Predicting Stock Performance Using 10-K Filings

A Natural Language Processing Approach Employing Convolutional Neural Networks

Master's Thesis

COPENHAGEN BUSINESS SCHOOL
HANDELSHØJSKOLEN

Kasper Regenburg Jønsson
MSc Finance and Investments
Student number: 71638

Jonas Burup Jakobsen
MSc Finance and Accounting
Student number: 41792

Supervised by
Thomas Plenborg
Thomas Riise Johansen

May 15, 2018
Characters: 137,891
Pages: 88

Abstract

This paper aims to predict company-specific performance based on the textual elements of 10-K filings. Due to their limited information processing capacity, investors need time to incorporate the information contained within the textual content of the 10-K filings into the market. This delay generates an opportunity for the investors to earn abnormal returns using automated text analysis. Using word embeddings to represent the text as input to a convolutional neural network (CNN), we analyze the text of over 29,000 10-K filings from 2010 to 2017. We find that company-specific stock performance is predictable. Furthermore, we control the results for known risk factors using the Fama-French 5 factor model finding that the investors are able to generate significant risk-adjusted returns based on the classifications of the CNN. Based on the findings, we propose several implications. Firstly, we confirm that the textual elements of the 10-K filings contain information which the investors currently do not fully utilize. Secondly, we contribute to the validity of using deep learning models when predicting company-specific performance. Lastly, we provide a practical tool for the investors, the regulatory entities, and the respective company to analyze the textual elements of the 10-K filings.

Contents

Chapter 1

Introduction

The annual report has been shown to be one of the most important external documents of a company (P. Hájek and Olej 2013). It is a vital tool which helps the management communicate the strategy and financial performance of the company, which serves as the foundation for investor valuations. However, the information content of the reports does not seem to be efficiently incorporated by the financial market. Ball and Brown (1968) and Bernard and Thomas (1989) showed how the market deviates from the fundamental value of stocks following an earnings announcement. Discarding the assumption that investors have unlimited information processing capacity can aid in explaining this apparent flaw in the market efficiency hypothesis (Engelberg 2008). As a result, it takes time for agents to incorporate the information of the annual report in the stock price. This delayed response to the information is important to examine, because it creates an opportunity for investors to earn abnormal returns (Fortuny et al. 2014). In this paper, we examine whether it is possible to predict company-specific performance by processing the textual information. In order to do this, we implement a deep learning algorithm to create a model that can analyze the textual elements in annual reports instantly and subsequently classify the company in one of five portfolios based on their expected performance.

An annual report contains a mix of hard information, defined as the accounting information (the numbers), and soft information, defined as the text. Following this definition, the hard information is easy to compare across firms (e.g. $8 dividends versus $12) and regardless of who collects it. Soft information is not as easily comparable across firms and the meaning depends on who collects the data (e.g. "risk of business has increased" is an ambiguous statement and will be interpreted differently) (Engelberg 2008). As a result, the hard information in the reports is incorporated into the stock prices much quicker than the soft information. In accounting, researchers have used several different methods to try to quantify the soft information enabling the prediction of company performance. The most popular methods involve the creation of parameters to measure one or more aspects of the text, such as readability, length, and sentiment. The first paper to link linguistic features of the annual report to company performance was Li (2008). Li finds evidence of a negative correlation between readability (measured with the Fog Index score) and company performance. Lang and Stice-Lawrence (2015) find that changes in length, boilerplating[1], and comparability in annual reports are correlated with economic outcomes of companies. These studies, among others, indicate that the soft information in annual reports indeed can be used for predicting company performance.

However, these methods of analyzing text are vulnerable due to their simple constructions. For example, the very popular *Bag of Words* model does not take the order of the words into account. The sentence "although the year was not good, we did meet our targets" would be classified the same way as "although the year was good, we did not meet our targets", which is highly problematic. The difference between the two sentences would not be caught by the simple approaches but it is vital to the investors. Moreover, when it comes to sentiment, the most common method used is classifying texts based on a chosen dictionary of words manually assigned to pre-defined categories[2] (Loughran and McDonald 2016). Obviously, one must consider the possible bias of the author, and their preconceptions of the meaning of certain words, but also take into account the general changes in language and its uses. Fortunately, more advanced techniques, like Word2Vec and GloVe, have been developed to quantify the soft information. The idea behind these methods is that individual words are represented numerically by vectors that have been trained on

[1]A measure describing how much text was reused from the previous annual report

[2]The most used categories are Positive and Negative

large amounts of textual data (e.g. the Wikipedia website). This allows the representations to contain very specific and fine-grained aspects of language (Pennington, Socher, and Manning 2014). The representations have proven successful in other Natural Language Processing tasks (NLP), however, their application in accounting research has been sparse. One of the main reasons is that they are not well-suited to use as input in the classic prediction models, such as simple regressions.

Advancements in the field of machine learning and deep learning has made the utilization of more complex inputs, such as word vectors, viable. In combining textual analysis (hereunder NLP), deep learning, and stock prediction, researchers have primarily analyzed financial news and social media posts. For example, Pinheiro and Dras (2017) and Ding et al. (2015) used financial news to predict stock returns using a recurrent neural network (RNN) and convolutional neural network (CNN), respectively, with promising results. They use text directly without any intermediate measure of sentiment or readability. Kraus and Feuerriegel (2017) analyzed the German equivalent to 8-K reports using GloVe to represent the text. They used multiple variants of RNNs to predict the direction of company-specific stock prices with a high accuracy. These results both inspired us to use deep learning in combination with textual analysis, as well as contribute to the validity of the methods. However, to the best of our knowledge, no one has yet attempted to predict company-specific performance based on the soft information of annual reports by use of these techniques. For this reason, we propose and test the following hypothesis.

1.1 Hypothesis

Hypothesis: Firm-specific stock performance is predictable by analyzing the textual elements of annual reports through natural language processing with deep learning using a convolutional neural network.

1.2 Scope

We limit the geographical scope of this research paper to stocks listed exclusively on the US stock market seeing as training a convolutional neural network requires a vast amount of data. The US has the longest and most consistent data track record with comprehensive data registration from 1994 and forward (SEC 2010), in addition to the more than 5.500 listed companies. This well-documented track record makes it the ideal market for the data requirements of the implemented model.

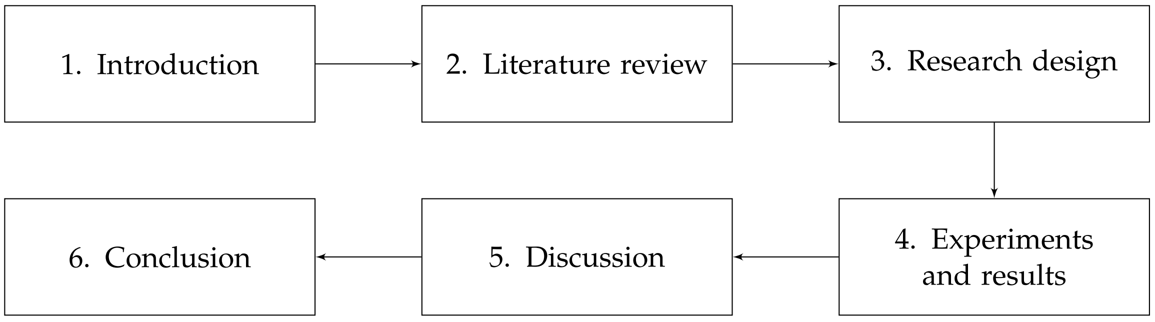
In addition, only publicly traded companies will be included in the analysis of this research paper. In order to evaluate the quality of our predictions, we need to be able to match a specific company report to the stock's following price development to see how the market reacted to that specific report. Looking at the requirements which make a company need to publicly file annual reports (SEC 2013), it is clear that some private companies are required to report these documents as well. Since the private companies do not have any public performance record, it is not possible to implement them in the model.

Furthermore, we have chosen only to use annual reports as our text source. As mentioned, annual reports are the fundamental public disclosure of companies' economic and financial position. The reports are required by law to be reported at specific intervals and with a given content and structure. Opposite to 10-Qs, 10-K filings are required to be audited and they contain a much more comprehensive overview of the company and its performance. The additional soft information in the annual reports makes them ideal for our research purposes, since our proposed model can leverage the extra text to create better pattern recognition.

1.3 Structure of the Paper

This paper intends to follow the format and approach of a scientific paper. Therefore, we organize the remainder of the paper as illustrated in figure 1.1.

Figure 1.1: Project structure



In chapter 2 we examine the previous works in the field's relevant to the research hypothesis. We confirm that the hypothesis is worth pursuing and that our combination of methods is a novel approach. In chapter 3 we present the overall research design of the paper. We describe the data sources used, the research methodology, and how we process the data. A short introduction to neural networks and convolutional neural networks is included to give a basic understanding of the presented learning algorithm. Furthermore, we refer to appendix A for a comprehensive explanation of the neural network theory needed to fully understand the inner workings of our proposed model. Although the comprehensive explanation has been placed in the appendix to keep a steady flow, it is an integral part of the paper. In chapter 4 we present our base-case model, our optimization process, and the results of the best performing model. In chapter 5 we discuss the key methodological elements of the paper and how they influence the results. Finally, in chapter 6 we summarize the final results, we discuss the implications of the results, and propose avenues in which to explore for future research.

Chapter 2

Literature Review

2.1 Introduction

A review of the related literature is relevant to help answer the research questions as it serves multiple purposes. Firstly, it serves as a sanity check, whether the desired topic is something worth pursuing. We show that there is a plethora of research in the field of textual analysis with an accounting perspective, including using NLP and machine learning. Secondly, it ensures that we are not trying repeat work that has already done. We found examples of our methods being used separately but none in the exact combination we employ. Lastly, it serves as a guideline of what is already understood about the subject, best practices, and what is possible or impossible.

Table 2.1: Article selection overview

Process	Remaining articles
Extraction from Scopus	1840
Articles not cited	889
Irrelevant titles	194
Irrelevant abstracts	81

Table 2.1 shows the process of selecting the relevant articles. We use Scopus as the main article database as it has some attractive features, such as advanced query search and drill-down tools. The first step is to create a search query to locate the papers that could be relevant to this paper, resulting in 1840 matches. The authors,

titles, publishers, abstracts, and number of times cites were extracted from Scopus. As a proxy for research quality, articles from 2016 and older which were not cited at least once a year since they were published were excluded. This narrowed the pool down to 889 articles. Subsequently, articles with titles not relevant were removed (e.g. neural networks used in medical diagnostics) resulting in 194 articles. The final sorting was made based on the articles' abstracts, removing all irrelevant to this paper, finalizing the search to 81 articles. Acknowledging that this structured approach potentially left out relevant material, any articles cited in the 81 papers that seem relevant are also included. We show an overview table of the mentioned articles at the end of the literature review in table 2.2 and 2.3.

The rest of the chapter is structured the following way: firstly, we present the main three prediction goals of analyzing text in accounting. Afterward, we highlight the different methods of analyzing text. Following this, we point out the different sources of text used in the literature and, lastly, present the models used.

2.2 Purposes

Prediction of corporate events (fraud detection and bankruptcy detection) and index or stock performance are the three main prediction objectives, which prior literature has focused on in accounting (Amani and Fadlalla 2017). Textual analysis has also been used for various retrospective applications such as monitoring the quality of historical accounting data and inventory optimization (Amani and Fadlalla 2017), but since none of the retrospective objectives are the focus of this paper, they are omitted from the literature review.

2.2.1 Fraud detection

Detecting fraudulent behavior of a company has always been a difficult challenge due to the lack of universal indicators (Goel and Uzuner 2016). Even if such universal indicators existed, companies would know how to construct their disclosures to avoid being investigated, making effective detection impossible. Despite the lack of indicators, researchers in accounting have found some interesting results from analyzing text. Humpherys et al. (2011) are able to predict fraudulent disclosures with a 67.3% accuracy, testing multiple prediction models based on 202 management discussion and analysis sections (MD&A) of annual reports. They create eight categories (e.g. specificity, complexity, uncertainty etc.) and calculate several ratios based on the frequency of the words belonging to the respective categories finding that the fraudulent texts have distinct linguistic cues. Similar results are found in a recent study by Goel and Uzuner (2016). They employ a more advanced method for analyzing the text based on the software *Diction* as well as a more advanced prediction model based on machine learning (Support Vector Machine) achieving an impressive 81.8% accuracy on their best model.

2.2.2 Bankruptcy detection

Early detection of bankruptcy is very valuable for all stakeholders, thus attracting researchers. Cecchini et al. (2010) develop custom dictionaries from MD&A sections of 10-K filings, which show significant prediction power of up to 80% accuracy. From this, they conclude that the soft information in the annual reports contains valuable information in excess of what is reflected in the financial statements. Hájek and Olej (2013) use a finance-specific dictionary to create sentiment measures as inputs to both Neural Networks and Support Vector Machine (SVM). They find that models with both the sentiment measures and financial ratios have 1-3% better accuracy in comparison to models based solely on financial ratios, adding to the idea that the text contains valuable information.

2.2.3 Stock Performance

Predicting stock performance is most often the purpose of textual analysis in accounting (Colm and Sha 2014). Generally, stock performance is predicted in one of two ways: as a specific return (i.e. X%) or as one category (e.g. positive/negative, buy/hold/sell).

Price et al. (2012) analyzes conference calls with a dictionary approach and tries to predict the cumulative absolute return (CAR) following the conference calls. (P. Hájek, Olej, and R. Myšková 2013) also predicts specific stock returns. They use a mix of textual and numeric inputs in their model, finding that the non-linear models performed much better than the base-case linear model.

Kraus and Feuerriegel (2017) predict the direction of absolute and abnormal returns of stock performance where up (1) is positive performance and down (0) is negative performance using an advanced deep learning model. Khedre et al. (2017) also try to predict the direction of stock performance. Hájek and Boháčová (2016) compare the classification ability of some of the simpler machine learning models (e.g. Naïve Bayes and Support Vector Machine) with the more advanced neural networks, finding best performance with the advanced models.

2.3 Textual Measures

As mentioned earlier, researchers have created several measures to quantify the information that is expressed in the text. These measures are then used as quantitative inputs to the respective prediction models. As a result, the text itself is not the input. The most commonly used measures are readability and sentiment (Loughran and McDonald 2016). Others do not calculate any textual measures but instead use more complex numerical representation as input for the prediction objective.

2.3.1 Readability

The most commonly used measure for estimating readability is the Gunning Fog index. It is a score based on average sentence length and number of complex words (words with more than two syllables) that is used to estimate the number of years of formal education a person needs to comprehend a piece of text on a first reading.

The first and very influential paper on readability and company performance is Li (2008). Li finds a significant negative correlation between earnings and the Fog index score of annual reports (i.e. firms with annual reports scoring a high fog index have lower earnings). Although the result is interesting, the real impact of the study is that Li was the first to link linguistic features of the annual report to actual firm performance (Loughran and McDonald 2016). The same negative correlation can be seen when looking at analysts' forecasts. Lehavy, Li, and Merkel (2011) find forecast accuracy to be better and analyst dispersion to be lower for companies with annual reports that have lower Fog index scores. Lo, Ramos and Rogo (2017) offer an alternative link between Fog index scores and performance. They confirm their hypothesis that firms that have managed earnings to beat their benchmark (often last year's performance) have more complex disclosures. This indicates that the relationship found by Li (2008) and others is not necessarily linear but non-monotonic around the benchmark.

Despite the widespread use of the Fog index in accounting literature, it has some fundamental flaws as pointed out by Loughran and McDonald (2014a). Annual reports contain many multi-syllable words that the Fog index scores as complex but would be easily understood by investors. For example, the most common 'complex' words in annual reports include words such as *company*, *financial*, *management*, and *customer* (Loughran and McDonald 2016). Obviously, these words can easily be comprehended by investors. Another issue with the Fog index is the fact that the words in a sentence can be rearranged so it makes no sense, while still obtaining the same score by the Fog index.

As an alternative measure of readability, Loughran and McDonald (2014a) suggest using the natural log of gross 10-K filing file size. They find a significant positive correlation between their alternative measure (the log file size as a proxy for complexity) and stock return volatility, earnings surprises and analyst dispersion (Loughran and McDonald 2014a). Although arguably better and easier to implement, the proposed alternative measure is not perfect either. Certain events have previously shown to affect corporate disclosures such as the Enron accounting scandal (Loughran and McDonald 2016). Loughran and McDonald (2016) argue that researchers, in general, should shy away from using readability and instead focus on the broader topic of information complexity.

2.3.2 Sentiment

Instead of focusing on how readable a piece of text is, sentiment analysis tries to extract the tone of the text. There are two well-established methods of sentiment analysis in the literature: the bag-of-words/dictionary approach and the machine-learning approach (Colm and Sha 2014). Both methods try to capture the sentimental level of the text.

Bag-of-Words

In the bag-of-words model text (e.g. sentences, paragraphs or documents) is represented by vectors based on word frequencies. Hence, a piece of text is represented by a list (vector) of the count of each word in the text. Comparing two of these vectors indicates how similar the texts they represent are; if they have roughly the same distribution of words, the texts are supposedly similar in meaning. Obviously, some words appear very frequently (e.g. the, and, to etc.) and yet they add little meaning to the text. To account for this, the term frequencies are often adjusted by the inverse frequencies in other documents. That is, the less the term appears in other documents in the entire corpus, the more relevant it is to the text's specific topic. This method is known as TF-IDF (term frequency-inverse document frequency).

Dictionary Approach

The widely used dictionary approach is, in its essence, a bag-of-words approach where the words that are counted have been narrowed down by manually creating a dictionary of words that reflect the sentiment the user wants to analyze. The words are often chosen to have a common theme such as positive, negative, uncertainty, etc. A score for each text can then be generated. Usually, the sum of the category or net sentiment (e.g. positive minus negative word count) is calculated. This enables the user to measure and compare texts based on the scores.

The earliest research on sentiment analysis in accounting is based on the *Harvard General Inquirer Word Lists (GI)* (Colm and Sha 2014). The dictionary contains 182 categories (or themes) with the negative category being the largest with 2291 words[3]. Tetlock (2007) links the tone based on the GI word lists of the Wall Street Journal's "Abreast of the Market" to daily stock market levels. He finds that high media pessimism results in downward pressure on the stock prices. He also finds that unusually high or low pessimism results in high trading volumes. Although Tetlock (2007) inspired subsequent research on sentiment tone on stock market performance, his results are not consistent across other types of text inputs. Li (2010) analyzes the tone of MD&A sections of annual reports using three different common English dictionaries (including the GI) and finds that using the dictionaries could not predict future performance. He concluded: "This result suggests that these dictionaries might not work well for analyzing corporate filings" (F. Li 2010, p. 1050).

Henry (2010) was the first to create a finance-specific dictionary. Henry finds that investors' reactions are affected by the tone in earnings press releases. Albeit the novel approach, the dictionary is very short (85 negative and 105 positive) and has left out somewhat obvious words such as *loss, impairment, gain* and *advantage*. Adding to the argument that domain-specific dictionaries are necessary, Loughran and McDonald (2011) (hereafter L&M) show that 73.8% of all the words listed as negative in the Harvard GI dictionary are not considered negative in a financial context. This is both a problem with general terms such as *cost, capital* and *liability* as well as sector-specific words such as *cancer* and *mine* which are common words in the biotech and mining industry, respectively. As an alternative, L&M develops six word lists covering the categories of negative, positive, litigious, uncertainty, strong modal, and weak modal. The word lists are created by examining the 5% most common words in their dataset of 10-K filings from 1994 to 2008 (Loughran and McDonald 2011). As a comparison to Henry's negative word list, L&M's contains 2337 words, making it considerably more detailed. L&M shows how their new word lists outperformed the commonly used Harvard GI in every aspect. Hence, it comes as no surprise that L&M's finance-specific word lists have become predominant in recent studies (Colm and Sha 2014). The following studies have used the L&M dictionary to analyze the sentiment of text in order to predict stock performance: Myšková et al. (2018), Hájek

[3]More information can be found at http://www.wjh.harvard.edu/~inquirer/3JMoreInfo.html

et al. (2016), Ahmad et al. (2016), Chen et al. (2014), Li et al. (2014), Hájek and Olej (2013), and Hájek et al. (2013).

Although the dictionary approaches have shown great results, they have also been criticized for their simplicity. First, they are subject to the creator's subjective presumptions of what words belong to which category. This is particularly true for homophones which can have significantly different meaning depending on the context. Second, they do not take the words' context into account. When calculating the scores for each text, the composition of the text is not considered. As mentioned earlier, the sentences "although the year was good, we did not meet our targets" and "although the year was not good, we did meet our targets" are scored the same by the dictionary approach. Despite these drawbacks, it is still a very popular method. One of the explanations to this could be that it is very simple to implement compared to the machine learning approaches (Colm and Sha 2014).

Machine Learning

In addition to the models described above, researchers have developed more advanced methods of assigning a sentiment score to a piece of text based on machine learning. These techniques rely on statistical inference to infer the meaning of text (F. Li 2010). The most common model used is the Naïve Bayes (NB) model (Loughran and McDonald 2016). The model is trained using supervising learning. That is, a fraction of the corpus is manually labeled in categories (e.g. positive, negative, bullish, bearish, etc.). This input (text) – output (category) data is then used to train the model, so it learns to recognize the patterns of the chosen categories. Afterward, the model can be used to categorize the rest of the corpus. Sentiment features can then be calculated and used as input for predictive models.

Antweiler and Frank (2004) were the first to use NB to classify text in finance. They measure the sentiment of Yahoo! Finance and Raging Bull[4] messages to predict stock performance and trading volume. They find that the messages help predict

[4]Yahoo! Finance and Raging Bull are two social media platforms for sharing content related to investing.

volatility, trading volume and stock returns. Despite stock returns being significant, they are economically too small to be profitable after trading costs. Sprenger et al. (2014) also use NB to classify roughly 250,000 tweets as bullish or bearish from the social media Twitter. Their more recent results confirm the result of Antweiler and Frank (2004) in that there is valuable information in the microblogging community that is not fully incorporated into the market. Li (2010) also uses NB to classify annual reports in four tones: positive, negative, neutral and uncertain. He manually labels 30,000 observations to be able to train his NB model. He finds that based on the MD&A section of annual reports, the tone is positively associated with future earnings.

The NB is very effective for large corpora of text and eliminates the subjective elements of the researcher once the model has been trained. However, the model has some disadvantages. First, the manual labeling of the training data is prone to errors made by the labeler. Second, the results are hard to recreate by others since the initial variables of the model are set randomly. Lastly, one of the assumptions of the model is that words are conditionally independent (i.e. the probability of each word appearing in a sentence is not affected by which words are in the sentence). This is obviously a rough assumption but empirically it does not seem to influence the model (F. Li 2010).

Word Embeddings

Instead of calculating a measure based on some aspect of the text, word embeddings have the *meaning* of the text embedded in their numerical representation (Pennington, Socher, and Manning 2014). The embeddings are trained without any interference from the user (i.e. unsupervised learning), eliminating any risk of introducing bias. This way of representing text have shown impressive results on word analogy tests, word similarity tests, and information retrieval tasks (Pennington, Socher, and Manning 2014). Word embeddings have also shown to improve prediction results in accounting literature (Kraus and Feuerriegel 2017). Sohangir et al. (2018) argue that word embeddings work well with Convolutional Neural Networks, although they do not specifically predict company performance.

2.4 Text Sources

Three main categories of text sources have been analyzed in the literature: financial news (e.g. The Wall Street Journal, Reuters, analyst reports, etc.), social media (e.g. Twitter, Yahoo! Finance blogs, SeekingAlpha, etc.) and corporate announcements, including annual reports (Kraus and Feuerriegel 2017) (Nardo, Petracco-Giudici, and Naltsidis 2016) (Colm and Sha 2014) (F. Li 2010). All three types have shown significant results in predicting stock performance.

As mentioned earlier, Tetlock (2007) was one of the first and very influential studies using financial news from The Wall Street Journal. Ahmad et al. (2016) use over 5.1 million news articles to study how the time-varying nature of firm-specific returns. They find that the tone of news periodically affects company-specific returns. However, they highlight two interesting points: first, they find that companies have longer periods of time where the sentiment of the media has no effect on stock returns. Additionally, when the sentiment does impact returns, they are sometimes quickly reversed indicating that the sentiment information was just noise. Other times, the impact lasts, implying that the information is fundamentally relevant for the company. They conclude that when the returns are lasting, it is not necessarily because of inefficient markets but because it takes time for investors to process the soft information in the news.

Social media posts have also been linked to stock return. Antweiler and Frank (2004) and Sprenger et al. (2014), as mentioned earlier, analyze posts on social media platforms. Chen et al. (2014) use almost 100,000 posts from the investment forum SeekingAlpha to predict stock returns and find significant results even after controlling for other sources e.g. financial analyst reports and newspaper articles.

As pointed out above, Li (2008) was the first to link linguistic features of annual reports to stock performance. Li's paper sparked the interest of many researchers to analyze annual reports. Li (2010) looked at the forward-looking statements (FLS) in the MD&A section of annual reports where he associates better performance, less return volatility and lower accruals to positive FLS. Loughran and McDonald (2014) argue that the MD&A sections in some reports have been spread out in the relevant other sections of the 10-K filing and that you should look at the entire document as in the case of Hájek and Olej (2013), and Hájek et al. (2013). The two papers analyze the text in annual reports to predict stock performance and financial distress, respectively.

Lang and Stice-Lawrence (2015) took another approach and tested how accounting quality in annual reports has developed. They test how length, boilerplating, and comparability has changed over time and concluded that the overall quality has gone up measured by more disclosure, less boilerplating, and greater comparability. Contrary results are presented by Dyer et al. (2017) who find negative development in length, readability, redundancy, boilerplate, specificity, and the ratio of hard and soft information. The differing conclusions can partly be explained by the way the development was analyzed. Lang and Stice-Lawrence (2015) use relatively simple measures whereas Dyer et al. (2017) use a more advanced technique called Latent Dirichlet Allocation (LDA) as well as a five times larger data pool. Using LDA, they find that the majority of the development is caused by three new mandatory requirements: fair value disclosure, internal control disclosure, and risk factor disclosure. All things equal, this increased length, readability etc. makes it harder to process the soft information in the annual reports, thereby, increasing the potential for automated text analysis as we are proposing in this paper.

2.5 Model Use

A number of models have been used in prior literature to predict stock prices ranging from the simple linear models to the very complex deep learning models. Wisniewski and Yekini (2015) analyze UK annual reports with a dictionary approach and use the dictionary scores as inputs to a linear model. They find that the themes of *activity* and *realism* predict increasing stock price. Sprenger et al. (2014) use machine learning to classify Twitter messages and a subsequent simple linear regression to predict stock returns. Venturing into machine learning territory, Qiu et al. (2014)

and Wu et al. (2014) both use SVM to predict stock performance from 10-K filings and financial news, respectively. Using only time-series data as input, Rather et al. (2015) combine both a linear model with a Recurrent Neural Network (RNN) and find improved prediction performance, showing promising results for the more advanced neural networks. Instead of using textual inputs, Dingli and Fournier (2017) use multiple numerical inputs (e.g. historical prices, currency exchange rates, indices, etc.) with a CNN to predict stock prices. To accompany the way input data must be structured for CNNs, they arrange it in an 8x8 matrix with each value corresponding to an input parameter. They are able to predict the next month's price direction (up/down) with 65% accuracy.

Although they do not predict stock performance, Sohangir et al. (2018) use Convolutional Neural Network (CNNs). They show that CNNs are better at classifying the sentiment of stock related text pieces from the social media StockTwits. They combine the CNN with word embeddings and argue that this combination work very well together because the CNN considers the order of the words which the vector representations ignore as a result of the way they are trained. The fact that they find CNNs to be best at predicting sentiment also indicates that the network 'understands' the text better than other methods. Kraus and Feuerriegel (2017) apply a different type of deep learning model called recurrent neural network with long short-term memory layers (RNN LSTM) to predict stock returns on corporate 8-K filings in the Germany stock market.

2.6 Key Takeaways

We see the development of textual analysis in accounting going towards more advanced machine learning methods, including neural networks. This development has been fueled by the continuous improvement of natural language processing methods, new types of prediction models as well as an increasing amount of textual data to be used for training the complex models. We wish to follow this development and therefore construct the following research design.

Table 2.2: Overview of articles used in the literature review

Author	Year	Purpose	Object analyzed	Textual measure	Prediction model
Ahmad et al.	2016	Stock performance	Financial news	Dictionary	Vector autoregressive (VAR)
Antweiler and Frank	2004	Stock performance	Social media posts	Sentiment (with NB)	Simple regression
Cecchini et al.	2010	Bankruptcy detection	MD&A	Dictionary	SVM
Chen et al.	2014	Stock performance	Social media posts	Dictionary	Simple regression
Dingli and Fournier	2017	Stock performance	Numerical data only	-	CNN
Dyer et al.	2017	Development of 10-K filings	10-K	Multiple measures	-
Farzaneh and Fadlalla	2017	Literature review	-	-	-
Goel S., Uzuner O.	2016	Fraud detection	MD&A	Dictionary	SVM
Hájek and Boháčová	2016	Stock performance	10-K	Dictionary	Multiple models
Hájek and Olej	2013	Bankruptcy detection	10-K	Dictionary	SVM
Hájek et al.	2013	Stock performance	10-K	Dictionary	Neural network and SVR
Henry	2008	Stock performance	Earnings press realeases	Dictionary	Simple regression
Humpherys et al.	2011	Fraud detection	MD&A	Dictionary	Multiple models
Kearney and Liu	2014	Literature review	-	-	-
Khedr et al.	2017	Stock performance	Financial news	Sentiment	K-nearest neighbors
Kraus and Feuerriegel	2017	Stock performance	German corporate announcements	Word embeddings	RNN (LSTM)
Lang and Stice	2015	Development of 10-K filings	10-K	Multiple measures	-
Lehavy et al.	2011	Analyst forecast accuracy	10-K	Fog-index	Simple regression

Table 2.3: Cont. overview of articles used in the literature review

Author	Year	Purpose	Object analyzed	Textual measure	Prediction model
Li	2008	Stock performance	10-K	Fog-index	Simple regression
Li	2010	Stock performance	MD&A	Sentiment (with NB)	Naïve Bayes
Li et al.	2014	Stock performance	Financial news	Dictionary	SVM
Lo et al.	2017	Earnings management	MD&A	Fog-index	Simple regression
Loughran and McDonald	2016	Literature review	-	-	-
Loughran and McDonald	2011	Proposes a finance-specific dictionary	-	-	-
Loughran and McDonald	2014	Proposes new readability measure	-	-	-
Myšková et al.	2018	Stock performance	Financial news	Dictionary	Multiple models
Nardo et al.	2016	Literature review	-	-	-
Pennington et al.	2014	Proposes the GloVe word embedding algorithm	-	-	-
Price et al.	2012	Stock performance	Conference calls	Dictionary	Simple regression
Qiu et al.	2014	Stock performance	10-K	Simple Bag of words	SVM
Rather et al.	2015	Stock performance	Numerical data only	-	RNN
Sohangir et al.	2018	Test classification ability of deep learning models	Social media posts	Word embeddings	Multiple models
Sprenger et al.	2014	Stock performance	Social media posts	Sentiment (with NB)	Simple regression
Tetlock	2007	Stock performance	Financial news	Dictionary	Simple regression
Wisniewski and Yekini	2015	Stock performance	10-K	Dictionary	Simple regression
Wu et al.	2014	Stock performance	Social media posts	Character embeddings	SVM

Chapter 3

Research Design

In this chapter, we present the data sources, methodology, and the theoretical framework used to answer our research hypothesis. The data section describes the data sources and any preprocessing of the data. Hereafter we present our methodology, where we explain how we use a convolutional neural network to predict firm-specific stock performance. Following this is an explanation how we further process the data, ensuring that it is in the correct format for our learning algorithm. Finally, we include a presentation of the theory of neural networks, convolutional neural networks, and how they are trained.

3.1 Datasets

Convolutional neural networks (CNNs) require that the input data is labeled with associated output in order to be trained. The input variables consist of annual reports gathered from a preprocessed dataset published by Loughran and McDonald (2018) and we use stock returns as our associated output. As stock performance, we calculate absolute and abnormal returns on specific stocks using CRSP stock data. To join the two datasets, we use data from Compustat.

3.1.1 Loughran and McDonald

Similar to Hájek and Olej (2013), and Hájek et al. (2013), we use the textual elements of annual reports of publicly traded companies in the US. Additionally, we use annual reports from 2010 to 2017 as the input to the CNN. Loughran and McDonald collected the 10-K filings (annual reports) from the US Securities and Exchange Commission's Electronic Data Gathering, Analysis, and Retrieval system (SEC EDGAR) and McDonald has made this data set available for researchers (McDonald 2018). They process the text by removing XML data format tags which include text formatting and markup tagged elements (e.g. XBRL and HTML tags). Also, they removed tables, excess lines, and any non-character objects. Non-character objects include pictures, other file formats, and characters not included in ISO-8859-1.

The processed data from Loughran and McDonald ranges from 1994 to 2017 and consists of 1,029,938 annual and quarterly reports. They split annual and quarterly forms into one of three categories: regular forms, amended forms, and forms stating a change in fiscal year (McDonald 2018). We only include 10-K filings in this paper as they are the initial reports evaluated by stakeholders and therefore what the market price reactions reflect. By excluding all other reports than the 10-Ks filing, we get a total of 175,965 reports.

3.1.2 CRSP

The CRSP data set is a widely accepted data set used in the academic world for stock information analysis. The data set contains returns including dividend for all traded shares on all stock exchanges in the US.

3.2 Methodology

This section describes the model and methods used to answer our research hypothesis. We first present the proposed prediction model. Next, we describe the overall steps taken and which tools we used to conduct the research.

3.2.1 Prediction Model

Like Sohangir et al. (2018), we use a convolutional neural network and similar to Kraus and Feuerriegel (2017) we use it to predict stock performance. Based on the textual elements of the annual reports, the network is constructed to classify each company in one of five portfolios according to how the model predicts their future stock performance will be.

The convolutional neural network is a learning algorithm that can be used both for supervised and unsupervised learning. We train the network with supervised learning and, subsequently, use it as a model to classify how companies will perform on the stock market. Supervised learning is when an algorithm is trained by feeding it with both input and output data, opposite to unsupervised learning where the algorithm only takes in an input (Bishop 2006). After the training process, the algorithm becomes a model.

3.2.2 Research Structure

The preparation of our data is the first step towards answering our research hypothesis. This process involves cleaning and processing the data. Subsequently we train our convolutional neural network by taking the textual part from a company's annual report as input and its future stock performance, separated into portfolios, as output. Using a trial and error approach, we identify and train a base-case model. Afterward, we attempt to optimize the performance of the base-case model by structuring nine tests by tweaking the settings of the model individually and then merge the best settings in a combined experiment. Based on the experiments, we select the model with the best performance and report its results thus answering our hypothesis. To test the robustness of the results, we additionally control for the well-known Fama-French 5-factor model. This additional control also helps us determine whether we generalize patterns already found in previous studies. The specific methodology of the experiments, how we evaluate the experiments, and the results will be explained in their respective sections.

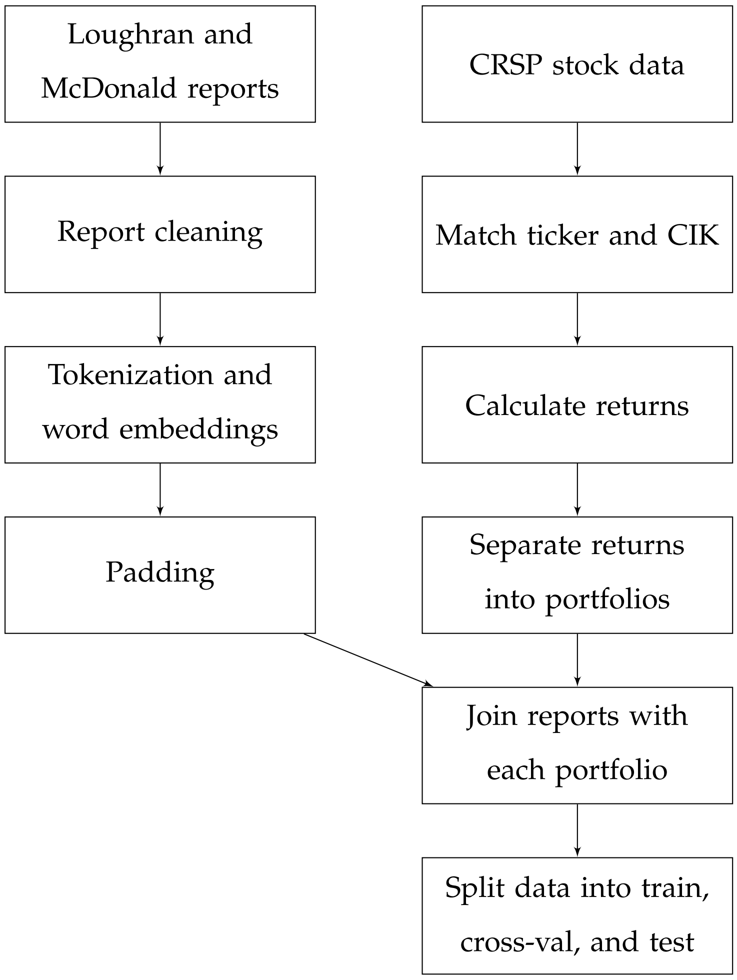
3.2.3 Tools and Software

We structure and train our Convolutional Neural Network (CNN) with the programming language Python using the Keras toolkit with TensorFlow as backend. To do this, we use over 4,000 lines of self-written agile and dynamic code to both clean and process the data, and to further structure and run the learning algorithm. The size of the code is equivalent to 180 pages of text and took more than 800 hours to write. The code is disclosed on the following website: `https://www.dropbox.com/sh/4pgyhjoge48alok/AAC8w2IAu07rvQa04eCG2s3Ja?dl=0`

3.3 Data Processing

In this section, we explain how we process the textual Elements of annual reports and the stock returns. The process is illustrated in figure 3.1.

Figure 3.1: Overview of the data processing

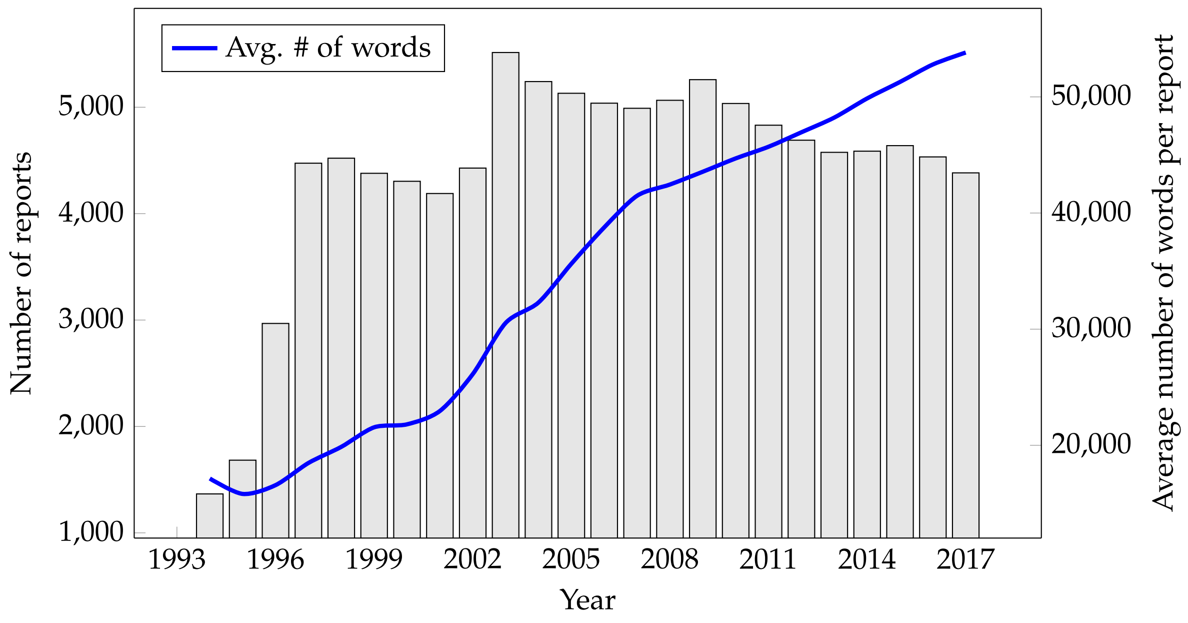

Loughran and McDonald reports
Report cleaning
Tokenization and word embeddings
Padding
CRSP stock data
Match ticker and CIK
Calculate returns
Separate returns into portfolios
Join reports with each portfolio
Split data into train, cross-val, and test

Firstly, we separate all words so that each report is represented by a long list of words. Secondly, we show how we a pre-trained word embedding to replace each word in the list with a vector representation of the given word and, thus, each report is represented by a matrix of quantified word features. Thirdly, we describe the process of padding the text to standardize its format, which is the last step of our text processing. After this, we explain the process of preparing the stock data. This process includes matching the ticker and CIK values, explaining how we calculate the returns, how we assign a portfolio to each company, and how we join the text and the assigned portfolio number together.

3.3.1 Processing the Textual Elements of Annual Reports

From the Loughran and McDonald data set, we excluded all non-10-K filings returning a set of 175,965 reports. Some of the remaining companies are not publicly traded, as they are private companies that the SEC still require to file the 10-K filing (SEC 2013). In order to filter out the private companies, we use the described CRSP data set. In accordance, we assume companies not in the CRSP dataset are not publicly traded companies. In order to validate this assumption, we look at a random sample of 50 reports. No public companies are found in the sample, thereby confirming our assumption. We, therefore, remove all companies that do not exist in the CRSP dataset, returning a total of 105,824 reports ranging from 1994 to 2017. Figure 3.2 shows the development in the number of 10-K filings and the average number of words in each filling in the aforementioned time period.

Figure 3.2: Development in the number of reports and the average number of words per report



Since 2007, a steady increase in the average number of words is noticed, which confirms the findings of Dyer et al. (2017). Furthermore, this tendency is viewed as a positive development for the proposed research model as it proves more difficult for the investors to comprehend the text.

To get a homogeneous input, we choose to narrow down the timespan of the data to 2010-2017. It is desirable that the text in the annual reports be as homogeneous as possible so that the model training becomes as efficient as possible. Historically, new regulatory requirements have been implemented on a continuous basis (Bommarito II and Katz 2017). This means that the regulatory adjustments required by SEC or other regulatory agencies have not previously been clustered in large reforms. These individual new requirements have a large impact on the way businesses report their 10-K filings (e.g. the implementation of SFAS 157 on Fair Value Measurement or Item 1a on business risk) (Dyer, Lang, and Stice-Lawrence 2017). For the input data not to be majorly affected by such a new requirement, we follow Bommarito et al. (2017). They view regulatory references as a proxy for annual report complexity. They find that the regulatory requirements and the lengths of the 10-K filings have changed over the last three decades, however, the number of words and regulatory references per filing has reached a status quo between 2010 and 2011. Thus, we limit our data set to only contain reports from 2010 and onwards, resulting in a total of 37,274 reports.

3.3.2 Cleaning Data

To ensure the quality of the data, we took random samples of 50 reports to look for potential improvements. Based on these samples we made the following corrections to the original data set to remove any elements that could create noise for the learning algorithm:

- Most of the reports had page numberings in different forms (e.g. standard numbers or roman numerals), which in a digital form of the reports appear as a number on a separate line. We remove the empty lines, the page numbers, and any other characters surrounding the page numbers.

- We remove all tabulating characters. Tabulates are often used in the table of contents of the reports and to structure other tables throughout the reports. Loughran and McDonald have removed all of the tables, however, some of the tabulating characters remained.
- We further removed all appearances of the word "PART" in upper case and subsequent numbering. Companies use "PART" as an indicator for the beginning of a new section.
- We remove all HTML tagging used by Loughran and McDonald to indicate the header of a report and any exhibits.
- We remove several words that create noise (e.g. the, for, in, a, etc.)

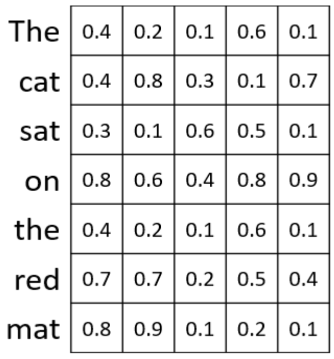
3.3.3 Tokenization and Word Vector Representation

After the above cleaning, we tokenize each word for each report. Tokenizing is the task of separating each element of a sentence into individual items. If we use the following sentence as an example: "The cat sat on the red mat", we would get a list of ["The", "cat", "sat", "on", "the", "red", "mat"] by tokenizing the text. This process is a necessary step towards preparing the text for the correct format, which is required for the convolutional neural network.

At this point, each report has been transformed from one long text document to a long list of the separate words of the reports. As mentioned in the literature review, quantifying the information contained in the text is a difficult task. The standard approach of creating a measure of sentiment (either by a dictionary or machine learning approach) is not well-suited for the learning algorithm in this paper. Instead, we follow the method of Collobert et al. (2011) and Kim et al. (2014) who use multiple word features represented as word-specific vectors. By replacing words with quantifiable features of the word, a machine can process the meaning of the word. Mikolov et al. (2013) present two unsupervised methods to efficiently train word vector representations using either the continuous bag-of-word model (CBOW) or the skip-gram model. The CBOW model tries to predict a center word depending on the surrounding words whereas the skip-gram model tries to predict the surrounding words based on a given center word. From Mikolov's skip-gram model, Pennington et al. (2014) presented GloVe, an improved method to efficiently train word vector representations. We use the GloVe word vector representations in our convolutional neural network. In order for vector representations to be considered good measures for a text, Schnabel et al. (2015) argue that word representation must mirror the linguistic relationship between the words in vector space. The GloVe vector representations have shown to be able to correctly recognize complex word analogies from the words' Euclidean distance (i.e. the distance between vectors). For example, the relationship "a king is to a man as a queen is to a woman" is encoded in the operation $king - man + woman = queen$. Similarly, the operation $Paris - France + Poland = Warsaw$ has shown to be true. These examples indicate that a deeper relationship between the words is understood and encoded in the word embeddings.

For our purpose, we use word vectors that have been pre-trained on 6 billion words from Wikipedia with a vocabulary of the 400,000 most frequent words. Each word is represented by a 300-dimensional vector which has shown to be the optimal dimensions (Pennington, Socher, and Manning 2014). To prepare the input for the CNN, we substitute each word with its vector representation, thereby creating an $n \times 300$-dimensional matrix where n is the number of words extracted from each report. Revisiting the short sentence from earlier and representing each word with a horizontal vector of five variables we get the structure in figure 3.3.

Figure 3.3: Words represented as vectors



The	0.4	0.2	0.1	0.6	0.1
cat	0.4	0.8	0.3	0.1	0.7
sat	0.3	0.1	0.6	0.5	0.1
on	0.8	0.6	0.4	0.8	0.9
the	0.4	0.2	0.1	0.6	0.1
red	0.7	0.7	0.2	0.5	0.4
mat	0.8	0.9	0.1	0.2	0.1

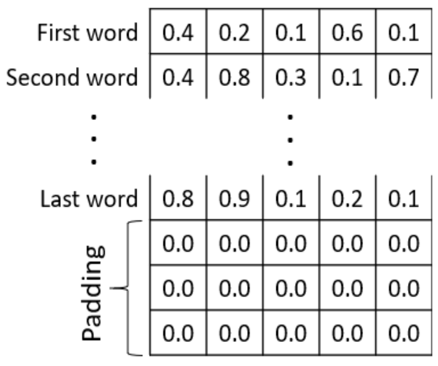
While all the values in the example matrix are arbitrary numbers between 0 and 1, these word vectors 'stacked' in a matrix serve as the input to a convolutional neural network.

3.3.4 Padding

One commonly known challenge in quantitative textual analyses is dealing with differences in sentence length (Collobert et al. 2011). This constitutes a challenge since convolutional neural networks require input to be the same dimensions for each sample. A very popular solution is to pad the input text. Padding is the process of modifying the textual input to fit specific predefined borders/format. In our case, padding works in two ways:

1. If an annual report is shorter than the set maximum number of characters, the padding adds zero vectors until the text fits the specified requirements. Featured in figure 3.4 is an illustration of this process.

Figure 3.4: Padding example

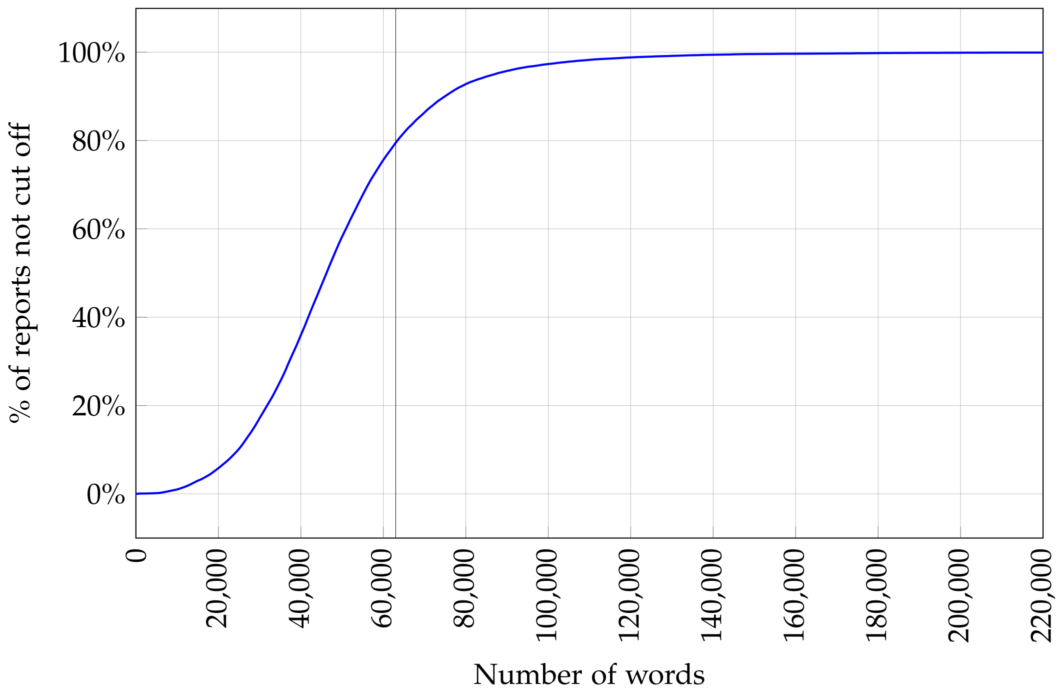


First word	0.4	0.2	0.1	0.6	0.1
Second word	0.4	0.8	0.3	0.1	0.7
⋮			⋮		
Last word	0.8	0.9	0.1	0.2	0.1
Padding	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0

2. If an annual report is longer than the set maximum number of characters, the padding removes the excess text from the report (i.e. if a report contains 100,000 words and the limit is 80,000, the last 20,000 characters are not included as input).

The number of words we extract from each report (i.e. the dimensions of the input) has an impact on the number of parameters the learning algorithm must train. For every extra word included, the learning algorithm gets approximately one extra parameter in our base algorithm. Hence, it is desirable to reduce the number of input words. Extracting too few words, however, can lead to important meaning being excluded from the reports. Figure 3.5 shows a function of how many percentages of our input, which should not have any words removed (y-axis) as a function of how many words we extract (x-axis).

Figure 3.5: Accumulated reports over number of words per report



As a compromise between computational efficiency and input quality, we choose to extract the first 63,000 words of each report. This limits the number of reports that require text to be reduced to 20% and includes 92.2% of *all* words across the entire corpus. As a result, we lose minimal meaning while keeping the number of parameters in our model within a reasonable range.

3.3.5 Matching Ticker and CIK

A direct link cannot be made directly between the company-specific identification number in the annual reports (CIK codes) and the identification numbers in the CRSP dataset (ticker). In an effort to overcome this, we gather data from Compustat, which enables us to match the SEC CIK codes with their respective ticker. The Compustat data also contains a time variable that eliminates the problem of companies which change their ticker.

In order to calculate stock performance, we collect stock prices and returns from CRSP on all exchange-traded stocks in the US from 2010 to 2017. As mentioned earlier, companies that are not included in the CRSP database are excluded. However, companies that have previous records but are not currently trading should also be discarded because they are not a part of the available market. Therefore, we discard annual reports if we do not have any stock data at the time of release of their 10-K filings. This narrows the total number of observations from 37,274 to 29,304. In addition, we conduct an additional manual check of 50 reports that were discarded to make sure no actively traded companies were removed by mistake. The excluded cases consist of companies that have gone private, declared bankrupt, buyouts, and other delisted companies that all still report 10-K filings. Thus, we assess that no companies which are publicly traded were removed.

3.3.6 Calculating Stock Returns

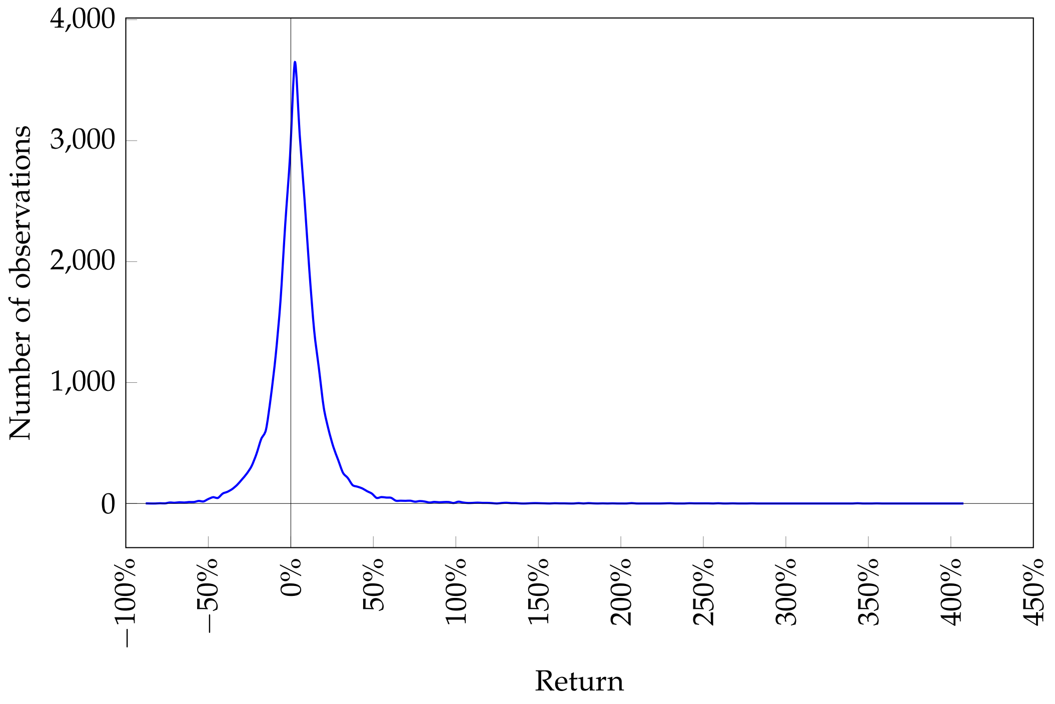
We measure stock performance as the absolute return of a given performance period. Based on daily observations, we calculate the return using equation 3.1:

$$r_{period} = \prod_{t=1}^{N}(1 + r_t) - 1 \tag{3.1}$$

Where r_{period} is the total return for the selected prediction period. We chose 60 days[5] as previous studies have used this time frame to examine the lag of information absorption as pointed out by (Price et al. 2012). If t denotes the day a report is released, we predict performance from $t + 2$ to $t + 62$ days after the release of a report. We choose $t + 2$ so there is ample time for the hard information to be absorbed by the market. One could argue for using $t + 1$ instead, however, in the case of a report being released after trading hours, we would be using the return from t to $t + 1$, which is not desired. In addition, we test for $t + 32$ and $t + 92$ days as well to see if the time frame makes a difference. The distribution of all the 60 days stock returns included in the input is presented in figure 3.6.

[5]equivalent to 41 trading days

Figure 3.6: Distribution of 60 days returns



As expected, the distribution of the actual returns is right-skewed. As the reader may notice, we do not have any observations of –100% return (i.e. company bankruptcies). The CRSP dataset has no clear indication of bankruptcy. We, therefore, exclude reports from companies in our training set who declare bankruptcy within the period we calculate the stock performance. Excluding the samples has no significant effect on the training process and from the discussion in chapter 5, we argue that it has no significant impact on the results.

3.3.7 Separating Returns into Portfolios

As mentioned in the literature review, two general ways of predicting stock performance exist: by absolute stock return or by category. Historically, convolutional neural networks have been used for classification tasks and following this, the model implemented in this paper is trained to classify reports into portfolios based on the company's stock performance. The model will be able to analyze the text of a report and assign it to the correct portfolio from 1 to 5, where we expect the returns to increase with the portfolio numbers. We have chosen five portfolios as a compromise of two conflicting factors.

On one hand, increasing the number of portfolios (e.g. to 10) helps us improve two things: 1) it makes it possible for the model to differentiate performance more finely and thereby better detect the best/worst performers. 2) it makes it easier to test whether or not the best performing companies are 'rewarded' for having higher risks relative to the others (i.e. does the risk of the companies in the portfolios increase with the portfolio number, indicating that we predict risk and not abnormal returns). On the other hand, increasing the number of portfolios punishes the training of the learning algorithm. When training the model, there is no indicator of the magnitude of the degree of a wrong guess. If the actual portfolio for a given observation is portfolio 5, the algorithm is punished equally if it guesses portfolio 4 as if it guesses portfolio 1. As a result, the weights are adjusted equally even though a smaller adjustment would be optimal if the model is very close to the true value as opposed to being far off. Thus, increasing the number of portfolios could lead to the model over-adjusting its weights even though it is recognizing patterns correctly.

The thresholds of the five portfolios (i.e. the values that guide which portfolio number a predicted value should be assigned to) are recalculated on a quarterly basis. This time frame is chosen in order to make sure that a company's performance is compared to companies that also released a 10-K filing in the same period. Thus, we avoid that all reports from a specific quarter with, for instance, good market conditions ending up in one portfolio and vice versa. To calculate the thresholds, we sort all the returns observations in a quarter. We then set the threshold values to the 20th percentile values to ensure that 1/5 of the values end up in each portfolio.

3.3.8 Splitting the Data for Training, Validation and Test

At this point, we a have complete data set of 29,304 reports which each consists of a 63,000 x 300 matrix with a corresponding portfolio classification. As a standard machine learning practice, we split up our dataset into three parts: a training set, a cross-validation set, and a test set (holdout set). The training set is used in the training phase to adjust the weights of the model. The cross-validation set is analyzed to correct the model's settings as to improve the model and ensure that model does not overfit to the training data. As the settings are set based on the results of the cross-validation set, we may also overfit the cross-validation set. We, therefore, use the holdout set to test how the model performs on unfamiliar data.

Typically, the total data set is shuffled randomly and then the splits are made based on the entire dataset. However, shuffling before splitting the data will ignore the chronological order which poses a problem when dealing with time-series data (Kraus and Feuerriegel 2017). For example, if the model had been trained on reports from 2017, where many companies reported a big drop in oil prices, and the model subsequently had to predict a report for an oil-dependent company in 2015, it would have the unfair advantage of being able to look into the future. As a result, the model would perform very well, although that kind of setting would never be possible in the real world. Thus, we aim for a more realistic research setup and choose to order our data in chronological order. From this argument, we use reports released from 2010 Q1 to 2016 Q3 as our training set and the reports from 2017 Q1 through 2017 Q3 are equally split in the cross-validation and test set. We leave out 2016 Q4 as many of the calculated stock performance of the Q4 reports depend on stock changes in 2017 Q1. We further remove 2017 Q4 reports because we do not have stock data from 2018 Q1. Additionally, we want each portfolio in the cross-validation and test set to have an equal number of observations in each portfolio in order to avoid any skewed distributions in the observed data. To do this, we shuffle the 2017 data, split them equally into the five portfolios, and randomly select half from each portfolio to use as cross-validation set and test set, respectively. Throughout the optimization process, we do not test the models on the holdout set and we, therefore, avoid any bias in the optimization process.

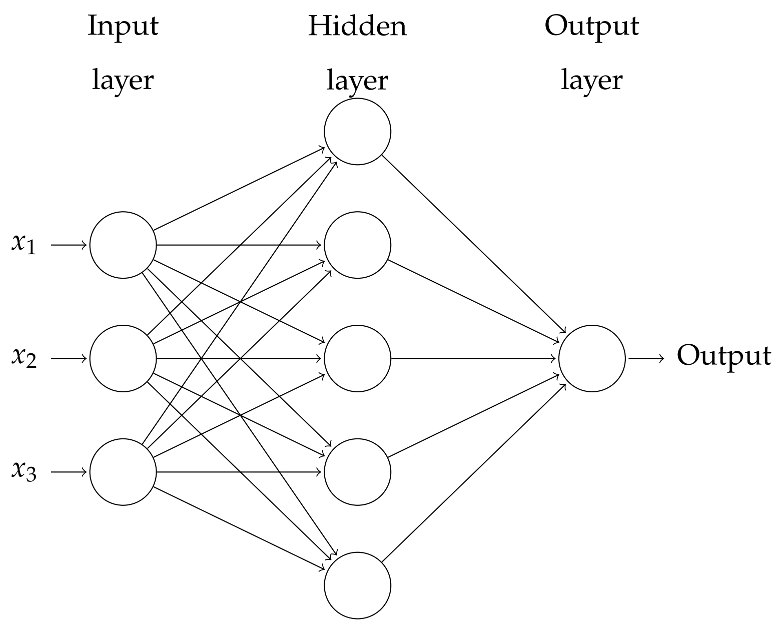
3.4 The Theoretical Foundation

This section first covers the structure of neural network learning algorithms and building on this understanding, we then present the structure of convolutional neural networks. Based on the structures, we present how the networks are trained. We further refer to the appendix A where we have included a more in-depth explanation of neural networks and convolutional neural network. We highly recommend any readers with no or little knowledge of the topic to also read this in order to get a more comprehensive understanding of the different terms and processes used in the field of neural network. The theory refers to Goodfellow et al. (2016) and Bishop's (2006) books on the field.

3.4.1 Neural Networks

Artificial Neural Networks (ANNs) or simply Neural networks (NNs) build on the understanding of the human brain (McCulloch and Pitts 1943) and they have the ability to find complex patterns in data by imitating logical operations (Bishop 2006). NNs consist of neurons which are structured in dense layers. Neurons are information processing units that receive an input, processes the input, and then passes on an output. Neurons are connected to every other neuron in the neighboring layers by connection weights. An illustration of a neural network is shown in figure 3.7, with a flow from left to right.

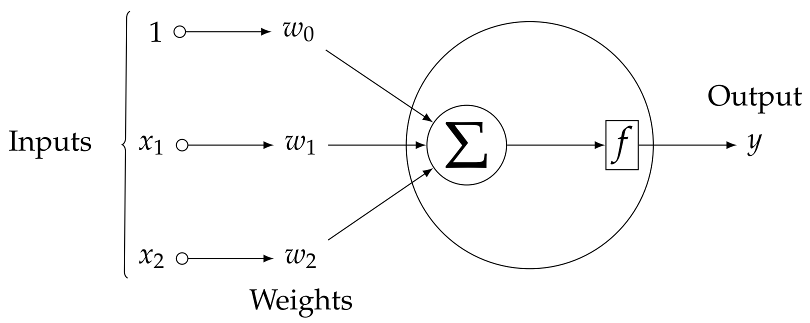
Figure 3.7: Example of a neural network



The circles indicate the neurons and the lines represent the connection weights. The first layer from the left is the input layer followed by a hidden layer and the output layer. A neural network with more than one hidden layer is a deep structured network, wherefrom the term deep learning derives. The neurons in the input layer do not process the input while neurons in the following dense layers contain activation functions. An activation transforms an input (from a previous neuron) into a non-linear output. We use an activation function called Rectified Linear Unit (ReLU) in our learning algorithms. ReLU evaluates a single value and returns the value if it is positive and zero if it is negative.

The output is then passed on to the next layer. All values going into a neuron are first multiplied by the connection weights and then summed together. Figure 3.8 shows how a neuron receives inputs from multiple neurons, transforms the inputs, and outputs a single value.

Figure 3.8: A close-up illustration of a neuron



A constant of one is also inserted into the neuron. The constant multiplied with a bias weight constitutes the bias of the activation function.

The whole process from input to the final output can be written as a function, which takes in an input vector $\boldsymbol{x}$ and a weight tensor $\boldsymbol{W}$ (multidimensional matrix) which holds all weights in the network. The function then outputs $\boldsymbol{Y}$, which in our case is the predicted portfolio.

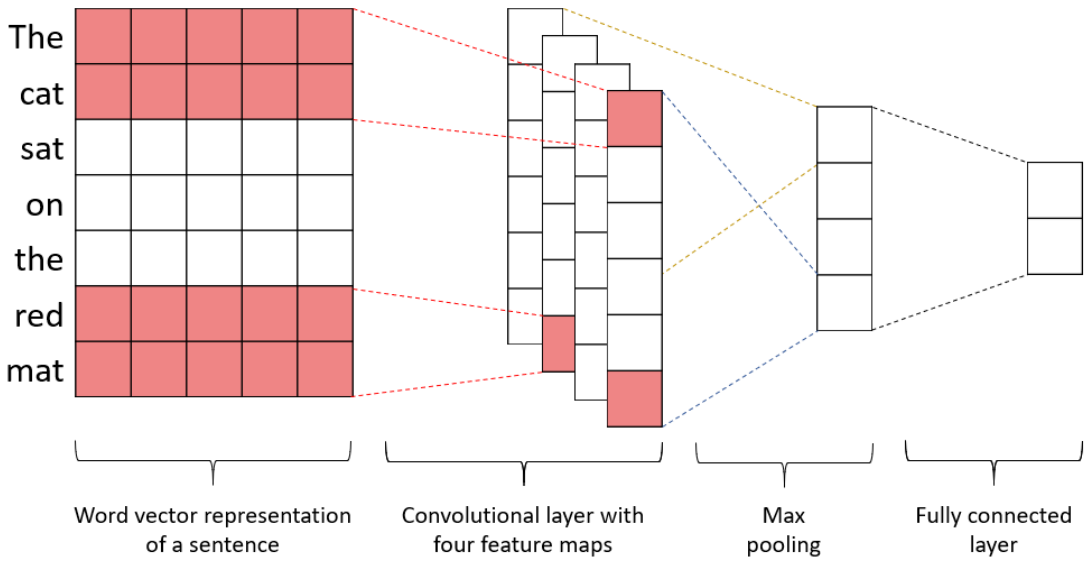
$$f(\boldsymbol{x}, \boldsymbol{W}) = \boldsymbol{Y} \tag{3.2}$$

3.4.2 Convolutional Neural Network

Convolutional neural networks (CNNs) is a special type of neural networks. A CNN can like the simple neural network, be represented by the same function as equation

3.2. A neural network becomes a convolutional network when it has at least one convolutional layer. CNNs have proven to be effective in classifying sequential text inputs with different lengths together with word embeddings (Kim 2014). They get their name from using convolutional layers.

Figure 3.9: A convolutional neural network for sentence classification


The
cat
sat
on
the
red
mat
Word vector representation of a sentence
Convolutional layer with four feature maps
Max pooling
Fully connected layer

A CNN is shown in figure 3.9. The left-most matrix is the textual input. The rows correspond to the input words and the values in the columns are the numerical representations of the words as previously illustrated. The convolutional layer processes the input of the layer by applying a filter which is used to 'scan' the input. The convolutional layer applies the filter with a given receptive area size (i.e. filter size), which iterates through the input matrix from top to bottom. In the shown example the red filter size represents a receptive area size of two. On the first iteration, the network reads the words "The" and "cat". If the filter moves one word each iteration, the second iteration reads the words "cat" and "sat". Thus, the filter may overlap the input (i.e. the word "cat" overlaps the two iterations). A filter creates an output of the previous layer which is called a feature map. A convolutional layer may have several filters. Each filter focuses on different characteristics of the input (e.g. specific word use, the general tone of the text, etc.)[6]. A pooling layer often follows a convolutional layer. Opposite to the convolutional layer, the filter of the pooling layer does not

[6]These patterns are hard to decipher because of the complexity of the model.

overlap to make sure that the feature maps are only 'scanned' once. The pooling layer compresses the input by selecting only the most important information to be used in the next layer of the model. As a result, the computational load becomes much lighter. We use the standard max pooling operation as the pooling function which outputs the max value (i.e. the most important) in the receptive area. A CNN often has several sets of convolutional and pooling layers. The layers of convolution and pooling find and compresses patterns in the input across several words. To combine the structural information with logical operations, convolutional neural networks have one (or more) regular dense layers after the last pooling layer. To go from a multidimensional pooling layer to a dense layer, we use a flattening layer which takes the neurons in the pooling layers (which are in the format of a matrix) and vectorizes them. The convolutional neural network concludes in a dense layer that has as many neurons as the number of outcomes (in our case five outcomes).

3.4.3 Training the Network

A neural network is a learning algorithm prior to and during its training. After the algorithm has been trained, it can be used as a model. We use our convolutional neural network as a classifier as we train it to classify a given annual report into a portfolio based on the expected stock performance. Compared to the simpler linear algorithms (e.g. the linear regression), neural networks require a significantly larger amount of data because the models have significantly more parameters that need to be tuned.

A neural network is trained by optimizing the generalization of the relationship between the $\boldsymbol{x}$ and $\boldsymbol{Y}$ in the function $f(\boldsymbol{x}, \boldsymbol{W}) = \boldsymbol{Y}$. By changing the weights ($\boldsymbol{W}$), the learning algorithm learns to recognize patterns in $\boldsymbol{x}$ that can explain $\boldsymbol{Y}$. Optimizing a network requires two methods: 1) a method to measure how 'badly' the network performs and 2) a function to determine how much the weights should be changed.

The first part is called the loss function and, as the name indicates, it calculates the loss of the network function previously mentioned in equation 3.2. Given we have a sample consisting of a pair of input and actual output, we use the inputs to calculate the predicted output of the function. The loss function then returns a value of the difference between the predicted output and the actual output. High differences in the predicted and the actual output equal a large loss value and vice versa.

The second method uses the loss to determine how much the weights of the network should be changed. This method is called an optimizer which mainly uses the partial derivatives of the calculated loss with respect to each weight in the network in order to adjust the weights. In other words, it uses the marginal error of each weight to adjust that individual weight. Thus, the more a single weight contributed to the overall error, the more it gets adjusted. To efficiently calculate the partial derivatives, we use a method called backpropagation (Rumelhart, Hinton, and Williams 1986), which is thoroughly explained in appendix A.

Changing the weights based on the partial derivatives may cause over-adjustments to the weights and therefore make the network perform worse than prior to the weight update. The weight changes are therefore multiplied by a dampening constant. The constant is called the learning rate as it adjusts the learning from each sample and thus prevents the updates from over-adjusting.

Neural networks may have several million weights and only half as many training samples. A network with many weights relative to the number of training examples has an increased risk of overfitting since each weight can potentially adapt to a single training sample. Dropout and L^2 are two regularization methods which prevent the network's weights to adjust to a single set of input and output. Dropout randomly deactivates neurons in a layer at a given dropout rate, making it impossible for weights to adapt to a single sample. The L^2 or Tikhonov regularization punishes extreme weight values by adding the square of the weights to the loss function. Weights are then exponentially punished as the absolute value of the weights increases.

Depending on the optimizer, the algorithm uses one or more samples before it updates the weights. A network's batch-size determines the number of samples included before each update to the weights is calculated. A batch-size of five means that five reports are run through the training process before the weights are updated. When the learning algorithm trains, it iterates over the training data set, one batch at a time. A learning algorithm often iterates more than once over the entire training set before reaching a local or global optimal point. Moreover, an epoch is defined as one iteration of the entire training data. Multiple epochs are often used to allow the model to fine-tune its parameters. Adding dropout and regularization cannot completely prevent a neural network from overfitting if the algorithm is trained over many epochs. To stop the network from overfitting, we include an early stopping mechanism which stops the training process after a given number of epochs if the model's performance on the cross-validation data set does not improve.

Chapter 4

Experiments and Results

In this chapter, we present the optimization process and the final results. Firstly, we present the training process and settings of the base-case model. Secondly, we attempt to optimize the base-case model by varying four selected settings. Each of the settings are then tested with two new variations while keeping all other parameters from the base-case model constant, resulting in eight new variations of the base-case model. Based on the result of these experiments, we try to combine the settings of the best performing models in a single model. Lastly, we present the results of the best performing model, out of the ten models, and compare it to the base-case model.

4.1 Experiments

4.1.1 Training Process of the Base-Case Algorithm

As suggested by Collobert (2011), our approach to finding the base model has been rather experimental by implementing a trial-and-error approach. Thus, in the process of the finding of a base-case learning algorithm, we had two objectives: 1) to get an accuracy score on the validation data above 20%, 2) to prevent the algorithm from overfitting the training data. In order to check if the model's training process runs as intended, we plot the model's accuracy and loss as a function of the number of epochs run. We plot both the model performance on the training and cross-validation data.

Figure 4.1: Training process accuracy

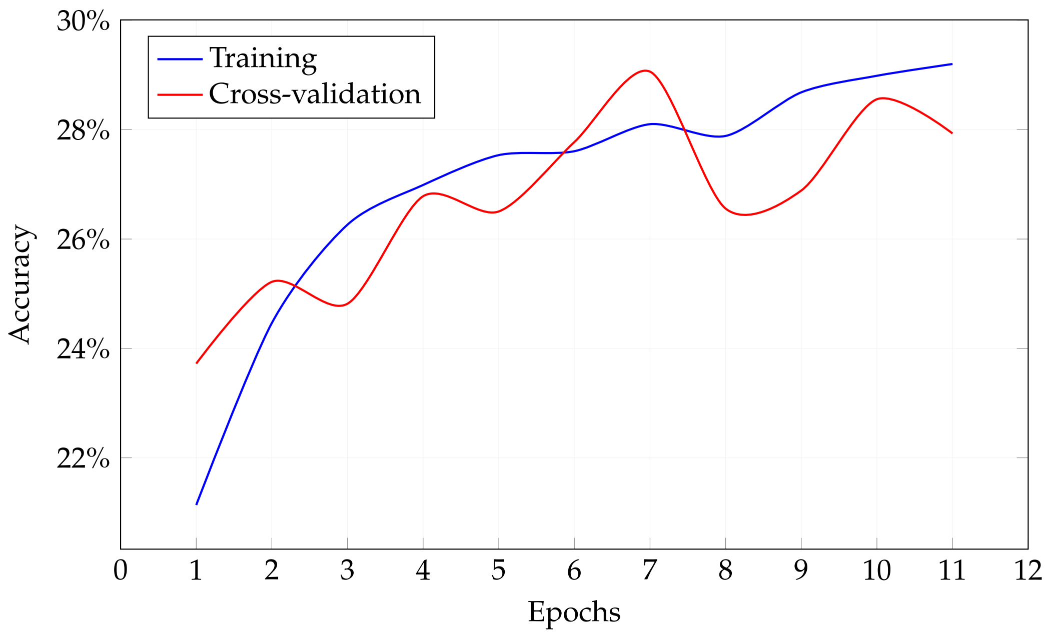
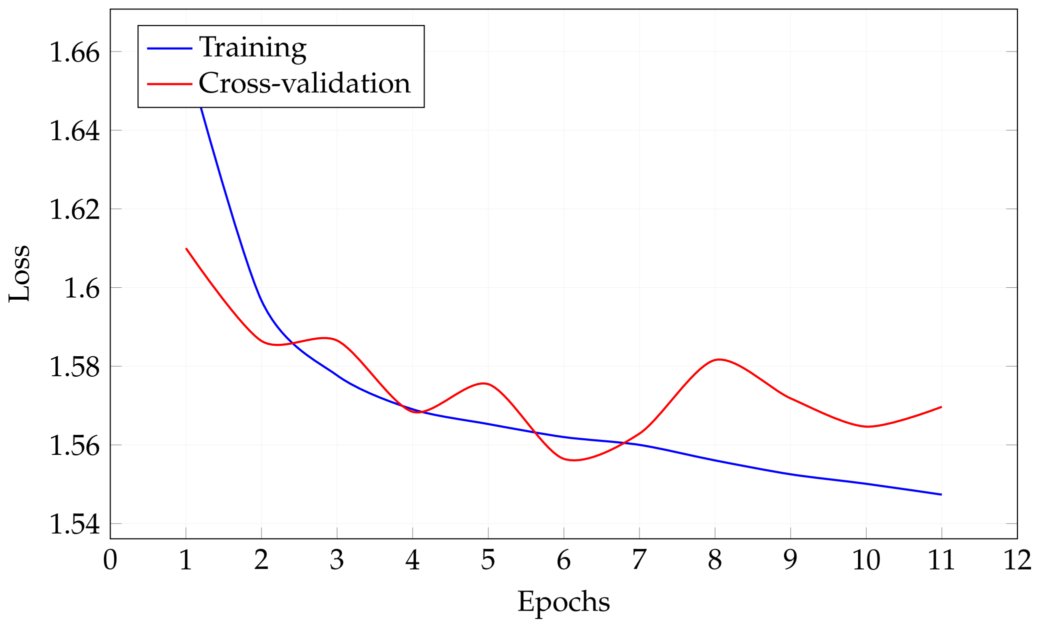


Figure 4.2: Training process loss



In figure 4.1 and 4.2 we present the accuracy score and loss, respectively for the base-case model. The accuracy score on the training and cross-validation data increases and sequentially stagnates. The performance is significantly above 20%, thus, our first objective has been achieved. To make sure we achieved our second objective, we look at both the accuracy and the loss graphs. The validation accuracy hits its maximum at the seventh epoch and then stagnates while the training accuracy continues to increase. The pattern is a classic sign of a learning algorithm which begins to overfit the parameters to the training data. Similar to the findings in the accuracy graph, we see that the loss of the cross-validation set starts to increase at the seventh epoch while the loss of the training data still decreases. In order to avoid this overfitting, we apply an early stopping-function which stops the training if the validations accuracy does not reach a new maximum after five epochs. Thus, stopping early prevented the model from overfitting and the second goal of the training was achieved.

4.1.2 Settings of the Base-Case Model

In this section, we specify the settings of our base-case convolutional neural network based on the training process in the previous section. The base-case model is illustrated in figure 4.3 and has the following settings:

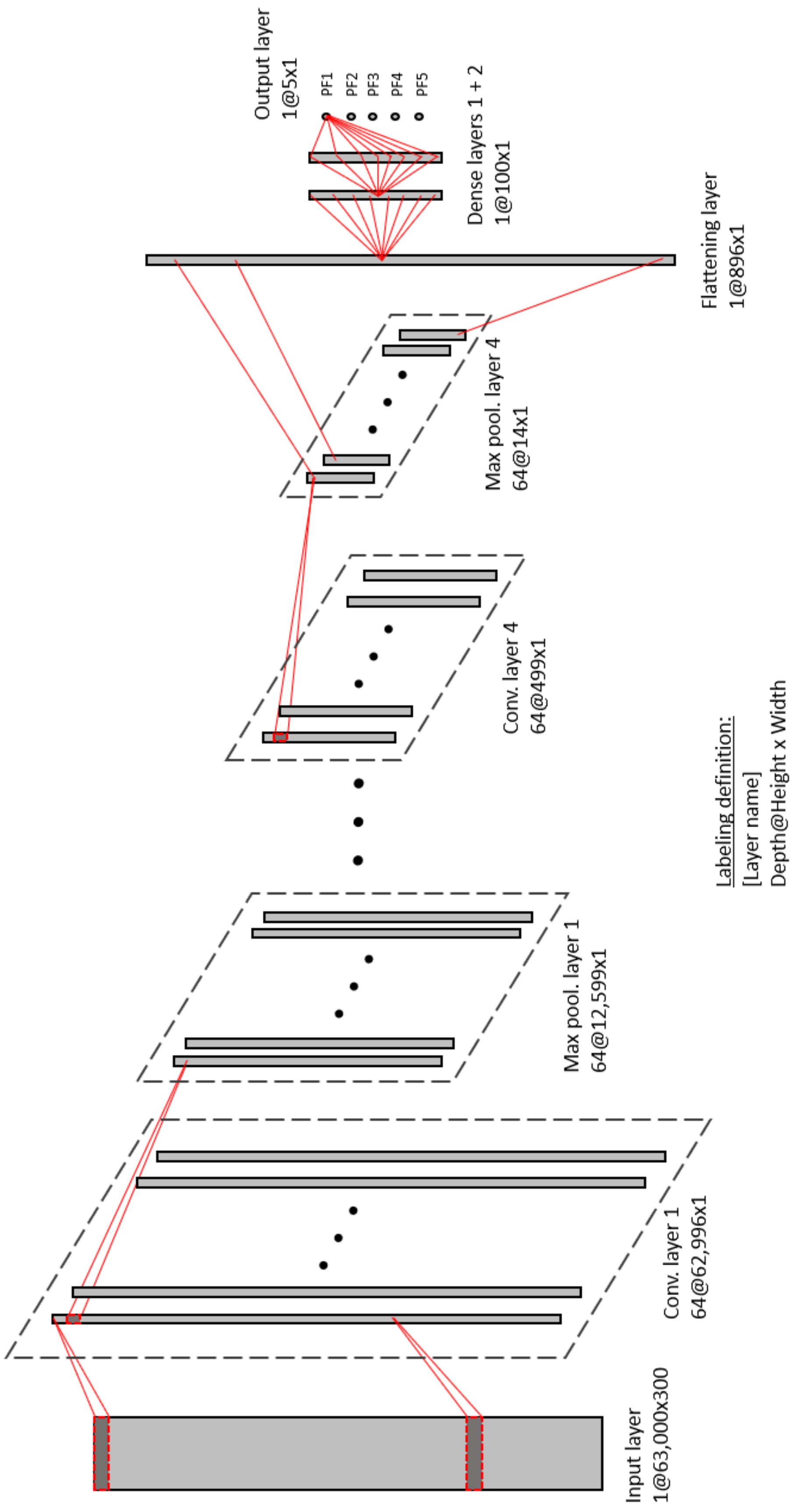
- One input layer
- Four convolutional layers each followed by a max pooling layer.
- 64 filters for each convolutional layer and therefore 64 feature maps. Picking more than 64 feature maps increases the number of trainable parameters while decreasing the number of feature maps potentially limiting the algorithm from finding complex patterns.
- A receptive area of five for both the convolutional and max pooling layers. The convolutional, thus, covers five words per iteration of which the max pooling layer then extrapolates the most important features from the five words.
- To go from a multidimensional layer to a dense layer, we use a flattening layer between the last pooling layer and the first dense layer.
- We end the network with three dense layers where the final dense layer is the output layer. The output layer has five neurons, each one corresponding to one of the portfolios.

- 100 neurons for each of the two hidden dense layers.
- A learning rate of 0.0005 to avoid the weight changes from becoming too large.
- A dropout rate of 15% following each pair of convolutional and max pooling layer, meaning we randomly disable 15% of all neurons from the output of the pooling layers.
- A dropout rate of 50% and a receptive area size of 35 in the last max pooling layer. By heavily compressing the last convolutional layer, we ensure optimized feature filtering from the prior layers, while substantially decreasing the number of trainable parameters.
- A regularization term is added to the loss function to likewise prevent the model from overfitting. We use the L^2 parameter regularization presented in the theory section, where we use a regularization constant of 0.0002. Setting the regularization constant too high, however, would make the model unable to adjust the weights properly and no patterns would then be detected.
- A batch-size of 10. This is the number of reports passed into the network before updating the weights. We choose 10, as we find that changes to the weights, based on individual reports, generate noise rather than generalizations. On the contrary, passing all the training samples in the model for each update would be computationally inefficient.
- An optimizer called ADAM[7]. The optimization algorithm uses the first-order derivative to update the weights. The optimizer used is rather complex, and it has an adaptive learning rate while it reduces the learning variance of the weight changes. (Kingma and Ba 2014). ADAM proves to be much more efficient than other optimizers on batch-sizes under 50.
- Padding with a maximum of 63,000 words, as explained in the methodology.
- Rectified Linear Unit (ReLu) as our activation function. As mentioned in the theory section, ReLu is widely used and has proven effective when used in CNNs for NLP tasks (Kim 2014).

[7]ADAM is a combination of the optimizers Momentum and RMSProp

- An early stopping-mechanism with a patience of five epochs. The model stops training if it sees a stagnation or a reduction in the performance on the cross validation set for five consecutive epochs.
- The categorical cross-entropy loss function which equals the average log-loss of all the outcomes.

Figure 4.3: Illustration of the base-case model



4.1.3 Optimization Experiments

We select the following four model settings for testing: Number of layers, number of feature maps, portfolio return time period, and receptive size of the convolutional and pooling layer. We test two variations of each setting. For a comprehensive overview of the tested variations, see table 4.1:

Table 4.1: Overview of model settings

Parameter	Settings		
	Small-scale	Base-case	Big-scale
Sets of convolutional layers	3	4	5
Number of feature maps	32	64	96
Receptive size	3	5	7
Number of days predicted	30	60	90
Number of dense layers	-	2	-
Neurons in each dense layer	-	100	-
Dropout rate	-	0.15	-
Regularization constant	-	0.0002	-
Optimizer	-	ADAM	-
Batch-size	-	10	-
Padding (words)	-	63,000	-
Activation function	-	ReLu	-
Early stopping (number of epochs)	-	5	-
Loss function	-	Cross-entropy	-

"-" indicates no changes to settings

Testing two different variations for *all* the settings would be the optimal optimization method to find the best model, however, it would require us to make over 20 different models. To keep the number of models within a reasonable range, we have selected the four parameters which create the biggest changes to the overall structure of the model. Additionally, as mentioned earlier, some of the parameters, such as the optimizer and activation function, have been proven to work well with our choice of model and are therefore not tested. The number of words in our padding was chosen based on the analysis of the distribution of words so this parameter is not tested either.

4.1.4 Evaluation of the Experiments

This subsection covers the evaluation method for choosing the settings for the combination model, followed by the results of the experiments (including the combination model). We use accuracy expressed in equation 4.1 as our evaluation measure for the experiments and use the measure to decide which combination of parameters we should use in our combined learning algorithm. We only change the settings if the experimental model performs better than the base-case model. If not, we keep the settings of the base-case model. We finally identify the model with the overall highest accuracy and report the final results for this model.

$$Accuracy = \frac{Number\ of\ true\ positives}{Total\ number\ of\ predictions} \tag{4.1}$$

The measure can be misleading if the distribution of outcomes is not even. By splitting both the training and the validation data equally into each portfolio we get a uniform distribution of observations in each portfolio, eliminating this problem. We, therefore, expect a 20% accuracy if we find no predictive information in the textual elements of the annual reports. In table 4.2 we present the accuracy score for the all the test models.

Table 4.2: Accuracy of the experiments in %

Parameter	Settings		
	Small-scale	Base-case	Big-scale
Sets of convolutional layers	**<u>28.9</u>**	28.0	25.9
Number of feature maps	26.2	**<u>28.0</u>**	26.7
Receptive size	**<u>28.2</u>**	28.0	28.1
Number of days predicted	27.6	**<u>28.0</u>**	25.0

Changing the number of feature maps did not improve the accuracy of the model. As mentioned earlier, the number of feature maps decides how many patterns the model examines. Reducing the number of feature maps can cause the model to contain too few filters to generalize the patterns. Contrarily, increasing the number of feature maps can cause the model to contain too many parameters, and thereby reducing performance. Moreover, changing the number of days predicted did not improve the accuracy either. If we only predict from the second day to 30 days after the release of the annual reports, it is very likely that the investors have not fully incorporated the information into the market. Similarly, increasing the period may allow subsequent information to be released (e.g. a 10-Q report), thus making the information in the annual report less relevant.

Thus, the only two models which outperformed the base-case model were the small-scale versions of the number of layers and receptive size settings. Table 4.3 presents the characteristics of the combined model.

Table 4.3: Settings of the combined model

Parameter	Combined model
Sets of convolutional layers	3
Number of feature maps	64
Receptive size	3
Number of days predicted	60

The combined model achieves an accuracy of 26.9%. Comparing this score to the accuracies presented in table 4.2, it is clear that combining the best performing features of the tested models did not improve the performance. Thus, optimization of the base-case model is not a simple case of combining the top performers. The inherently complex and non-linear structure of the learning algorithm causes unintended interactions that decrease the accuracy of the combination model (Goodfellow, Bengio, and Courville 2016).

Based on the accuracy scores presented in this section, we identify the three-layered model with an accuracy of 28.9% on the validation set as performing the best, out of the ten tested models.

4.2 Results

In this section, we present the performance of the three-layer model and compare it to the base-case model. First, we present the average return for each of the portfolios for each quarter and the weighted return over the three quarters. Acknowledging that these returns are not adjusted for risk, we later control for the well-known Fama-French 5-factor model. We then look at the confusion matrices of the two models to get a deeper understanding of the prediction distributions.

An important thing to note is the fact that we now move from the optimization process, where the cross-validation data is analyzed, to the presentation of the results which is based on the test data. In addition, we ask the reader to recall that we only report on the first three quarters in 2017 since we do not have the complete actual performance of all the reports released in 2017 Q4 and, thus, cannot confirm our predictions for these reports.

4.2.1 Absolute Performance

In table 4.4, we see that the classifications of the three-layer correctly capture the actual weighted returns over the three quarters. By testing the weighted returns in a cross-sectional regression against the portfolios, we find that the three-layer model significantly predicts the company-specific stock performance and we, therefore, accept our research hypothesis. By testing whether the portfolio number can significantly explain the weighted return, as seen in equation 4.2, we find that the resulting coefficient is significant at a 5% confidence level.

$$R_{PF} = c + x \cdot PF \tag{4.2}$$

Where R_{PF} is the portfolio return, c is the constant and x is the coefficient for the portfolio number, PF.

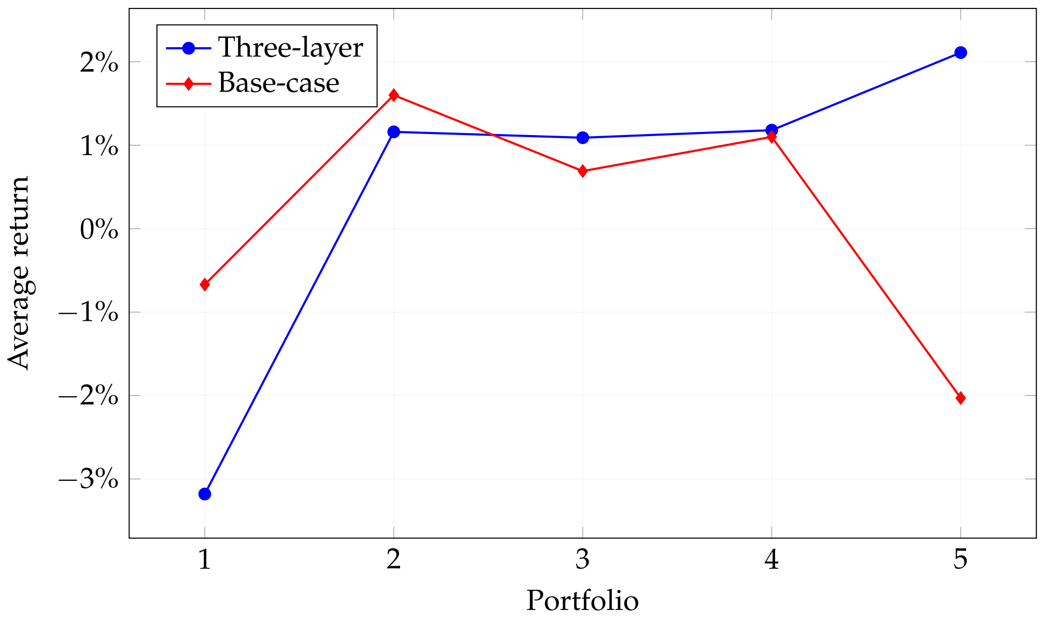
Table 4.4: Average return per portfolio in %

	Base-case				Three-layer			
Portfolio	Q1	Q2	Q3	Weighted return	Q1	Q2	Q3	Weighted return
1	-1.13	-1.49	5.85	-0.67	-3.82	-1.71	3.16	-3.18
2	1.41	1.46	4.07	1.60	0.30	1.38	11.42	1.16
3	0.61	-1.27	3.56	0.69	1.13	-0.90	2.60	1.09
4	0.60	5.93	2.52	1.10	0.38	8.83	3.49	1.18
5	-2.57	-3.88	6.52	-2.03	1.94	-0.45	6.65	2.11

The weighted return is calculated based on the assumption that you would invest the same amount in each company. The weighted returns show what one would generate over the full period from buying the stocks from a given portfolio two days after their 10-K filing dates and holding the given shares for 60 days.

The three-layer model generates an increasing weighted average return over the portfolios from 1 to 5, where portfolios 2 through 4 have near identical returns. Additionally, the model's ability to capture the tails of the weighted returns for portfolio 1 and 5 is excellent as the returns diverge from the rest.

Figure 4.4: Weighted return over portfolio number



Comparing the results to the base-case model, there are no clear similarities in figure 4.4. The returns of the base-case model do not increase with the portfolio number and it predicts neither the tails correctly (portfolio 1 and 5), nor the middle consistently, for either of the quarters or the weighted return. Had you, hypothetically, invested an equal amount in each company in the three periods, the total return for portfolio 5 would have been –2.03%, as shown by the weighted return. Thus, even though the model correctly classified 26.1% of all reports, the misclassified reports seem to disturb the overall performance of the individual portfolios to a point where no economic reasoning can be derived, when looking at the raw returns. In subsection 4.2.3 the confusion matrices will increase the understanding of the results.

Both models seem to struggle with the reports in Q2 and Q3. For both models, the returns vary significantly between the portfolios in a manner which is different from the linear relationship we expect. The behavior of the models could be explained by looking at the distribution of the time in which the 10-K filings are released. 86% of them are released in Q1 whereas the rest are evenly split between Q2 and Q3 (i.e. 7% in each). This means that out of the total attention an investor must give the analysis of the 10-K filings, 86% must be placed in Q1. However, the assumption that investors have unlimited information processing capacity is unrealistic (DellaVigna and Pollet 2009). For example, Dellavigna and Pollet (2009) find that it is possible to generate abnormal returns based on investors' inattention to earnings releases on Fridays supposedly because the investors are distracted by the impending weekend. Thus, it may appear that the amount of information to be processed in Q1 is too large for the investors. As a result, generating returns is easier in Q1. The amount of information to be processed is significantly lower in Q2 and Q3 and the information of the released 10-K filings is thus absorbed in the market faster, reducing the potential for generating returns.

4.2.2 Abnormal Performance

Even though the three-layer model shows significant results, we cannot yet determine whether the model mimics known risk factors. For example, if companies classified in portfolio 1 and 5 have high systematic risk, we would just be modeling the market beta. In order for the patterns of the classification to be unique, we must exclude any effects of known factors. Using the Fama-French 5-factor model, we calculate the company-specific abnormal returns. When adjusting for risk, we determine whether the information captured by the model is unique and thereby the robustness of the model. To adjust the company-specific returns, we subtract the expected returns calculated with the Fama and French 5-factor model (Fama and French 2015). Fama and French show that the size, value, profitability, and investment patterns of a firm explain the company-specific returns. Each factor contains a premium and a company-specific beta value. Fama and French present the factors in a single equation which equals the expected company-specific return, R_{it} at time t:

$$R_{it} = a_i + R_{Ft} + b_i(R_{Mt} - R_{Ft}) + s_i SMB_t + h_i HML_t + r_i RMW_t + c_i CMA_t + e_{it} \quad (4.3)$$

Where the upper-case letters represent the premia generated from being exposed to the given factors and b_i, s_i, h_i, r_i, c_i are the beta values of the premia. $R_{Mt} - R_{Ft}$ is the market premium, where R_{Ft} represents the risk-free rate at time t. The remaining four factors are as follows: SMB (Small-Minus-Big) explains how the company size affects the return, HML (High-Minus-Low) explains how the book-to-market ratio affects the return, RMW (Robust-Minus-Weak) explains how the profitability affect the return, and CMA (Conservative-Minus-Aggressive) explains how the investment patterns of a given company affect the return. Furthermore, a_i or α is the abnormal returns and e_{it} is the idiosyncratic risk which can be diversified. It is a common goal among investors to generate a positive α, as the investors are then able to overperform the benchmark.

We retrieved the daily five factors from Fama and French's website (French 2018) and calculate the betas based on returns one year prior through to the day before the filing. The 5-factor premia are calculated during the same period as the actual return (60 days).

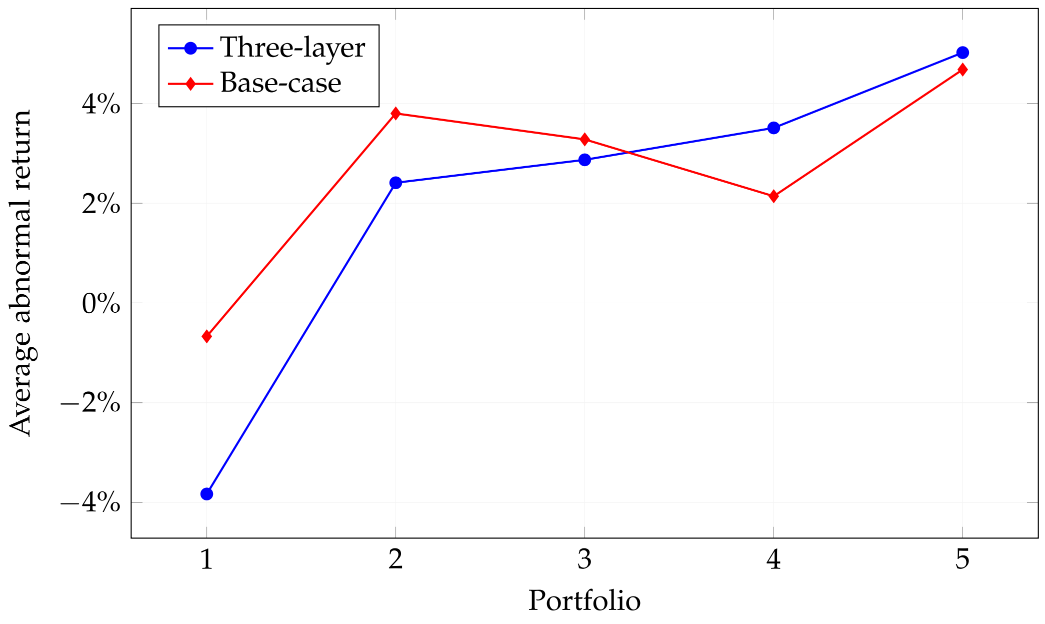
Table 4.5: Average abnormal return per portfolio in %

	Base-case				Three-layer			
Portfolio	Q1	Q2	Q3	Weighted return	Q1	Q2	Q3	Weighted return
1	-0.62	-2.80	0.82	-0.67	-4.05	-3.80	-1.11	-3.83
2	4.38	-0.79	1.04	3.80	2.17	-0.03	7.67	2.41
3	3.99	-1.49	-0.71	3.28	3.67	-2.84	-1.31	2.87
4	3.02	-1.13	-5.50	2.14	4.01	3.47	-2.60	3.51
5	6.15	-7.17	-1.90	4.68	5.89	-1.85	1.03	5.02

Subtracting the expected return and the risk-free rate from the realized return, we get the abnormal returns presented in table 4.5. For the three-layer model, we see that the weighted abnormal returns develop linearly over the portfolios from 1 to 5. Testing the significance of the development in a cross-sectional regression over the portfolios, we see a strong connection between the predicted portfolio number and the actual abnormal returns generated. Our presented three-layer model is, therefore, also significantly able to predict abnormal stock performance. We thus capture information not included in the Fama-French 5-factor model.

When adjusting for risks, it is seen in figure 4.5 that the performance of the base-case model improves too. The weighted return for portfolio 5 has particularly improved from –2.03% to 4.68%. However, the base-case model struggles somewhat with ordering portfolio 2 to 4 correctly.

Figure 4.5: Weighted abnormal return over portfolio number



4.2.3 Confusion Matrices

In order to get a better understanding of the above results, we take a deeper look at how the model classified the reports in the different portfolios. To do this, we construct the confusion matrices for the two models. The distribution of correct and incorrect predictions can aid us in seeing more clearly why the models performed the way they did. We perform a chi-square test of independence to determine if the classification distributions of the models are significantly different from a random classification distribution. We further use the binomial distribution test to see if the accuracy of each predicted portfolio is significantly greater from the benchmark of 20%.

Table 4.6: Confusion matrix for the three-layer model

		Actual					
		1	2	3	4	5	Total count
Predicted	1	**<u>204</u>**	98	85	53	132	572 (31%)
	2	16	**<u>35</u>**	30	29	23	133 (7%)
	3	29	98	**<u>107</u>**	114	55	403 (22%)
	4	10	16	21	**<u>25</u>**	12	84 (5%)
	5	109	121	124	147	**<u>147</u>**	648 (35%)
		368 (20%)	368 (20%)	367 (20%)	368 (20%)	369 (20%)	1,840 (100%)

The confusion matrix for the three-layer model of the test set is presented in table 4.6. Focusing on the predictions (rows), we see that the diagonal (true positive) is the *highest* for all portfolios, except for the third. This means that except for portfolio 3, the most frequently guessed portfolios are the correct ones. We also see that the majority of the reports are classified in either portfolio 1, 3 or 5. This would indicate that the three-layer model has trained itself to recognize three different types of patterns: one for bad performers, one for medium performers, and one for good performers. The performance is excellent if we want to implement the results in a long-short investment strategy. As Myskova et al. (2018, p.195) argues: "it is usually more important to correctly classify firms with a high risk (class 1), rather than those with a low risk (class 0)." In other words, we would rather be able to spot the winners and losers more confidently, as compared to being more confident around the average performing companies. The distribution of predictions helps explain how the model performed very well when adjusting for risk factors.

Table 4.7: Precision and sensitivity of the three-layer model in %

		Precision		
		Exact match	± 1 portfolio	Sensitivity
Predicted	1	35.7**	79.2**	55.4**
	2	26.3*	60.9	9.5
	3	26.6**	79.2**	29.2**
	4	29.8*	69.0*	6.8
	5	22.7*	68.1**	39.8**
		28.2**		28.2**

* = Significantly larger than 20% ($p < 0.05$)
** = Significantly larger than 20% ($p < 0.01$)

In table 4.7 the precision and sensitivity are presented for the three-layer model. Precision is equivalent to a portfolio-wise accuracy, which describes how many percents of the reports that we classify in a portfolio, that are classified correctly. We significantly outperform the benchmark of 20% in all portfolios with the three-layer model, with a total accuracy of 28.2%. Notably, we correctly predict 35.7% of portfolio 1 and 26.6% of portfolio 3. Recognizing that missing the correct portfolio by only one is more satisfying than by e.g. four, we look at the collective precision for ±1 portfolio of the actual portfolio[8]. Here we see an impressive 79.2% precision for portfolio 1 and 3 as well as 68.1% for portfolio 5. This indicates that when we classify a specific portfolio, we are significantly closer to the correct portfolio than randomly guessing. Except for the ±1 precision for portfolio 2, these results are significant beyond the 5% level as tested with the binomial distribution.

Sensitivity measures how many percents of the actually observed instances of a portfolio we correctly classify. Again, portfolio 1, 3, and 5 appear to lie significantly above the benchmark, whereas portfolio 2 and 4 are below. The sensitivities of the portfolios 1, 3, and 5 added together is far from the expected 60% because of the uneven distribution of predictions. We do not see it as a weakness of the model because, as mentioned before, we would rather have a high precision than have a high sensitivity.

[8]The tails are adjusted by a factor of 1.5 to be comparable to the other portfolios

Table 4.8: Confusion matrix for the base-case model

		Actual					
		1	2	3	4	5	Total count
Predicted	1	**248**	154	126	120	208	856 (47%)
	2	36	**57**	57	60	56	266 (14%)
	3	49	112	**131**	138	70	500 (27%)
	4	18	29	32	**34**	24	137 (7%)
	5	17	16	21	16	**11**	81 (4%)
		368 (20%)	368 (20%)	367 (20%)	368 (20%)	369 (20%)	1,840 (100%)

Table 4.8 shows the confusion matrix of the base-case model. If we compare the matrix with the three-layer's confusion matrix, we see that the base-case classifies significantly more reports in the low portfolio numbers than the three-layer model. Only 11% of the reports are predicted in portfolio 4 and 5. We see almost half of the predictions in portfolio 1.

Table 4.9: Precision and sensitivity of base model in %

		Precision		
		Exact match	± 1 portfolio	Sensitivity
Predicted	1	29.0**	70.4**	67.4**
	2	21.4	56.4	15.5
	3	26.2**	76.2**	35.7**
	4	24.8	65.7	9.2
	5	13.6	50.0	3.0
		26.1**		26.1**

* = Significantly larger than 20% ($p < 0.05$)
** = Significantly larger than 20% ($p < 0.01$)

Table 4.9 shows the precision and sensitivity of the base-case model. Here, the model only achieves an accuracy overall of 26.1%, as compared to 28.2% of the three-layer model. We also see that the precision of the base-case in portfolio 5 is 13.6%. The 13.6% precision explains the low returns of portfolio 5 in the absolute performance of the base-case. Only the precision and sensitivity for portfolio 1 and 3 of the base-case are significantly greater than the benchmark of 20%. Thus, when a report is not classified in one of those portfolios, we cannot confirm that it is any better than the benchmark of 20%. When comparing the precision scores to the three-layer, we see that all the portfolios perform worse in the base-case model. We can, therefore, conclude that the three-layer model is superior to the base-case model in all aspects.

Chapter 5

Discussion

In this chapter, we identify and discuss five elements of the research approach which influence the results. Firstly, we discuss possible limitations of the model as a result of the chosen time frame of the training data. Secondly, we discuss the impact of removing the companies which go bankrupt during the 60 days prediction period. Thirdly, we discuss the effect of using pre-trained word embeddings. Fourthly, we debate how the learning algorithm can be perceived as a 'black box' and how this affects the optimization process. Finally, we discuss the effect of other potential factors not explained by the Fama-French 5-factor model.

5.1 Predictive Power over Time

To achieve a homogeneous sample of annual reports, we use reports from 2010 to 2017. The reason for this is to make the generalization of patterns easier for the learning algorithm than in the scenario of varying report contents. Because of the changing regulatory requirements, we, therefore, cannot ensure that the model's results will be significant in the future. Examining the leading S&P500 Stock Index (*S&P500 Stock Index* 2018), we also see that the market has undergone a long positive trend since mid-2010. As a result, the model is trained in a bullish market and it is thus difficult to determine the model's performance in different market states. Consequently, we view the limited training time period as a limitation to how well the model can perform in the future. To overcome this limitation, we propose to train the model continuously, which would enable the model to adapt to the regulatory changes and different market conditions.

From the literature review, we see a trend in the research moving towards using advanced machine learning techniques to analyze elements from both corporate information and financial news. Given this development, we cannot deny the possibility that the market players adapt to the new findings. When a sufficient amount of investors begin to trade based on new information to gain abnormal returns, they also even out the abnormality and thus remove the opportunity to generate abnormal returns in the future. On the other hand, as mentioned in the literature review, one of the challenges with machine learning approaches is that they are hard to replicate due to random initialization. Our model is trained using a learning algorithm where the weights are randomly initialized. Therefore, building an exact replication of our model is very difficult. As a result, the effect of others replicating the model is somewhat mitigated.

5.2 Bankruptcies

In the dataset used to train and test the model, we exclude companies that go bankrupt within the second to 62 days after the release of their annual reports. As mentioned before, these companies could not be included because there was no clear indicator for bankruptcy in the stock data from CRSP. Excluding bankruptcies would in many cases be a significant issue for testing the efficient market hypothesis.

In 2017, 70 companies went bankrupt of which 31 are placed in our test data. Five out of these 31 companies went bankrupt within 62 days after the release of their annual report (BankruptcyData 2017). The remaining 26 companies went bankrupt after the 62 days prediction window and are thus considered as any other company during the period we predict. As a result, classifying these 26 reports correctly has the same importance as predicting companies that do not go bankrupt later in the year.

Since the five company filings were not included in the original test set, we manually process their reports and run them through the three-layer model. We want the model to classify them in portfolio 1, in order to reflect their bad performance. Indeed, three of the five reports are correctly classified in portfolio 1 while the model classifies the last two in portfolio 5. Thus, a majority of the companies which go bankrupt during our prediction period are classified correctly. Consequently, it may seem counterintuitive that the model predicts the remaining companies in portfolio 5. However, the model may have recognized great financial distress which either concludes in bankruptcy or results in a turnaround generating potentially high stock returns. Seeing that only five out of the 1840 companies in the test dataset go bankrupt and that only two are misclassified, we show that not including the companies which go bankrupt does not have any negative effects on the results.

5.3 Trainable Word Embeddings

We replace words with their vector representation using the GloVe word embedding trained on Wikipedia. Similar to the earlier discussed dictionaries, the quality of the word embeddings depends on the context in which they have been trained (Huang et al. 2012). For example, word embeddings trained with texts from physics and chemistry will probably be ill-suited as word representations for psychology and philosophy texts. Kim (2014) shows that CNNs improve their predictive capabilities by letting the word embeddings be trainable. Training the embeddings while also training the model lets the embedding become more context-specific, which in the case of standard dictionary approaches showed improved results. This also allows the model to be able to generate word embeddings for words not originally included in the embedding vocabulary. Thus, letting our word embeddings be trainable could potentially provide improved results. However, due to computational limits, this by far exceeds what is possible with our setup. Currently, we are able to train a maximum of approximately 10 million parameters whereas the number of trainable parameters, including trainable word embeddings, would exceed 750 million parameters. Kraus and Feuerriegel (2017) find that implementing what they call transfer learning improves their predictive accuracy. Similar to the Loughran and McDonald finance-specific dictionary, Kraus and Feuerriegel train their own finance-specific word embeddings with the text from US 8-K filings and subsequently use them to analyze the German equivalent of 8-K reports. Unfortunately, they did not make the finance-specific embeddings publicly available for us to implement.

5.4 Dealing with a Black Box

From the experiments and the tested combined model, we see that combining the best performing settings did not result in an enhanced performance. The complexity of the convolutional neural network comes at the cost of transparency making it very difficult to measure what effect changing the settings has at a deeper level. The non-linearity, therefore, opens for further improvement of the identified model which may recognize other patterns not caught by our model. Thus, to fully optimize our model, we should experiment with all possible combinations of settings, including those we keep static in our experiments.

Methods for unfolding and visualizing CNNs in natural language processing has received little attention from researchers (J. Li et al. 2015). However, highlighting dominant text areas in the input can be done by following Östling and Grignonyte's (2017) approach. Highlighting the text areas which have the greatest influence on the final classification may be useful to get a better understanding of which patterns the model notices in different reports. The challenge, however, still lies with interpreting the text areas which the model notices. Some may be obvious, some may be difficult to make sense of, some may have no logical meaning, and yet some may even be misleading. Thus, these tools also have limitations.

5.5 Other Factors

As initially stated, it takes time for investors to incorporate the soft information from the annual reports in the stock price. Our results not only confirm this statement, but they also offer an advantage for potential investors to overcome the delay and, therefore, gain abnormal returns. However, we still acknowledge the fact, that other factors could explain the results. For example, Carhart (1997) shows that the variable *momentum* has explanatory power of the performance of mutual funds. Additionally, Asness et al. (2000) find that industry measures can explain company-specific stock returns. Controlling our results for momentum and industry factors would improve the robustness of the results, however, doing so is outside the scope of this paper. Additionally, we cannot exclude the option that the model significance derives from unknown phenomena not explainable with current known factors.

Chapter 6

Conclusion

We significantly predict stock performance from analyzing textual elements of annual reports in the US using a convolutional neural network. We present a three-layered convolutional neural network which on average and portfolio-wise significantly predicts the company-specific stock performance. By controlling for the Fama-French 5-factor model, we show that the model captures abnormal returns over the portfolios and that the returns steadily increase with the portfolio numbers which is the desired performance. Thus, we confirm our research hypothesis that company-specific performance is predictably with the proposed methodology.

However, some limitations apply to our findings. Firstly, the model has been trained in a bullish market. As a result, we do not know how the model performs in different market states. Secondly, because of the low transparency of the model, we cannot determine what specific patterns the model has found. Thus, other factors than the tested Fama-French model may describe parts of the results found by the model. However, if the returns are not reflected through the already established factors they may, indeed, arise from currently unknown patterns.

6.1 Contributions and Implications

The results found in this paper produce several implications. Firstly, we add to the existing research stating that the 10-K filings contain relevant soft information which the investors do not effectively capture. As mentioned in the literature review, Li

(2008) was the first to show that the linguistic features of the 10-K filing have predictive power of company-specific performance which this paper supports. Based on the absolute returns found by our model, we confirm that the information in the reports is relevant for performance evaluation purposes. In addition, we find significant abnormal returns, thus indicating that the information in the annual reports is not currently being fully utilized by the investors.

Secondly, we contribute to the validity of using natural language processing with deep learning models to predict stock performance. Although some of the model transparency is lost using the deep learning models, we find robust and significant findings similarly to Rather et al. (2015) and Kraus and Feuerriegel (2017). Seeing that better and more advanced model designs are continuously developed, we believe that there is an immense potential for deep learning models in accounting and finance research.

Thirdly, we provide an evaluation tool for the stakeholders of a company. Investors can benefit from the model presented in this paper by improving their capital allocation. The respective companies can use it as a tool to gauge how the financial markets are going to react to their 10-K filing. The regulatory entities can use it to investigate whether new legislations will improve the quality of the information in the 10-K filing. Thus, the model also serves as a practical tool to help improve the utilization of the 10-K filings for the company's stakeholders.

Lastly, our findings point to a semi-efficient version of the efficient market hypothesis. Although the model is evaluated on a subset of the entire market, our results indicate that the market is less than efficient during the second to 62 days after the release of the 10-K filing. As argued previously, one explanation to this could be that the assumption that investors have unlimited information processing capacity is unrealistic. In accordance with Dellavigna and Pollet (2009) and Engelberg (2008), our results support the argument of limited capacity, while also giving an effective solution to benefit from the limitation and thereby gain abnormal returns.

6.2 Future Research

In the optimization section we tested how varying the number of days the model predicts changed the model's performance. Like Price et al. (2012), we found 60 days to be the optimal prediction period, possibly, because of the time it takes for the market to incorporate all the information of a 10-K filing. We use reports from 2010 to 2017, however, as discussed in section 5.1, we cannot guarantee that the model performs at the same level when the time frame is changed. Increasing the overall time frame tests the model in two ways: 1) how it reacts to new legislation and 2) how it reacts to changing market conditions (i.e. bullish or bearish markets). Thus, we encourage others to thoroughly test how the model performs over a longer time frame.

Because of the nonlinear relationship between the best performing settings found in the optimization process, we also find it relevant for others to test a more exhaustive number of settings. Seeing that combining the best individually performing settings did not improve the performance, a combinatorial approach can be useful in finding the ultimately best performing model. Although taking significantly more time to complete, it yields interesting results. Firstly, the tests will reveal how well the model can ultimately perform. Secondly, and perhaps most importantly, it may help map out how the settings affect what patterns the model finds. In this paper, for example, we find significantly different prediction patterns of the base-case model and the three-layer model as a result of only changing the number of layers in the network. Thus, the combinatorial approach may improve the understanding of the black box by enabling the user to get a thorough overview of what patterns different settings create.

Bibliography

Ahmad, K. et al. (2016). “Media-expressed negative tone and firm-level stock returns”. In: *Journal of Corporate Finance* 37, pp. 152–172.

Amani, Farzaneh A. and Adam M. Fadlalla (2017). “Data mining applications in accounting: A review of the literature and organizing framework”. In: *International Journal of Accounting Information Systems* 24, pp. 32–58.

Antweiler, W. and M.Z. Frank (2004). “Is all that talk just noise? The information content of Internet stock message boards”. In: *Journal of Finance* 59, pp. 1259–1294.

Ball, Ray and Philip Brown (1968). “An Empirical Evaluation of Accounting Income Numbers”. In: *Journal of Accounting Research* 6.2, pp. 159–178. ISSN: 00218456, 1475679X. URL: http://www.jstor.org/stable/2490232.

BankruptcyData (2017). *Bankruptcy Data*. URL: http://www.bankruptcydata.com/ (visited on 04/21/2018).

Bernard, Vl and Jk Thomas (1989). “Post-Earnings-Announcement Drift: Delayed Price Response or Risk Premium?” In: *Journal of Accounting Research* 27, pp. 1–36. URL: https://EconPapers.repec.org/RePEc:bla:joares:v:27:y:1989:i::p:1-36.

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc. ISBN: 0387310738.

Bommarito II, Michael J. and Daniel Martin Katz (2017). “Measuring and Modeling the U.S. Regulatory Ecosystem”. In: *Journal of Statistical Physics* 168.5, pp. 1125–1135. URL: https://doi.org/10.1007/s10955-017-1846-3.

Carhart, Mark M. (1997). "On Persistence in Mutual Fund Performance". In: *The Journal of Finance* 52.1, pp. 57–82. ISSN: 00221082, 15406261. URL: http://www.jstor.org/stable/2329556.

Cecchini, Mark et al. (2010). "Making words work: Using financial text as a predictor of financial events". In: *Decision Support Systems* 50, pp. 164–175.

Chen, H. et al. (2014). "Wisdom of crowds: The value of stock opinions transmitted through social media". In: *Review of Financial Studies* 27, pp. 1367–1403.

Collobert, Ronan et al. (2011). "Natural Language Processing (almost) from Scratch". In: 12, pp. 2493–2537.

Colm, Kearney and Liu Sha (2014). "Textual sentiment in finance: A survey of methods and models". In: *International Review of Financial Analysis* 33, pp. 171–185.

DellaVigna, Stefano and Joshua M. Pollet (2009). "Investor Inattention and Friday Earnings Announcements". In: *The Journal of Finance* 64.2, pp. 709–749.

Ding, Xiao et al. (2015). *Deep Learning for Event-driven Stock Prediction*. Buenos Aires, Argentina. URL: http://dl.acm.org/citation.cfm?id=2832415.2832572.

Dingli, A. and K.S. Fournier (2017). "Financial time series forecasting - a deep learning approach". In: *International Journal of Machine Learning and Computing* 7, pp. 118–122.

Dyer, T., M. Lang, and L. Stice-Lawrence (2017). "The evolution of 10-K textual disclosure: Evidence from Latent Dirichlet Allocation". In: *Journal of Accounting and Economics* 64, pp. 221–245.

Engelberg, Joseph (2008). *Costly Information Processing: Evidence from Earnings Announcements*. URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1107998.

Fama, Eugene F. and Kenneth R. French (2015). "A five-factor asset pricing model". In: *Journal of Financial Economics* 116.1, pp. 1–22. URL: https://EconPapers.repec.org/RePEc:eee:jfinec:v:116:y:2015:i:1:p:1-22.

Fortuny, Enric Junqué de et al. (2014). "Evaluating and understanding text-based stock price prediction models". In: *Information Processing and Management* 50, pp. 426–441.

French, Kenneth R. (2018). *Current Research Returns*. URL: http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html (visited on 04/12/2018).

Goel, S. and O. Uzuner (2016). "Do Sentiments Matter in Fraud Detection? Estimating Semantic Orientation of Annual Reports". In: *International Journal of Intelligent Systems in Accounting and Finance* 23, pp. 215–239.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press.

Hájek, P. and J. Boháčová (2016). "Predicting abnormal bank stock returns using textual analysis of annual reports – A neural network approach". In: 629, pp. 67–78.

Hájek, P. and V. Olej (2013). "Evaluating sentiment in annual reports for financial distress prediction using neural networks and support vector machines". In: *Communications in Computer and Information Science* 384, pp. 1–10.

Hájek, P., V. Olej, and R. Myšková (2013). "Forecasting stock prices using sentiment information in annual reports - A neural network and support vector regression approach". In: *WSEAS Transactions on Business and Economics* 10, pp. 293–305.

Haykin, Simon (1999). *Neural Networks - A comprehensive foundation*. 1st ed. Hamilton, Ontario, Canada: Pearson.

Henry, Elaine (2010). "Are Investors Influenced By How Earnings Press Releases Are Written?" In: *International Journal of Business Communication* 45, pp. 363–407.

Huang, Eric H. et al. (2012). "Improving Word Representations via Global Context and Multiple Word Prototypes". In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*. ACL '12. Jeju Island, Korea: Association for Computational Linguistics, pp. 873–882. URL: `http://dl.acm.org/citation.cfm?id=2390524.2390645`.

Humpherys, Sean L. et al. (2011). "Identification of fraudulent financial statements using linguistic credibility analysis". In: *Decision Support Systems* 50, pp. 585–594.

Khedr A.E. Salama S.E., Yaseen N. (2017). "Predicting stock market behavior using data mining technique and news sentiment analysis". In: 9, pp. 22–30.

Kim, Yoon (2014). "Convolutional Neural Networks for Sentence Classification". In: *CoRR* abs/1408.5882. URL: `http://arxiv.org/abs/1408.5882`.

Kingma, Diederik P. and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980. URL: `http://arxiv.org/abs/1412.6980`.

Kraus, Mathias and Stefan Feuerriegel (2017). “Decision support from financial disclosures with deep neural networks and transfer learning”. In: *Decision Support Systems* 104, pp. 38–48.

Lang, M. and L. Stice-Lawrence (2015). “Textual analysis and international financial reporting: Large sample evidence”. In: *Journal of Accounting and Economics* 60, pp. 110–135.

LeCun, Yann et al. (1999). “Object Recognition with Gradient-Based Learning”. In: *Shape, Contour and Grouping in Computer Vision*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 319–345. URL: `https://doi.org/10.1007/3-540-46805-6_19`.

Lehavy, Reuven, Feng Li, and Kenneth Merkley (2011). “The Effect of Annual Report Readability on Analyst Following and the Properties of Their Earnings Forecasts”. In: *The Accounting Review* 86, pp. 1087–1115.

Li (2008). “Annual report readability, current earnings, and earnings persistence”. In: *Journal of Accounting and Economics* 45, pp. 221–247.

Li, Feng (2010). “The Information Content of Forward-Looking Statements in Corporate Filings — A Naïve Bayesian Machine Learning Approach”. In: *Journal of Accounting Research* 48, pp. 1049–1102.

Li, Jiwei et al. (2015). “Visualizing and Understanding Neural Models in NLP”. In: *CoRR* abs/1506.01066. eprint: `1506.01066`. URL: `http://arxiv.org/abs/1506.01066`.

Li, X. et al. (2014). “News impact on stock price return via sentiment analysis”. In: *Knowledge-Based Systems* 69, pp. 14–23.

Lo, Kin, Felipe Ramos, and Rafael Rogo (2017). “Earnings management and annual report readability”. In: *Journal of Accounting and Economics* 63.1, pp. 1–25.

Loughran, Tim and Bill McDonald (2011). “When Is a Liability Not a Liability? Textual Analysis, Dictionaries, and 10-Ks”. In: *The Journal of Finance* 66.1, pp. 35–65. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-6261.2010.01625.x`.

— (2014a). “Measuring readability in financial disclosures”. In: *Journal of Finance* 69, pp. 1643–1661.

— (2014b). “Regulation and financial disclosure: The impact of plain English”. In: *Journal of Regulatory Economics* 45.1, pp. 94–113. ISSN: 1573-0468. URL: https://doi.org/10.1007/s11149-013-9236-5.

— (2016). “Textual Analysis in Accounting and Finance: A Survey”. In: *Journal of Accounting Research* 54, pp. 1187–1230.

McCulloch, Warren S. and Walter Pitts (1943). “A logical calculus of the ideas immanent in nervous activity”. In: *Bulleting of Mathematical Biophysics* 5.

McDonald, Bill (2018). *Software Repository for Accounting and Finance*. URL: http://sraf.nd.edu/data/.

Mikolov, Tomas et al. (2013). “Distributed Representations of Words and Phrases and their Compositionality”. In: *CoRR* abs/1310.4546. URL: http://arxiv.org/abs/1310.4546.

Myšková, Renáta, Petr Hájek, and V. Olej (2018). “Predicting abnormal stock return volatility using textual analysis of news - a meta-learning approach”. In: *Amfiteatru Economic* 20, pp. 185–201.

Nardo, M., M. Petracco-Giudici, and M. Naltsidis (2016). “Walking down wall street with a tablet: A survey of stock market predictions using the web”. In: *Journal of Economic Surveys* 30, pp. 356–369.

Ng, Andrew (2018). *Lecture notes from Coursera course: Machine Learning*.

Östling, Robert and Gintare Grigonyte (2017). “Transparent text quality assessment with convolutional neural networks”. In: pp. 282–286.

Pennington, Jeffrey, Richard Socher, and Christoper D. Manning (2014). “Glove: Global Vectors for Word Representation”. In: pp. 1532–1543.

Pinheiro, Leonardo Dos Santos and Mark Dras (2017). “Stock Market Prediction with Deep Learning: A Character-based Neural Language Model for Event-based Trading”. In: *Proceedings of the Australasian Language Technology Workshop (ALTA)* 27.5, pp. 6–15.

Price, S.M. et al. (2012). “Earnings conference calls and stock returns: The incremental informativeness of textual tone”. In: 36, pp. 992–1011.

Qiu, X.Y., P. Srinivasan, and Y. Hu (2014). “Supervised learning models to predict firm performance with annual reports: An empirical study”. In: *Journal of the American Society for Information Science and Technology* 65, pp. 400–413.

Rather, A.M., A. Agarwal, and V.N. Sastry (2015). “Recurrent neural network and a hybrid model for prediction of stock returns”. In: *Expert Systems with Applications* 42, pp. 3234–3241.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1986). “Learning representations by back-propagating errors”. In: *Nature* 323.6088, pp. 533–536. URL: `http://dx.doi.org/10.1038/323533a0`.

S. Asness, Clifford, Burt Porter, and Ross L. Stevens (2000). “Predicting Stock Returns Using Industry-Relative Firm Characteristics”. In:

Schnabel, Tobias et al. (2015). “Evaluation methods for unsupervised word embeddings”. In: pp. 298–307.

SEC, ed. (2010). *Important Information About EDGAR*. URL: `https://www.sec.gov/edgar/aboutedgar.htm` (visited on 02/2018).

— (2013). *The Laws That Govern the Securities Industry*. URL: `https://www.sec.gov/answers/about-lawsshtml.html`.

Sohangir, S. et al. (2018). “Big Data: Deep Learning for financial sentiment analysis”. In: *Journal of Big Data* 5.

S&P500 Stock Index (2018). URL: `https://finance.yahoo.com/quote/%5EGSPC?p=%5EGSPC` (visited on 05/01/2018).

Sprenger, T.O. et al. (2014). “Tweets and trades: The information content of stock microblogs”. In: *European Financial Management* 20, pp. 926–957.

Tetlock, Paul C. (2007). “Giving Content to Investor Sentiment:The Role of Media in the Stock Market”. In: *The Journal of Finance* 62, pp. 1139–1168.

Wisniewski, T.P. and L.S. Yekini (2015). “Stock market returns and the content of annual report narratives”. In: *Accounting Forum* 39, pp. 281–294.

Wu, D.D., L. Zheng, and D.L. Olson (2014). “A decision support approach for online stock forum sentiment analysis”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 44, pp. 1077–1087.

Appendix A

A Basic Guide to Convolutional Neural Networks

We build a convolutional neural network to predict a company's stock performance based on its respective annual report. To understand how a convolutional neural network works, it is necessary to explain the basics of a simple neural network: how they are structured and how they are trained. Afterward, we show how the convolutional neural network is developed from the structure of artificial neural networks and how CNNs can be used in natural language processing. The presentation of artificial neural networks uses the intuition of Bishop (2006) alongside Andrew Ng's online Machine Learning course (Ng 2018), while the deep learning parts for the convolutional neural network mainly refers to Ian Goodfellow et al. (2016). We provide this in-depth guide for readers not working in the field or readers without prior knowledge of neural networks.

A.1 Neural Networks

In the endeavor of creating artificial intelligence, McCulloch and Pitts (1943) presented a method to solve logical calculus by imitating the structure of the human brain. The human brain contains billions of cells (neurons) that are interconnected in an extremely complex system. Specific patterns of input trigger different neurons resulting in different outcomes (i.e. interpretations in the brain). McCulloch and Pitts proposed the Artificial Neural Network (ANN), or simply Neural network (NN), which is defined as:

Artificial Neural Network: "… a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use." (Haykin 1999, p. 24)

The Artificial Neural Network is the basic model of a neural network which, similar to the brain, consists of neurons. McCulloch and Pitts, published in 1943 were many years ahead of their time as neural networks constitute a great part of today's Machine Learning.

Before and during the training phase, neural networks are not models but rather learning algorithms that produce models. Like regressions, neural networks take a set of inputs and output and generates a generalization of the relation between the input and output.

Figure A.1: Example of a neural network

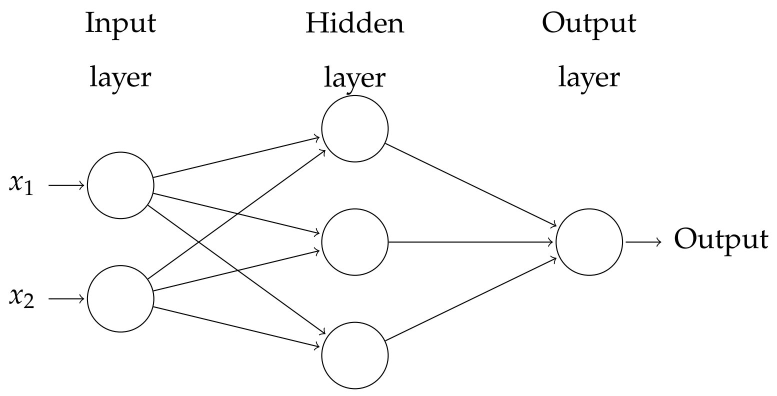


Figure A.1 illustrates a neural network with one input layer, a hidden layer, and an output layer. The circles represent neurons where the input layer is constituted by the left-most neurons. The middle layer is a *hidden layer* with three neurons and the output layer, to the right, consists of a single neuron.

A.1.1 Neurons

A neuron is an information processing unit which receives an input, then processes the input, and outputs the result. Neurons connect with other neurons through weighted connections in a feedforward structure. Thus, all neurons in one layer connect with all neurons in the next layer. Neurons do not connect with neurons in the same layer and the output of a neuron is never sent backward in the network. The lines in figure A.1 indicate the weighted connections.

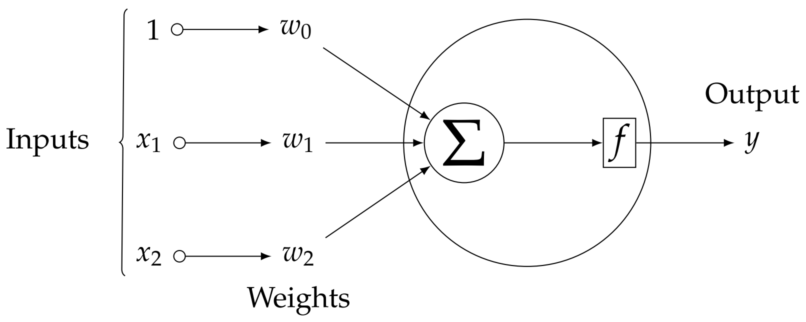
The neurons in the input layer contain the inputs to the neural network. The neurons in the input layers do not process their inputs, however, neurons in the hidden layers and the output layer process their inputs by applying an activation function.

A.1.2 Activation Functions

In figure A.2 a close-up look of a neuron shows how said neuron receives the sum-product of the inputs (i.e. the previous neurons' output) and their weights. The neuron inserts the sum-product into the activation function and the activation function then outputs result z which is used as the input for the next layer.

Besides inserting the weighted inputs, the sum function also includes an additional input. The additional input has a static value of one while its weight varies. The static value multiplied by its weight constitutes the bias of the neuron.

Figure A.2: A close-up illustration of a neuron



One of the most frequently used activation functions for neural networks is the logistic function, also known as the sigmoid function which is shown in equation A.1.

$$f(y) = \frac{1}{1+e^{-y}} \tag{A.1}$$

The logistic function receives an input and outputs a value between 0 and 1. The binary attributes of the logistic function make a network of logistic functions able to imitate logical operations. Take, for example, a non-linear prediction problem where the desired prediction is whether a company is going to perform well or poorly the next year. We use two input variables: the company's revenue and its costs. When using multiple inputs, the input neurons are vectorized as $x =$ [revenue, cost]. To illustrate the function's attributes, we use an imaginary company called A Basic Company (ABC). We want the activation function to be able to output a 1, when the costs (including the bias) exceed the revenue, or 0 if the revenue exceeds the costs (including the bias).

ABC has revenues of 100 and costs of 80. Assuming the weight associated with the revenue input is equal to -1, the weight associated with the cost is equal to 1, and the bias weight[9] is equal to 30 the sum-product is equal to:

$$(-1)\cdot 100 + 1\cdot 80 + 1\cdot 30 = 10$$

The output of the activation function is thus:

$$\frac{1}{1+e^{-10}} \approx 1$$

However, if costs decrease to 60, we get

$$(-1)\cdot 100 + 1\cdot 60 + 1\cdot 30 = -10$$

and the result of the activation function is

$$\frac{1}{1+e^{10}} \approx 0$$

Thus, when the cost and bias exceed the revenues, the activation function outputs a 1 indicating a bad performance and vice versa. The output of the activation function

[9]Remember that the value of the bias is always equal to one

can then be used by the subsequent layers. Other activation functions can be used to imitate different logical operations.

A.1.3 Layers

A neural network consists of layers, each containing a given number of neurons. In figure A.1, we use three dense layers, where the middle is a hidden layer. All layers that are not input or output layers are called hidden layers. ANNs follow a feedforward structure which means that each layer passes its outputs to the next layer, starting with the input layer and ending with the output layer. One iteration of calculating the final output based on a given input is called feedforward or forward propagation.

Altogether, a neural network can be formalized as the function:

$$f(\boldsymbol{x}, \boldsymbol{W}) = \boldsymbol{Y} \tag{A.2}$$

Where $\boldsymbol{x}$ is the input vector and the weight tensor (multidimensional matrix), $\boldsymbol{W}$, which contains all the weights of the network. The tensor $\boldsymbol{W}$ can contain several thousand connection weights depending on how complex the network is.

A.1.4 Training the Network

Training a neural network manually is a tedious process, although it has been automated by computers. Training requires sets of inputs and actual outputs (samples) from which it optimizes the weights so the error in the calculated output compared to the actual output is minimized. To do so, the algorithm needs a way to measure the error of the network.

A.1.5 Loss Functions

Applying a loss function to the result enables the neural network to determine the error of the predicted output. When calculating a forward pass in a prediction network, the output ranges between 0 and 1. For example, if a network outputs a probability of good performance of 90% for a company and the company actually performed poorly (0%), the simplest form of loss function calculates an error of $0\% - 90\% = -90\%$.

Many loss functions exist, but the cross-entropy loss function is the most popular. In this paper, we use categorical outcomes as output and we, therefore, use the categorical cross-entropy loss function which is presented in formula A.3:

$$J(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{y}) = -\frac{1}{N}\sum_{n=1}^{N}[y_n log(\hat{y}_n) + (1 - y_n) log(1 - \hat{y}_n)] \tag{A.3}$$

Where y_n is the actual outcome and $\hat{y}_n$ is the estimated outcome. Thus, when the actual outcome is 1 ($y_n = 1$), the $y_n \cdot log(\hat{y}_n)$ part of the function is calculated, and when the actual outcome is 0 ($y_n = 0$), the $(1 - y_n) \cdot log(1 - \hat{y}_n)$ part of the loss function is calculated.

The objective of the training phase is thus to minimize the loss function by changing the weights. Continuing with the example above, if the model outputs a value of 90% and the actual outcome is that the company performs poorly (0%) we get the following loss:

$$-\tfrac{1}{1}(0 \cdot log(90\%) + (1 - 0)log(1 - 90\%) = 2.302585$$

The loss is deemed large. However, if the company performs well, we get:

$$-\tfrac{1}{1}(1 \cdot log(90\%) + (1 - 1)log(1 - 90\%) = 0.105361$$

We see that the loss is substantially smaller than before because the estimate is closer to the actual outcome.

A.1.6 Backpropagation

Adjustment of the weights is an optimization problem, where the objective is to minimize the loss function by changing the weights of the neural network. The partial derivatives of the loss function with respect to each of the connection weights are the foundation of optimizing the network, as presented in formula A.4:

$$\frac{\partial}{\partial \boldsymbol{W}} J(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{y}) \tag{A.4}$$

A computer can easily overcome calculating the partial derivatives of the connection weights in our small example neural network. Some networks, however, have more than one hundred neurons in each layer with more than 10 layers. Calculating the derivative of each weight individually would be computationally heavy and inefficient. Rumelhart et al. (1986) show an effective way to overcome the computational inefficiencies of the optimization task by using the advantages of the chain rule of derivatives. The backpropagation algorithm starts with the feedforward pass where each step in the network is calculated. In addition to calculating all the steps, the backpropagation also calculates and stores the partial derivatives for each of the calculated steps with respect to the input of each step. In other words, the network calculates a measure which is useful when determining the error contributed by each weight to the overall error. After all the steps are calculated, the network traces back the error which each individual weight contributed to the total error using the chain rule. The weights are then simultaneously adjusted in order to reduce the loss function using an optimizer. Through this process, the error becomes smaller and smaller which improves the performance of the learning algorithm. When the training is completed, the model is ready to be used for predicting unfamiliar data.

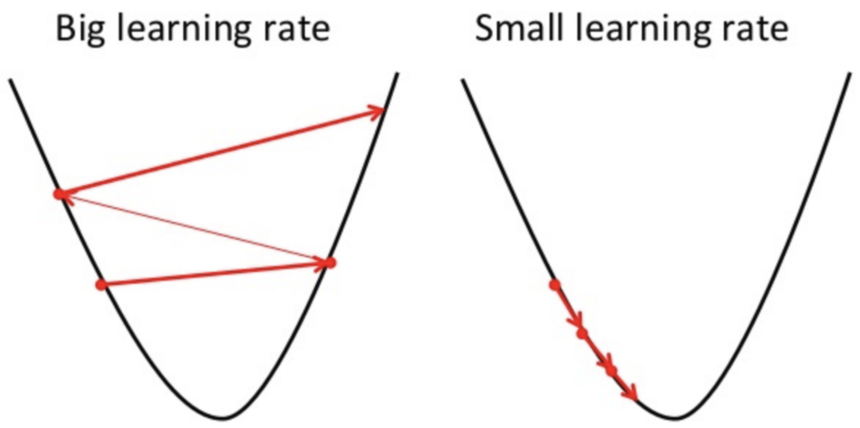
A.1.7 Optimizers

An optimizer calculates the adjustment to the weights based on the partial derivatives calculated with the backpropagation method. Optimizers are mathematical algorithms which exploit the power of the derivatives to minimize or maximize an objective. In machine learning, this objective is often to minimize the loss. Instead of updating the weights for each training sample, multiple samples are often assembled in batches where the batch-size determines the number of samples. As a result, optimizers update the weights once it has calculated all the samples in a batch. Furthermore, a single training iteration is defined as calculating one feedforward pass, using backpropagation to calculate the partial derivatives. When all training samples in a batch have been calculated, the optimizer updates the weights. Depending on the neural network and the task, the learning algorithm requires many epochs to fully learn the patterns of the input. An epoch is defined as one iteration throughout the entire training dataset.

A.1.8 Learning Rate

The changes in the weights calculated with the optimizer can in some cases become too large. If this happens, the optimal weight values will never be computed as the changes are too large for the fine-tuning needed to find the optimum. Figure A.3 shows a two-dimensional representation of the training loss of a network.

Figure A.3: Example of overshooting



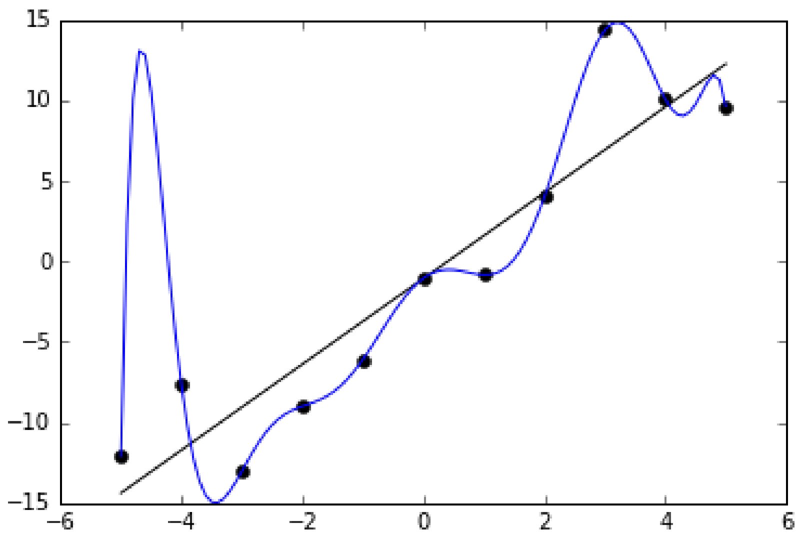
Neural networks use a learning rate to overcome the problem of changing the weights excessively. The learning rate is a constant which dampens the learning process. The change to each weight is thus reduced to only take small steps towards the final model. The learning rate is set depending on the optimization method and the model task in general.

When initiated, models tend to be far away from the optimal point which creates a problem when applying a learning rate. Observing that if the learning rate is small, the model will take many small steps towards the optimal point. Also, if the learning rate is too small, the model never comes close to the optimal weight settings, resulting in a bad performance. However, if the learning rate is high, the model moves towards the optimum in bigger steps, but it will eventually encounter the problem of changing the weights too much in each iteration. Adding a decay factor to the learning rate allows the learning rate to decrease each time the weights are updated. Weight changes then take big steps in the beginning, however, as the process of training proceeds the decay diminishes the learning rate.

A.1.9 Overfitting and Regularization

Overfitting happens when the model fails to generalize the patterns in the input but instead only adjusts to the specific training data. In machine learning, overfitting often happens because the weights of the model take on extreme values which allows the model to fit very specific patterns. As a result, the model performs badly when it predicts unfamiliar data. Figure A.4 shows a model which overfits the samples. The blue line correctly explains the relationship between every data point, however, it would not perform well on any additional unfamiliar data points. The standard linear regression performs significantly better at generalizing the overall trend of the data points.

Figure A.4: Example of overfitting



Regularization is a method used to reduce the problem of overfitting. As a result, the model's performance of the training data may decrease, however, its accuracy on unfamiliar data increases. Many regularization methods penalize large weights by adding the size of the weights to the loss function. The additional loss prevents the weights from becoming extremely large or small because the optimizer would reduce their size in order to reduce the overall loss. By limiting the neurons to adapt to specific training samples, the learning algorithm generalizes the training data instead of overfitting the given training samples. For instance, in a case with many features but few samples, models tend to overfit the training samples.

Instead of simply adding the weight values to the loss function, adding the squared weights as the regularization term has advantages when combined with the backpropagation algorithm.

$$J(\boldsymbol{W},\boldsymbol{y}) = -\frac{1}{N}\sum_{n=1}^{N}[y_n log(\hat{y}_n + (1-y_n)(log(1-\hat{y}_n)] + \frac{1}{2}\lambda \boldsymbol{W}^2 \tag{A.5}$$

Formula A.5 includes the L^2 parameter regularization, or the Tikhonov regularization at the end of the loss function, which is used in the network in this paper. The λ term works similarly to the learning rate. This term regulates how much the weights' values should be accounted in the loss function similar to how the learning rate regulates how much of the weight changes should be applied when optimizing the model.

A.1.10 Random Initialization of Weights

Before training begins, it is necessary to initialize the weights of the neural network. Setting the weights to zero makes the backpropagation produce identical changes to all weights forcing the neurons in each layer to adapt to the same pattern. Initialization of weights is, therefore, often completed by setting the weights to a random number between -1 and 1.

A.2 Convolutional Neural Network

This section presents how a convolutional neural network develops from the understanding of neural networks. We demonstrate how convolutional neural networks can be used in natural language processing.

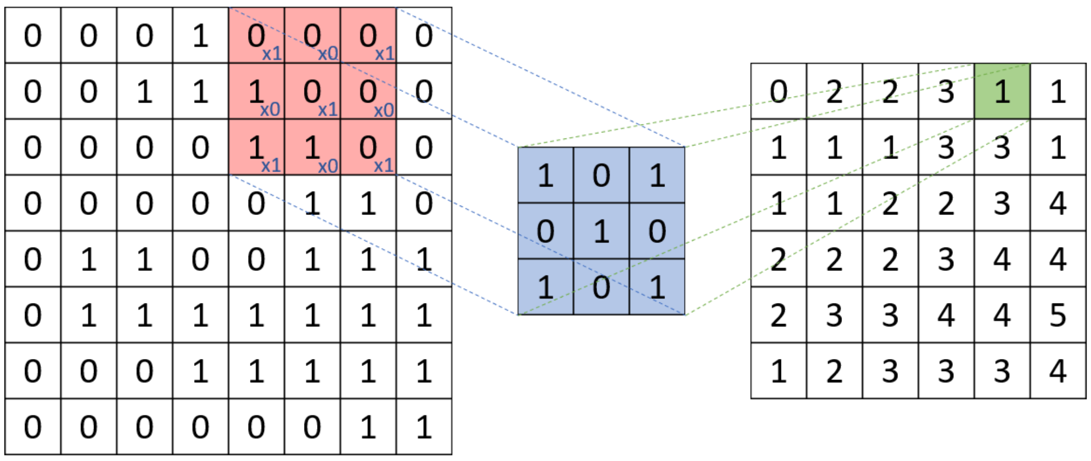
A.2.1 Inspiration from Vision

LeCun et al. (1999) introduced convolution neural networks as a learning algorithm able to recognize objects in images. Convolutional neural networks (CNN) are neural networks, which include convolutional layers, which compress the information in a prior layer to the next by using filters. The filters are fitted to specific patterns throughout the training process of the model. Collobert et al. (2011) later found that CNNs also perform well at semantic parsing and sentiment classification. Unlike neural networks, convolutional neural networks may effectively process great amounts of sequential and multidimensional information.

A.2.2 Convolutional Layers

Convolutional neural networks get their name from adding convolution layers to a neural network. A convolutional layer is useful when input data is multidimensional. For example, imagine a small image that consists of 8x8 pixels. The image has a total of 64 pixels. The image can be represented as a matrix where each value represents a specific pixel. The values of the matrix represent the blackness of the pixels. A value of one represents a completely black pixel and a zero represents a white pixel. The matrix has the same size as the image. Reshaping the matrix into a vector removes the relationship between the pixels. Thus, using the matrix representation of an image in an artificial neural network is not possible as artificial neural networks do not allow multidimensional inputs. Convolutional layers pose the solution to this limitation of the standard neural network. In a convolutional layer, a filter iterates over the two-dimensional input to look for specific patterns. The size of the filter determines how big a range of inputs it examines on each iteration. In figure A.5, the filter in blue is 3x3 pixels large and thus scans a range of nine values of the matrix on each iteration.

Figure A.5: Example of a convolutional layer

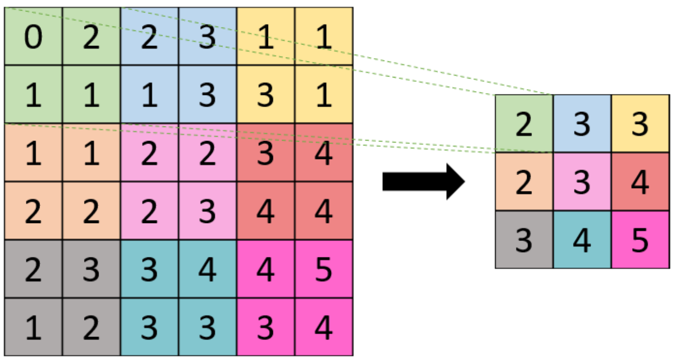


The filter outputs a compressed version of the input called a feature map. The feature map's values are calculated as the sum-product of the filter values and the input values. The values of the filter determine which patterns the filter finds, and it is also what is being trained to detect the important patterns of the input. By changing the filters, the network adapts to the input data by recognizing patterns that appear in the inputs. A convolutional layer often has several filters where each filter has a different composition. Different filters notice different features of the images and pass these on to the next layer. The size of the filter can be changed to fit the input images and the size of the filter is referred to as the receptive area size of the convolutional layer.

A.2.3 Pooling Layers

Convolutional neural networks often have pooling layers after each convolutional layer. Like the convolutional layer, the pooling layer applies its function on an area of the previous layer one iteration at a time. Opposite to the convolutional filter layer, the pooling layer does not overlap the input data. By not overlapping, each value of the convolutional layer gets evaluated once to make sure only the most important information is passed through to the next layer. Applying a receptive size of 2x2 in the pooling filter, we get the results in figure A.6 where each color matches the input and corresponding pooling values.

Figure A.6: Example of a max pooling layer

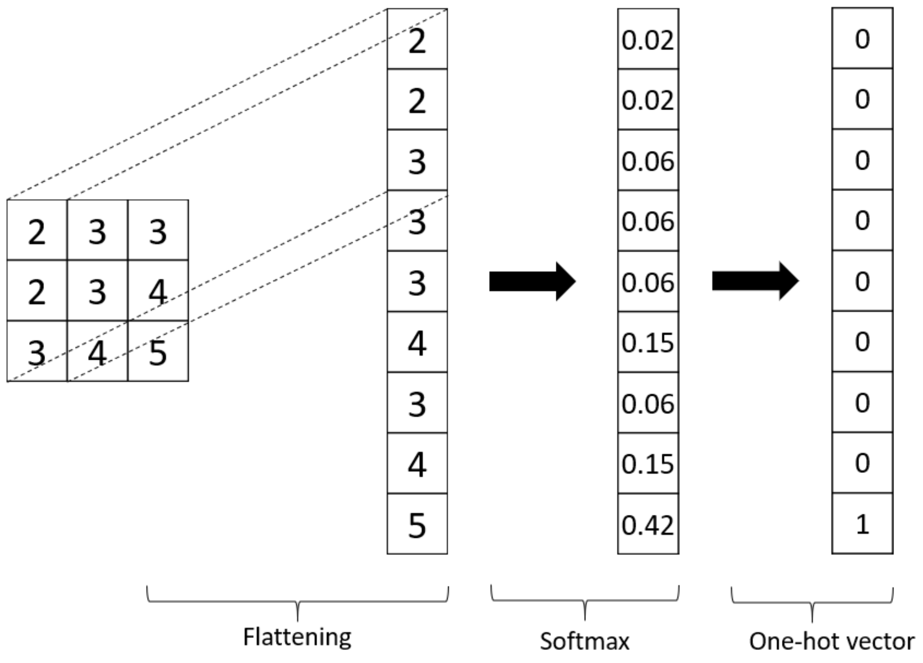


The max pooling function outputs the maximum value of the area for each non-overlapping area. It compresses the input while still passing on the relevant attributes of the previous layer.

A.2.4 Flattening Layer

To make the final output of the model interpretable, the network eventually needs to be flat like the standard neural network. A flattening layer converts a two-dimensional matrix into a one-dimensional vector. For example, applying a flattening layer to a 3x3 matrix from a previous layer transposes the matrix into a 9x1 vector as illustrated in figure A.7. The layers following a flattening layer are fully connected layers which are the same as in an ordinary neural network, as described earlier. The network then ends with an output layer with a number of neurons that match the number of possible outcomes.

Figure A.7: Example of a flattening layer, softmax function, and a one-hot vector



A.2.5 Softmax

With several outcomes, we cannot use the value of the output layers as it is not a unit of measurement which cannot be interpreted. To transform the outputs from the final layer, we calculate the values as probabilities. To do this, we use the softmax function as the activation function of the final layer. The softmax function transforms the final layer's output into a probability. Formula A.6 shows how the function is calculated.

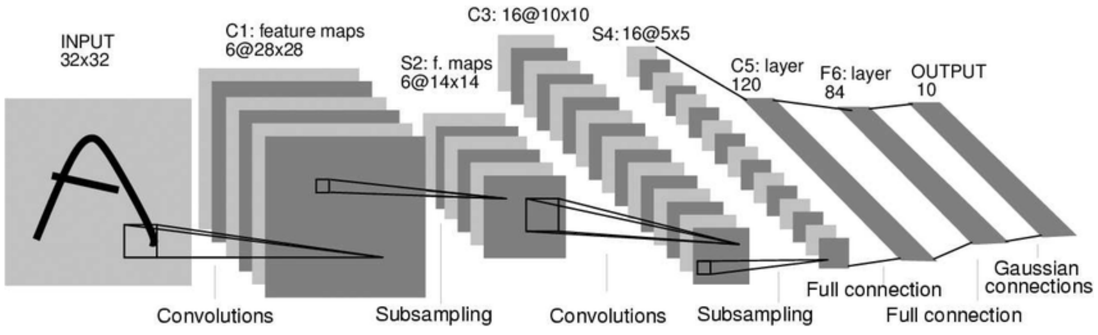
$$\sigma(\boldsymbol{x}_j) = \frac{e^{xj}}{\sum_i e^{xi}} \tag{A.6}$$

Other techniques for transforming multiple outcomes into a probabilistic measure exist, but the softmax is by far the most commonly used. In figure A.7, we show the softmax function applied to the nine neurons from the flattening layer. Usually, the outcome with the highest probability is chosen as the output. As shown in the figure, the softmax is represented as a vector and is transformed into a *one-hot* vector where all values are zero except the predicted outcome which has the value of one. The final one-hot vector is what is used when calculating the loss function.

A.2.6 Putting the Parts Together

Adding a convolutional layer and a pooling layer transforms the standard neural network into a convolutional neural network. By also adding flattening layer, we enable the model to handle classification problems. CNNs may have several sets of convolutional and pooling layers. Figure A.8 shows the first illustration made by LeCun et al. (1999), where an image of a hand-written letter "A" is used as input.

Figure A.8: A complete convolutional neural network



After the input layer, the next layer applies six different filters to create six feature maps. Labeled as "subsampling", the model compresses the prior layer in a pooling operation. After an additional set of convolution and pooling layers, the model is flattened and it ends in a fully connected layer with 10 outcomes. By applying the softmax function to the output layer, the network returns the probability of each outcome.

A.2.7 Convolutional Neural Network in Natural Language Processing

Instead of using images, Collobert et al. (2011) show a way to use text as the input in a CNN. Revisiting the data section, where we explained how words can be represented as vectors with word embeddings. By replacing each word in a text with a horizontal vector (the word's embedding), we create a matrix for each text or sequence of words. Using the matrix as input gives the network the same capabilities as in the previous illustration in figure A.8.

The Receptive Area and Word Embeddings

As mentioned earlier, the receptive area is the size of the area that a convolution layer and a pooling layer examines when iterating through the previous layer. In figure A.5 we saw how a convolutional layer examines all possible 3x3 areas of the input. However, when working with word embeddings we want to include all columns of the previous layer as *all* the numbers representing a word should be examined at the same time. Thus, we only convolute on the input matrix in a vertical manner.