spMMMP at GermEval 2018 Shared Task: Classification of Offensive Content in Tweets using Convolutional Neural Networks and Gated Recurrent Units

Dirk von Grünigen*
Fernando Benites
Pius von Däniken
Mark Cieliebak
Zurich University of Applied Sciences (ZHAW)
CH-8400 Winterthur
dirk@vongruenigen.com
{benf,vode,ciel}@zhaw.ch

Ralf Grubenmann*
SpinningBytes AG
Albanistrasse 20
CH-8400 Winterthur
rg@spinningbytes.com

Abstract

In this paper, we propose two different systems for classifying offensive language in micro-blog messages from Twitter ("tweet"). The first system uses an ensemble of convolutional neural networks (CNN), whose outputs are then fed to a meta-classifier for the final prediction. The second system uses a combination of a CNN and a gated recurrent unit (GRU) together with a transfer-learning approach based on pretraining with a large, automatically translated dataset.

1 Introduction

Sentiment Analysis was a major focus for text analytics in the last few years. Recently it became clear that only differentiating between positive and negative opinions is insufficient for some practical applications. Nowadays many website maintainers are requested to remove offensive content and monitor the discussions on their websites and social networks. This creates an overwhelming need for automated classification and removal of posts which could cause legal issues.

Although there are resources and research on some languages, e.g. English (Davidson et al., 2017; Waseem and Hovy, 2016), most languages have little or no resources on the matter. The GermEval Shared Task 2018 aims to tackle the problem of offensive language within micro-blog posts from Twitter ("tweets") written in German.

In this report, we propose two classifiers for identifying offensive content in tweets. Our experiments show that using embeddings created from large amounts of unsupervised in-domain data has a beneficial impact on the results. We rely on state-of-the-art convolutional neural networks (CNNs) and ensemble strategies, which have shown to achieve competitive results on sentiment analysis (e.g. Deriu et al. (2016)).

2 Task Description

The organizers of the shared task provided a dataset with 5009 samples. Each sample contains a tweet and two types of labels, one for each sub-task: The first label is for the binary-classification task ("Task I") and hence only distinguishes between *offensive* and *non-offensive* content. The second label discriminates between four different classes, of which 3 are different types of offensive content: *abuse*, *insult* and *profanity* and the fourth label for *non-offensive*. The second subtask is very unbalanced, with the labels distributed as: 3321 non-offensive, 1022 abusive, 595 insult and 71 profanity.

3 System Descriptions

In the following two sections, we describe our two proposed systems. System I is built on an ensemble of convolutional neural networks (CNN) whose outputs are consumed by a meta-classifier for the final prediction. This system is optimized to work as a classifier for the binary classification task ("Task I"). System II is based on the CNN+GRU architecture proposed by Zhang and Luo (2018). An important component of both systems is the use of diversified and enriched word embeddings to grasp the semantic context of the words. Both approaches are cutting-edge for specific but related text classification tasks and are therefore well suited to the problem domain, although they have not been di-

*Equal Contribution

Proceedings of GermEval 2018, 14th Conference on Natural Language Processing (KONVENS 2018)
Vienna, Austria – September 21, 2018

rectly compared to date.

4 System I

Deep learning models based on convolutional neural networks (CNN) are state-of-the-art for a number of text classification tasks, in particular in sentiment analysis (Kim, 2014; Kalchbrenner et al., 2014; Severyn and Moschitti, 2015a; Severyn and Moschitti, 2015b; Johnson and Zang, 2015), which is closely related to the domain of detecting offensive content in text. The system proposed by Mahata et al. (2018) has proven to perform exceptionally well in the domain of classifying medication intake from tweets. Based on this, we also trained multiple shallow CNNs and combine them into an ensemble in a similar fashion.

4.1 Preprocessing

The data is processed by lowercasing the tweet and normalizing numbers and removing "|LBR|" tokens, which signify a newline in a tweet. Depending on the embeddings used further down the process, as detailed in Section 4.3, we used different tokenization strategies. For vanilla word2vec and fastText embeddings, we used the NLTK `TweetTokenizer` (Bird et al., 2009). On the other hand, for the subword byte-pair embeddings (Sennrich et al., 2016), we used the Google `sentencepiece`[1] tool.

As the last step, we applied the hashtag splitting procedure described below to split up hashtags into their distinctive parts, since hashtags can convey a lot of the intention of a tweet. Finally, we converted the tokenized tweets into a list of indices, which was used to select the corresponding word embeddings. Furthermore, we enriched the word-embeddings with word-based polarity values.

Word Polarity Values: In offensive texts in tweets, often very polarising words are used (e.g. racial slurs or insults). To take advantage of this fact, we incorporated polarity values for each word in the used dataset. For that purpose, we employed three different resources: A multi-domain sentiment lexicon for German from the IGGSA website[2], the list of insults in German from the website hyperhero.com[3] and a list of racial slurs in German from the website hatebase.org[4]. The polarity values in the lexicon range from -1.0 (negatively polarising) to +1.0 (positively polarising). The average of all polarity values provided for each word in the lexicon provided to the system an additional feature. This sentiment lexicon was extended with the words from the list of German insults from the website hyperhero.com and from the list of racial slurs from hatebase.org to it. Further, we assigned a negative polarity (i.e. -1.0) value to these additional words. We then generated a one-hot encoded vector with 11 polarity-classes for each word in the dataset by discretizing the continuous polarity values. These vectors were stacked on top of each of the word embedding vectors before being passed to the convolutional network.

Hashtag Splitting: Hashtags are problematic in tweets, since sometimes they are composed of multiple words (e.g. "#ThisIsASingleHashtag") and hence would be out-of-vocabulary for the word embeddings most of the time. But they are crucial to understand the real meaning behind a tweet: For example the meaning of a tweet with the hashtag "#sarcasm" might be understood completely different without adding this hashtag. To tackle this problem, we implemented a hashtag splitting procedure using the CharCompound[5] tool (Tuggener, 2016). It is a simple but elegant solution, which uses ngram probabilities and returns different splits for each word with a certainty value for each split. We applied the splitting procedure recursively to the hashtags to ensure that we split all compounds. We set the certainty threshold to 0.8 and stopped when no split with a certainty greater or equal to this threshold could be found.

4.2 Base CNN

The base CNN for the ensemble consists of multiple, shallow convolutional layers. Each convolutional layer consists of the following components, in the listed order:

- Word embeddings layer that converts an indices-vector into a sentence-matrix.
- Dropout layer (Srivastava et al., 2014) as a regularization measure.
- Convolution operation for the feature extraction.
- Batch normalization layer (Ioffe and Szegedy, 2015) to speed up the training.

[1] https://github.com/google/sentencepiece
[2] https://sites.google.com/site/iggsahome/downloads
[3] http://hyperhero.com/de/insults.htm
[4] https://www.hatebase.org/search_results
[5] https://github.com/dtuggener/CharSplit

Hyperparameter	*Value*
Number of Conv. Kernel	200
Conv. Kernel Sizes	[2, 3, 4, 5, 6]
Conv. Kernel Stride	1
Conv Kernel Dilation	0
Number of Neurons in Hidden Layer	4096
Dropout Probability (after word-embeddings layer)	0.4
Dropout probability (after conv. operation)	0.3
Dropout probability (between fully-connected layers)	0.4
Max. Input Length	200

Table 1: Hyperparameters used for the base CNN in System I. Only one kernel size was used per convolutional operation, but we used 5 convolutional layers, each using one of the sizes for its kernels.

- Another dropout layer.
- Max-pooling layer to reduce the dimensionality of the output.
- ReLU activation function (Nair and Hinton, 2010) to squeeze the output values into the range $[0, +\infty)$.

In total there are five of these layers, all using the same hyperparameters (see Table 1), except for the kernel size in the convolution operation. The sentence-matrix is fed to each of these parallel convolutional layers and the resulting output vectors are concatenated, resulting in a vector with 1000 values. This vector is then forward propagated through two fully connected layers, which then output two logit values for the two classes (i.e. ”not offensive” and ”offensive”). A visualization of the base CNN model is depicted in Figure 1.

Hyperparameters: The hyperparameters used in the base CNN of System I can be seen in table 1. The max-pooling operation was performed as global max-pooling. This implies that each of the convolution operations outputs 200 distinct values, because we configured each convolution operation to use 200 different kernels. As a result of using 5 different convolutional layers having 200 output values each, the vector, which is forwarded to the fully-connect layer, contains 1000 values.

Initialization and Optimization of Parameters: All parameters, except for the biases, of the base CNN were initialized using the `Xavier Normal` initialization (Glorot and Bengio, 2010) with the gain value set to 1. The biases were initialized to 0. We used the `Adam` optimizer (Kingma and Ba, 2014) for the optimization of the network parameters, including the word embeddings. `Adam` dynamically adapts the learning rate for every parameter in the network by using first- and second-order information. We used a learning rate of 0.001, 0.9 and 0.999 as the beta coefficients for computing the running averages of the gradients, a weight decay value of 0.0005 and an epsilon value of 10^{-8}. As the loss function, we employed the cross-entropy loss between the expected, one-hot encoded label vector and the output of the CNN after being passed through a Softmax layer.

4.3 Word Embeddings

Word embeddings are omnipresent today when performing any natural language processing, especially with deep learning models. Due to our approach of using several of the previously described base CNNs, we decided that we would initialize each of these with another kind of word embeddings. We use different kind of word embeddings to get an diversified view of the data, which helps with our ensembling approach.

The following types of word embeddings were used:

- `fastText (SpinningBytes-FT)` embeddings (Bojanowski et al., 2017; Joulin et al., 2017) with 300 dimensions trained on a large corpus of German tweets (”sb-tweets”) provided by SpinningBytes[6]. These are currently not publicly available.
- `fastText (fasttext-Wiki)` embeddings with 200 dimensions pretrained on the texts from the German Wikipedia corpus. These can be downloaded via the fastText GitHub page[7].
- `word2vec (SpinningBytes-W2V)` (Mikolov et al., 2013) embeddings with 200 dimensions, also trained with the ”sb-tweets” corpus. These can also be downloaded from the SpinningBytes website.
- `fastText Byte-Pair Embeddings (Spinningbytes-BP)` embeddings with 100 dimensions for the case where subword tokenization (Sennrich et al., 2016) was performed, trained with the ”sb-tweets” corpus. For the tokenization, we used the previously mentioned Google `sentencepiece` tool. These embeddings are not publicly available at the moment.

[6] http://spinningbytes.com
[7] https://github.com/facebookresearch/fastText/

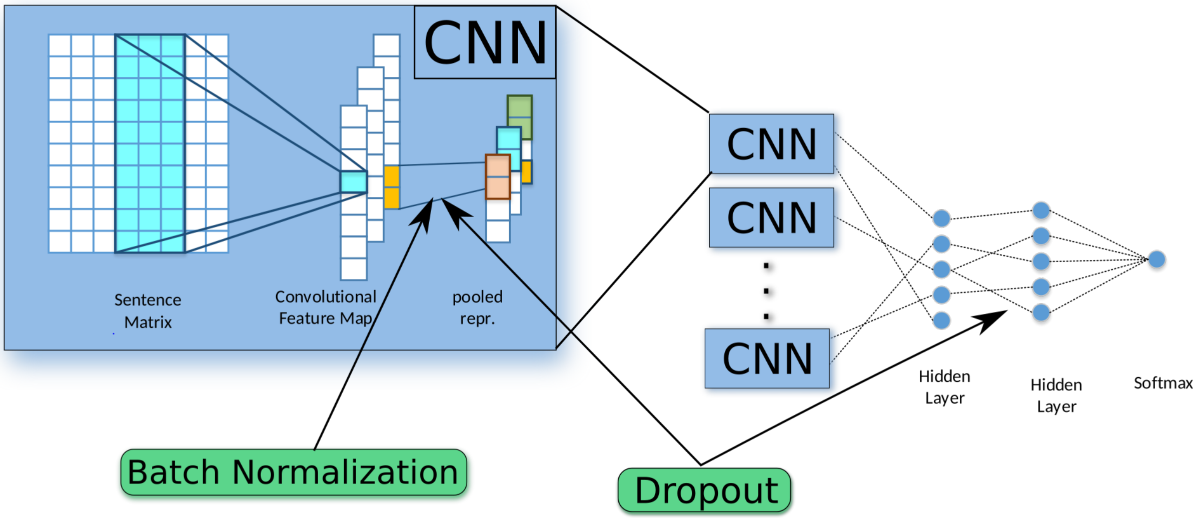

Figure 1: Visualization on the structure of the base CNN model.

4.4 Training Procedure and Ensembling of Classifiers

We decided to train our models in a similar fashion as Mahata et al. (2018): First, we split the data provided by the organizers randomly into a training and holdout dataset, where the training dataset contains 90% of the provided data and the other 10% is used as for the holdout dataset. We train each of the different models by doing k-fold cross-validation (with $k = 5$) over said training data and use the evaluation dataset for performing early stopping if the performance on it did not improve for more than 20 epochs with respect to the macro F1-score. Each of the models trained on each fold is then stored for later usage in the ensemble. This results in 20 base CNNs in total, 5 for each of the 4 different CNNs initialized with the word embeddings listed in Section 4.3.

Class Weights: Only 33.7% of the samples in the provided data contain offensive content, whereas 66.3% do not. We used class weights to counter this imbalance in the label distribution. For this we computed class weights, which are then used to rescale the loss function when performing the back-propagation. The following formulae were employed:

$$C_O = \frac{|L_N| + |L_O|}{2 \cdot |L_O|} \quad (1)$$

$$C_N = \frac{|L_N| + |L_O|}{2 \cdot |L_N|} \quad (2)$$

where $|L_O|$ is the number of offensive samples, $|L_N|$ the number of samples with non-offensive content in the provided dataset. C_O and C_N are the resulting class weights for offensive and non-offensive samples respectively.

4.5 Meta Classifiers

As described before, we trained the same base CNN with different word embeddings on different parts of the training data using k-fold cross-validation. Moreover, we concatenated the outputs of these 20 models on the training dataset and used them in conjunction with the labels to train different meta-classifiers. We experimented with different strategies for meta-classification (see Table 3 in Section 6) and used hyper-parameter optimization while training them.

5 System II

Following Zhang and Luo (2018), our second architecture utilizes both CNN and Gated Recurrent Units (GRU, Cho et al. (2014)). It uses three different embeddings and an attention layer, which are described in detail in the following.

5.1 Preprocessing

Additionally to the preprocessing of System I, user mentions (*@username*) were removed, words containing dots were split and special characters / |: ; & \ were removed. German stopwords[8] were also removed from the input string. Words not present in the embeddings were replaced with an **UNK** token.

[8] https://github.com/stopwords-iso/stopwords-de

5.2 CNN + GRU

The model consists of two CNN+GRU architectures, one for word-embeddings and one for subword embeddings, which are later concatenated together, along with a Smiley-feature vector, before being used by a fully connected Softmax layer to get predictions of the model. To prevent overfitting, dropout of 0.5 was added before every convolutional as well as the final layer. ReLU was used as activation function for all convolutional layers. An overview of the architecture is shown in Figure 2.

Word embeddings architecture: fastText embeddings of 200 dimensions each for uni- and bigrams in a tweet are concatenated to get a 100x400 feature matrix. Tweets are limited 100 tokens. 1d convolutions with 100 feature maps and kernel sizes of 3, 4 and 5, and kernel sizes 2 and 3 with dilations of 2 and 3, respectively, are then applied to the feature matrix separately. The dilated convolutions are meant to simulate the *skipped CNN* proposed in (Zhang and Luo, 2018). The results are max-pooled by a factor of 4 and concatenated along the feature axis. This is then passed to a bi-directional GRU unit. The hidden states at each time step of the GRU are then combined by an attention layer (Xu et al., 2014), yielding a feature vector containing 1000 values.

Subword embeddings architecture: This architecture largely mirrors the word embeddings architecture, but takes subword tokenized embeddings as input. Due to the smaller nature of subword tokens, a maximum sentence length of 150 is enforced. The architecture is adjusted to yield the same 1000 dimensional feature vector as in the word-embeddings architecture.

Emoji embeddings: A list of 751 Unicode emojis (Kralj et al., 2015) is used to count the occurrences of different emojis in the tweets. A linear transformation is applied to the emoji feature vector to reduce dimensionality to 200.

Final layer: The output of all three parts of the architecture is concatenated to yield a 2200 dimensional feature vector. A fully connected layer with Softmax is used to get the final output of the architecture, with 2 and 4 dimensions for the coarse and fine tasks, respectively.

5.3 Transfer Learning

Due to the relatively small amount of training data, the model was pretrained on a related task. To our knowledge, only one other hate speech corpus in German is available (Ross et al., 2016). But there are two large corpora for hate speech detection available in English, namely (Davidson et al., 2017) and one provided by Lukovnikov[9]. To get as close as possible to the target domain, the English hate speech corpora were automatically translated[10] to German. The model was jointly trained on the related German and English corpora until train scores stopped improving. Then the last layer of the network was discarded and retrained on the actual data provided for the Shared Task.

5.4 Semi-Supervised Retraining using the Test Dataset

To extend the training set, we used a similar semi-supervised approach to Jauhiainen (2018). For that purpose, our system is first trained on the training dataset and then used to classify the test dataset. Predictions on samples of the unlabeled test dataset with a confidence higher than 0.75 are then used as additional labeled data to augment the training set. We treat the output of the Softmax layer as the confidence score. The classifier is then trained again on the augmented training dataset. The results can be seen in Tables 2 and 3 for the systems labeled with *Semi*.

6 Experiments

We performed several tests on the labeled training data. As described above, we randomly selected 10% of the training data as test data. We then trained on the training data and evaluated the systems on the test data. This procedure was repeated five times in order to estimate an average and standard deviation of the performance.

We compared our results to a baseline which consisted of an SVM using TF-IDF feature weighting. The data preprocessing was performed by tokenizing the tweets with the mentioned `TweetTokenizer` and the `GermanStemmer` from the `stem.snowball` module of `NLTK`. We also compared the single classifiers of System I versus the results using different meta classifiers. We evaluated the results with the F-1 macro average measure. The results are depicted in Table 3.

In task I, the meta classifiers had a remarkable impact. Logit Averaging provided an advantage over the other approaches and improved the overall classification performance by more than 3 points

[9]https://github.com/lukovnikov/hatespeech

[10]https://translate.google.com/

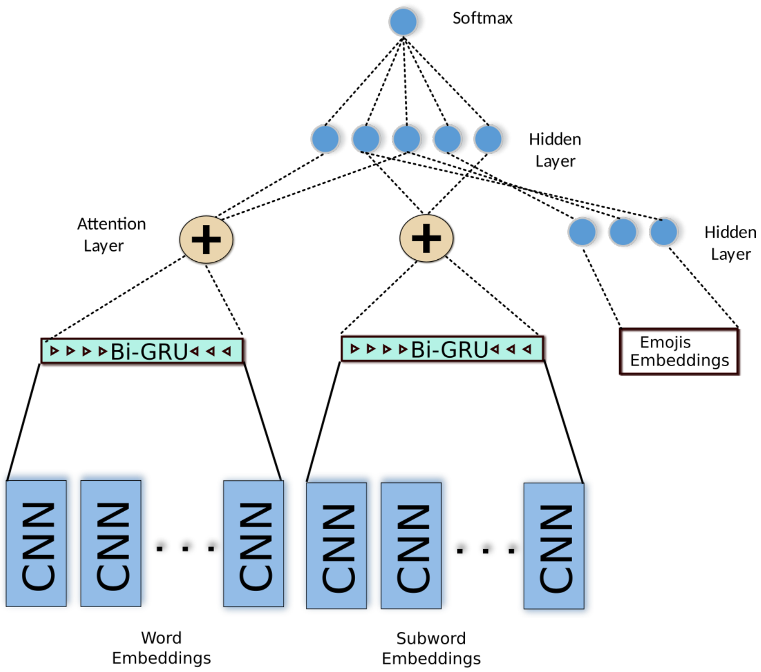

Figure 2: Visualization on the structure of the CNN + GRU model.

with respect to the F-1 macro score in comparison to the best performing single classifier (see Table 3). This confirms the results of Mahata et al. (2018). Other meta-classifiers, such as Random Forests, Logistic Regression and Linear SVM were close, though the single classifiers were also in this range. The System I results showed that the embeddings can have a decisive impact on the results of the classification systems. These systems had a big margin to the Multilayer Perceptron meta classifier, which performed last in the results and also has the largest variance in the performance. The SVM baseline performed worse comparing to the other single classifier approaches.

Using the semi-supervised routine can make a decisive difference on the performance, as can be seen from the System II results. Especially for task II, we see that the semi-supervised approach was 4 points better. Interestingly, the baseline SVM performed best in this task.

7 Submitted Runs

7.1 For Task I

The following runs were submitted to the GermEval organizers for Task I:

- **spMMMP_coarse_1**: System I, best model out of 15 runs.

System II	***F-1 macro***
SpinningBytes-CNN+GRU	0.4100 ± 0.0363
SpinningBytes-CNN+GRU Semi	0.4549 ± 0.0324
SVM	$\mathbf{0.4797 \pm 0.0346}$

Table 2: Results for the CNN+GRU classifier on task 2. All reported scores are the performance on the holdout dataset from each specific run, measured in F1-score (macro) over the `OFFENSIVE`, `ABUSIVE`, `INSULTING` and `OTHER` labels for the 4-class classification task.

- **spMMMP_coarse_2**: System I, second-best model out of 15 runs.
- **spMMMP_coarse_3**: System II with semi-supervised augmented training data, best model out of 5 training runs.

7.2 For Task II

The following runs were submitted to the GermEval organizers for Task II:

- **spMMMP_fine_1**: System II without semi-supervised augmented training data, best model out of 5 training runs.
- **spMMMP_fine_2**: System II with semi-supervised augmented training data, best model out of 5 training runs.

System	*F-1 macro*
SVM	0.7266 ±0.0212
System I	
Single Classifiers	
SpinningBytes-FT CNN	0.7547 ± 0.0160
SpinningBytes-W2V CNN	0.7656 ± 0.0143
fastText-Wiki CNN	**0.7703 ± 0.0102**
SpinningBytes-BP CNN	0.7354 ± 0.0188
Meta Classifiers	
Random Forest	0.7843 ± 0.0188
Majority Vote	0.6813 ± 0.0329
Logit Averaging	**0.8048 ± 0.0138**
One Trigger	0.6304 ± 0.0223
Logistic Regression	0.7762 ± 0.0308
Linear SVM	0.7686 ± 0.0334
Multilayer Perceptron	0.6638 ± 0.1299
System II	
SpinningBytes-CNN+GRU	0.7454 ± 0.0168
SpinningBytes-CNN+GRU Semi	0.7684 ± 0.0087

Table 3: Classification results on the task I training data. All reported scores are the performance measures in F1-score (macro) over 5 randomly different tests on the holdout set.

- **spMMMP_fine_3**: SVM with TF-IDF and semi-supervised augmented training data.

8 Conclusion

In this paper, we described our two different approaches to tackling the problem of detecting offensive content in micro-blog posts from Twitter in the context of the GermEval 2018 Competition.

The first system used an ensemble of the same CNN base model initialized with different types word embeddings. These models are then used in combination with an output-averaging approach to generate the final prediction. A preliminary evaluation of the system showed that it achieves an average F1-score (macro) of 80% on average on randomly chosen holdout datasets on the binary classification task.

The second system used a combination of a CNN and GRU architecture with two different type of word embeddings. The preliminary evaluation on a randomly chosen holdout set showed that it could achieve a performance of 45% with respect to the macro-averaged F1-score over all four labels from the multi-label classification task.

References

Bird Steven, Loper Edward and Klein Edward. 2009. *Natural Language Processing with Python*. O'Reilly Media Inc.

Bojanowski Piotr, Grave Edouard, Joulin Armand and Mikolov Tomas. 2017. *Enriching Word Vectors with Subword Information*. *Transactions of the Association for Computational Linguistics*. Association for Computational Linguistics.

Cho Kyunghyun, Van Merriënboer Bart, Gulcehre Caglar, Bahdanau Dzmitry, Bougares Fethi, Schwenk Holger, Bengio Yoshua. 2014. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. *arXiv preprint arXiv:1406.1078*. arXiv.org.

Davidson Thomas, Warmsley Dana, Macy Michael, Weber Ingmar. 2017. *Automated Hate Speech Detection and the Problem of Offensive Language*. *Proceedings of the 11th International AAAI Conference on Web and Social Media*. ICWSM.

Deriu Jan, Gonzenbach Maurice, Uzdilli Fatih, Lucchi Aurelien, De Luca Valeria and Jaggi Martin. 2016. *SwissCheese at SemEval-2016 Task 4: Sentiment Classification using an Ensemble of Convolutional neural networks with distant supervision*. *Proceedings of the 10th International Workshop on Semantic Evaluation*, 1124–1128. Association for Computational Linguistics.

Deriu Jan, Lucchi Aurelien, De Luca Valeria, Severyn Aliaksei, Müller Simon, Cieliebak Mark, Hoffmann Thomas and Jaggi Martin. 2017. *Leveraging Large Amounts of Weakly Supervised Data for Multi-Language Sentiment Classification*. *Proceedings of the 26th International Conference on World Wide Web*, pages 1045–1052. International World Wide Web Conferences Steering Committee.

Glorot Xavier and Bengio Yoshua. 2010. *Understanding the Difficulty of Training Deep Feedforward Neural Networks*. *Proceedings of the thirteenth international conference on artificial intelligence and statistics* 2010, pages 249–256. ACM.

Ioffe Sergey and Szegedy Christian. 2015. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. *Proceedings of the 32nd International Conference on International Conference on Machine Learning* volume 37. JMLR.org.

Jauhiainen Tommi, Linden Krister and Jauhiainen Heidi. 2018. *HeLI-based Experiments in Swiss Germ Dialect Identification* (in Press). *Proceedings of the Fifth Workshop on NLP for Similar Languages, Varieties and Dialects (VarDial)*.

Johnson Rie and Zhang Tong. 2015. *Semi-Supervised Convolutional Neural Networks for Text Categorization via Region Embedding*. *NIPS 2015 - Advances in Neural Information Processing Systems*. Association for Computational Linguistics.

Joulin Armand, Grave Edouard, Bojanowski Piotr and Mikolov, Tomas. 2017. *Bag of Tricks for Efficient Text Classification. Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics* volume 2, short papers. Association for Computational Linguistics.

Kalchbrenner Nal, Grefenstette Edward and Blunsom Phil. 2014. *A Convolutional Neural Network for Modelling Sentences. ACL - Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Kim Yoon. 2014. *Convolutional Neural Networks for Sentence Classification. EMNLP 2014 - Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Kingma Diederik and Ba Jimmy. 2014. *Adam: A Method for Stochastic Optimization. arXiv prepr.int arXiv:1412.6980*. arXiv.org.

Kralj Novak Petra, Smailović Jasmina, Sluban Borut, Mozetič, Igor. 2015. *Sentiment of emojis. PLoS ONE* Volume 10. PLoS ONE

Mahata Debanjan, Friedrichs Jasper, Shah Rajiv Ratn and Hitkul. 2018. *# phramacovigilance-Exploring Deep Learning Techniques for Identifying Mentions of Medication Intake from Twitter. arXiv preprint arXiv:1805.06375*. arXiv.

Mikolov Tomas, Sutskever Ilya, Chen Kai, Corrado Greg and Dean Jeff. 2013. *Distributed Representations of Words and Phrases and their Compositionality. NIPS 2013 - Advances in Neural Information Processing Systems*. Curran Associates, Inc.

Nair Vinod and Hinton Geoffrey E. 2010. *Rectified linear units improve restricted boltzmann machines. Proceedings of the 27th international conference on machine learning (ICML-10)*. Omnipress.

Ross Björn, Rist Michael, Carbonell Guillermo, Cabrera Benjamin, Kurowsky Nils, Wojatzki Michael. 2016. *Measuring the Reliability of Hate Speech Annotations: The Case of the European Refugee Crisis. Proceedings of NLP4CMC III: 3rd Workshop on Natural Language Processing for Computer-Mediated Communication*. Bochumer Linguistische Arbeitsberichte.

Sennrich Rico, Haddow Barry and Birch Alexandra. 2016. *Neural Machine Translation of Rare Words with Subword Units. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* volume 2, long papers. Association for Computational Linguistics.

Severyn Aliaksei and Moschitti Alessandro. 2015. *Twitter Sentiment Analysis with Deep Convolutional Neural Networks. Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM.

Severyn Aliaksei and Moschitti Alessandro. 2015. *UNITN: Training Deep Convolutional Neural Network for Twitter Sentiment Classification. SemEval 2015 - Proceedings of the 9th International Workshop on Semantic Evaluation*. Association for Computational Linguistics.

Srivastava Nitish, Hinton Geoffrey, Krizhevsky Alex, Sutskever Ilya and Salakhutdinov Ruslan. 2014. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research* volume 15. JMLR.org.

Tuggener Don. 2016. *Incremental Coreference Resolution for German. PhD Thesis*. University of Zurich.

Tuggener Don. 2018. *Evaluating Neural Sequence Models for Splitting (Swiss) German Compounds* (in press). *Proceedings of the 3rd Swiss Text Analytics Conference - SwissText 2018*. ceur-ws.org.

Waseem Zeerak and Hovy Dirk, 2016. *Hateful symbols or hateful people? predictive features for hate speech detection on twitter. Proceedings of the NAACL student research workshop* 2016, pages 88–93. NAACL.

Xu Kelvin, Ba Jimmy, Kiros Ryan, Cho Kyunghyun, Courville Aaron, Salakhudinov Ruslan, Zemel Rich, Bengio Yoshua. 2014. *Show, attend and tell: Neural image caption generation with visual attention. International conference on machine learning* 2015, pages 2048–2057. icml.cc.

Zhang Ziqi, Luo Lei. 2018. *Hate Speech Detection: A Solved Problem? The Challenging Case of Long Tail on Twitter. arXiv preprint arXiv:1803.03662* arXiv.org.