a second data set for evaluation is to show that Convolutional Neural Networks (CNNs) also work for data coming from sources other than Wikipedia talk pages. Therefore, YouToxic English provides comments on videos from the YouTube platform.

	Overall
Total number of comments	1,000
No Label	538
Toxic	462
Abusive	353
Hate Speech	138

Table 7.2.: The size of YouToxic data set created as part of this thesis. In contrast to the Kaggle data set there are no predefined train and test sets. Therefore, the overall numbers are given. This data set is much more balanced than the Kaggle set with roughly half the comments containing a label.

For the creation of this data set the tagging tool in chapter 6 has been utilized. The tool has been filled with comments coming from 13 different videos about the unrest in Ferguson, Missouri in 2014. [2] This topic has been chosen because it is highly controversial and provides a sufficient amount of toxicity in its comments. As can be seen in chapter 6 the tagging tool provides the ability to give a comment up to 13 different labels. However, for this work there are only three of them used: *Toxic*, *Abusive* and *Hate speech*. For a more fine-grained classification, the data set does not provide enough data.

In this data set the label *Toxic* is seen as a parent label for both *Abusive* and *Hate speech*. If a comment is labeled as either one of the two child labels it is also labeled as the parent label. This is ensured by the tagging tool used for creating this data set.

The actual size of the data set can be seen in table 7.2. In contrast to the Kaggle data set this one is not divided into train and test set. For this reason, the table gives the sizes for the whole set. There are exactly 1,000 comments available where 538 do not contain any label and the other 462 are at least labeled as *Toxic*. It can be seen that this data set is relatively balanced compared to the Kaggle data set.

Coming from two different sources the data sets show significant differences in the structure of the comments. The comments coming from Wikipedia discussion pages are on average more complex. This complexity exposes itself in a higher number

[2] https://en.wikipedia.org/wiki/Ferguson_unrest

	Kaggle	YouToxic
Number of characters	391	186
Punctuation rate	0.051	0.039
Uppercase rate	0.052	0.053

Table 7.3.: Comparison of both data sets regarding the shape of the containing comments. Each of the values denotes the mean over all comments. It can be seen that the average comment in the Kaggle data set is more complex due to a higher number of characters and a higher punctuation rate. The rate of uppercase characters is very similar.

of characters and also a higher punctuation rate. The uppercase rate, on the other hand, is very similar in both data sets.

7.2. Hyperparameters

To get good results out of a neural network the choice of suitable hyperparameters is of high importance. This section gives an overview of the parameters used for the experiments in this work.

The basic network structure is the same for all architectures and all experiments. An embedding layer is used to convert the tokens to vectors of size 300. It can be seen in equation 7.1 that the number of parameters to be optimized in this layer highly depends on the number of unique tokens known by the model.

$$n_{params} = 300 \times n_{tokens} + 300 \tag{7.1}$$

The number of parameters in this layer constitute the vast majority of the number of all parameters in the model. Therefore, the preprocessing pipeline has a high effect on the overall model size.

After the embedding layer a 1D spatial dropout layer with dropout rate 0.2 is applied. This technique is intended to reduce overfitting of the model. In contrast to regular dropout, the spatial version of it randomly zeros out whole rows of the layer before. Applied to textual input this makes the network zeroing out a whole token vector during training.

CNN Architecture	Filter sizes			Number of filters		
	L1	L2	L3	L1	L2	L3
Singlelayer	3	-	-	150	-	-
Multiwindow	3/4/5	-	-	100/100/100	-	-
Multilayer	3	3	-	150	150	-
Dilated	3	3	3	150	150	150

Table 7.4.: This table shows the hyperparameters of the convolutional layers in the four evaluated architectures. L1-L3 denote the particular convolutional layer. For the multiwindow CNN the numbers given separately for the three convolutional blocks in the first layer.

The embedding layer with dropout applied is followed by the particular convolutional parts. These parts are particular to the different architectures and can be found later in this section. The higher layers of the network consist of a fully connected layer of size 100 where regular dropout is applied to with a rate of 0.4. The output of the network is calculated with another fully connected layer where the size depends on the number of classes in the data set.

An overview of the details of the convolutional parts for each network can be found in table 7.4. It can be seen that all the layers use filters of size 3. With this setting filters in the first layer learn to detect patterns which spread over a maximum of 3 tokens. The only exception is the single layer CNN with multiple window sizes where the sizes 3, 4 and 5 are used. Similarly, all layers use 150 filter maps with the exception of the multiwindow architecture where 100 are used for each filter size. The dilated CNN uses dilation rates 1, 2 and 4 for its three layers. The third convolutional layer has a receptive field size of 14 tokens with these hyperparameters.

7.3. Training

This section gives information about the training process of the neural networks. Due to the similar architectures and also similar data sets used most parts of the process are the same for all experiments.

One of the parts where the experiments partially differ is the initialization of weights. In all those experiments where pretrained embeddings are used these values are

taken as initial weights for the embedding layer. In all other cases the embedding layer is initialized with random values taken from a uniform distribution ranging from -0.05 to $+0.05$. The subsequent layers (fully connected and 1D convolutions) use Xavier uniform initialization [20]. In this initialization method the range of the uniform distribution depends on the number of incoming and outgoing connections of the particular neuron.

For optimizing the parameters of the network the Adam algorithm is used [34]. The parameters of the algorithm are set to the suggested values in the original paper ($\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$). As a loss function, the experiments are performed using binary cross-entropy, where the cross-entropy for each label is calculated and the mean of them is taken. The definition for a single label can be found in equation 7.2.

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N} [y_i \, log \, \hat{y}_i + (1 - y_i) log(1 - \hat{y}_i)] \tag{7.2}$$

Training is done for 5 epochs and uses a batch size of 64 comments. All the values in this section are based on findings in the literature on similar work. Therefore, there is no evaluation done on finding the optimal values for the task of toxicity classification.

8. Evaluation

This chapter gives information about the results of all experiments. In the first section, the evaluation results of four different CNN architectures are provided. The second section shows the measures of different preprocessing techniques and how they contribute to the classification process. In section 8.3 information about what the neural networks actually learn can be found. This includes the importance of tokens for the classification process and the features which the convolutional filters learn to detect.

For performance evaluation of the different models, F1 score is chosen as the primary measure. However, for the comparison of neural network architectures, ROC AUC is used as a second measure. The intuition behind using an alternative measure is an improved comparability with the results of the Kaggle challenge the main data set is taken from. F1 score is chosen as the primary one over ROC AUC because it draws a more realistic picture of the model's actual capabilities of finding toxic comments in these imbalanced data sets.

The details about the software packages used for implementing the experiments and also the hardware used for training and evaluating the models can be found in appendix A.

8.1. Comparison of Architectures

The main research goal of this thesis is the evaluation of CNNs suitability for the toxicity classification task. For answering this question the four architectures presented in chapter 5.2 are evaluated on the data set of the Kaggle toxic comment classification challenge. The results are intended to show the raw performance of a CNN on the data set without using sophisticated preprocessing techniques. Therefore, the experiments shown in this section only use a minimum amount of

preprocessing. The techniques applied to the data consists of lowercasing of all characters and tokenization using the NLTK TweetTokenizer[1].

To provide a better comparability with results from Kaggle the experiments in this section use pretrained word2vec embeddings trained on data obtained from Google News[2]. Using these embeddings slightly improves the results which can be seen when comparing the results in this section with the ones in the preprocessing section in 8.2 where random initial embeddings are used.

The actual results of the four evaluated architectures can be found in table 8.1. The numbers shown in the table are the micro-averaged scores over all six labels in the Kaggle data set. All experiments have been performed 5 times and the mean and standard deviation is presented here. It can be seen that the two single-layer architectures provide the best performance for this data set with an F1 score of roughly 0.64. The multilayer CNN which applies two convolutional layers achieves a lower score than the aforementioned ones. The worst-performing architecture is the dilated CNN, which includes three convolutional layers. These results suggest that the additional complexity of additional layers is not needed for the task on this data set.

The precision-recall curves of the singlelayer CNN with multiple filter sizes can be seen in figure 8.1. It shows the curves for all labels in the Kagge data set. It can be seen in there that the model performs worse for labels with less training data available. The more prominent labels *toxic*, *obscene* and *insult* provide both higher precision and higher recall for most parts of the curve.

As mentioned in the introductory part of this chapter the ROC AUC score show overly optimistic results for the classifiers even though the actual classification capabilities. The highest ROC AUC score of these models is almost 0.98 whereas the same model only reaches the aforementioned F1 score of 0.64. This can be attributed to the strong label imbalance in the data set. The difference between these two

[1] https://www.nltk.org/api/nltk.tokenize.html#module-nltk.tokenize.casual
[2] https://code.google.com/archive/p/word2vec/
[3] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/discussion/52557
[4] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/discussion/52612
[5] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/discussion/52762
[6] https://www.kaggle.com/prashantkikani/pooled-gru-with-preprocessing
[7] https://www.kaggle.com/demesgal/lstm-glove-lr-decrease-bn-cv-lb-0-047
[8] https://www.kaggle.com/jhoward/nb-svm-strong-linear-baseline

Model	F1	ROC AUC
Singlelayer CNN	0.6398 ±0.0045	**0.9790** ±0.0003
Singlelayer CNN with multiple filter sizes	**0.6401** ±0.0072	**0.9790** ±0.0006
Multilayer CNN	0.6327 ±0.0029	0.9776 ±0.0008
Dilated CNN	0.6290 ±0.0059	0.9763 ±0.0006
Kaggle 1^{st} place[3]	-	0.9885
Kaggle 2^{nd} place[4]	-	0.9882
Kaggle 3^{rd} place[5]	-	0.9880
Kaggle GRU + GloVe embeddings[6]	-	0.9800
Kaggle LSTM + GloVe embeddings[7]	-	0.9779
Kaggle Logistic Regression[8]	-	0.9772

Table 8.1.: This table shows the results of all four evaluated CNN architectures on the Kaggle data set. The numbers in the table are the micro-averaged scores over all labels in the data. Each of the experiments has been performed 5 times and the mean and standard deviation are given in this table. It can be seen that the singlelayer architecture with multiple windows sizes performs best for both F1 score and ROC AUC. The results of the Kaggle challenge only give ROC AUC results because this is the official measure of the challenge.

measures can easily be seen when comparing the precision-recall curves in 8.1 and the ROC curves in 8.2.

For a comparison of the models in this work with state-of-the-art methods, the table includes results from the Kaggle toxic comment classification competition. As mentioned before they only provide ROC AUC scores, therefore, a comparison by F1 score cannot be done here. The first place solution uses a bidirectional GRU architecture which uses multiple pre-trained embeddings. They also rely on augmenting the data set by adding machine-translated comments. Multiple models were trained and stacked to create a single output in the end. The second best model of the competition is an ensemble classifier using RNNs, CNNs and gradient boosting machines. This model also relies on using multiple pretrained embeddings and machine translation for augmenting the data set. The third-best model again uses an ensemble of a variety of classifiers as LSTM, GRU, logistic regression etc.

All highly ranked models obtain their good results by heavy data preprocessing and augmentation and also on training multiple models and aggregating them in various

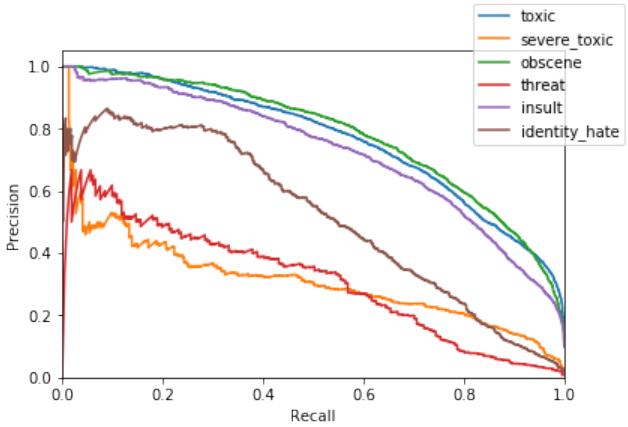


Figure 8.1.: This figure shows the precision-recall curves of the singlelayer CNN with multiple filter sizes. It shows the curves for all labels in the Kaggle data set. It can be seen that the curves for *toxic*, *obscene* and *threat* show a higher performance and are also stabler. This is due to the much higher number of training samples for these three labels.

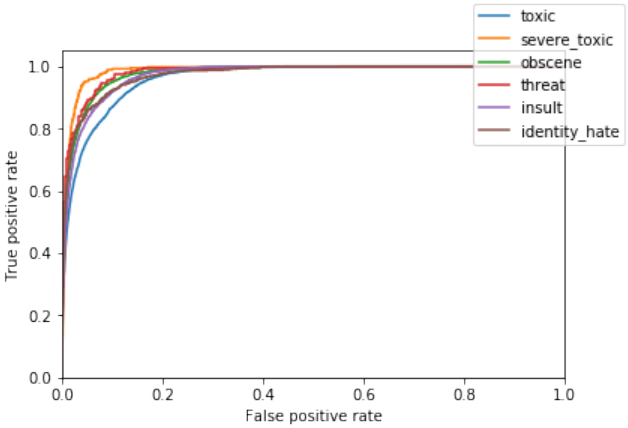


Figure 8.2.: In contrast to figure 8.1 this figure shows the ROC curves for all labels. When comparing it to the precision-recall curves these ones show an overly optimistic picture of the actual classification capabilities. Even the labels with less data available (*severe_toxic*, *threat*, *identity_hate*) show a very good ROC curve.

ways. To evaluate the raw suitability of a CNN for classifying toxic comments, however, the models in this section only apply a minimum amount of preprocessing and train a single classifier. Therefore, the comparison should be done with other raw classifiers in the Kaggle competition. Three different results can be found in table 8.1. It can be seen that an LSTM with a very similar amount of preprocessing and pretrained GloVe embeddings performs similarly to the singlelayer models in this thesis. A similar result is obtained by applying logistic regression to TF-IDF vectors. A model using a GRU network and uses a slightly higher amount of preprocessing obtains a ROC AUC score of 0.98.

8.2. Comparison of Preprocessing Techniques

In contrast to the previous section, the experiments in this sections focus on the steps done before feeding a CNN with the data. They are intended to show the effectiveness of various preprocessing techniques on the toxicity classification task. Effectiveness, however, is not only shown by comparing the performances of the evaluated approaches. In addition, to that, this section gives information about the number of unique tokens in the data set and how they influence training time and the number of parameters in the model.

All the results given here are obtained by applying the preprocessing techniques individually to the Kaggle data set. The only preprocessing technique which is applied across all experiments is lowercasing of all characters. As mentioned in the methods chapter in 5.2.1 these experiments use the singlelayer CNN with multiple filter sizes. The hyperparameters of the model are explained in chapter 7.2. All experiments in the previous section use pretrained Word2Vec embeddings trained on Google News. However, to only focus on the preprocessing and to prevent the pretrained embeddings from influencing the performance of certain preprocessing techniques the embeddings here are randomly initialized. Therefore, the baseline model here achieves a slightly lower score than the corresponding model in previous experiments.

An overview of the results can be found in table 8.2. It can be seen that - with the exception of reducing the length of character sequences - all preprocessing techniques improve the performance of the classifier. The improvements range from very small ones when removing all or parts of punctuation to relatively significant

improvements when performing stemming, lemmatization or replacing all words not found in a dictionary.

In addition to an increased performance, these preprocessing techniques also provide other advantages. The results in table 8.2 state that all these techniques also reduce the number of unique tokens in the data set. This makes the classifiers more robust because they need to focus on a smaller number of features in the data. Also, stemming and lemmatization transform the tokens to their root form, which helps the classifier treat different shapes of the same word the same way.

As briefly explained in chapter 7.2 the number of unique tokens in the data set is directly proportional to the number of parameters in the embedding layer. The parameters embedding layer represents the majority of parameters in the whole model. Therefore, these preprocessing techniques also reduce the overall size of the neural network. Replacing unknown tokens in the data set with a special token, as explained in section 5.1.2, is a very drastic technique for reducing the number of tokens. The dictionary which is used for determining whether a token is known or not, is the aforementioned Word2Vec model trained on Google News. All tokens which can be found in there are considered as known. When applying this technique, there are only 23% of all unique tokens left in the data.

This reduction is also visible in the training time per epoch, although to a smaller extent. Reducing the number of unique tokens from the baseline of 326,175 to 74,212 results in a training speedup from 680 seconds to 545 seconds per epoch, which is a reduction of around 20%. When using other preprocessing methods this improvement is not that strong anymore. Removing all punctuation from the comments leads to a training time of 665 seconds per epoch.

In addition to applying individual preprocessing techniques to the data set, this thesis also includes experiments on combinations of these approaches. Table 8.2 shows the results of the evaluated combinations. The first four experiments focus on combining the two approaches for removing punctuation with both stemming and lemmatization. It can be seen from the results table that all of these combinations achieve a similar performance. Moreover, this performance is similar to the individual performance of stemming and lemmatization. The main difference between the four combined preprocessing strategies is the number of unique tokens, which ranges from 223,000 to 279,000. This means the reduction rate varies varies between 14% and 31%. The fifth combined approach replaces unknown tokens and lemmatizes all remaining words in the data set. The result of this experiment

again shows a very similar performance as all the other combined preprocessing strategies. When compared to the individual result of replacing unknown tokens, however, this combination achieves a lower F1 score. An advantage compared to the other ones is the even more drastic reduction in the number of unique tokens. When applying this strategy to the Kaggle data set the number gets reduced by more than 80%.

8.3. Feature Importance

When applying machine learning models in production systems it is important to know what the model actually learns and which features influence the model's decisions. CNNs are considered as being black boxes and hardly interpretable due to the huge number of parameters and the complex structure. The results in this section try to shed light on the knowledge of a CNN trained on classifying toxicity in online comments.

For the analysis, the trained baseline model of the preprocessing experiments is used. This means that, again, there are no pretrained embeddings involved. The analysis of the model is done using two different approaches.

The first one is based on a model analysis tool called LIME [50]. This tool takes a comment which should be fed into the model and transforms it by creating new comments where random tokens are removed. The scores for these altered comments are then predicted by the model. Based on the differences to the full comment LIME calculates the contribution of each token to the scores for each label.

As a starting point, the first analysis focuses on which tokens the model sees as important when classifying a comment as toxic. An exemplary explanation of a prediction result can be found in figure 8.3. This is a comment from the Kaggle data set which is labeled as *toxic*, *obscene* and *toxic*. It can be seen in the figure that the trained model correctly predicts all of the labels, toxicity and obscenity even with a very high score. The LIME analysis shows which words are important for the classification of all labels. In the example shown in the image the most important tokens for all three labels are *suck* and *moron*. A human doing this classification task would see the exact same tokens as important for the results.

Preprocessing	F1	Unique tokens	Training duration per epoch
Baseline	0.6358 ±0.0085	326,175	680s
Reduce length	0.6357 ±0.0090	325,464	680s
Remove punctuation	0.6380 ±0.0090	268,928	665s
Remove punctuation weak	0.6386 ±0.0066	290,831	670s
Stemming	0.6430 ±0.0103	270,849	645s
Lemmatization	0.6448 ±0.0037	314,708	675s
Replace unknown tokens	**0.6451** ±0.0092	74,212	545s
Remove punctuation + stemming	0.6444 ±0.0078	223,050	620s
Remove punctuation + lemmatization	0.6449 ±0.0059	257,380	640s
Remove punctuation weak + stemming	0.6443 ±0.0054	245,122	630s
Remove punctuation weak + lemmatization	0.6443 ±0.0087	279,304	650s
Replace unknown tokens + lemmatization	0.6443 ±0.0081	64,418	540s

Table 8.2.: The first part of the table shows the results of all evaluated preprocessing techniques applied individually on the data set. Results in the second part of the table are obtained by combining preprocessing approaches. All the experiments done here use the single layer CNN with multiple windows sizes shown in 5.2.1. The results represent the micro-averaged F1 score over all labels in the Kaggle data set. All experiments are performed 5 times and mean and standard deviation of the F1 score can be seen in the table. The number of unique tokens is calculated by applying the preprocessing techniques and using the NLTK TweetTokenizer. Training time can be improved by reducing the number of tokens. The hardware and software setup which produces these training times can be found in appendix A.

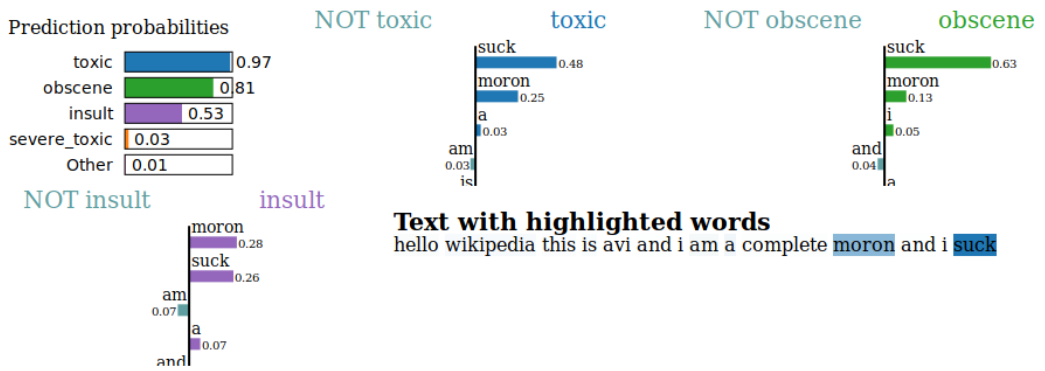
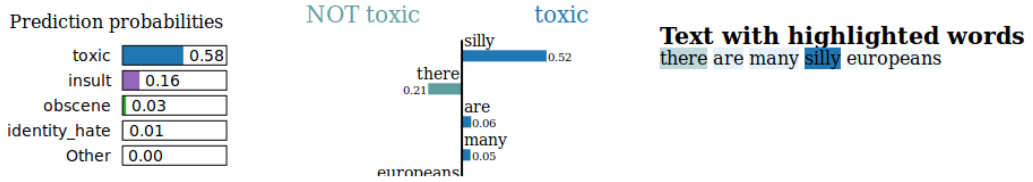
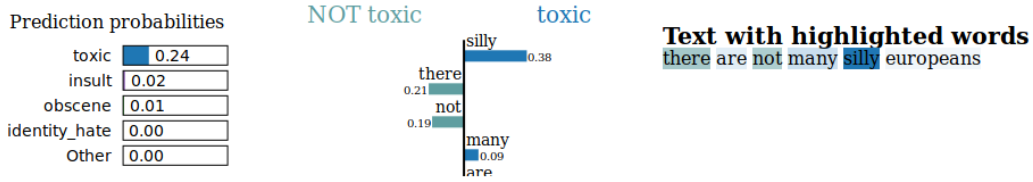


Figure 8.3.: In this figure the LIME explanations of the predictions for this toxic comment are shown. It can be seen that this comment receives very high scores for the labels *toxic* and *obscene*. Also the label *insult* gets a relatively high score assigned. The words *suck* and *moron* contribute most to the high scores. This matches the explanation a human would give for this comment.



(a) An artificial comment which is predicted as toxic.



(b) The same comment as in a), but with a *not* added to the sentence to make it non-toxic.

Figure 8.4.: These figures show the prediction difference between a toxic comment and its negated form. It can be seen that the added *not* token decreases the toxicity score from 0.58 to 0.24.

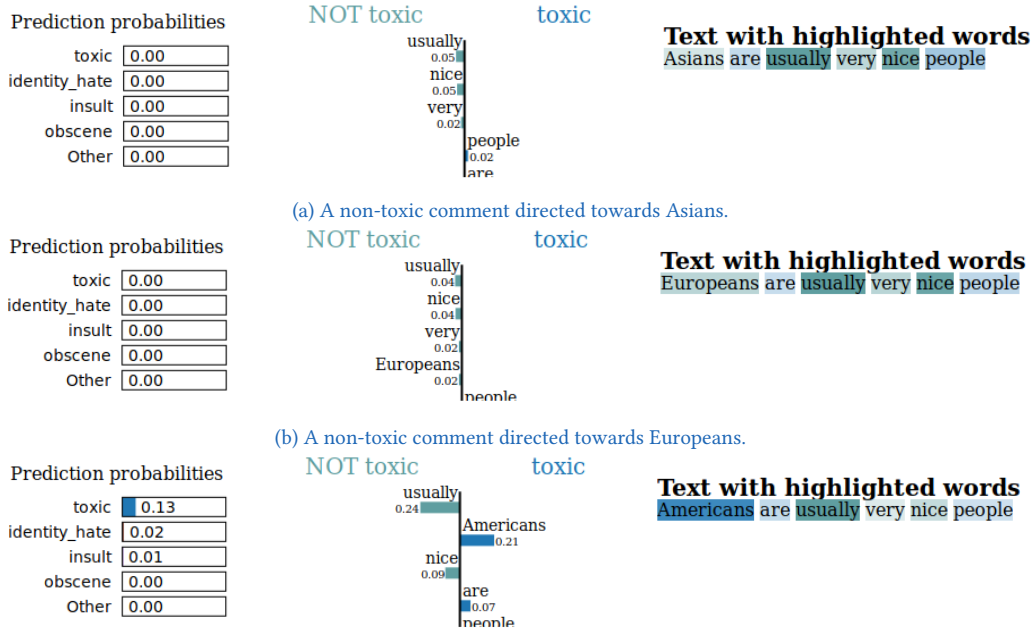
In chapter 2.1 it is stated that a major challenge in this and similar tasks is the proper treatment of negations in the text. For evaluating the behavior of this model when adding negations to the text an simple artificial comment which contains toxicity is created:

> there are many silly europeans

This comment is then transformed to a non-toxic comment by adding a *not* to the text. After evaluating these two comments with LIME it can be seen in figure 8.4 that the negated form is correctly labeled as non-toxic. The word importance shows a strong influence of the *not* token for classifying the comment as not toxic. This is, however, only one side of the story. In many other cases, the negation does influence the result, but not strong enough to change the predicted labels. This means the comment would have a lower toxicity score but is still seen as a toxic comment in the end.

Another important criteria for a good model is a bias-free prediction, especially when planning to use it in production systems. An example of such a bias would be a different prediction for two similar comments - one of the directed towards males, the other one towards females. To test the model for such a behavior three similar non-toxic sentences are created. The first one talking about *Asians*, the second one about *Europeans* and the last one about *Americans*. In an ideal model, these three comments would obtain the same toxicity score. In the model trained for this experiment, however, the predictions are different. Figure 8.5 shows the LIME explanations for these comments. The words *Asians* and *Europeans* are treated as slightly non-toxic. In contrast to that, the token *Americans* is treated as highly toxic and increases the overall toxicity score of the comment from 0 to 0.13. This shows that there is at least a bias based on the geographical region learned from the Kaggle data set.

The analysis method before is oriented towards the influence of single words to the overall classification scores. The following approach, on the other hand, focuses on the patterns which filters in the convolutional layer learn to detect. To get a better understanding when particular filters detect a pattern, the highest activation of each filter map for each comment in the test set is analyzed. The location of the highest activation is then mapped to a region in the comment to get the actual texts which are detected. Due to the size of the test set, only 10 comments with a very high activation for the particular filter are examined.



(a) A non-toxic comment directed towards Asians.

(b) A non-toxic comment directed towards Europeans.

(c) A non-toxic comment directed towards Americans.

Figure 8.5.: The figures show LIME explanation for three similar non-toxic comment directed towards Asians, Europeans and Americans. It can be seen in from the word highlightings that *Asians* and *Europeans* have a slightly non-toxic effect on the score. *Americans*, however, is a word with a strongly negative connotation by itself.

Example 1	Example 2	Example 3
. the author	, idiot ,	mentioned in the
. please deal	, immature idiot	are in the
on the article's	, renders an	joined in the
. the limits	, suicides grossly	groups in the
edited the introduction	a bitch "	name in the
. the links	is ridiculous .	weigh in if
. please use	is horrible .	article in the
. the verb	is a fact	rating in the
. please do	then * adam	specify in the
. please engage	, torein *	books in the

Table 8.3.: This table shows examples of the highest activating text regions of selected convolutional filters of size 3. Example 1 seems to detect the beginnings of new sentences, especially in combination with the tokens *the* and *please*. This seems like a detector for positive beginnings in sentences Example 2 focuses on content with a higher toxicity level, preferably after commas in the comments. The third example is a very strong detector on the pattern *in the*.

Three examples of size 3 filters can be found in table 8.3. When having a closer look at the first example one notices the presence of a period at the beginning of 8 out of 10 regions with high activations. This suggests that this filter detects the start of a new sentence in a comment. Additionally, there are no obscene or in any other way negatively connoted words in the detected regions. This observation combined with the presence of *please* in 4 regions makes it likely to be a filter for non-toxic comments. The regions in row 3 and 5, however, do not follow these patterns.

The second example in the table shows filter which is very likely a detector for toxic regions in a comment. It shows regions with high activations where insults or negativity is present in 6 out of 10 samples. It also seems to have even higher activations when toxicity occurs immediately after a comma. Like in the first exemplary filter there are again regions where the reason for activation is not immediately visible. The region "*, torein **" for example does not contain an actual English word but, nevertheless, is detected with a high activation.

Column 3 gives an example of an easily interpretable filter. 9 out of the 10 regions with the highest activation values end with the pattern *in the*. Therefore, this filter can be considered very robust in detecting this pattern.

All three examples share the property of being interpretable features. But in many other filters, the detected patterns are not easily interpretable for humans. They detect seemingly random combinations of tokens and make it difficult to assess their influence on the network. It can also be seen in the examples that there are sometimes outliers which do not match the main pattern of other detected regions by the same filter.

More examples of highly activated regions in filters can be found in appendix B.

8.4. Evaluation on YouToxic Data

So far the experiments have shown the performance of CNNs on data obtained from Wikipedia discussion pages. The question is, however, if they work for data coming from other sources similarly well. For showing the performance on online comments from another platform the singlelayer CNN with multiple filter sizes is evaluated on the YouToxic English data set. The details of the data set can be found in section 7.1.2.

The evaluation on this data set contains three different experiments. The first one utilizes the singlelayer CNN with multiple window sizes trained on the Kaggle data set. This model is then used to predict labels on the YouToxic comments. Such an experiment gives an intuition of how similar the data sets are in terms of patterns which are used for predicting the labels. The limitation of this experiment is, that the labels are not the same in the Kaggle and YouToxic sets. Therefore, only the matching labels are evaluated. In the second experiment, the same architecture is trained on the YouToxic data from scratch. The results of this experiment are intended to show if the network can be trained with such a small amount of data.

Due to the small number of comments in the YouToxic set a third experiment is performed. In this one, transfer learning is applied to reuse the knowledge learned from the Kaggle data set. The hypothesis is, that this approach improves the classification performance because the model utilizes the knowledge obtained from 160,000 comments in the other data set. For this experiment, the baseline model of the preprocessing experiments in section 8.2 is taken. The output layer is cropped and a new output layer which maps the labels in this data set is added on top of the network. This new output layer again applies sigmoid as its activation function and

Dataset	Label	F1 Score
YouToxic	Toxic	0.6039
YouToxic	Abusive	-
YouToxic	Hate speech	0.0000
Kaggle	Toxic	0.6655
Kaggle	Identity Hate	0.5523

Table 8.4.: This table shows the performance of the model trained on the Kaggle data set when being applied to YouToxic comments. The last two rows give the performance on the Kaggle data set for a better comparison. It can be seen that the performance on the toxic label is not much lower than on the original data set. The hate speech label, however, is not classified correctly for any given comment in the YouToxic data set. This can be seen from the 0.0 F1 score of this label. For the *abusive* label, there is no matching label in the Kaggle data set. Therefore, an evaluation is not possible.

the weights are initialized randomly. The remaining layers of the network use the trained weights of the model trained on the Kaggle data set as initial weights.

The results of the first experiment, where a model trained on Kaggle data is applied to the YouToxic set, can be found in table 8.4. The performance on the *toxic* label is in a similar region as on the Kaggle data set. For the *hate speech* label, the *identity hate* label in the Kaggle data set is identified as a matching label for comparison. From the table, however, it can be seen that the F1 score is exactly 0, which means that there were no comments classified correctly. For the label *abusive*, there is no matching label in the Kaggle data set. Therefore, an evaluation cannot be done.

The results of experiments 2 and 3 are presented in table 8.5. Training of the models is done for a maximum of 20 epochs with 5-fold cross validation for splitting the data into train and test set. Like all other experiments in this thesis, these ones are run 5 times and the mean and standard deviation of the F1 score is calculated.

In the left column of the table the results of the model trained from scratch are given. The overall F1 score of the model on the YouToxic data set is around 0.607 which is slightly lower than the same architecture evaluated on the Kaggle data set. However, when only looking at the performance on the *Toxic* label the results show a different picture. The Kaggle model results in an F1 score of around 0.666 whereas the trained model in this experiment obtains a score of 0.678. The performance on the label *Abusive* also shows a comparatively high F1 score. The third label

Dataset	Label	From Scratch	Transfer Learning
YouToxic	All	0.6068 ±0.0055	0.6118 ±0.0101
YouToxic	Toxic	0.6779 ±0.0084	0.6801 ±0.0110
YouToxic	Abusive	0.6231 ±0.0096	0.6273 ±0.0109
YouToxic	Hate speech	0.2175 ±0.0360	0.2367 ±0.0188
Kaggle	All	0.6358 ±0.0085	
Kaggle	Toxic	0.6655 ±0.0102	

Table 8.5.: This table shows the performance of the model trained on the YouToxic data set. The left results show the performance with of a model trained from scratch with randomly initialized weights. The right ones denote the performance when using transfer learning from a model trained on Kaggle data. All experiments are performed 5-times with 5-fold cross validation. For an easier comparison, the scores on the Kaggle data set are included in the table. The records with label *All* denote the micro-averaged F1 score of all labels in the particular data set. The results show that the model performs better on the Kaggle data set. When only looking at the toxicity label, the performance on the YouToxic data set is slightly higher. It can also be seen that the model using transfer learning has an edge over the model trained from scratch for each label.

in the YouToxic data set, *Hate speech*, on the other hand, does only achieve a F1 score of 0.218. This is also the reason for the lower overall score compared to the performance on the Kaggle data set.

The values on the right hand side in table 8.5 show the performance of the model using transfer learning. It can be seen from there that this model performs slightly better on each of the three labels and also achieves a higher micro-averaged result.

9. Discussion

In the previous chapter, the results of the experiments are shown. This chapter, however, is intended to discuss the findings with respect to the research questions which are stated in chapter 1.2.

The main research question is the question of the suitability of CNNs for classifying toxicity in online comments. For this four different architectures have been developed and evaluated on the Kaggle data set. The results of these experiments show that simpler models perform better on this task than deeper ones with more than one layer. This matches literature found on this topic by Conneau et al [9]. They try deep convolutional neural networks on text classification and found out that they work very well. However, they need larger data sets for training than the Kaggle one in this thesis.

An interesting finding of the experiments is the equal performance of both single-layer architectures. The simpler one only uses a single convolutional block with filter size 3 in the convolutional layer. In contrast, the second one uses three different blocks with filter sizes 3, 4 and 5. The hypothesis for developing the multi-size architectures was that this model is capable of detecting patterns spreading over a longer range of tokens. Results show, however, that their performance is very similar. This suggests that patterns important for classification do not spread over more than three tokens most of the times. This finding is also backed by the analysis of filters in the convolutional layer. There are many of them which only detect 1- or 2-grams in the comments.

When comparing the results of the models in this thesis to the results of the Kaggle competition, there are some findings which can be concluded. First of all, when looking at Kaggle models with a similar amount of preprocessing it can be seen that they deliver a similar performance to the CNNs in this work. Independently of whether they use LSTMs, GRUs or logistic regression with TF-IDF vectors, their performance lies within a narrow range. The winning results of the

Kaggle competition, on the other hand, show a higher performance compared to the mentioned models with little preprocessing applied. When looking at the models of the winners, it can be seen that they heavily rely on data augmentation, preprocessing, ensembling of classifiers and blending of models. This leads to the conclusion that the classifier itself is not the deciding factor for a well-working classification model.

The models in this thesis work well on 3 out of 6 labels in the Kaggle data set. On the remaining three labels the results show a very weak precision-recall curve in section 8.1. This is due to the much lower number of training comments available for these labels than for the other three. The label *thread* for example only contains 478 samples for training out of 160,000 samples in the whole data set.

All in all, the results of the experiments suggest that CNNs are indeed suitable for being used in toxicity classification. Their performance on raw data sets is similar to those of other classifiers. It must be ensured, however, that there is enough data for training available as the weak performances on the labels *severe_toxic*, *threat* and *identity_hate* show.

The first sub research question of this thesis is the influence of different preprocessing techniques to the classification performance. For this question multiple preprocessing strategies have been implemented and evaluated with respect to their performance, the resulting number of unique tokens in the data set and the training time.

In contrast to the experiments for the architecture comparison, the models for the preprocessing comparison do not use pretrained word embeddings. A finding when comparing the same architecture with and without pretrained embeddings gets visible when looking at the standard deviation of the F1 scores. The models without the pretrained weights produce a higher deviation in their results. This can be explained by the random initialization of the initial embeddings which have a high influence on the end result of the trained model. The models with pretrained embeddings start from the same initial embedding weights in every run of the experiment. The only randomly initialized weights in these models are the parameters in the higher layers.

The first and probably most surprising finding in the results of the preprocessing techniques is the performance when removing all punctuation and special characters from the data set. As stated in section 5.1.2 the hypothesis for this technique

was that it shrinks the model but also removes important information from the comments. This would, in the end, harm the performance of the model. The result of this technique is quite the contrary to the hypothesis, the classification performance even improved after removing all these characters. An explanation for this behavior can be found when having a closer look at the data set. It includes many samples with misplaced punctuation and also many comments completely without punctuation. It seems that they are not in all cases a reliable indicator for sentence structure. Another reason for this finding is the presence of many non-latin letters in the data set. These characters do not add any value to the classifier. Therefore, removing them might further improve the performance.

Another well-working preprocessing technique is replacing all unknown tokens with a special unknown token. For this pretrained word2vec embeddings are taken as a dictionary of known tokens and all tokens which are not found in there are replaced. This technique provides the best performance of all evaluated ones in this thesis. This might be explained by the removal of misspelled words or words in other languages which do not provide much semantic for the classifier. Replacing them by the special token still enables the classifier to detect that a comment uses non-standard language. In addition to providing the best performance, this technique also shrinks the model to the smallest size. This is especially important for systems where computational resources are limited. The number of parameters in the embedding layer to be trained is reduced by more than three quarters. This does not only reduce the overall size of the model but also reduces the time needed for training it. In the experiments, the training duration when using this technique is reduced by around 20% compared to the baseline.

An interesting observation can be found in the number of unique tokens when comparing stemming and lemmatization. Despite being two techniques with similar goals the outcome of these two is very different. Lemmatization reduces the number of unique tokens only by 3.5% whereas applying the Porter2 stemmer results in a reduction by 17%. This finding combined with the higher performance of the lemmatization approach leads to the conclusion that the choice between these two depends on the needs of the system. If the goal is a model as accurate as possible, the lemmatizer should be favored. If the model should, however, be applied to a system with limited computational power the stemmer leads to a smaller and faster model without sacrificing the performance too much.

The preprocessing experiments in this thesis are all performed individually. In a real

model, there are usually multiple of these approaches applied together. Even though techniques work well on their own, combinations of them might lead to a lower overall performance because important information gets lost in the processing. This thesis takes up on this point and evaluates combinations of the previously mentioned preprocessing techniques.

Experiments are done on the combination of removing punctuation in the strong and weak form with either stemming or lemmatization. The results show that all of these four approaches achieve a very similar F1 score which is similar to the individual performance of stemming and lemmatization. This suggests that removing punctuation does not add value in terms of a more accurate classification to stemming and lemmatization. However, the reduction of the number of unique tokens after applying one of these four approaches varies between 14% and 31%. It seems that these strategies combine the performance gain of stemming/lemmatization with the smaller model and faster training when removing punctuation. Therefore, even though the performance does not change, such a combination might be the preferred approach in some cases. Another evaluated combination is replacing all unknown tokens in the data set first and then lemmatizing the remaining tokens. Performance-wise, this approach performs very similar to the other combinations, but worse than only replacing all unknown tokens without lemmatization. When looking at the reduction of the number of unique tokens, this is the most rigorous approach. After applying it the number decreases by more than 80%. From the evaluation of combined preprocessing techniques it can be concluded that stemming and lemmatization improve the performance when being applied to the raw data set. But they also limit the possible performance when being combined with other approaches. This indicates that stemming and lemmatization remove important information from the comments.

The second subquestion in this thesis is focused on understanding the knowledge which is contained in a trained model. For this, such a model is analyzed in two ways. The first part consists of an examination for which regions in the text the filters in the convolutional layers have high activation values. And it also focuses on the question if the patterns detected by the filters can be interpreted by humans. The analysis shows that some of these filters are easily interpretable. Patterns in their highly activated text regions show for example the beginning of new sentences after a period or toxicity immediately after a comma. Most of the visible patterns do not seem to spread over more than 2 or 3 tokens in the text, even in the filters of size 5. As already mentioned earlier this might also be the main reason why the multi-

filter-size architecture does not give a better performance than the architecture with a single filter size.

In addition to those filters which are interpretable, there are many filters which learn to detect seemingly random token sequences. It is not clear from the analysis if these filters learn to detect hidden but important patterns or if they are completely useless for the classification. A similar question arises when looking at the outliers of filters with otherwise clearly visible patterns. Do those outlying regions share hidden patterns with the visible patterns detected by the filter or are these just a product of coincidentally similar embeddings for different tokens. This is also a question which can not be answered from the analysis done as part of this thesis.

The second part of the model analysis consists of using the model analysis tool LIME to create explanations for the classification of a comment. It can be seen from the explanation in the evaluation section that the model sees the same words as important for classification as a human would do. Such behavior makes the model easier interpretable for their developers.

This easier interpretability makes it also possible to find drawbacks in the model. The example in section 8.3 shows a comment where the negation is well perceived. This is not the case for all of the evaluated comments, though. In many comments, the negation is detected by the classifier and the score shifts in the correct direction. But the shift is not strong enough so that the classifier still predicts the same label as without the negation. For a well-working model, the importance of a negated toxicity needs to be strong enough to change the prediction.

Another important finding of this analysis is the incorporation of biased word embeddings in the model. In section 8.3 the evaluation shows a comparison between three comments. All of them have a similar non-toxic meaning, but all of them are directed to people from other geographic regions. Two of these regions are slightly positively connoted, whereas the third one introduces toxicity to the prediction. This is an issue which is important to be resolved before using such a model for real systems. A model for classifying toxicity must not introduce biases which are themselves toxic against various characteristics of people.

From this analysis, it can be seen that a closer examination of trained models is a necessary step before applying them elsewhere. Data sets do not always have a sufficient amount of samples for detecting the aforementioned issues. Therefore, a

manual evaluation for understanding parts of the model is as important as a good score on the test set.

The last subquestion of this thesis is oriented towards the applicability of CNNs to data from different sources. So far all the results discussed in this chapter focus on comments from Wikipedia talk pages. Section 8.4 closes this gap and evaluates the singlelayer architecture with multiple filter sizes on data gathered from YouTube comments.

In a first evaluation, a model trained on the Kaggle data set is applied for predicting the labels on the YouToxic data set. The results show that this approach works well for the *toxic* label. This suggests that there are similar patterns necessary for classifying toxicty in these two data sets. For the *hate speech* label, on the other hand, this approach does not work. There is not a single instance classified correctly. This behavior is expected, because the topics of the two data sets are very different and hate speech occurs in different forms based on the context. So the patterns learned for predicting *identity hate* in the Kaggle data set is not helpful for predicting *hate speech* on YouToxic comments.

The data set containing YouTube comments is very small compared to the Kaggle data set. For this reason, training a model on this data set is done in two different ways. The first experiment uses the same approach as in previous experiments - the weights are initialized randomly and then trained on the data set. To overcome the issue of the small data set the second trained model uses a transfer learning approach. In this method, the weights from a model trained on the Kaggle data set are taken as initial weights for the new model. This way, the knowledge of the Kaggle model can be utilized for the training on the YouToxic data set.

From the results, it can be seen that the overall results are in a similar region compared to the experiments on the Kaggle data, even with such a small data set. Moreover, the results show that the model using transfer learning performs better than the model trained from scrach on each label. This indicates that transfer learning gives an advantage for such small data sets, even though the difference is small in this experiment. It also suggests that toxicity shows a similar structure in comments coming from Wikipedia discussion pages and YouTube. This is also supported by the finding before, where the model trained on the Kaggle data set achieves good results on YouToxic comments.

In addition, there are two more observations when comparing the results on the Kaggle data with those on the YouToxic data. Firstly, when only looking at the label whether a comment is toxic or not the performance is higher on the YouToxic data set than on the Kaggle data set. This is especially surprising given the sizes of the two data sets. The Kaggle set contains 30 times more toxic comments than the YouToxic one. However, the high performance on YouTube comments might also be partially explained by their simpler structure. In section 7.1.2 there is a comparison between the two data sets. It can be seen that the YouToxic data shows a lower number of characters on average per character. Moreover, the punctuation rate is also lower compared to the data from Wikipedia discussion pages.

In contrast to the high results on toxic comments, the classifier does not show a strong performance on the *Hate speech* label. Hate speech comes in large variety of forms and shapes and, therefore, needs more data for training a strong classifier. Also, the transfer learning approach is not helpful for this task, because it improves the performance only from an F1 score of 0.21 to 0.23. This behavior is expected because of the different topics of the two data sets. In addition to the larger variety, the structure of hate speech often depends on the environment where the comment is posted. This suggests that the patterns learned for the *identity_hate* label in the Kaggle data set are not very helpful for classifying hate speech in YouTube comments.

The evaluation on the YouToxic data set shows that the same architecture does also work on comments from different online platforms. The results suggest, that CNNs also work for data sets where the number of samples is relatively small. According to the observations, it is also possible to transfer the knowledge learned from one data source to a new model for a second data source. This approach gives results which are a little higher than the ones with the model trained from scratch in the experiments. This is a good starting point for cases where there is not enough data available.

To conclude up the findings of this thesis, CNNs are indeed suitable for being trained on classifying toxicity in online comments. The experiments show that they perform similarly well as other classifiers on the raw Kaggle data sets. Moreover, a good performance is also achieved on the YouToxic data set which shows that the architectures work on comments from another source too. For improving the performance of a CNN on this task the data needs to be preprocessed in a beneficial way. In addition to the better classification performance, preprocessing is also able

to reduce the size of the model or improve the time needed for training. After training the model a careful evaluation needs to be done to prevent unwanted biases in the classifier. Experiments in this thesis show a bias on the geographic region of people.

10. Conclusion

In this chapter, the findings of this thesis are concluded and recommendations for working on this topic are given. Additionally, recommendations for further research in this area are presented.

In the related work for the toxicity classification task in section 3.1 it can be seen that most approaches use SVMs, a form of RNNs or even unsupervised approaches. This work shows that neural networks containing convolutional layers are also able to perform this task. The evaluation presents results which are on a similar level as other approaches on the data set of the Kaggle Toxic Comment Classification challenge. Moreover, singlelayer architectures have an edge over the deeper ones in this evaluation. To show that this performance is not a finding specific to comments from Wikipedia discussion pages as in the Kaggle data set, a second data set called *YouToxic English* has been created. Even on this small data set the CNN shows good results. Due to the size of the YouToxic data set transfer learning is applied to make use of the prior knowledge gained from the first data set. It can be seen from the results that CNNs obtain slightly higher scores than the model trained from scratch.

One of the reasons for the performance gap between the results in this thesis and the top results in the Kaggle competition is the extensive use of preprocessing. To get a better understanding which preprocessing techniques are useful for CNNs this thesis does an evaluation on some of these approaches. The observations show that some drastic methods show a significant performance gain compared to the baseline result. Additionally, these approaches shrink the size of the overall model by the highest rate and reduce the time needed for training. However, this thesis does only include a basic evaluation of how preprocessing approaches work in combination.

An important step before using a trained classifier in production is an analysis of the knowledge contained in the model. Such an evaluation limits the risk of

unpredictable behavior when being fed with certain data. This thesis presents approaches on how to do this analysis and shows that this knowledge is partially interpretable by humans when using the right approaches.

In section 2.1 it is stated that negations are a common challenge for classifiers in the NLP domain. Observations in this thesis reveal that a CNN is able to detect them. However, in many cases, their influence is not strong enough to turn the prediction towards the correct label. In addition to that, the analysis of the importance of tokens for the prediction shows another issue. Some words which are supposed to be neutral do have a bias towards toxicity or non-toxicity. An example in this work is a different level of toxicity based on the geographic region of a person. Such a bias is to be absolutely avoided for such a sensitive topic.

To sum up this thesis, the use of CNNs is indeed suitable for classifying toxicity in online comments because of a similar performance compared to other classifiers. Simpler architectures should be preferred as a first step. For achieving a state-of-the-art performance preprocessing is inevitable. This thesis suggests that preprocessing techniques leading to a simpler data set both increase performance and reduce training time and model size up to a certain degree. After successfully training a model an in-depth analysis of the contained knowledge should be done. This is important for detecting unwanted prediction behavior and biases in the model.

After the evaluation of CNNs for toxicity classification, there are three recommendations for people who want to work on a similar topic.

1. **Start simple!** Simple architectures are already able to perform well on some NLP tasks like toxicity classification. Add complexity only if needed.
2. **Know your goals!** Different preprocessing techniques have different effects on the model, like an increased performance, a smaller size or faster training. Define your goals early enough to know which techniques are worth being applied.
3. **Analyze your model!** After training a model on this or similar tasks, analyzing the knowledge is absolutely necessary. There is always the danger of an unwanted bias when training from user generated texts.

10.1. Further Work

This thesis does an evaluation of the capabilities of CNNs for classifying toxicity. However, there are still topics not covered by this work. Some recommendations for further research are given in this section.

Previously in this chapter, it is stated that this thesis only includes basic experiments on combinations of preprocessing techniques. In practice, there is hardly ever only a single technique involved. Future work might take up this point and do a more extensive analysis of combinations of the presented approaches.

In the evaluation chapter, this work shows patterns which a trained model learns to detect. While this leads to a better understanding of the model, the analysis does not give information about the influence of these pattern to the prediction. Moreover, there is also the open question about the effect of patterns which are not interpretable by humans. Do these patterns contribute to the prediction in a meaningful way or will these patterns be ignored in later layers of the model?

The evaluation of the importance of tokens for the classification shows a significant bias of supposedly neutral words. Therefore, a possible extension to this work would be research about model debiasing for CNNs applied to toxicity classification.

Last, but not least, a possible direction is the evaluation of the architectures and preprocessing techniques in this thesis on comments in other languages. For creating a YouToxic data set in other languages, the tagging tool presented in chapter 6 can be used to simplify this task.

A combination of this thesis and the possible future work presented in this section could give end-to-end recommendations for applying a CNN to classifying toxicity in online comments.

Bibliography

[1] J. Angwin and H. Grassegger. Facebook's secret censorship rules protect white men from hate speech but not black children. https://www.propublica.org/article/facebook-hate-speech-censorship-internal-documents-algorithms, 2017. Accessed: 2018-11-07.

[2] J. Angwin, M. Varner, and M. Tobin. Facebook enabled advertisers to reach 'jew haters'. https://www.propublica.org/article/facebook-enabled-advertisers-to-reach-jew-haters, 2017. Accessed: 2018-11-07.

[3] S. Bai, J. Z. Kolter, and V. Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[4] S. Bhattarai. Exploring deep learning in combating internet toxicity. 2018. Accessed: 2019-01-15.

[5] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[6] C. D. Brown and H. T. Davis. Receiver operating characteristics curves and related decision measures: A tutorial. *Chemometrics and Intelligent Laboratory Systems*, 80(1):24–38, 2006.

[7] Cambridge. Hate speech. https://dictionary.cambridge.org/dictionary/english/hate-speech. Accessed: 2018-10-10.

[8] Y. Chen, Y. Zhou, S. Zhu, and H. Xu. Detecting offensive language in social media to protect adolescent online safety. In *Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Confernece on Social Computing (SocialCom)*, pages 71–80. IEEE, 2012.

[9] A. Conneau, H. Schwenk, L. Barrault, and Y. Lecun. Very deep convolutional networks for text classification. *arXiv preprint arXiv:1606.01781*, 2016.

[10] S. Cook. Cyberbullying around the world – which country has the most victims? https://www.comparitech.com/internet-providers/cyberbullying-statistics/, 2018. Accessed: 2018-11-07.

[11] F. Del Vigna, A. Cimino, F. Dell'Orletta, M. Petrocchi, and M. Tesconi. Hate me, hate me not: Hate speech detection on facebook. 2017.

[12] Ditch the Label. The annual bullying survey 2017. https://www.ditchthelabel.org/research-papers/the-annual-bullying-survey-2017/, 2017. Accessed: 2018-08-19.

[13] L. Dixon, J. Li, J. Sorensen, N. Thain, and L. Vasserman. Measuring and mitigating unintended bias in text classification. In *available at: www. aies-conference. com/wp-content/papers/main/AIES_2018_paper_9. pdf (accessed 6 August 2018).[Google Scholar]*, 2018.

[14] C. dos Santos and M. Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 69–78, 2014.

[15] Facebook. Community standards enforcement preliminary report. https://transparency.facebook.com/community-standards-enforcement, 2018. Accessed: 2018-11-07.

[16] R. Fei-Fei. L. and fergus and p. perona. learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. In *CVPR Workshop on Generative-Model Based Vision*, 2004.

[17] S. V. Georgakopoulos, S. K. Tasoulis, A. G. Vrahatis, and V. P. Plagianakos. Convolutional neural networks for toxic comment classification. *arXiv preprint arXiv:1802.09957*, 2018.

[18] N. Ghamrawi and A. McCallum. Collective multi-label classification. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 195–200. ACM, 2005.

[19] C. H. E. Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth International Conference on Weblogs and Social Media (ICWSM-14). Available at (20/04/16) http://comp. social. gatech. edu/papers/icwsm14. vader. hutto. pdf*, 2014.

[20] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[21] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. 2007.

[22] B. Habert, G. Adda, M. Adda-Decker, P. B. de Marëuil, S. Ferrari, O. Ferret, G. Illouz, and P. Paroubek. Towards tokenization evaluation. In *Proceedings of LREC*, volume 98, pages 427–431, 1998.

[23] Z. S. Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[24] H. He and E. A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, Sept 2009.

[25] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[26] I. Hendrickx, S. N. Kim, Z. Kozareva, P. Nakov, D. Ó Séaghdha, S. Padó, M. Pennacchiotti, L. Romano, and S. Szpakowicz. Semeval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals. In *Proceedings of the Workshop on Semantic Evaluations: Recent Achievements and Future Directions*, pages 94–99. Association for Computational Linguistics, 2009.

[27] H. Hosseini, S. Kannan, B. Zhang, and R. Poovendran. Deceiving google's perspective api built for detecting toxic comments. *arXiv preprint arXiv:1702.08138*, 2017.

[28] Jacobellis v. Ohio. *378 U.S. at 197.* 1964.

[29] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153, Sept 2009.

[30] Jigsaw. Perspective api. https://www.perspectiveapi.com/. Accessed: 2018-10-05.

[31] R. Johnson and T. Zhang. Convolutional neural networks for text categorization: Shallow word-level vs. deep character-level. *arXiv preprint arXiv:1609.00718*, 2016.

[32] Y. Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[33] Y. Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751. Association for Computational Linguistics, 2014.

[34] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[36] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[37] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[38] S. Li. *Application of Recurrent Neural Networks In Toxic Comment Classification.* PhD thesis, UCLA, 2018.

[39] N. Lomas. Germanys social media hate speech law is now in effect. https://techcrunch.com/2017/10/02/germanys-social-media-hate-speech-law-is-now-in-effect, 2017. Accessed: 2018-08-03.

[40] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[41] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[42] T. Mullen and N. Collier. Sentiment analysis using support vector machines with diverse information sources. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, 2004.

[43] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[44] T. H. Nguyen and R. Grishman. Relation extraction: Perspective from convolutional neural networks. In *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*, pages 39–48, 2015.

[45] M. A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[46] C. Nobata, J. Tetreault, A. Thomas, Y. Mehdad, and Y. Chang. Abusive language detection in online user content. In *Proceedings of the 25th international conference on world wide web*, pages 145–153. International World Wide Web Conferences Steering Committee, 2016.

[47] S. J. Pan, Q. Yang, et al. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[48] R. Pandarachalil, S. Sendhilkumar, and G. S. Mahalakshmi. Twitter sentiment analysis for large-scale data: An unsupervised approach. *Cognitive Computation*, 7(2):254–262, Apr 2015.

[49] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[50] M. T. Ribeiro, S. Singh, and C. Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144, 2016.

[51] B. Ross, M. Rist, G. Carbonell, B. Cabrera, N. Kurowsky, and M. Wojatzki. Measuring the reliability of hate speech annotations: The case of the european refugee crisis. *arXiv preprint arXiv:1701.08118*, 2017.

[52] C. Silva and B. Ribeiro. The importance of stop word removal on recall values in text categorization. In *Proceedings of the International Joint Conference on Neural Networks, 2003.*, volume 3, pages 1661–1666 vol.3, July 2003.

[53] S. O. Sood, J. Antin, and E. F. Churchill. Using crowdsourcing to improve profanity detection. In *AAAI Spring Symposium: Wisdom of the Crowd*, volume 12, page 06, 2012.

[54] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014.

[55] D. Svozil, V. Kvasnicka, and J. Pospichal. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, 39(1):43–62, 1997.

[56] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[57] N. Thain, L. Dixon, and E. Wulczyn. Wikipedia Talk Labels: Toxicity. 10.6084/m9.figshare.4563973.v2, 2017. Accessed: 2018-11-15.

[58] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler. Efficient object localization using convolutional networks. *CoRR*, abs/1411.4280, 2014.

[59] P. D. Turney. Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 417–424. Association for Computational Linguistics, 2002.

[60] W. Warner and J. Hirschberg. Detecting hate speech on the world wide web. In *Proceedings of the Second Workshop on Language in Social Media*, pages 19–26. Association for Computational Linguistics, 2012.

[61] Z. Waseem. Are you a racist or am i seeing things? annotator influence on hate speech detection on twitter. In *Proceedings of the first workshop on NLP and computational social science*, pages 138–142, 2016.

[62] Z. Waseem, T. Davidson, D. Warmsley, and I. Weber. Understanding abuse: a typology of abusive language detection subtasks. *arXiv preprint arXiv:1705.09899*, 2017.

[63] E. Wulczyn, N. Thain, and L. Dixon. Wikipedia detox. doi:10.6084/m9.figshare.4054689, 2016. Accessed: 2018-11-15.

[64] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[65] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[66] D. Zeng, K. Liu, Y. Chen, and J. Zhao. Distant supervision for relation extraction via piecewise convolutional neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1753–1762, 2015.

[67] D. Zeng, K. Liu, S. Lai, G. Zhou, and J. Zhao. Relation classification via convolutional deep neural network. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 2335–2344. Dublin City University and Association for Computational Linguistics, 2014.

[68] X. Zhang, J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.

[69] C. Zhou, C. Sun, Z. Liu, and F. Lau. A c-lstm neural network for text classification. *arXiv preprint arXiv:1511.08630*, 2015.

Appendix

Appendix A.

Experimental Environment

Information about the hardware used for training and evaluating the models in this thesis:

- CPU: Intel i7-7700
- GPU: Nvidia GeForce GTX1070
- RAM: 16GB DDR4

All experiments are implemented using the following software packages with their corresponding versions:

- CPU: Python 3.6.1
- Keras 2.1.3
- Tensorflow-gpu 1.5.0
- NumPy 1.15.4
- NLTK 3.2.4
- CUDA 9.0.176

Appendix B.

Additional Filter Activation Analysis

Appendix B. Additional Filter Activation Analysis

Example 1	Example 2	Example 3
thanks for experimenting	of a political	state colleges -
stands for the	of femininity you	” dr .
thanks for blocking	burn the cell	not necessarily correct
thanks for the	of vandalism ,	are often supplied
thanks for giving	of lumbini buddha	samuel beckett infobox
. for example	of the oldest	establish both points
solution for this	carries a political	sexes often strive
you for understanding	of this article	publishing various segments
account for yourself	blow a 3	in 1994 -
reason for banning	follow the basic	‘ ’ oxford

Table B.1.: These are three examples of highly activated comment regions of filters with size 3. The first one detects the word *for*, often in combination with *thanks*. The second one detects the word *of*. Example 3, however, detects token sequences which seem random to humans.

Example 1	Example 2
gay as hell ! he	paulus , reached the volga
burn you to hell if	uses . perhaps the former
, idiot , jerk ,	truce was reached in minsk
death is part of the	. the ‘ ’ oxford
a bitch . :: :	its own format , with
burn in hell .	giving the reader the possibility
burn the cell contents ,	the second defines the social
+ torein , torein *	. we voted to keep
jews not testing as jews	these studies show the endothermic
, suicides grossly misrepresents their	the author describes the unverifiable

Table B.2.: These are two examples of highly activated comment regions of filters with size 5. The first seems like a detector for toxicity. Example 2 looks like it detects rather random, but neutral, phrases.