Final Project for lab course "Python for Engineering Data Analysis"
Department of Electrical and Computer Engineering
Technical University of Munich

Tweet Text Sentiment Analysis using LSTMs

A Deep Learning Approach evaluating custom trained Word Embeddings

Stefan Rummer [03709476] ✉ **and Adrian Brünger [03715546]** ✉

Department of Electrical and Computer Engineering, Technical University of Munich

✉ stefan.rummer@tum.de
✉ adrian.bruenger@tum.de

September 11th, 2021

Abstract — In this analysis project, a data set [3] containing US airline tweets labeled according to their sentiment has been processed in order to train a deep neural network (DNN) that serves as a text sentiment classifier model. The authors aim is to evaluate the performance of a custom trained word vectorization approach in the context of preprocessing and dimension reduction for tweet text analysis. In addition, the final model is applied to recently extracted tweet data related to a selected US airline to demonstrate the model performance in real-world use cases.

1 Introduction

Given that the social network Twitter is widely accessible and heavily used to express individual thoughts on specific topics online, it can be regarded as a valuable source to gather information on user opinions concerning certain issues. Especially, for purposes of business, it has become of great importance to constantly monitor online sentiments based on textual data. In the following it is assessed up to which extent one can achieve such insights with limited training data and processing power available.

2 Theoretical Foundations

2.1 Basics of Natural Language Processing

Natural Language Processing (NLP) is a range of computational techniques that focus on processing and analyzing natural language texts [2]. In an NLP context, this project's goal is to train a DNN with sentiment-labled phrases/word sequences drawn from a text corpus with a limited vocabulary in order to predict sentiment-labels of previously unseen word sequences. Concretely, the DNN assigns an example three probabilities for each sentiment class (negative, neutral and positive), motivated by the words the word sequence contains and the context they occur in.

2.1.1 Tokenization and Padding

To generate sensible training data from preprocessed (as described later on in subsection 3.2) sequences of words, the authors approach is to represent each of the sequences as a vector of fixed length. To accomplish that, each word in the text corpus is seen as a token. A fixed vocabulary size v is defined such that the $v-1$ most frequent tokens in the text corpus are kept for further analysis, while all other tokens are regarded as out of vocabulary (oov). This is done to maximize the probability that a token on average occurs with a meaning and in a context that it is commonly used in and therefore to minimize variance in meanings and contexts of a token. Furthermore, model performance is increased. Each token is assigned to a unique integer from 1 to $v-1$ based on its frequency (most frequent word gets assigned to 1), while 0 represents the padding token. Padding tokens are then needed to pad all tokenized sequences to a fixed maximum sequence length l by using the "keras.preprocessing"-library.

2.1.2 Word Embedding and Dimension Reduction

So far each phrase is represented by a sequence of tokens in $[0, v-1]$ of length l, while each token is seen as a feature. To reasonably encode the tokens one would have to use One-Hot encoding to avoid introducing a ordinal relationships through a language-wise meaningless tokenization order and arbitrary focus on larger token values in the weight optimization of the DNN. That would result in a v-dimensional orthogonal feature space, which is performance heavy and secondly might lead to degraded learning due to data sparsity [4]. Avoiding these problems, word embeddings are used to reduce the dimension of the feature space to a variable dimension $d < v$. Word embeddings are (abstract) vector representations of words that can further encode interesting characteristics of the vocabulary, such as similarities of words.

2.2 Suitable Neural Network Models

2.2.1 Recurrent Neural Networks (RNN)

RNNs are -contrary to feed forward networks- able to "memorize" past input tokens by introducing cyclic relationships between previous and current hidden states, which enables the model to predict the current output depending on prior tokens in an input sequence [5].

2.2.2 Long Short Term Memory (LSTM)

In an LSTM, the recurrent hidden layer units of a RNN are replaced by purpose-built memory cells. As a result, they may be better at finding and exploiting long range dependencies in sequence-like data. LSTMs are furthermore addressing the vanishing and exploding gradient problems of conventional RNNs [12]. Due to (most) natural language heavily relying on dependencies between word-tokens in a word-token sequence, LSTMs offer great potential of modeling natural language [5] [12]. To analyze the context of a word-token it is necessary to access both past and future tokens. For this type of task Bidirectional LSTMs can be utilized [5], which justifies the implementation of such a network in the model pipeline (subsection 4.1).

3 Fundamental Data Analysis

3.1 Data Set Statistics

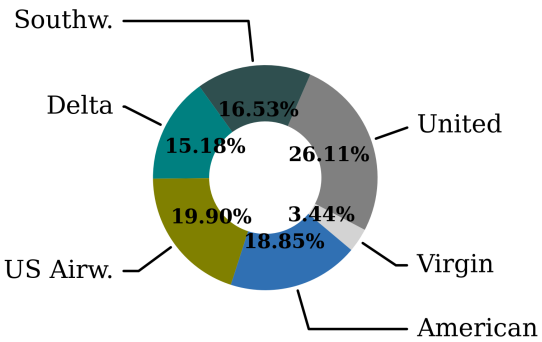

Figure 1 Proportion of tweets included in the data set.

The data set used in this analysis [3] contains a total amount of 14640 tweets from six major US air lines, which have been labeled according to their sentiment. The fundamental distribution of sentiment classes for each respective airline has been illustrated in Figures 2 and 1. Not only one can observe a great imbalance towards negative tweets, but also the amount of tweets available for each airline greatly varies.

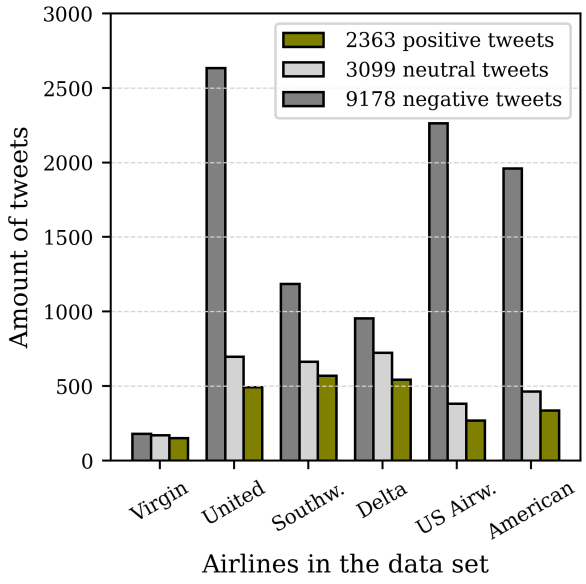

Figure 2 Distribution of sentiments for each airline represented in the provided data set.

3.2 Tweet Data Preprocessing

Previous to tokenization and padding, every tweet of the data set undergoes a series of modifications to provide optimal conditions for training the model. Using a dedicated python library called "emoji", all emoticons and graphic symbols are extracted from the tweet text and then added to the very end of the tweet, each emoji only once. Furthermore, the tweet texts are converted to be fully lowercase and unnecessary white spaces or tabulators are deleted. Also, hyperlinks, mentions of other Twitter users, retweet indicators and all remaining non-letter symbols are removed. The sentiment labels are being One-Hot encoded.

4 Model Structure and Implementation

4.1 Keras based Model Pipeline

The DNN-LSTM illustrated in Figure 3 is constructed with the "keras.layers"-API. First off an "Embedding" layer is utilized to incorporate the advantages shown in subsubsection 2.1.2 in a trainable, supervised fashion. Each row of the weight-matrix of the "Embedding" layer represents the embedding vector of the word-token that equals the row index. The layer takes the preprocessed, tokenized and padded sequences as an input and outputs a $l \times d$ matrix per training batch.

A "SpatialDropout1D" layer is used to randomly deactivate 40% of 1D input feature maps (embedding vectors/rows in the "Embedding" output matrix) to prevent the model from over-fitting to common "word neighborhoods" that occur in natural language [13].

The "bidirectional LSTM" layer functions as described in subsubsection 2.2.2 with the number of units equaling the embedding dimension d. A dropout, as well as a recurrent dropout of 20% each are added to further prevent the model from over-fitting.

Finally, a "Dense" layer is used to process the output of the "LSTM" layer to a three dimensional model output for categorical crossentropy loss computations, using the One-Hot encoded sentiment labels. Due to its universally great performance "Adam" is used for optimizing [6].

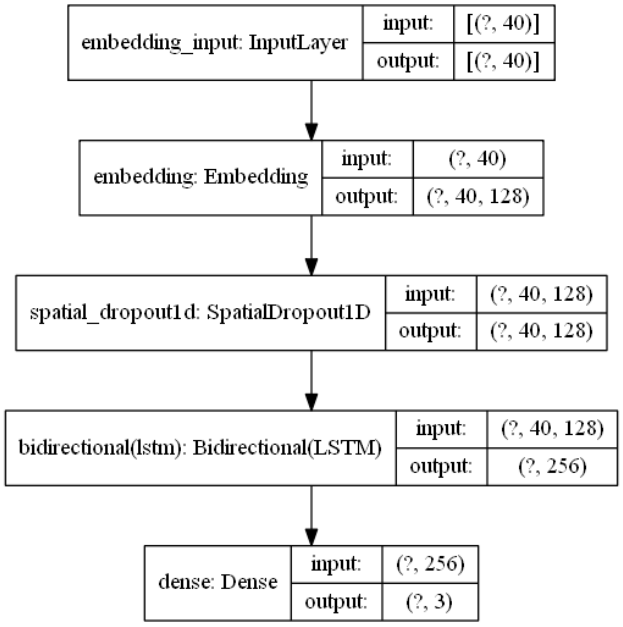

Figure 3 Keras LSTM model layer architecture.

4.2 Custom trained Word Embeddings

As an approach to increase model performance, pretrained embeddings can be used to replace the trainable weights of the "Embedding" layer (Figure 3) [8]. In the following, continuous Skip-gram Word2Vec embeddings are trained using negative sampling. For a given target word the idea is to predict words (context words) in a window around the target word by training a NN model with target- context pairs (Skip-grams), filled with negative samples (words, that do not lay in the context of the target word). Assuming that similar (target) words occur in similar contexts, the learned (target) word embeddings are also likely to represent these similarities.

4.2.1 Skip-grams and negative sampling

Training examples are generated by first randomly sampling positive Skip-gram pairs for each sequence of the form (target word; context word) using a sampling table. The sampling table ensures that, assuming the word frequencies in the text corpus follow Zipf's law [1], higher frequent words, such as stop words, that do not encode much information for the model to learn from, are sampled less often. The context words are drawn from a window of size 4 (2 words befor and after the target word). Secondly, for each positive Skip-gram 4 negative samples are drawn from the vocabulary by again using a Zipfian distribution. In total each training example has the form (target word; (positive) context word, 4 negative context words). Since the idea is to predict true context words for a given target word, each training example is labeled $[1, 0, 0, 0, 0]$.

4.2.2 Word2Vec Model

The structure of the used Word2Vec model shown in 4 functions as follows: Target words are fed into the target model (marked with "t"), that returns an embedding vector of size d. Context words are fed into the context model (marked with "c"), which returns a $5 \times d$ matrix containing embedding vectors of the positive and negative context words. The model updates the embedding weights, and therefore the embedding vectors, by minimizing categorical crossentropy loss from the logits, obtained by calculating the dot product of the target embedding vector with the context embedding matrix and $[1, 0, 0, 0, 0]$. Again "Adam" is used for optimizing. It is important to note that the training of word embeddings aims at minimizing training loss in order to learn the given vocabulary instead of generalizing well on unseen data.

4.2.3 Embedding Visualization

For visualizing the resulting embedding vectors (Figure 5) a Principal Component Analysis (PCA) was used to reduce the dimensions of the embedding vector space from $d = 256$ to $d_{PCA} = 2$, displaying a variance proportion of $5,86\%$. From a linguistic standpoint the visualization works better with a rather small vocabulary ($v \approx 500$), since the cardinality of the used data set is relatively small and the use of words out of context and grammatical structure on Twitter is

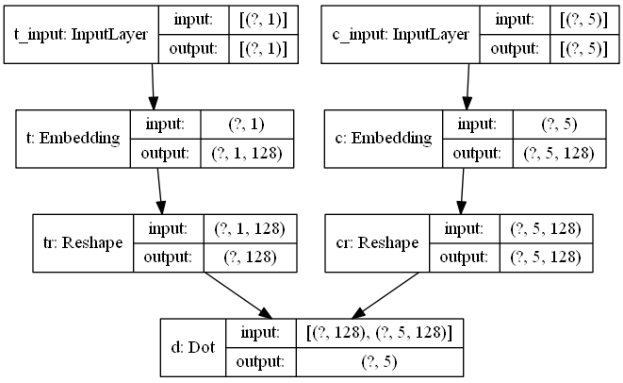

Figure 4 "Keras Word2Vec Skip-gram model architecture

expected to be quite common. Thus, the variance in meaning and context of less frequent words is assumed to be large, which might result in two word vectors being close (regarding euclidean distance), without any linguistic context. As an example, in Figure 5 the words "seat", "destination" and "broken", that are close to "plane" are linguistically sensible in the context of "plane", but belong to totally different groups of words and can also be used in different contexts. That is most likely caused by occurrences of these words almost exclusively in the context of "plane" and not in different contexts that would, by assumption in subsection 4.2, reduce the similarity of the words and therefore increase their distance in the embedding space to "plane".

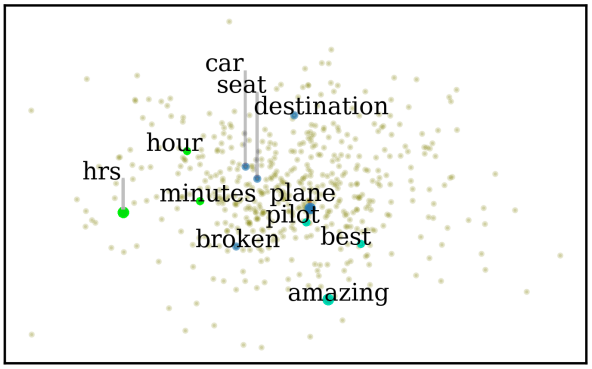

Figure 5 Closest word vectors to "hrs" (green), "plane" (blue), "amazing" (turquoise) within a 2D vector space.

4.3 Further Hyperparameters and Training

For the given data set text corpus a vocabulary size of $v = 5000$ and an embedding dimension of $d = 128$ has proven to be the best performing hyperparameter choice, capturing the majority of variance in the vocabulary and structure in the word sequences while avoiding noise through less frequent words. The maximum length of the sequences l is set to equal the longest sequence in the text corpus.

The data is divided in a test- and validation set containing 20% of the generated examples each. Training examples are fed into the model in batches of size 64 to increase the learning efficiency. The model with the lowest validation loss after 20 epochs is saved.

5 Performance and Results

With the trainable embedding configuration and hyperparameters discussed above, the authors reached a loss of 0.4536 and an accuracy of 0.8207 on the test set with the training curves shown in Figure 6. The respective confusion matrix is shown in Figure 7. Since the majority of examples is labeled "negative", it is plausible that False-"neutral"- and False-"postive" rates are much larger than the False-"negative" rate. From a practical standpoint, "negative" labeled text that is predicted "positive" and vice versa have the greatest impact on the performance assessment. Hence, with $5,43\%$ of test examples falsely labeled as described above, the model performs well considering the quality and quantity of the underlying data (subsection 3.1).

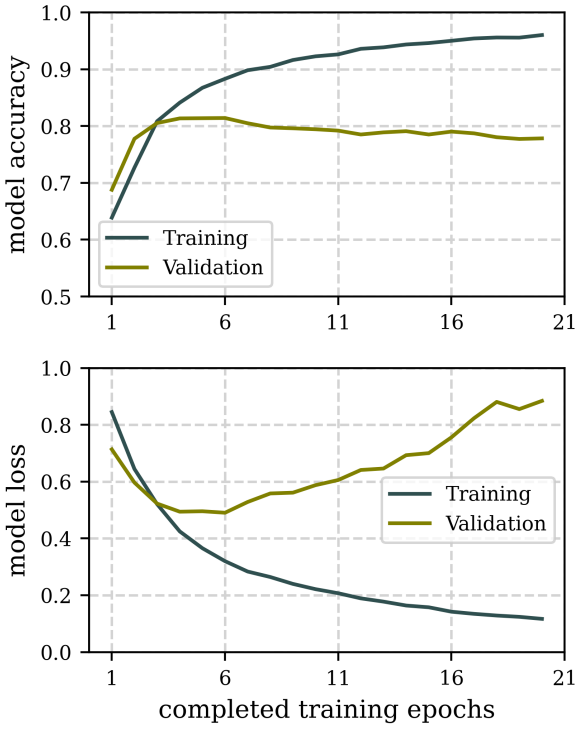

Figure 6 Evolution of accuracy and loss values over completed model training epochs, for train and test data sets.

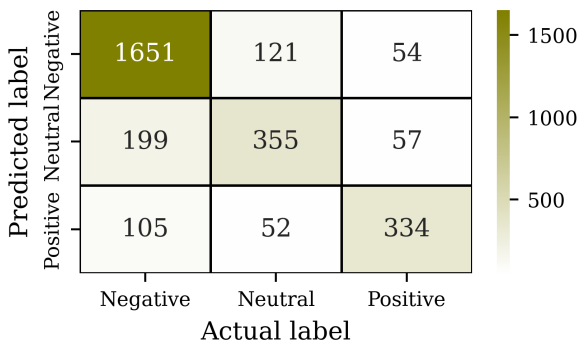

Figure 7 Confusion matrix representing the characteristics and performance of the final sentiment classifier model.

Using the pre-trained embeddings a slightly worse performance of 0.5046 loss and 0.8087 accuracy on the test set was achieved by removing the "SpatialDropout1D" layer and adding a "Dense" layer with the units equalling the vocabulary size v to compromise the loss of trainable weights when freezing the embedding layer weights. The loss in performance is assumed to be caused by the sub-optimal quality of the pre-trained embeddings for the chosen vocabulary size.

6 Application in selected Use Cases

In order to demonstrate potential use cases of the implemented NN model, samples of the 250 most recent and but also popular tweets available for US Delta airline (@delta) [14] were fetched from Twitter via the "Tweepy"-API [11]. This analysis was conducted within a time frame ranging from 18:58 to 22:39, on the 5th of September 2021. Extracted tweets were then processed by the sentiment classifier. For each set of sentiment scores, a final score value was calculated in a way that the positive score was reduced by the amount of the negative score, what indirectly also takes the magnitude of the neutral score into perspective. The resulting score values, as well as the distribution of sentiments detected, were visualized in Figure 8. The mean sentiment can be interpreted as the average Twitter user opinion about Delta Air Lines.

7 Conclusion

Based on the aforementioned results, one can conclude that the implementation of a deep learning model to analyse tweet sentiments is possible and can yield real time insights with sufficient accuracy.

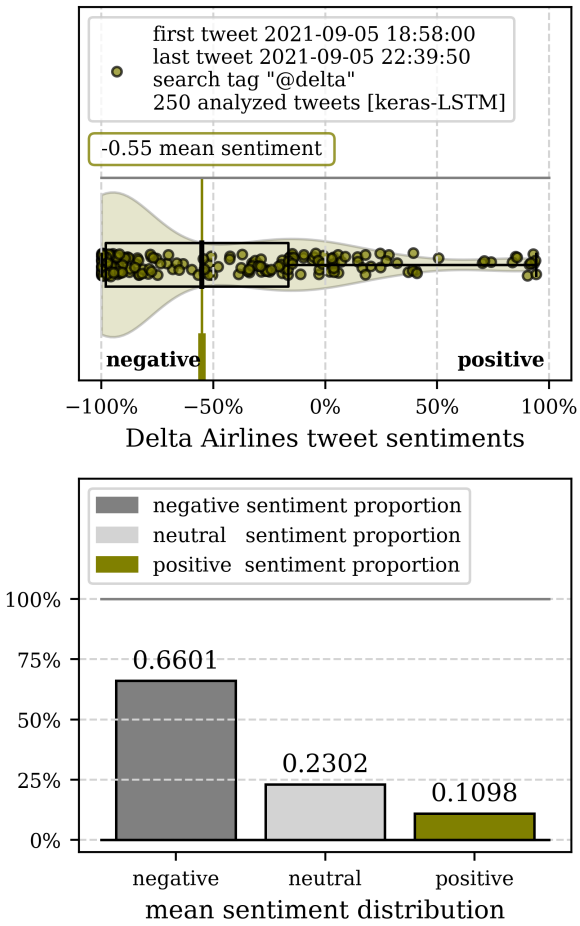

Figure 8 Average Twitter sentiment for Delta airlines from the 05.09.2021, 250 Tweets within approx. four hours.

8 Outlook and Improvements

8.1 Data related Improvements

According to the initial analysis (subsection 3.1), the data set provided lacks a uniform distribution among the tweet classes represented. It contains way more negative tweets than positive or neutral ones. For future approaches, one might aim for a more balanced but most importantly a larger data set to achieve better performing classifier models. The latter would also improve the quality of pretrained embeddings for a larger vocabulary size, that is necessary to display structure and context for further processing with a LSTM network. In addition, previous to training, a dropout filter for all tweets that lack sufficient confidence regarding their sentiment label can be implemented.

8.2 Embedding related Improvements

The quality of the embeddings implemented in this project may have been constrained by the limited amount of training data available in the US airlines

data set. An idea to improve the embeddings for the data at hand would be to adopt Stanford's GloVe embeddings, which are trained on a 6 billion token text corpus from Wikipedia and Gigaword [10].

The sentiment classifier model in this project is based on a word vectorization method to create suitable embeddings and therefore by design only captures the similarity of words within the defined vocabulary vector space. The (Word2Vec) embeddings actually only learned to classify the word sense for each expression individually. Consequently, it delivers a single representation for a word that is the same regardless of its context [7].

One possible approach to improve on this major flaw would be the implementation of a Transformer based sentiment classification model, which is able to evaluate words in the context of their respective sentences. Stacked Transformer blocks are capable of capturing even distant dependencies between phrase patterns and can shift their attention towards specific keywords within a sentence [9]. Furthermore, such an advanced approach may also perform significantly better in handling contextual noise within sentences, such as ignoring irrelevant phrases that might be interpreted in a misleading way by the NLP model used.

References

[1] Lada A Adamic and Bernardo A Huberman. "Zipf's law and the Internet." In: *Glottometrics* 3.1 (2002), pp. 143–150.

[2] Ronan Collobert et al. *Natural Language Processing (almost) from Scratch*. 2011. arXiv: 1103.0398 [cs.LG].

[3] Crowdflower's Data for Everyone library. *Kaggle Data set: Twitter US Airline Sentiment*. URL: https://www.kaggle.com/crowdflower/twitter-airline-sentiment?select=Tweets.csv. (last accessed: 20.08.2021).

[4] Fei Huang and Alexander Yates. "Distributional Representations for Handling Sparsity in Supervised Sequence-Labeling". In: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. Suntec, Singapore: Association for Computational Linguistics, Aug. 2009, pp. 495–503. URL: https://aclanthology.org/P09-1056.

[5] Zhiheng Huang, Wei Xu, and Kai Yu. *Bidirectional LSTM-CRF Models for Sequence Tagging*. 2015. arXiv: 1508.01991 [cs.CL].

[6] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[7] Antonio Lopardo. *Word2Vec to Transformers*. URL: https://towardsdatascience.com/word2vec-to-transformers-caf5a3daa08a. (last accessed: 02.09.2021).

[8] Tomás Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality". In: *CoRR* abs/1310.4546 (2013). arXiv: 1310.4546. URL: http://arxiv.org/abs/1310.4546.

[9] Usman Naseem et al. "Transformer based Deep Intelligent Contextual Embedding for Twitter sentiment analysis". In: *Future Generation Computer Systems* (2020). DOI: https://doi.org/10.1016/j.future.2020.06.050. URL: https://www.sciencedirect.com/science/article/pii/S0167739X2030306X.

[10] Jeffrey Pennington, Richard Socher, and Christopher D Manning. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[11] Pablo Rivera. *Twitter Teepy API*. URL: https://www.tweepy.org/. (last accessed: 05.09.2021).

[12] Haşim Sak, Andrew Senior, and Françoise Beaufays. *Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition*. 2014. arXiv: 1402.1128 [cs.NE].

[13] Jonathan Tompson et al. *Efficient Object Localization Using Convolutional Networks*. 2015. arXiv: 1411.4280 [cs.CV].

[14] Wikipedia. *Delta Air Lines*. URL: https://de.wikipedia.org/wiki/Delta_Air_Lines. (stock ticker: DAL, based in: Atlanta, Georgia, USA).