Automatic Dispatching of Issues
Using Machine Learning

Automatisk fördelning av ärenden genom maskininlärning

Fredrik Bengtsson
Adam Combler

Supervisor: Ahmed Rezine
Examiner: Cyrille Berger
Upphovsrätt

Detta dokument hålls tillgängligt på Internet - eller dess framtida ersättare - under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/.

Copyright

The publishers will keep this document online on the Internet - or its possible replacement - for a period of 25 years starting from the date of publication barring exceptional circumstances.

The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/.

© Fredrik Bengtsson
Adam Combler
Abstract

Many software companies use issue tracking systems to organize their work. However, when working on large projects, across multiple teams, a problem of finding the correct team to solve a certain issue arises. One team might detect a problem, which must be solved by another team. This can take time from employees tasked with finding the correct team and automating the dispatching of these issues can have large benefits for the company. In this thesis, the use of machine learning methods, mainly convolutional neural networks (CNN) for text classification, has been applied to this problem. For natural language processing both word- and character-level representations are commonly used. The results in this thesis suggests that the CNN learns different information based on whether word- or character-level representation is used. Furthermore, it was concluded that the CNN models performed on similar levels as the classical Support Vector Machine for this task. When compared to a human expert, working with dispatching issues, the best CNN model performed on a similar level when given the same information. The high throughput of a computer model, therefore, suggests automation of this task is very much possible.
Acknowledgments

Thanks to the telecommunications company and their employees for their help with understanding the domain and performing our experiments. A special thanks to our supervisor Kim and our closest manager Marcus for all their support. We also want to thank our supervisor and examiner, from Linköping University, Ahmed Rezine and Cyrille Berger. Finally, we are grateful for the valuable feedback and opposition from Elina Lundberg and Erica Gavefalk.
# Contents

Abstract iii  
Acknowledgments iv  
Contents v  
List of Figures vii  
List of Tables viii  
Abbreviations ix  

## 1 Introduction 1  
1.1 Motivation 2  
1.1.1 Telecom Company 2  
1.2 Aim 2  
1.3 Research Questions 3  
1.4 Delimitations 3  

## 2 Theory 4  
2.1 Natural Language Processing 4  
2.1.1 Raw Feature Reduction 5  
2.1.2 Numerical Feature Reduction 5  
2.1.3 Feature Transformation 7  
2.2 Machine Learning 7  
2.2.1 Supervised Learning 8  
2.2.2 Support Vector Machines 9  
2.3 Neural Networks 10  
2.3.1 Activation Functions 11  
2.3.2 Deep Learning 13  
2.3.3 Convolutional Neural Networks 13  
2.3.4 Training and Tuning 17  
2.4 Performance Measures 20  
2.5 Interviews 23  
2.6 Related Work 24  
2.6.1 Text Classification 24  
2.6.2 Issue Classification 25  

## 3 Method 27  
3.1 Frameworks 27  
3.2 Hardware 28  
3.3 Initial Datasets 28  
3.3.1 Contents of the Issues 28
# List of Figures

2.1 Word Embeddings ................................................................. 7  
2.2 SVM .................................................................................. 10  
2.3 ReLU .................................................................................. 12  
2.4 Convolution example ........................................................... 14  
2.5 Dropout example ................................................................. 16  
2.6 Example of confusion matrix ................................................ 23  
3.1 Creation of datasets .............................................................. 30  
3.2 Training/validation and test split .......................................... 30  
3.3 Distribution of issues across teams for dataset A and B, after preprocessing ......................................................... 33  
3.4 The number of words in issues of dataset A and B .................. 34  
3.5 The number of characters in issues of dataset A and B ............ 34  
3.6 Char-level CNN ................................................................. 37  
3.7 Multichannel CNN (characters + words) ............................... 39  
3.8 User experiment landing page ............................................. 42  
3.9 Information used for prediction in user test ............................ 42  
3.10 Making predictions in user test .......................................... 43  
3.11 Tag NN architecture ............................................................ 44  
4.1 Top-3 overlap of word- and character-level CNN predictions (dataset A) ................................................................. 46  
4.2 Top-3 accuracy, accuracy and macro F1-score of all models (dataset A) ................................................................. 47  
4.3 The per team top-3 accuracy of the models (dataset A) .......... 48  
4.4 Multichannel CNN (characters + tags) top-3 confusion matrix (dataset A) ................................................................. 49  
4.5 Overlap of human and model correct classifications (dataset A_D) ............................................................................. 50  
4.6 Human vs model scores (dataset A_D) .................................... 51  
4.7 Human and model per team top-3 accuracy (dataset A_D) ........ 51  
4.8 Multichannel CNN (character + tag) top-3 confusion matrix (dataset A_D) ................................................................. 52  
4.9 Human expert top-3 confusion matrix (dataset A_D) .............. 53  
A.1 Distribution of issues across the teams in the subsets of dataset A ................................................................. 76  
A.2 Distribution of issues across teams for dataset A_D ................ 77  
A.3 Distribution of issues across the teams in the subsets of dataset B ................................................................. 78  
B.1 Entire page for human expert prediction ................................ 80
## List of Tables

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Preprocessing settings</td>
<td>31</td>
</tr>
<tr>
<td>3.2 Dataset splits</td>
<td>32</td>
</tr>
<tr>
<td>3.3 Research Questions mapping to experiments</td>
<td>36</td>
</tr>
<tr>
<td>3.4 Word- and Character-level CNN hyperparameter tuning ranges</td>
<td>38</td>
</tr>
<tr>
<td>3.5 Tuned parameters for word- and character-level CNN</td>
<td>38</td>
</tr>
<tr>
<td>3.6 Multichannel CNN (characters + words) hyperparameter tuning ranges</td>
<td>39</td>
</tr>
<tr>
<td>C.1 SVM grid search results</td>
<td>81</td>
</tr>
<tr>
<td>C.2 Word-level CNN random search step 1 results</td>
<td>83</td>
</tr>
<tr>
<td>C.3 Word-level CNN random search results second run</td>
<td>84</td>
</tr>
<tr>
<td>C.4 Word-CNN grid search results</td>
<td>85</td>
</tr>
<tr>
<td>C.5 Character-level CNN random search results for first run</td>
<td>86</td>
</tr>
<tr>
<td>C.6 Character-level CNN random search results for second run</td>
<td>86</td>
</tr>
<tr>
<td>C.7 Multichannel CNN random search results for first run</td>
<td>87</td>
</tr>
<tr>
<td>C.8 Multichannel CNN random search results for second run</td>
<td>88</td>
</tr>
<tr>
<td>C.9 Multichannel CNN random search results for third run</td>
<td>89</td>
</tr>
<tr>
<td>C.10 Tag NN random search results</td>
<td>90</td>
</tr>
</tbody>
</table>
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BoW</td>
<td>Bag-of-Words</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional Neural Networks</td>
</tr>
<tr>
<td>EDA</td>
<td>Exploratory Data Analysis</td>
</tr>
<tr>
<td>FN</td>
<td>False Negatives</td>
</tr>
<tr>
<td>FP</td>
<td>False Positives</td>
</tr>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>NN</td>
<td>Neural Networks</td>
</tr>
<tr>
<td>OD</td>
<td>Original Dataset</td>
</tr>
<tr>
<td>RQ</td>
<td>Research Question</td>
</tr>
<tr>
<td>ReLU</td>
<td>Rectified Linear Units</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machines</td>
</tr>
<tr>
<td>SG</td>
<td>Stacked Generalization</td>
</tr>
<tr>
<td>SGD</td>
<td>Stochastic Gradient Decent</td>
</tr>
<tr>
<td>TF</td>
<td>Term Frequency</td>
</tr>
<tr>
<td>TN</td>
<td>True Negatives</td>
</tr>
<tr>
<td>TP</td>
<td>True Positives</td>
</tr>
<tr>
<td>TF-IDF</td>
<td>Term Frequency-Inverse Document Frequency</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
</tbody>
</table>


Imagine being part of a small team of developers working on a small software project. You might know everyone in the team, and you might divide the work across the team depending on the expertise of each individual. If you encounter a problem, you can just ask whoever is the most proficient within the team for assistance, or whoever wrote the code. Now, instead imagine that the software project is huge. Tens, or maybe hundreds of teams collaborate, and the work is divided so that each team is assigned some specific part of the project. Now, in such a case if you encounter a problem, it is possible that no one in your group is an expert on the part of the software you are having problems with. As a result, you might have to create an issue that should be resolved by the correct corresponding team; but how do you know which one to send the issue to?

For large software companies, trying to solve this problem might be expensive, time consuming and error prone [1]. One approach is to assign a person or a team of people to distribute these issues. The cost of this solution would be the salary of the workers in this team and the time consumption depends on the efficiency and size of the team. The errors would come from the limits in the human ability to correctly classify incoming issues to the correct team. Another solution is to let the teams pick the issues that they want to solve themselves, from a long list of issues. However, there might be issues that no one wants to solve. Then these issues have to be assigned anyway, for example through the first suggested approach. If the assignment could be done automatically instead, there would be a huge potential to save both time and money in many software companies [1].

Machine learning introduces the possibility of being able to automate such tasks. The descriptive text within each issue can be transformed into features, that in turn can be used by a machine learning model for classification. One method allows the words or the characters to be represented as vectors, and as a result, a text can be represented as a matrix of these vectors. This allows for deployment of similar machine learning models to those commonly used with great success in for example image processing [2], [3]; convolutional neural networks (CNN). For images, the pixels are commonly the smallest unit of input to the CNNs. However, for texts it is not as clear cut; sometimes it is the words and sometimes it is the characters. This thesis aims to investigate how the level, character- or word-level, of these vectors
affect the results of a CNN classifying issues with mixed languages in a team-based setting. In addition, it investigates the performance of CNN models using character-level, word-level or a combination of both levels, in this setting. The performance is also compared to a well-established baseline, a Support Vector Machine (SVM) using Term Frequency-Inverse Document Frequency (TF-IDF) features. Finally, an investigation is conducted regarding the performance of a human expert compared to the best of the CNN models, to give an indication of the usefulness of an automatic system in this context.

1.1 Motivation

Many companies use issue tracking systems; for example Jira\(^1\) alone has over 50 000 customers according to Atlassian [4]. Somehow, these issues have to end up at a person or team assigned with resolving the issue. If any part of the dispatching process is done manually, an automatic solution has the potential to speed up the process, saving both time and effort [1].

1.1.1 Telecom Company

This master’s thesis research is conducted at a large international telecommunication company. At this company there is a large number of projects going on at the same time, with many teams connected to each project. For a single project there may be hundreds of new issues generated each week. Furthermore, the company uses an issue tracking software called Jira\(^1\) to track these issues. Within the telecom company there is a support organization tasked with supporting the research and development teams. It is this part of the company which is the focus for this thesis.

The approach used for dispatching issues in the support organization is to let teams involved in a project select which issues out of a large backlog is best suited to the team. However, sometimes issues do not get selected by any team. For these cases, there are employees tasked with manually dispatching the issues to one of the teams. The teams have some distinct areas of expertise, but with a rather large overlap in competences. Hence, multiple teams might be equally good choices for solving some issues. Note that even though the support organization divides issues into projects, the same team can work on multiple projects. Using a machine learning system, which assigns the correct team to these issues, this repetitive and time consuming every-day task can be solved by a computer instead of a human. This could save both time and money for the company.

1.2 Aim

This thesis aims to investigate how CNNs can be used to classify issues in a team-based organization. Character- and word-level embeddings are put against each other, as well as used in unison. To understand how the different text representations affect the features learned by the CNN. Furthermore, the CNN models for text classification are compared with a well established baseline method. Lastly, to estimate the potential benefits of a automated system, the best CNN is compared to the performance of a human expert.

\(^1\)https://www.atlassian.com/software/jira
1.3 Research Questions

RQ1: Do word- and character-level CNNs, trained on issues, produce complementary features?

Many recent advancements in the field of text classification have involved CNNs, sometimes using word embeddings and sometimes character embeddings [5]–[8]. The issues in the international telecom company might contain different languages, computer generated text and company specific terminology such as server and project names. As seen in [7], there might be a benefit in combining character and word embeddings in a text classification context with multiple languages, specifically sentiment classification of twitter posts. This suggests that the CNN learns different information from the two representations in their domain; the goal with this research question is to investigate if the effect occurs in the domain of issue classification as well. Therefore, this question includes the creation of three models; a character-level CNN, a word-level CNN and a joined character- and word-level CNN. The joined CNN uses one channel each for the two representations of the text and is based on the architecture described in [7]. The goal is to answer this research question by comparing the results of the three models.

RQ2: How do the CNN models (of RQ1) compare to the commonly used SVM with TF-IDF model?

A baseline commonly used in the domain of text classification is using TF-IDF as the text descriptor, and a linear model to classify; in this case an SVM. The SVM specifically has been used for this purpose by among others, Johnson and Zhang [6]. This question aims to investigate how the CNN models compare to this established baseline. The comparison is mainly in terms of predictive performance, however, other factors like ease of implementation and training time will be taken into consideration. As the dataset used for this thesis is not available for outside parties, the comparison to a baseline was determined to be of high importance. Furthermore, without a proper baseline it is hard to determine what might be a reasonable result for a model on this specific task.

RQ3: How do humans with expert domain knowledge compare to an automatic system when dispatching issues across multiple teams?

The current solution for dispatching issues at the telecom company is, as described in Section 1.1.1, done in two ways. Either an issue is taken from backlog by the team or it is dispatched to a team by another employee, which is tasked with dispatching issues. The aim of this research question is to determine the viability of using an automatic issue dispatching system, based on the best CNN model created as a part of RQ1. Therefore, the performance of an employee with expertise in dispatching issues at the company is compared to the performance of the model.

1.4 Delimitations

The thesis is limited to dispatching in a team-based environment and is, therefore, not investigating dispatching to individuals. This is because the thesis is conducted at the telecom company, where the work, as in many companies, are divided on a team-level.
2 Theory

Relevant terms and methodology are covered in this chapter, to lay the foundation necessary to understand the work in this thesis. Firstly, as this thesis relies on a dataset consisting of text, natural language processing is introduced. The aim is to introduce common strategies to prepare text based data for use in machine learning applications. Secondly, the field of machine learning is explained in broad terms; most importantly defining Support Vector Machines. Thirdly, neural networks are described in detail. In particular, it describes common neural network layers used in convolutional neural networks; the type of networks which is the focus of this thesis. Fourthly, different performance measures used for the final machine learning systems are introduced. Furthermore, different types of interviews and the difference between them is covered. Finally, the theory section is concluded by covering related work in the area of text classification in general and, more specifically, issue classification.

2.1 Natural Language Processing

The field of Natural Language Processing (NLP) is a branch of Artificial Intelligence, with the aim to form models extracting information from natural language. Natural language can be both spoken and written language (text), consisting of words forming sentences and meaning. Natural language is classified as unstructured data, which contrasts to e.g. data stored in relational databases [9], [10]. Additionally, it is high dimensional, since each word can be seen as a dimension [9]. There are also other factors impacting the information contained within texts that humans can easily understand, but which are much harder to model, e.g. "reading between the lines" or context. Commonly NLP problems are presented in the form of texts, which is also the case for this study. To be able to extract useful information from text, NLP-systems usually follow two steps; preprocessing and modelling. This section will focus on the preprocessing step, whereas modelling is described further in Section 2.2.

The preprocessing step aims to prepare a text dataset for modelling, which in turn tries to model some NLP problem. As described in [9], the preprocessing step consists of three phases; raw feature reduction, numerical feature reduction, and feature transformation.
2.1.1 Raw Feature Reduction

The first step of the preprocessing is to transform texts into sets of features; a process called tokenization [10]. One way is to separate the words by whitespace and add them to a list in some programming language. After the tokenization step the aim is to reduce the dimensionality of the features, in other words, removing and altering the words of the text in different ways. Some of the most common methods are:

- lowercasing,
- stop words removal,
- stemming,
- and lemmatization.

Lowercasing, sometimes called case folding, refers to replacing all uppercase letters in a text with their corresponding lowercase ones [10]. Understandably, this reduces the vocabulary, as tokens which earlier only differed by casing now are represented using the same token. This can in some cases result in a loss of information for tokens where the casing significantly changes the meaning.

Stop word removal means that so-called stop words are removed from the text, and consequently from the vocabulary. These are common but non-contributing words; for English this might be words such as the, an and a. There are multiple lists for stop words available online, and as a part of many libraries [9], [10].

Stemming is the process of transforming words into their stem or root form [9], [10]. If stemming is applied to a tokenized text, it will reduce the number of features since all words with the same root will be represented by the same feature. Usually the stemming algorithms depend on some set of rules removing or replacing the suffixes or prefixes of words. One example of this is the Porter Stemming Algorithm [11]. If this algorithm would be used on the word ponies it would end up as poni, where the suffix -ies is replaced by -i.

Lemmatization is similar to stemming, but here the words are instead transformed into their dictionary base form (lemma) [9], [10]. Therefore, lemmatization depends on a dictionary for lookup of each word. For example, the word rang would first be entered into the dictionary, which would show that ring is the lemma. This lemma would then replace the original word in the set of features.

Token Expansion

To be able to better represent a text, sometimes additional tokens are concatenated to the tokenized version of the text in the form of N-grams. N-grams are combinations of \( n \) number of tokens where \( n \in \mathbb{N}^+ \). For example the bigrams, \( n = 2 \), of the sentence “I live in Cape Town” are “I live”, “live in”, “in Cape” and “Cape Town”. Using n-grams could help the classifier later on; in the previous example, the city name “Cape Town” might better explain what was intended than the separate words “Cape” and “Town”. The normal case of just using single words can also be referred to as unigrams. This method is also applicable to character-level tokens and works in the same fashion.

2.1.2 Numerical Feature Reduction

After raw feature reduction has been applied to a text dataset, each text of the dataset has been transformed into a set of text features, for example a list of lemmatized words. The purpose of
2.1. Natural Language Processing

the next step, is to transform the set of text features into a numerical representation; Jurafsky et al. [9] calls such a numerical representation a vector space model. Additionally, this step uses the properties of the vector space model to select the most important features. In this section multiple ways to accomplish this are explained. However, to be able to do so, some terminology must first be properly defined:

- **Term** is another word for text feature, and can be for example a word, a smiley or an expression/name such as “San Fransisco”.
- **Document** refers to a list of terms corresponding to one of the texts in the original text dataset. For example, if the text dataset would be a set of tweets, a document would be the list of terms corresponding to the words of a single tweet.
- **Corpus** is usually used to describe a set of documents, and will, in the case of this study, be used to refer to the set of documents corresponding to all texts in the original text dataset.

Explanations regarding different ways to transform a document into its corresponding vector space model follow in the remainder of this section.

**Bag-of-Words** (BoW) and **Term Frequency** (TF) refer to vector space models that represent documents by an unordered set of term weights, one for each type of term. The weight of each term is the count of the term in a document; the term frequency. Usually for term frequency weighing the term weights are also normalized to reduce the importance of frequent terms, by for example the log-function [10]. One way of reducing the number of features in a BoW-like model is to exclude the terms that are represented by the lowest weights.

**Term Frequency-Inverse Document Frequency** (TF-IDF) is another vector space model, similar to term frequency models; the difference is how the weight for each term is calculated. It is commonly used as a baseline in NLP-problems [8], [9], [12]. The idea behind TF-IDF is that the weight of a term should not only depend on the importance of the term within a document (TF) but also the importance of the term in relation to the corpus. This inter-document weight of a term $t$, inverse document frequency, can be expressed as

$$IDF_t = \frac{N}{DF_t},$$

where $N$ is the number of documents in the corpus and $DF_t$ is the document frequency of a term $t$; the number of documents within the corpus where the term $t$ is found. Then, the TF-IDF weight for a term $t$ is calculated as

$$TF-IDF_t = TF_t \cdot IDF_t.$$  

Similarly to just using the TF, the TF-IDF weight is usually also normalized [10].

**Word Embeddings** are a representation of words as dense vectors, where vectors which appear in similar context are closer in vector space. There are multiple ways to create these vectors; three methods are word2vec [13], Glove [14] and fasttext [15]. Arithmetic with these embedding vectors has been shown by Mikolov et al. [16] to yield interesting results. For example, using Glove word embeddings, it is true that

$$\text{vector(Stockholm)} - \text{vector(Sweden)} + \text{vector(Netherlands)} \sim \text{vector(Amsterdam)}.$$ 

Furthermore, Mikolov et. al used Principal Component Analysis (PCA) to project word embeddings into a two dimensional space for plotting. To further explain the concept of word
2.2. Machine Learning

embeddings, this approach is used in Figure 2.1 to illustrate the phenomenon expressed by Equation 2.3. In this case, Glove word embeddings with 100 dimensions are projected into two dimensions using PCA. Using this projection, the expression and amsterdam vectors are observed to lie close to each other, indicating that they express similar concepts.

2.1.3 Feature Transformation

Even though the vector space model of a dataset might contain fewer features than the original text dataset, because of for example lemmatization or BoW-modelling, it still might be very high dimensional. To combat this, one or several so-called dimensionality reduction algorithms could be used. These algorithms are aimed at finding the features that best explain the data. Best in this context could mean different things for different algorithms. The use of these algorithms stretches beyond the scope of this study, but some commonly used algorithms include Linear Discriminant Analysis (LDA) [10], Principal Component Analysis (PCA) or Locally Linear Embedding (LLE) [9].

2.2 Machine Learning

To better understand the methods used in this thesis, an introduction to the general field of machine learning is required. Therefore, in this section general terminology and learning algorithms regarding machine learning are presented.

Machine learning problem formulations originate from either a dataset or some way to generate data describing an event or object. This data can be in the form of for example sensory output from a robotic arm, chess moves, images of faces and internet news articles to name a few. Furthermore, the data consists of one or several features. Features are used to describe different properties of the event or object, referred to as a data point, in the dataset [17, ch. 5]. The set of features describing a single data point is often represented as a vector, where each dimension represents a single feature; called a feature vector [18]. In the case of news articles, the dataset would be the set of separate news articles whereas a feature could be a letter or a word inside an article. A feature vector would for example be a vector containing all the

Figure 2.1: Example a two dimensional projection, using PCA, of 100 dimensional Glove vectors. Where the point expression is the vector calculated in Equation 2.3. The image is inspired by a similar image of word2vec vectors in [16].
2.2. Machine Learning

separate letters or words of a single article. Now, Machine learning algorithms can be defined as algorithms that are used to learn something from data [17, ch. 5]. There are three different kinds of machine learning:

The first kind is **Reinforcement learning**, which can be introduced by a real-world example; a dog trainer teaching a dog to obey the command “sit” without showing it how. The dog trainer might say “sit”, and the dog rolls around, jumps or runs away, which leads to the trainer trying again and again. However, at some instance when the trainer says “sit” the dog sits, maybe by accident, and the trainer feeds it candy. Since the dog is rewarded for sitting when the trainer said “sit”, it slowly learns to obey the command. In the case of machine learning, the learning algorithm would be the dog, the data would be the actions of the dog, and the only thing given to the system is a reward (the candy) for completing the task or taking a certain action (sitting). The algorithm can then, by exploration (such as running, jumping or rolling) find a solution, given that it is rewarded when completing the task, or punished for doing it wrong [19], [20, ch. 1].

The second kind of learning is called **Supervised learning**. Problems solved by supervised learning algorithms are stated in terms of a dataset containing features, and for each data point there is a corresponding label. The task for supervised learning algorithms is to predict the label of a data point given the features of the data point [17, ch. 5], [20, ch. 1]. This is useful given new entries, where the label is unknown, since then the supervised learning algorithm can be used to predict the label. This is similar to how for example a parent might point at a lamp, while exclaiming “lamp”, when trying to teach its child the name of the object. Even though lamps can be very different in appearance, given time, the child will learn that the word lamp corresponds to all these different looking objects. The child will also be able to recognize new lamps it has never seen before, and knowing that they should be referred to as lamps as well. In this case, the child would be the machine learning model, while the parent represents the dataset (given the visual appearance of a lamp, the parent provides the label “lamp”).

Finally, the last type of machine learning is **Unsupervised learning**. When performing unsupervised learning, the algorithm only has access to a dataset of entries, but no labels. The goal is instead to find structures within the dataset, for example distribution of the entries across different features or groups of similar entries (clustering) [17, ch. 5], [20, ch. 1].

In addition to these types of machine learning, there can be combinations where properties from the different types intertwine. However, this study utilizes only supervised learning, and therefore it is covered more in-depth in the remainder of this section whereas the other types are left for the reader to explore further in other literature.

### 2.2.1 Supervised Learning

A supervised machine learning algorithm is used to solve some task by the use of a labeled dataset [17, ch. 5], [20, ch. 1]. However, these tasks can be formulated in many different ways. Therefore, the tasks are categorized into different task types. The two most general task types are presented within this section.

Classification tasks are stated as learning a function \( y = f(x, \Theta) \) where \( x \) is an input feature vector, \( \Theta \) is a set of model parameters and \( y \) is a label which, in this context, is called a class. For a classification task, there is always a finite set of classes [17, ch. 5], [20, ch. 1]. A simple example would be the task of classifying the subject of an image. Here \( x \) would be the pixels of an image and \( y \) would be the predicted subject of the image.
Regression tasks, similarly to classification tasks, also try to find a function $y = f(x, \Theta)$, but here $y$ instead represents a continuous variable. Therefore, there is an infinite set of values $y$ could be [17, ch. 5], [20, ch. 1]. A typical regression task would be trying to predict a stock value, given for example its most recent previous values, CEO and partner companies stocks.

Training, Validation and Test Set

The dataset used for supervised learning is often divided into three parts; the training set, the validation set and the test set, with no overlap between the sets. The training set is used by the model to fit the function $f$ as well as possible to the data points within it. However, sometimes the model is too simple, which results in $f$ not being able to adapt to the training data enough (underfitting). It could also be too complex, which could make the function $f$ perfectly fit with all training points, without performing good when used on new data (overfitting). To cope with these types of problems, most machine learning algorithms have hyperparameters. These impact the model’s behavior in different ways, but are fixed settings that are not tuned by the training itself. To be able to determine how to set the hyperparameters of a model to achieve as good performance as possible, multiple models with different settings of the hyperparameters need to be trained and evaluated. However, the training set can not be used for this evaluation, since it will not show, for example, if the model has overfitted or if it is performing well. This is what the validation set is used for, as it consists of data points not directly used to update the model’s parameters during training. Therefore the validation set can, for example, be used to give an indication of when the model starts to overfit, by comparing the model’s performance on the training and validation sets. The final set, the test set, is used to estimate the generalization after training, and may be used only once. This generalization error shows how well the model is estimated to perform given new data in the same problem domain. The validation set cannot be used for this purpose, since it has influenced for example the settings of the hyperparameters [17, ch. 5].

2.2.2 Support Vector Machines

A Support Vector Machine (SVM) is in its essence a binary classifier, using a decision boundary to distinguish between the two classes, as seen in Figure 2.2. In this example, the classes are circles and triangles. A new incoming data point will be classified by the SVM as either a circle or a triangle depending on which side of the decision boundary it is located. The training of the SVM is essentially finding a good location for its decision boundary, using the training data. In particular, the algorithm tries to maximize the margin between the classes. In Figure 2.2 this means making the distance between the decision boundary and the class boundaries (the dashed lines) as large as possible. Additionally, the data points that define the class boundaries are called support vectors, and are framed by a black square in Figure 2.2 [21], [22, ch. 4].

An SVM can use either soft or hard margins. If the SVM uses hard margins it does not allow any data point in the training set to be misclassified. In the linear case this means that if the classes are not linearly separable it is impossible to train the model as it will fail to converge. The solution is to use a soft-margin SVM, where a slack variable is introduced which allows some misclassification in the training set. In Figure 2.2, this would be equivalent of allowing some squares or circles outside the class boundaries defined by the support vectors, and even on the wrong side of the decision boundary. This can also result in a more robust model as it can create a decision boundary which is less dependent on noise in the data [21], [22, ch. 4].

Although SVM:s in their nature are binary classifiers, methods have been developed which allow them to be used for problems with multiple classes. One method commonly used is the one-vs-all method where separate binary SVM:s are trained for each class, such that they
separate the current class from all other data points in the dataset. Then for a new data point, all SVM:s are evaluated and the SVM giving the highest score decides the class [21].

![Decision boundary and Margin](image)

**Figure 2.2:** An SVM in a two-class problem; circles and triangles. The support vectors are marked by being surrounded by a black square border.

### Kernel Trick

The *kernel trick* is a method for implicitly mapping a feature vector into a high dimensional space but still keep the computational complexity low. The base of the kernel method is kernel functions. A kernel function is defined as

\[
K(x_i, x_j) = \phi(x_i)^T\phi(x_j),
\]

where \(x_i\) and \(x_j\) are feature vectors and \(x_i, x_j \in \chi\) where \(\chi\) is the input space. The function \(\phi\) is defined as

\[
\phi : \chi \rightarrow \mathcal{V},
\]

where the target is to have a space \(\mathcal{V}\), such that the points are linearly separable in that space. This space is of a higher dimensionality than the input space resulting in the model having more freedom in creating the decision boundary [22, ch. 6].

### 2.3 Neural Networks

Neural networks consist of multiple layers, where each layer performs operations which transform the data as it flows through the network. Commonly the layers of a neural network are split into three categories; input, output and hidden layers [23, ch. 1]. The input and output layer depend on the data and the task of the model. Where the input layer performs
2.3. Neural Networks

no computation, it simply passes the data representation into the model. Therefore, it is usually not counted when determining the number of layers in a model [17, ch. 6], [23, ch. 1]. In contrast, the output layer performs computations which make a linear transformation from the model’s internal feature representation into the output space. All layers between the input and output layers are the so-called hidden layers, which transform the data into a feature space representation [17, ch. 6].

The main type of computation in the layers of a neural network is linear transformations, which can be represented as matrix multiplications. Commonly an activation function is also applied to the result of the linear transformation of each layer. The purpose of the activation function can be, for example, to produce a non-linear mapping between the input and the output of a layer. One layer of a neural network can therefore be defined as

$$y = g(W^T x + b),$$

where $W$ is the weight matrix of the layer, $x$ a vector of inputs, $b$ is a vector of bias terms and $g$ is the activation function [17, ch. 6]. Most of the time $g$ is applied element-wise to the vector resulting from the matrix multiplication [17, ch. 6], [23, ch. 3]. Note that the weights and the bias terms are parameters which are learned from the data. The vector $y$ is the output of the layer in the form of a vector of scalars.

A neural network can also be represented as a computational graph, where each node is a computational unit. A layer is then represented as a set of nodes, where the number of nodes is the same as the amount of outputs of the layer [23, ch. 1]. As these layers can be connected in multiple different ways, different layer names are used to specify the purpose and connection type of a certain layer. For example, the most common layer type is called fully connected layer. The name originates from the fact that all inputs to the layer are connected to all nodes in the layer [23, ch. 8]. An example of fully connected layers can be seen in the network shown Figure 2.5:A. Since the output of each node in the fully connected layer depend on all inputs, it is trivial to write a fully connected layer using the definition of Equation 2.6. Assuming a fully connected layer with an activation function $g$ which is applied element-wise; the calculation of a single node in the fully connected layer can easily be written as

$$y_i = g(W_{i,j}^T x + b_i),$$

where $i$ is the row. The asterisk denotes the use of all elements along the dimension. $y_i$ is then the output, a scalar value, of the $i$:th node of the layer.

When viewing a neural network as a computational graph, a fully connected layer means that all outputs from the nodes in the previous layer, are used as input to each node in the next layer. Since the output of each node in a later layer depend on the output from an earlier layer, it is trivial to write a fully connected layer using the definition of Equation 2.6. Assuming a fully connected layer with an activation function $g$ which is applied element-wise; the calculation of a single node in the fully connected layer can easily be written as

$$y_i = g(W_{i,j}^T x + b_i),$$

where $i$ is the row. The asterisk denotes the use of all elements along the dimension. $y_i$ is then the output, a scalar value, of the $i$:th node of the layer.

2.3.1 Activation Functions

An activation function is a fixed non-linear function, where in most cases both the input and the output are vectors. The effect of an activation function on the neural network differs depending on the layer to which it is applied. If it is used in a hidden layer it creates a non-linear mapping which allows the neural network to create arbitrary decision boundaries and
therefore solve non-linear classification problems. In the output layer the activation function is more coupled with the loss function and therefore the task performed by the model. Using an activation function on the output layer converts the features space of the model into an output space which is more easily interpretable. Activation functions are commonly applied elementwise; however, some activation functions like for example softmax take the entire vector into account when producing the output [17, ch. 6], [23, ch. 1], [23, ch. 3].

ReLU

The Rectified Linear Units (ReLU) function

\[ z_i = g(y_i) = \max(0, y_i), \quad (2.9) \]

is commonly used as the activation function in the hidden layers of a neural network. A visualization of this function can be seen in Figure 2.3. Here, \( y_i \) is only the output of one of the nodes in a network layer, however, the activation function is applied to all nodes of that layer in the same way. The reason being that ReLU is similar to a linear function it also has the benefit of being easy to optimize. Research has also been made into improving ReLU, which has led to variants such as leaky ReLU. Two other activation functions are the hyperbolic tangent and logistic sigmoid. These where common for hidden layers before ReLU but they have the problem of vanishing gradients, as when the absolute value of the input grows it becomes saturated. Saturated in this case meaning that further increase to the value results in a minimal change to the output, and therefore the gradient is close to zero. The small gradient makes the gradient based learning commonly used in neural networks difficult [17, ch. 6].

![Figure 2.3: The Rectified Linear Units function, providing a non-linear mapping between its input \( y_i \) and output \( z_i \).](image)

Softmax

Another common activation function is called the softmax activation function, which is defined as

\[ p_k = g(y, k) = \frac{\exp(y_k)}{\sum_j \exp(y_j)} \quad (2.10) \]
where \( y \) is the output from all nodes of a network layer. The output \( p_k \) is calculated for each node output \( y_k \) where \( k \) represents the index in \( y \). In addition, \( p_k \) is bound between 0 and 1, and satisfies the condition \( \sum_k p_k = 1 \). The softmax activation function is often used on the output layer of the network, and given the properties of the softmax output \( p \), this makes it possible to interpret the output as class probabilities where each \( p_k \) corresponds to a class \( c_k \) [20, ch. 5].

### 2.3.2 Deep Learning

Neural networks are commonly stated as universal function approximators. The reason being that even a fully connected feedforward neural network with only one hidden layer, using a non-linear activation function, can approximate any reasonable function. The caveat being the need for often quite a large amount of nodes in that hidden layer. The main problem of having a large amount of nodes is the amount of parameters which needs training, making the learning task harder. So even though it is theoretically possible to use a neural network with a single layer, it is most of the time not practical. Multiple hidden layers can be used to reduce the number of parameters to train, by using fewer nodes in each layer [23, ch. 1]. A neural network with multiple hidden layers is referred to as a deep network. There is a multitude of different types of layers with different purposes which can be combined in deep network architectures.

### 2.3.3 Convolutional Neural Networks

A convolutional neural network (CNN) is a neural network consisting of at least one convolutional layer [24]. This section presents how this specific layer type works, and in addition, it explains some of the most common layer types used inside deep CNNs.

**Convolutional Layer**

Convolutional layers are used to process data which are in an array- or matrix-like form; examples include images, videos, texts, and audio signals. They are based on the mathematical operation convolution which, in its essence, takes two input signals and produces a third output signal. For example, assume we have an audio signal of a song, and the goal of filtering out the high frequencies to keep only the bass notes. To achieve this goal a convolution kernel, or just kernel, can be constructed in such a way that the result of a convolution between the kernel and the audio signal is the desired low frequencies (a low-pass filter). The good thing about the kernel is that the same kernel may be used for different audio files and through convolution produce their corresponding low frequencies. In the case of working with input to a machine learning system, the input signals are often discretized, finite and multidimensional. For such signals and in the simplest multidimensional case, that of two dimensions for the input \( I \) and the kernel \( K \), the convolution operation is defined as

\[
S(i, j) = (I * K)(i, j) = \sum_{m} \sum_{n} I(i - m, j - n)K(m, n)
\]

where \( S \) is the output; also known as a feature map [17, ch. 9]. In a convolutional layer, the input comes from the previous layer output, and the weights of the kernel are learned in similar fashion to that of the weights of a fully connected layer. The final output of the layer is obtained by applying an activation function to the feature map. Any neural network containing at least one convolutional layer is called a convolutional neural network [24].

Moreover, a kernel in a convolution layer can have a stride. When performing a convolution in two dimensions, the operation can be seen as flipping the kernel (rotating the matrix 180 degrees) and stepwise sliding it across the input matrix, performing element-wise multiplication in the overlap, and then adding these products together to produce one value in the
2.3. Neural Networks

feature map. An example is shown in Figure 2.4. The kernel moves in the direction of the dashed arrows, jumping down to the next row when the end of the row is reached. Strides are how much the kernel should move in each of these steps [17, ch. 9]. In the example, the stride is one, producing a 3x3 feature map. If the stride instead would be two, it would result in a 2x2 feature map, as the kernel would take two steps each time it moves across the input. Naturally, an even larger stride results in an even smaller feature map and vice versa.

Furthermore, one advantage of using a convolutional layer is that of parameter sharing. Since the kernel weights stay the same independent of the placement of the kernel during a convolution, only the weights of the kernel need to be stored. This can be significantly less than in a fully connected layer, where a weight for each element in the input need to be stored. In addition, similarly to how the low-pass filter in the example earlier could be used to filter out low frequencies of an audio file, these kernels can learn to detect specific features in the input array. For an image, this could for example be edges and corners [24].

![Diagram of convolution operation]

**Figure 2.4:** A convolution operation between an input 4x4 matrix (Input) and a 2x2 kernel. Here, the kernel flipping is left out, since when the weights of the kernel are learned, it means that the kernel flipping also can be learned implicitly if necessary (in this case, it is instead an operation called cross-correlation).
2.3. Neural Networks

Pooling Layer

A pooling layer is commonly used directly after a convolutional layer. It somewhat similar to the convolution operation, in that it can be seen as a window sliding across the input to the layer. This means that strides also can be applied to the pooling layer. However, the operation performed in each step is entirely different. Two common types are max pooling and average pooling. For max pooling, the maximum value of the input in each window region is propagated to the output of the layer. For average pooling, the average of the elements in the input of the window region is instead calculated and used as output.

There are several reasons why a pooling layer might be used. For example, it makes the network invariant to small translations in the input. Another positive effect is that the pooling layer reduces its output, in comparison to its input, and thereby reduce the number of parameters needed in the following layers of the network [17, ch. 9]. Since the output is reduced, the goal is to keep only the most relevant features.

Dropout

Since deep networks are usually complex, there is a risk that they will overfit to the training data, even with good hyperparameters. To combat this, one idea might be to train several simpler networks on random subsets of the training data. Then, for a given test data point, each of the trained networks predicted class probabilities are considered when determining the class of the test data point. Using several models together to determine the class of a given data point is referred to as bagging. One advantage is that there is less risk that the combined bagging model can overfit, since each of its member classifiers can be simpler models. However, training more complex models in this way would take a lot of time and put huge constraints on the systems where they would be used [17, ch. 7].

Dropout is used to adapt some of the advantages of bagging into a single deep neural network. It works by excluding (dropping) a random subset of the nodes in a layer, and may be applied to each layer of a deep network, even the input layer. The fraction of nodes excluded for a certain layer is called the dropout rate, and usually is around 0.5-0.8 for a hidden layer. This random dropout can be done many times during training, often before each minibatch of training samples. In each of these instances, the network can be seen as a new, simpler network as it is missing a subset of its nodes and connections. In turn, this also forces each sub-network to learn to classify the inputs to the network. In Figure 2.5 an example of a sub-network is shown at a specific training instance. After training, all nodes are used, resulting in multiple paths through the network stemming from the different sub-networks being able to classify each input. This is very similar to bagging, except that in this case, the weights of each sub-network are shared since they are used by many other sub-network configurations. In addition, the number of sub-networks trained could be much greater using a similar amount of training time, since not all sub-networks need to be trained from scratch because of the weight sharing [17, ch. 7], [25].

Batch Normalization

Several problems arise when the depth of a neural network is increased. As stated previously, with deep neural networks, there is always a risk of overfitting, due to increased adaptability. Furthermore, as the parameters of the layers are updated for each training step, the distribution of the input activations to each layer changes due to this update; referred to as the Internal Covariance Shift. As this requires the model to continuously adapt to the new distribution, it slows down the convergence time, making the model slower to train. In short, if problems could be solved, it would speed up training time, as well as making the model less likely to overfit [26].
2.3. Neural Networks

One approach to deal with internal covariance shift is called Batch Normalization, which has the added benefit of regularization. Here, Batch refers to the use of minibatches during training. A batch normalization layer normalizes each of its input’s dimensions over a minibatch, to have zero mean and a variance of one. In addition, it introduces the possibility for the layer to scale and shift the normalized input. The reason for this is to avoid limiting the output to certain regions of latter activation functions working in series with the batch normalization layer. For example, without the scale and shift, for the sigmoid activation, the output would be located in the linear region of the sigmoid, unable to produce a non-linearity [26].

As an example of batch normalization, assume we have all input vectors to a batch normalization layer

\[ B = \{x_1, ..., x_m\} \] 

(2.12)

of a minibatch of size \( m \). Here, \( x_i \) represents the \( d \)-dimensional output vectors from the previous layer. Then, batch normalization works by first finding the minibatch mean

\[ \mu_B = \frac{1}{m} \sum_{i=1}^{m} x_i, \] 

(2.13)

and the minibatch variance

\[ \sigma_B^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2, \] 

(2.14)
where the square operation is element-wise. Secondly, each input vector $\mathbf{x}_i$ is normalized by

$$
\hat{\mathbf{x}}_i = \frac{\mathbf{x}_i - \mu_b}{\sqrt{\sigma^2_b + \epsilon}}.
$$

(2.15)

Finally, the output of the batch normalization layer is given by

$$
y_i = \gamma \hat{\mathbf{x}}_i + \beta,
$$

(2.16)

where $\gamma$ and $\beta$ are element-wise scale and shift respectively. The values of these vectors are learned during the training of the network [26].

### 2.3.4 Training and Tuning

There are many different issues regarding training a neural network. First of all, the question of how to train the network arises. More specifically, how can we update the parameters $\Theta$ of a classifying network $y_i = f(\mathbf{x}, \Theta)$ where $y_i$ is the predicted class, and $\mathbf{x}$ the label? First off, some way to determine how well the network is performing during training is needed; for example a loss function. Then, we need to use the performance measure to decide on how to update the model weights, which is exactly the purpose of an optimizer. Both of these concepts are explained further within this section. Furthermore, concepts regarding how to set the hyperparameters of a neural network and how to deal with overfitting are introduced.

**Batches and Epochs**

When training a neural network the training data is often iterated over several times to update the model weights, by for example gradient descent. These iterations are called epochs, and the full set of training examples is called a batch. However, computing the weight updates can be very costly in terms of computations when using the full batch. To combat this, it is common to instead use minibatches, by dividing the full batch into smaller subsets. For most algorithms it is more efficient in terms of computations to compute approximate weight updates for each minibatch, rather than the exact updates by using the full batch [17, ch. 8]. There are several other advantages to using minibatches; it saves memory, since not the entire batch need to fit in memory at the same time, and it can introduce regularization which helps to prevent overfitting given that the minibatches are sufficiently small [27].

**Loss Function**

When training a network, the goal is to tune the parameters $\Theta$ of a function $\mathbf{p} = f(\mathbf{x}, \Theta)$ to get as close to the desired output $c$ for the input feature vector $\mathbf{x}$ as possible. Here $\mathbf{p}$ is the output vector which is as long as the number of classes, and each dimension corresponds to a class probability; the highest one determines the prediction of the network $(\max(\mathbf{p}))$. The question is, how do we determine how well a network is performing during training? Maybe more important, how can we measure that the parameters $\Theta$ are updated in such a way that the model performance increases?

In order to solve these problems, a loss function is used. One of the most common ones is called the negative log-likelihood (NLL), and is defined as

$$
\text{NLL}(\mathbf{y}) = -\log(p_c),
$$

(2.17)

where $p_c$ refers to the class probability of the desired class $c$ in the output vector $\mathbf{p}$. The total NLL loss for all training examples is then found summing all NLL losses for all input feature vectors to the network, called the cross-entropy loss, is defined as

$$
L_{\text{tot}}(\mathbf{P}) = \frac{1}{N} \sum_{i=1}^{N} \text{NLL}(p_{ci}),
$$

(2.18)
where \( P \) is the set of class probability vector outputs of the model given an input set of size \( N \). To gain the initial output probabilities \( p \) from the network, often the softmax activation function is used [17, ch. 5], [20, ch. 4], see Section 2.3.1.

**Gradient Descent for Neural Networks**

The parameters of the neural network are most commonly trained using gradient descent by calculating the gradient of the loss, as calculated by the loss function. Then the backpropagation algorithm is used to propagate the gradient of the loss through all of the layers of the network. Where backpropagation uses the chain rule for derivatives to calculate how each layer affected the final loss and thereby how the parameters of that layer should be tuned [23, ch. 1].

**Optimizer**

The goal of the optimizer when training a neural network is to minimize the loss function. Most commonly stochastic gradient descent (SGD) and variants of SGD are used for training neural networks. The SGD optimizer updates the parameters, for mini-batch \( t \), \( \Theta_t \) using

\[
\Theta_t = \Theta_{t-1} - \alpha g_t, \quad (2.19)
\]

where \( g_t \) is the gradient of the loss, \( \alpha \) the learning rate and \( \Theta_{t-1} \) the previous parameter values. The learning rate can for gradient descent based optimization be seen as the step size taken in the negative gradient direction. SGD differs from gradient descent by using the gradient of the loss for a mini-batch instead of the gradient for the entire training set, where each mini-batch is sampled from the training set. It can also be used for online learning with one training data point at the time, which can be seen as each mini-batch is of size one [17, ch. 5].

SGD can also be used with momentum, a method which increases the rate in which the model learns during training [17, ch. 8]. Let \( p \in [0,1] \) be the momentum of the model, then the current velocity can be defined as

\[
v_t = pv_{t-1} + g_t, \quad (2.20)
\]

where \( v_{t-1} \) is the velocity of the last mini-batch. Then instead of using the gradient \( g_t \) in Equation 2.19 to update the parameters the velocity \( v_t \) is used instead. There are some variants of the momentum but this one is used in the SGD optimizer of Pytorch [28]. There are multiple cases where momentum works well. One example is when moving over an area with weak gradient signals but all in the same direction, then the momentum will build up and it will move faster than if no momentum was used. It also is useful for reducing the impact of the noise caused by each mini-batch, and therefore also the gradient, being a sample [17, ch. 8].

Adam is a common optimizer with an adaptive learning rate. Adaptive learning rate in this context refers to the optimizer having individual learning rates for each parameter of the model [17, ch. 8]. In the case of Adam, these are calculated based on estimations of the first and second moments of the gradient. In this case the moment is calculated using linear interpolation of the gradient and the square of the gradient respectively. Such that the first and second moment for mini-batch \( t \) are defined as

\[
m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \quad (2.21)
\]

and

\[
v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \quad (2.22)
\]
respectively. Where \( g_t \) is the gradient of the current mini-batch, \( m_{t-1} \) and \( v_{t-1} \) the first and second moment of the last mini-batch respectively. The coefficients \( \beta_1, \beta_2 \in [0, 1] \) are parameters of the optimizer. Then bias-correction is applied to both the first and second moment, using

\[
\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.23}
\]

and

\[
\hat{v}_t = \frac{v_t}{1 - \beta_2^t}. \tag{2.24}
\]

Finally the parameters of the model \( \Theta_t \), for the current mini-batch \( t \), are updated using

\[
\Theta_t = \Theta_{t-1} - \frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{2.25}
\]

where \( \alpha \) is the learning rate (step size) and \( \epsilon \) a small value added for numerical stability [29].

Early Stopping

One major problem with training complex neural networks is that they tend to overfit to the training data. One way to spot that a network has overfitted is to observe how the errors change over the training epochs for the training and validation sets. Both errors would decrease to a certain point whereafter the validation error would go back up again, and the training error would continue to fall. This is not good, since the validation error is a kind of estimate of the test error; how well the model would perform in general. However, there is a simple and common solution to this problem; the use of early stopping. The basic idea is to stop at the point where the validation error increases again, in other words, to find the point where the network produces the smallest validation error [17, ch. 8], [20, ch. 5]. The naive approach would be to keep track of the validation error at the end of each epoch and stop when the validation error goes up. Then, the weights of the previous iteration would be returned, which would correspond to the network with the lowest validation error so far. However, since the validation error often is noisy, small peaks are common while the overall trend still might be a decreasing error over several epochs. This is a problem, since using the naive approach would probably stop the training early, due to this noise. As a result, often a patience of several epochs are used. This refers to the number of epochs that should be run after the best validation error is found. Then, if a model with a lower validation error is found, it is set as the new best model and the training runs for at least another patience amount of epochs. If no lower validation error is found, the model with the lowest validation error is returned, from a patience amount of epochs back [30].

Hyperparameter Tuning

As mentioned in Section 2.2.1, hyperparameters are settings of a machine learning model that are fixed during training, and could for example impact the model’s ability to fit the training data. For example, for an SVM the slack variable is a hyperparameter [21], and for a CNN it could be the kernel size or the stride [17, ch. 9], see also Section 2.3.3. However, these parameters can be tuned by testing several different settings and evaluating on the validation data. There exist several different approaches to search for these parameter settings; two of the most common ones are grid search and random search. Grid search is a simple and common approach to hyperparameter tuning. As an example, assume one hyperparameter should be tuned for a machine learning model. Then a set of values to test is firstly specified. One approach to this is selecting values on a logarithmic scale, to cover a large area of possible values. When the interval has been selected, grid search refers to testing all values in the specified interval, for the machine learning algorithm of
interest, and find the setting which gives the best performance. The grid search may then be repeated in an interval more closely to the best performance to tune the hyperparameter even further. If more than one hyperparameter should be tuned, the number of tests needed for grid search quickly escalates, since each combination of values in each of the hyperparameter intervals need to be tested [17, ch. 11].

Random search is an alternative to grid search. Here, different marginal distributions are instead defined for each hyperparameter setting. Then, to conduct a random search, for each hyperparameter, its corresponding distribution is sampled and used as a setting for training a machine learning model. This is repeated several times, resulting in multiple models with random settings of the hyperparameters. The search can be stopped at any time when the hyperparameters are deemed to be sufficiently good. Similarly to grid search, random search may be repeated in intervals that lie closer to the previously found hyperparameters, to find even better settings. It has been found that random search can be used to find a good set of hyperparameter settings much faster than grid search [31].

2.4 Performance Measures

How do we determine how well a machine learning model has solved its task? For supervised learning tasks, the predictions \( y \) of a model given features \( x \) from a dataset are compared to the correct class \( c \) by a performance measure. The remainder of this section presents a selection of performance measures common for machine learning model evaluation.

Before introducing the most common performance measures for classification tasks, it is useful to first define some basic concepts. To simplify, these concepts are first explained for machine learning algorithm solving a binary classification problem; a dataset containing only two classes. Then, an explanation regarding how these concepts generalize to more than two classes follow. In the binary case, for the purpose of the concept definitions, the classes will be called \textit{Positive} and \textit{Negative}. However, these names can be exchanged for any class name. The basic concepts in a binary classification problem are [32]:

- **True Positives (TP)** is the number of data points predicted as \textit{Positive} that was \textit{Positive} according to the label.
- **False Positives (FP)** is the number of data points predicted as \textit{Positive} that was \textit{Negative} according to the label.
- **True Negatives (TN)** is the number of data points predicted as \textit{Negative} that was \textit{Negative} according to the label.
- **False Negatives (FN)** is the number of data points predicted as \textit{Negative} that was \textit{Positive} according to the label.

In the best case, the sum of the TP and the TN should be equal to the total number of data points in the dataset used for evaluation (for example the test dataset). Anyhow, the sum of all four terms is always equal to the total number of data points in the evaluated dataset. Now, for a multi-class problem, these concepts are defined per class \( C_i \) where \( i \in \{1, ..., N\} \) and \( N \) is the total number of classes. Let the predicted class for a data point \( x_j \) be denoted as \( y_j \), and the correct class as \( c_j \) where \( j \in \{1, ..., P\} \). Here \( P \) is the number of data points in the evaluated dataset. Then the concepts above are defined per class according to [32] as

\[
TP_i = \sum_{j=0}^{P} [y_j = C_i, c_j = C_i]
\]  

(2.26)
2.4. Performance Measures

\[ FP_i = \sum_{j=0}^{P} [y_j = C_i, c_j \neq C_i] \]  
(2.27)

\[ TN_i = \sum_{j=0}^{P} [y_j \neq C_i, c_j \neq C_i] \]  
(2.28)

\[ FN_i = \sum_{j=0}^{P} [y_j \neq C_i, c_j = C_i] \]  
(2.29)

The brackets refer to a count operation, counting all elements in the evaluation dataset for which the conditions inside are true. Using the above definitions several classification performance measures can be defined. Two common ways of calculating performance measures for classification tasks having more than two classes are macro-averaging and micro-averaging [9], [33], [34]. Macro-averaging is defined by

\[ \text{macro } B = \frac{\sum_{i=1}^{N} B(TP_i, FP_i, TN_i, FN_i)}{N}, \]  
(2.30)

where \( N \) is the number of classes [32]. Furthermore, micro-averaging is calculated as

\[ \text{micro } B = B\left(\sum_{i}^{N} TP_i, \sum_{i}^{N} FP_i, \sum_{i}^{N} TN_i, \sum_{i}^{N} FN_i\right) \]  
(2.31)

[32]. In addition to the multi-class performance measures, performance measures for binary classification problems not using micro- or macro-average are expressed as

\[ \text{binary } B = B(TP, FP, TN, FN). \]  
(2.32)

For all three Equations 2.30, 2.31 and 2.32, \( B \) is defined as a function taking four arguments

\[ B = B(\varepsilon_{TP}, \varepsilon_{FP}, \varepsilon_{TN}, \varepsilon_{FN}), \]  
(2.33)

where for example \( \varepsilon_{TP} \) corresponds to one of \( TP_i, \sum_{i}^{N} TP_i \) or \( TP \) respectively for the three equations. \( B \), in turn, can be one of multiple different performance measures. Some of the most common performance measures are presented below in terms of the arguments of \( B \), which means they can be used for both multi-class (through micro- and macro-averaging) and binary classification problems [32].

**Accuracy** is the fraction of data points that were correctly classified. It is defined as

\[ \text{accuracy} = \frac{\varepsilon_{TP} + \varepsilon_{TN}}{\varepsilon_{TP} + \varepsilon_{FP} + \varepsilon_{TN} + \varepsilon_{FN}} \]  
(2.34)

[32]. The weakness of using accuracy is that it can be misleading for imbalanced datasets. Assume for example a dataset containing 100 patients which have been examined for fatal diseases. This dataset has two classes, fatal and non-fatal, which tells if the patient has a fatal disease or not. As it is usually more common to not have a fatal disease, assume there are 95 patients in the non-fatal class, and the remaining 5 in the fatal class. Then, the dataset is imbalanced as it contains more data points for the non-fatal class than for the fatal class. Now, if a model would predict 99 patients as not having fatal deceases, and classify one patient with a fatal disease correctly, it would result in a high accuracy (96%). At first this might seem good,
but it may have dire consequences; the four patients with fatal diseases wrongly classified as non-fatal would not be detected.

In the above example precision and recall might have been better options to measure the performance of the model for the imbalanced dataset. Precision is defined as

$$\text{precision} = \frac{e_{TP}}{e_{TP} + e_{FP}}$$  \hspace{1cm} (2.35)

and recall as

$$\text{recall} = \frac{e_{TP}}{e_{TP} + e_{FN}}$$  \hspace{1cm} (2.36)

[32]. The values of precision and recall are between one and zero, where one is the optimal value for both measures. If these measures would be used instead of accuracy in the fatal disease example, then we would have gotten an indication that the model did not work so well. Precision would be high, since the one patient predicted as having a fatal disease does have a fatal disease. However, the recall would be low, since only one out of the total four with fatal diseases were detected. This would be an indication that the model does not work well for this imbalanced dataset.

There is sometimes a trade-off between precision and recall; it easy to make one obtain a higher score by lowering the score of the other. This is true in the previous example as well. The model is not good at finding patients with fatal diseases, yielding a low recall, however, the one it found was correct, which gives a high precision. A harsh model is used for the benefit of precision, but gives a lower recall. On the other hand, if the model would classify many patients as being sick, this would give a higher recall, since it would probably find all sick patients, but a low precision, since many healthy patients would be classified as being sick. This means that it is necessary to take both metrics in to account to be able to determine the performance of a model. To be able to more easily determine the combined performance of both metrics together, a single metric called $F_\beta$-score is commonly used. It is the harmonic mean of precision and recall, and is defined as

$$F_\beta = \frac{\frac{e_{TP}}{(1 + \beta^2) \cdot e_{TP} + \beta^2 \cdot e_{FN} + e_{FP}}}{(1 + \beta^2) \cdot e_{TP} + \beta^2 \cdot e_{FN} + e_{FP}}$$  \hspace{1cm} (2.37)

[32]. $\beta$ in this case can be any number, but a common metric is the F1-score, with $\beta = 1$.

In conclusion, assume as an example micro F1-score should be used as a performance measure. Then firstly observe Equation 2.31 and Equation 2.33 in unison; the arguments of $B$ should be

$$e_{TP} = \sum_i^N TP_i, \ e_{FP} = \sum_i^N FP_i, \ e_{TN} = \sum_i^N TN_i \ \text{and} \ e_{FN} = \sum_i^N FN_i$$  \hspace{1cm} (2.38)

when micro averaging. Secondly, to specifically calculate the micro F1-score, $B$ in this case is substituted by the F1-score, see Equation 2.37, where $\beta = 1$. The final formula for the micro F1-score is then obtained by inserting the arguments from Equation 2.38 into the F1-score equation to obtain

$$\text{micro F1-score} = \frac{\sum_i^N TP_i}{(1 + 1^2) \cdot \sum_i^N TP_i + 1^2 \cdot \sum_i^N FN_i + \sum_i^N FP_i}.$$  \hspace{1cm} (2.39)

Other micro- and macro-averaged performance measures can be calculated in similar fashion, by combining the averaging formulas with the performance measure formula of interest through substitution of $B$. 

22
A method for visualizing the per class classification performance of a model is the *confusion matrix*. As the name suggests it is matrix and therefore, has two dimensions. An example of a confusion matrix with three classes is shown in Figure 2.6. As seen in the image the vertical-axis is the label, actual class, which is what the model should have predicted. Furthermore, the horizontal-axis is the predicted class, so what was actually predicted by the model. The idea is then to compare the label to the predicted class [35].

![Confusion Matrix Example](image.png)

*Figure 2.6: Example of a confusion matrix with three classes. Inspired by the example of Ting [35].*

In the example, Figure 2.6, each of the three class, 1, 2 and 3, has ten data points which were predicted by the model. Optimally, this confusion matrix would have a diagonal line from the bottom left to the top right, all with the value 10. As a result, all other positions would have the value zero and corresponding to all data points being correctly classified. In this example, if the label was one, for a data point, it is always correctly classified by the model. However, when the label was two, only four data points were correctly classified and the remaining six were classified as class three. Finally, if the label was three it was correctly classified half of the time, with a wrongly classified issues spread of the other two classes.

### 2.5 Interviews

This section introduces the major types of interviews, which are *structured*, *semi-structured* and *unstructured* interviews [36]. The advantages and disadvantages, as well as how they are commonly used, is covered for each of the interview types.

*Structured interviews* can be thought of as a guiding the interview subjects through a questionnaire. Structured interviews are planned out in detail before they start. As a result, the questions and their order are exactly the same for each interview, similar to a questionnaire [36]. Furthermore, the method of recording the responses to the questions is also standardized. The main advantage of this method, when compared to the other interview types, is a high degree of comparability, which comes from the well defined structure. However, this well defined structure also comes with disadvantages, such as the interviewer not being allowed to ask follow-up questions [37, ch. 6]. The main use of structured interviews is therefore when comparability is important or when specific questions needs to be answered [36], [37, ch. 6]. Therefore, it is a common method for descriptive research [37, ch. 6], which is research focused on determining what is happening in a certain situation at a specific point in time [37, ch. 1].

*Unstructured interviews* doesn’t use a predetermined list of questions, instead the discussion is more driven by the interview subject. The interviewer makes sure the discussion stays on topic but uses the answer of the subject to formulate the next question [36], [37, ch. 6]. According to Kothari [37, ch. 6], a standardized method of recording the responses is not nec-
2.6 Related Work

In this section an overview of work relevant for this thesis is described. In Section 2.6.2 the focus lies on work related to the specific task of classifying issues. Some of the sources used have focused on the classification of bug reports which can be seen as a subset of classifying issues. Because issues to a large part is a text description, the field of text classification is of importance to this work. Therefore, work relating to the classification of text in a broader sense is described in Section 2.6.1.

2.6.1 Text Classification

Y. Kim [5] proposed a CNN architecture which represents the input as a matrix of word embeddings, such that each row of the matrix is the embedding of one word. The convolution is then performed with a kernel of the same size as the word embedding, resulting in only a one dimensional motion in the convolution. The architecture was further extended by combining it with a character-level CNN when used for sentiment analysis of a multi-language twitter dataset, in a study by Zang et al. [7]. In that study it was shown that, for their dataset, the combined character- and word-level CNN improved the accuracy over the individual models.

Zhang and Wallace [38] studied the effect of hyperparameters for a word-level CNN, where the base architecture and parameters were the ones proposed by Kim [5]. The experiments where performed on nine different datasets. Zhang and Wallace [38] give multiple recommendations on which parameters to tune and also examples of ranges for the parameters. They saw that the size of the filter can have a large impact on the results and should be tuned. Also the amount of feature maps can have a large impact on the performance. In this thesis recommendations from Zhang and Wallace [38] are used when tuning the parameters of the CNN.

A character-level CNN uses the smallest component of a sentence which is similar to how CNN for images works on the pixel-level [8]. The first CNN architecture working with characters was introduced by Zhang et al. [39]. It showed promising results but was not the best performing on all datasets. Conneau et al. [8] created a deeper character-level CNN architecture, VDCNN, showing that as they used deeper versions of the architectures it improved the performance. The deeper versions of the architecture also outperformed the model created by Zhang et al. [39] on a majority of the datasets tested. Both models represent a sentence as a matrix, where one dimension is the encoding of each character. However, the encoding of
2.6. Related Work

characters differs as the first uses a sparse representation of 70-dimensional one-hot-encoded vectors [39] and the other uses 16-dimensional dense character embeddings [8].

2.6.2 Issue Classification

Jonsson et al. [12] investigated multiple classifiers on the task of dispatching issues based on teams. In the study it was shown that combining the best of the classifiers using Stacked Generalization (SG) increased the performance. SG is an ensemble method where the output of a set of classifiers is used as the input to one final classifier which determines the final class. The classification accuracy of four datasets of issues from a telecommunications company ranged between 57% and 89% for the SG model, which according to the authors was similar to the manual process used at that time. Finally the study also showed that using non-textual fields of an issue gave more promising results.

Helming et al. [40] proposed an approach for automatic issue assignment, but to a specific developer rather than to teams. A tool called UNICASE is used to represent what, in their paper, is referred to as work items. A work item is connected to a functional requirement, and functional requirements can be connected to other functional requirements. A functional requirement defines some desired feature of the system under development, and a work item defines something that needs to be done in order to implement this feature. Helming et al. [40] use this structure in their model-based approach. For an incoming work item, their model first finds the functional dependency connected to that work item. Then, all functional dependencies connected to that functional dependency are found. Finally, all resolved work items connected to all these functional dependencies are identified. The developer that had resolved the most of these work items is selected as the developer for the new work item. This model-based approach showed better performance than the SVM model. However, the model-based approach needs a connection between the issue and the functional requirements related to that issue. This connection does not exist in the data used for this thesis and is therefore not a viable solution.

Another method for issue classification, proposed by Zibran [33], is based on using topic modeling. In this case, the task was to classify issues into a set of 22 categories. The method used was supervised labeled latent dirichlet allocation (LLDA), which is a completely different approach than the one used in this thesis. Furthermore, Zibran used the comments of the issues as a part of the text used for classification. The comments in this thesis are considered to be posterior information, as these might have been added after the model would have performed the classification in a real world scenario. If that is the case, this can result in the model performing better than it would in reality.

Bhattacharya et al. [41] proposed an augmentation to the assignment of issues to specific developers based on so-called tossing graphs. Tossing graphs come from the concept of bug tossing, where a developer could for some reason not solve the task, and instead passes (tosses) it to another developer. The tossing graph used by Bhattacharya et al. [41] contains information about the portability for each new developer to be assigned, given the assigned developer, product, component and the amount of time since the tosses last activity in the project. The last activity is used to filter out developers which have left the project and is important because of the nature of open-source development. The system generates an ordered list of potential developers by first taking the three most likely developers according to the classifier. For the first and second developer the most likely new developer to be assigned is added to the list to the second and fourth place respectively, such that every other developer was picked by the classifier.

Research has also been made in the binary classification task of distinguishing bugs from non-bugs. Models like SVM, Naive Bayes and logistic regression, among other, have been applied
2.6. Related Work

to this task [34], [42]. Latent dirichlet allocation (LDA) has been used to extract topic which then was used as input to among others a Naive Bayes classifier [43]. The task of classifying bug and non-bug issues can be seen as an easier task than finding the correct team, as there might be many more classes in the case of teams.

Helming et al. [40] showed that their SVM classifier dropped in performance when evaluated using a history-based approach. The authors argue that this approach better reflects the actual amount of information known about a given item at the time of classification. Furthermore, according to Jonsson et al. [12] it is important to evaluate the model on issues sorted by the submission date, as this more realistic scenario gave lower scores than 10-fold cross-validation with stratification. Both of these studies have shown the importance of evaluating models on issues created after the training data to not artificially inflate the accuracy. In this thesis, all issues used for evaluation are created later than the last issue used for training.

In both [12]1 and [40], different types of neural networks were tried but not evaluated, as the training time was deemed too long. In a master’s thesis by Helén and Persson, a CNN was applied to the problem of assigning bug reports at a telecommunications company, but performed worse than a SVM classifier in the study [44]. The CNN architecture they used was the one proposed by Kim [5], which is also one of the architectures investigated in this thesis. However, instead of using externally created issues, as Helén and Persson, this thesis used internally creates ones. In further contrast, this thesis investigates the difference between using word- and character-level versions of Kim’s proposed CNN architecture, similarly to Zhang et al. [7], but in the context of issue classification.

SVM has been used as a baseline for text classification tasks in general [6], [45]. More specifically SVM has shown good results in the domain of issue classification [12], [40], [46], [47]2. As mentioned earlier, in the thesis by Helén and Persson, the SVM model outperformed their other models [44]. Therefore, it was decided to use an SVM classifier as the baseline in this thesis.

1In [12] the neural network is called multilayer perceptron
2In [12] the model is called SMO, which is a fast algorithm for training of SVM models
This chapter presents the software frameworks used to implement the investigated models. The systems used for training and evaluating the models are also specified. Furthermore, a description of the original dataset and the preprocessing used to construct the datasets used in the study is given. Finally, the experiments conducted to answer the research questions are explained. In short, different neural networks, mostly convolutional neural networks, a support vector machine and a human expert are tested as means of dispatching issues to the correct team.

3.1 Frameworks

The CNN models investigated within this study were implemented in Pytorch\(^1\). This is a high-level library to build neural networks using Python\(^2\). This framework was selected due to the fact that it is low-level enough to give the sufficient amount of freedom to implement complex network architectures, while being high-level enough to have implementations for a multitude of layers, optimizers and loss functions.

The SVM models and the evaluation metrics for all models were implemented using Scikit learn\(^3\). This is a Python library which implements many different machine learning algorithms, preprocessing, dimensionality reduction algorithms, evaluation metrics and much more. Furthermore, some of the preprocessing steps made use of the Python libraries NLTK\(^4\) and Gensim\(^5\). NLTK was used for lemmatization, tokenization and n-grams, while Gensim was used for stop word removal. Finally, the library Torchtext\(^6\) was used to generate mini-batches from the training, validation and test datasets for all CNN models.

\(^1\)https://pytorch.org/
\(^2\)https://www.python.org/
\(^3\)https://scikit-learn.org/stable/
\(^4\)https://www.nltk.org
\(^5\)https://radimrehurek.com/gensim/
\(^6\)https://torchtext.readthedocs.io/en/latest/
3.2 Hardware

All the experiments were conducted in two virtual machines (VM:s). The VM setups were:

- **Operating system**: Red Hat Enterprise Linux Server release 7.5 (Maipo)
- **CPU**: 6 physical cores of an Intel Xeon Gold 6132 CPU @ 2.60 Ghz simulating a single core each in the VM.
- **Memory size**: 32 GB

3.3 Initial Datasets

The aim, as described in Section 1.2, was to use machine learning to dispatch issues within a team-based organization; in this case a telecom company. To store the issues the company uses the issue tracking software Jira, which contains both current and historical issues. In addition, the telecom company has several active projects, where each active project has a set of issues and teams connected to them. Some of the projects are also grouped into larger sets of projects in a tree-like structure. The potential use for the investigated models of this thesis would be in each of the individual projects where all issues share the same set of teams.

To reflect the problem domain at the company as well as possible the main dataset, dataset A, only contains issues from one of the active projects. This project was chosen because it has the largest amount of historical issues and also has the highest rate of new issues being created. Where the higher rate made it a larger target for automation, as it would have a greater impact, and the larger number of issues results in more examples for the model. However, in order to better generalize across different projects a larger dataset, dataset B, was created for tuning the parameters of the models. This dataset contained the issues from multiple of projects, and dataset A was simply the issues from a single one of these projects. Dataset A also spans over a slightly longer time period, about a month longer. In addition to the data fields present in dataset B, dataset A also contained some project specific fields.

3.3.1 Contents of the Issues

The issues at the telecom company contained a number of different data fields. Some of the fields were the same for all projects, and some were specific to certain projects. Project-specific fields are commonly categorical and are in this thesis referred to as *tags*. These tags are added to issues dependent on the needs of a project. Out of the fields that were common for all issues, the following were used in some way to construct the two datasets:

- the **title**;
- the **description** describing the problem;
- the **creation date**, when the issue was reported;
- the **resolution date**, the date when the issue was resolved;
- the **status**, describing the state of the issue (such as to-do, doing, resolved or rejected);
- the **resolver**, the team which resolved the issue; and
- the **project**, the project for which the issue was created.
In addition, three tags were used specifically for dataset A. These tags were related to the physical location and technology used where the error occurred. They were used exclusively to answer RQ3, and added based on information gained from a human expert dispatching issues for that project. Dataset B contains multiple projects, therefore, the types of tags available are not the same for all issues. Because of this, dataset B does not contain any tags.

### 3.3.2 Construction of Datasets

Both datasets were created in the same way, where the data fields of the issues served different purposes in the construction. Firstly, the status field was used to save only the issues which had been resolved. This was done to avoid the problem where an issue had been wrongly dispatched to a team and later re-dispatched to another one, as this could cause errors in the model. Afterwards, the team field was used to remove issues missing a team. Since all issues then had a corresponding team, the team was set as the label of each data point in the two datasets. Furthermore, the title and description fields were merged into a single text field. This field was later preprocessed and turned into different feature representations used by the different models. Moreover, the project field was used to select the issues belonging to the specific project represented by dataset A. The tag fields relevant to this project were saved as categorical features specifically to dataset A, in addition to the other fields. Finally, the creation date and resolution date fields were used to select the time span for the issues in dataset A and B. For dataset B and A, no issue was created or resolved earlier than 2017-05-12, and no later than 2019-02-28 and 2019-03-29 respectively.

An example of the issue selection process can be found in Figure 3.1. Here, the original issue dataset is at the top, dataset B second, and finally dataset A. Row 2 and 3 from the original issues are selected for dataset B, since their status is resolved and they have a corresponding team that solved the issues. For dataset A, the project column is also considered. Only the issues from a single project are selected; in this case project 1. Both the creation and resolution date are used to select issues from the predefined time interval of each dataset, which in this example contain all issues. Both the creation and the resolution date of each issue are used to select the issues which are within the predefined time interval, in this example all issues.

An additional note is that the new Text Field in dataset B and A is a concatenation of the Title and Description fields, and that the Resolver field in the original dataset table is called Label in dataset A and B. Furthermore, the tag fields related to project 1 in the example are saved for dataset A, as seen in Figure 3.1.

The creation and resolution date fields had an additional purpose, to select the age of the issues that were used. This second purpose was to split dataset A into a test set and a training/validation set. All issues created after 2019-02-28 were used in the test set, and the issues resolved before that date were used in the training/validation set. This temporal split was to replicate a real-world scenario where issues in the future cannot be used to predict the issues of today, since they do not yet exist. An illustration of the split is shown in Figure 3.2. The region called Unused in the figure refers to the issues that were created before the resolution date threshold 2019-02-28 but solved after the creation date threshold 2019-02-28. These issues were purposely left out as the information inside these issues would not be complete at the time of training, if the model would have been used in a real-world scenario. For dataset B, no such temporal split was used since this dataset was only used for hyperparameter tuning and does, therefore, not require a test set.

Finally, the training/validation set of dataset A and the entirety of dataset B were split into two subsets each; a training set and a validation set. The split was random with no overlap, using an 80/20 split, such that both training sets contained 80% of the data and the remaining 20% was in the validation sets.
3.3. Initial Datasets

**Figure 3.1:** The selection of data points for dataset A and B from the original issue dataset. Dashes represent that there should be no value for that column at that point, and <MISSING> represents a value that is missing for some reason.

**Figure 3.2:** How dataset A was split based on the issues resolution and creation dates into a training/validation set and a test set.
3.4 Preprocessing

In this section two parts of the preprocessing performed are covered. Firstly, the raw feature reduction techniques are covered, which is a set of preprocessing techniques used to reduce the number of features in the data. Secondly, custom preprocessing techniques are covered; techniques which are specific for the data used in this thesis. Numerical feature reduction, however, is not covered in this section. Instead, as the numerical feature reduction differs between the experiments, it is described for each experiment independently.

3.4.1 Raw Feature Reduction

The raw feature reduction concerned the text field of dataset A and B. The outcome of this step was two new fields; one where each character of each issue was a feature (character-level), and the other where each word was a feature (word-level). To construct these two fields, a pipeline was set up to try different combinations of four preprocessing techniques; whether or not to perform lowercasing, remove stop words, keep only alphanumeric characters, or to lemmatize using an English dictionary. This testing of different preprocessing techniques was done independently for the word-level and the character-level fields.

The method for selecting which preprocessing techniques to use was to test different preprocessing settings, independently, on both the word- and character-level fields. The dataset used for this task was dataset B. Specifically, to compare the different preprocessing settings, models were trained using the training set and then the performance of the validation set was used to compare the different preprocessing settings. The models used for deciding on the preprocessing differs between the word- and character-level fields. For the word-level field, the word-level CNN and the SVM, created for the experiments, were used. For the character-level field only the SVM was used, since the character-level CNN was too slow to train for this purpose. However, it is worth mentioning that at least in the word-level case the best preprocessing settings was the same for both the word-level CNN and the SVM. The representation of characters to the SVM is based on TF-IDF exactly like for the SVM used in the experiment, but using characters instead of words as the tokens. The final preprocessing settings stemming from this empirical testing are presented in Table 3.1.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Character</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Stop word removal</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Only alphanumeric</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Lemmatization</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 3.1: The preprocessing settings for the character- and word-level datasets respectively, determined empirically using character-level SVM, word-level SVM and word-level CNN.

3.4.2 Custom Preprocessing

Dataset A and B were imbalanced since some of the teams in the dataset had solved only a hand-full of issues, while others had solved hundreds or thousands. To balance the datasets somewhat, data points labeled with teams which had solved less than 50 issues over almost two years in the training set were removed. Since, teams with so few issues solved were considered as inactive.

While random searching for the parameters of the character-level CNN in Experiment B, the computer ran out of memory. This was due to the fact a few issues contained extremely long computer generated output, two orders of magnitude more characters than the mean. As a
result, the number of numerical feature representation was larger than the 32 GB of RAM available in the VM. To be able to continue the parameter tuning, the character-level field for all data points were truncated to 10 000 characters.

3.5 Final Datasets Before Experiments

In this section statistics regarding the sizes, features and distributions of dataset A and B, as they were after the raw feature reduction and custom preprocessing steps, are presented. This is the versions of the datasets which then are used for all experiments. Firstly, the total sizes of the two datasets are displayed in Table 3.2 along with the relative sizes of their respective training, validation and test subsets. Secondly, in Figure 3.3, the distributions of data points, or issues, across the different teams are presented for dataset A and B. Observe that both datasets are imbalanced, with only three teams covering 47.34 % and 37.8 % of all issues in dataset A and dataset B respectively. Finally, the sizes of the issues in terms of words and characters are shown in Figure 3.4 and 3.5. The distributions are shown on a logarithmic x-axis, as there are big differences in the number of words and characters between the longest and shortest issues. The distribution across teams for the train, validation and test sets of dataset A, as well as the train and validation sets for dataset B, is shown in Appendix A. Also in Appendix A, is the distribution across teams of dataset A

<table>
<thead>
<tr>
<th>Dataset</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total issues</td>
<td>15506</td>
<td>38318</td>
</tr>
<tr>
<td>Training set</td>
<td>76.96 %</td>
<td>80.17 %</td>
</tr>
<tr>
<td>Validation set</td>
<td>19.46 %</td>
<td>19.83 %</td>
</tr>
<tr>
<td>Test set</td>
<td>3.58 %</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3.2: The size of dataset A and B, along with the percentage of the total issues in each of their respective training, validation and test subsets. In addition, the number of teams (classes) for the two datasets and the number of classes for each tag in dataset A. Finally, the vocabulary sizes are the amount of different tokens found in the training set of dataset A and B respectively.
3.5. Final Datasets Before Experiments

Figure 3.3: Distribution of issues across the teams in dataset A and B, after pre-processing. The two datasets are imbalanced, as there are classes containing thousands of issues at one end and close to the used cut off, 50 issues, on the other.
3.6 Evaluation

All models were evaluated using the performance measures micro accuracy, macro F1-score, micro F1-score and micro top-3 accuracy. In the case of this thesis, micro accuracy and micro F1-score will be equivalent for the conducted experiments, and will be referred to as just accuracy. Micro and macro averaging as well as the accuracy and F1-score are covered in detail in Section 2.4, while top-3 accuracy will be explained further in this section. The performance
measures were used both for parameter tuning and for comparing the different models. Finally, this section will cover a visualization designed for this thesis, referred to as a top-3 confusion matrix; it was created to further investigate the top-3 prediction performance.

Top-3 accuracy uses three most likely classes, as predicted by a model, instead of just the most likely class as in the ordinary accuracy measure. More precisely, a prediction is counted as correct if any of the three, according to the model, most likely classes are correct. As a comparison the ordinary accuracy measure could also be thought of as top-1 accuracy. Top-3 accuracy was especially useful for this thesis because of the large overlap between teams in competence areas, such that multiple teams could be equally viable for a task. Therefore, top-3 accuracy was determined to better reflect the real world usefulness of the model. Furthermore it was also useful to determine the suitability of the model as a part of a recommendation system, where the three best predictions are the recommendation made to the user.

Furthermore, for the evaluation the top-3 accuracy was selected as the main parameter for determining model quality. In concrete terms, the top-3 accuracy was the main determiner of which hyperparameters to use from the parameter tuning and which model that was the best one. For cases where the top-3 accuracies were similar, the other metrics were used to determine which one was better.

The top-3 confusion matrix was designed by combining the top-3 accuracy with a confusion matrix. This was done by using the top-3 predictions, the three most likely classes according to the model. In contrast, the ordinary confusion matrix only uses the most likely class. First, the top-3 predictions were recorded for all data points used for evaluation. Second, the top-3 predictions for all data points with the same label were counted; this results in a number for each predicted class for that label. This corresponds to a row in the matrix, similar to a row in the example confusion matrix in Section 2.4. However, because the top-3 confusion matrix uses three predictions of each data point the sum of a row is three times the number of data points corresponding to the label of that row. In contrast, the sum of a row in the ordinary confusion matrix is the same as the number of data points corresponding the label of the row. Furthermore, the rows were normalized, so each sum to one, to be more easily interpretable. Which was also useful because the dataset, as mentioned in Section 3.5, was imbalanced. Without the normalization each row would have a different sum, making it harder to compare the rows. Finally, because the model only could predict a class one time for each data point, the highest value in the top-3 confusion matrix, after normalization, was 0.33.

3.7 Experiments

The experiments conducted to answer the research questions are presented in this section. A mapping on how the research questions, given in Section 1.3, relate to the experiments is given in Table 3.3. In the first experiment, Experiment A, a SVM was implemented. As a SVM is faster to implement and run using the methods of this thesis, it was beneficial to implement and conduct this experiment first, as it could serve as a baseline on what to expect in the other experiments. Furthermore, it was deemed useful as a tool for evaluating the implementation of the preprocessing step. Experiment A was used to answer RQ2, together with Experiment B and C. Both experiment B and C were also used for RQ1, since the first experiment investigates the performance of word-level and character-level CNNs independently, and the second investigates the joint performance. Finally, all experiments are considered for RQ3, as the best of the different models are used to construct the final model, which in turn is compared to human performance measured in Experiment D.
3.7. Experiments

<table>
<thead>
<tr>
<th>Experiment</th>
<th>RQ1</th>
<th>RQ2</th>
<th>RQ3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B, C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A, B, C, D</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.3: An overview on how the conducted experiments relate to the research questions.

For parameter tuning all experiments, except Experiment D, used dataset B. Specifically, the training set was used for training and the validation set was used for evaluation. Furthermore, the validation set was also used for early stopping of the neural networks. When evaluating the hyperparameters, the metrics described in Section 3.6 were used. For Experiment D the parameter tuning was done directly on dataset A because the model relied on the tags only present in that dataset. In that case the model was trained using the training set of dataset A and the parameters were evaluated based on the performance on the validation set of dataset A.

Evaluation of the models was performed on dataset A; the models were trained on the training set and then evaluated on the test set. During training of all neural networks, the validation set of dataset A was used for early stopping. Once again, there is a slight difference with Experiment D; the models were evaluated on both the test set of dataset A and dataset A_D. The final evaluation of all models also used the metrics described in Section 3.6.

3.7.1 Experiment A - SVM

The pre-processed word-level field of dataset A was transformed into TF-IDF form using the `TfidfVectorizer`-class of Scikit learn. Additionally, token expansion was used in the transformation, such that the vectors contain one and two grams of the tokens. First, the training set was used to calculate the IDF-weights. Then, the validation and test sets were transformed into TF-IDF form using these weights.

The SVM was implemented using the `LinearSVC`-class of Scikit learn. Furthermore, the SVM has a parameter, C, which is a slack variables. A larger value of C results in a higher penalty for misclassification during training, reducing the number of possible margins. By using a value closer to zero the model is allowed to have more misclassifications, which in turn allows for more choices when selecting the margin. To maximize the performance of the SVM, the C parameter was tuned on dataset B. Tuning was performed using grid search over the set $C \in 0.3, 0.6, \ldots, 2.1$, the results from all iterations are shown in Appendix C. The top-3 accuracy differs only by 0.009 between the best and the worst result. Therefore, the other metrics were also taken into account when determining C. The model with the best top-3 accuracy had worse accuracy and F1-score than the model with third best top-3 accuracy. However, the accuracy and F1-score only differed by 0.004 and 0.0003, respectively. Therefore, the model with the best top-3 accuracy was chosen. Finally, the SVM was trained and evaluated, using $C = 0.6$ as given from the tuning, on the word-level TF-IDF representations of dataset A.

3.7.2 Experiment B - Word- and Character-Level CNNs

The architecture used for the CNNs in this experiment is shown in Figure 3.6. It is the architecture used in both [5] (for word-level), and in [7] (for both word- and character-level). The embedding vectors were set to be 50-dimensional. Consequently, the kernel width, the width of the kernels in the convolution layer, were also set to 50. All embedding vectors were randomly initialized by sampling each vector from a normal distribution of mean 0 and standard deviation of 0.05. A dropout rate of 0.5 was also used on the next to last layer, in other words, the feature maps above E in Figure 3.6. The loss was calculated using cross-entropy in
combination with softmax activation in the output layer, and the optimizer used was Adam, with an initial learning rate set to 0.001. Early stopping with patience 10 was used to decide when to stop training. In addition, the training datasets for both dataset A and B were divided into minibatches of size 128 when using the character-level field and size 256 when using the word-level field. The issues within each minibatch were padded to the size of the longest issue in that batch.

Figure 3.6: A: An example input to the word-level CNN. B: An example input to the character-level CNN. C: An embedding layer, replacing each input word or character with the corresponding vector. D: Using the kernel height and number of kernels presented in Table 3.5, D shows the feature maps resulting from these convolutions and after applying the ReLU activation function. E: 1-max pooling over the feature maps. This is a special type of max-pooling, which takes the largest value from each feature map and combines them into a vector. F: A fully connected layer connects the previous layer with the output layer, which is as wide as the number of classes.

In the first part of the experiment, the hyperparameters kernel height, the height of the kernel in the convolution layer, and the number of kernels were tuned. The number of feature maps generated from the convolutional layer is the same as the number of kernels used. The hyperparameter tuning was performed independently for both models using dataset B. The parameter searching algorithm called random search, described in the hyperparameter tuning part of Section 2.3.4, was used for the main part of the tuning. Where, for each parameter range, the algorithm sampled the values uniformly. For both models the random search was run two times, where the parameter ranges of the second run were chosen based on the result of the first run. The first run was stopped when it was clear which parts of the parameters were performing poorly and which were performing well. Then the parameter range was changed to remove clearly inferior choices and in some cases expanded, if the best values were close to the edge of the range. Because of the relatively short training time, in comparison to the character-level CNN, an additional grid search was performed for the word-level CNN to further improve the hyperparameters. The parameter range for the grid search was determined using the results from the random searches. The sets of parameters used for each step of the parameter search is shown in Table 3.4, for both of the models.
3.7. Experiments

<table>
<thead>
<tr>
<th>Model</th>
<th>Search</th>
<th>Kernel height Set</th>
<th>Number of Kernel Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word-level CNN</td>
<td>Random Search - first run</td>
<td>${1,2,\ldots,11}$</td>
<td>${10,20,\ldots,300}$</td>
</tr>
<tr>
<td>Word-level CNN</td>
<td>Random Search - second run</td>
<td>${1,2,\ldots,5}$</td>
<td>${20,40,\ldots,400}$</td>
</tr>
<tr>
<td>Word-level CNN</td>
<td>Grid Search</td>
<td>${1,2,3}$</td>
<td>${180,190,\ldots,250}$</td>
</tr>
<tr>
<td>Character-level CNN</td>
<td>Random Search - first run</td>
<td>${1,2,\ldots,11}$</td>
<td>${20,40,\ldots,300}$</td>
</tr>
<tr>
<td>Character-level CNN</td>
<td>Random Search - second run</td>
<td>${3,5,6,7,8,9}$</td>
<td>${120,140,\ldots,300}$</td>
</tr>
</tbody>
</table>

Table 3.4: The ranges used for the different steps of hyperparameter tuning for the two models. The kernel height set are values used by the algorithms for the height of the kernel of the convolution layer.

The final parameters are shown in Table 3.5. As described in Section 3.6, the top-3 accuracy is the main metric used to determine the best parameters. If two models were close in top-3 accuracy, the other metrics were also considered. For the character-level CNN the best and second-best result only differed by 0.002 in top-3 accuracy. Both models had similar parameters, only differing by 10 in the number of kernels and with the same kernel height. The model with highest top-3 accuracy also had the highest accuracy, however, the other model had a higher F1-score. Because of the small differences in the metrics, 0.0006 and 0.001 respectively, the choice was made to go with the model with highest top-3 accuracy. For the word-level CNN both the best and second-best, when rounded to five decimals, had the same top-3 accuracy. The choice of parameters where still trivial as the results where better across all other metrics for one of the models, which was then chosen. Similar to the results of the hyperparameter tuning of the character-level CNN, the only thing differing the best and second best model was a slight change in the number of kernels used. All the data generated from the parameter searches, which where used for these decisions, can be found in Appendix C. Finally using parameters determined during the tuning phase the models are trained and evaluated on dataset A.

<table>
<thead>
<tr>
<th>Model</th>
<th>Kernel Height</th>
<th>Number of Kernels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word-level CNN</td>
<td>2</td>
<td>250</td>
</tr>
<tr>
<td>Character-level CNN</td>
<td>4</td>
<td>280</td>
</tr>
</tbody>
</table>

Table 3.5: Parameters chosen for word- and character-level CNN, based on results from parameter tuning.

3.7.3 Experiment C - Multichannel CNN (Characters + Words)

The text based multichannel architecture is shown in Figure 3.7. It was constructed from the word- and character-level CNNs from Experiment B, and is very similar to the one presented in [7], with the addition of a dropout layer. The right channel of the network corresponds to the word-level CNN and the left channel corresponds to the character-level CNN. The embedding layers of each channel was initialized in the same way as in Experiment B; the weights were drawn from a normal distribution with a mean of 0 and a standard deviation of 0.05. To combine the channels a Concatenation layer is used, which concatenates the two output vectors from the the two channels. Note that, the last fully connected layer in each channel, before the concatenation layer in the multichannel architecture, was modified to no longer have the same number of outputs as there were classes. Instead, the number of nodes used here was tuned using random search. In addition, the dropout rate in the final dropout layer and the batch size when training were also tuned with random search, using dataset B. During the random search, values for the parameters are sampled uniformly from the sets shown in Table 3.6.
3.7. Experiments

**Figure 3.7:** The architecture of the multichannel CNN (characters + words). The right channel uses word tokens and the left one character tokens. The concatenation layer merges the two channels by concatenating their respective vector outputs.

<table>
<thead>
<tr>
<th>Search</th>
<th>Batch Size Set</th>
<th>Overlap Nodes Set</th>
<th>Dropout Rate Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Search - first run</td>
<td>[1, 2, 4, 8, 16, 32, 64]</td>
<td>[100, 110, \ldots, 300]</td>
<td>[0.3, 0.35, \ldots, 0.85]</td>
</tr>
<tr>
<td>Random Search - second run</td>
<td>[4, 8, 16, 32, 64]</td>
<td>[100, 110, \ldots, 300]</td>
<td>[0, 0.1, 0.2, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65]</td>
</tr>
<tr>
<td>Random Search - third run</td>
<td>[4, 8, 16, 32, 64]</td>
<td>[300, 310, \ldots, 600]</td>
<td>[0, 0.1, 0.2, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55]</td>
</tr>
</tbody>
</table>

**Table 3.6:** The ranges used for the different steps of hyperparameter tuning for the multichannel CNN, which uses character- and word-level representations.

For both the parameter tuning and later the training of the final model the Adam optimizer was used, with an initial learning rate of 0.001. Furthermore, cross entropy was used as the loss function and early stopping was used to decide when to stop training, with a patience of 10 epochs.

All results from the random search are found in Appendix C. The difference in top-3 accuracy between the two best models was only 0.0008, therefore, the other metric are used for determining the hyperparameters of this model as well. The model with the worse top-3 accuracy had a better accuracy and F1-score, with a difference of 0.013 and 0.028 respectively. Because,
the difference in the other two metrics was significantly larger than the difference in top-3 accuracy, the model with better accuracy and F1-score was chosen over the one with highest top-3 accuracy.

The final settings for these parameters were selected based on the top-3 accuracy on the validation set of dataset B; the batch size was set to 16, the dropout rate to 0.1 and the number of outputs in the fully connected layers of both channels to 380. Finally, the multichannel network was trained and evaluated on dataset A, using both the word- and character-level fields as inputs.

### 3.7.4 Experiment D - Human Expert

The goal of this experiment was to compare the issue dispatching of a human expert, at the telecom company, to a machine learning system created for the same task. Therefore, the first step was to gain an understanding of how the human expert dispatches issues. This was done by conducting an interview with two employees working with issue dispatching and therefore, considered to be experts in issue dispatching. Then, a experiment to test how one of the human experts dispatches issues was designed. Finally, the predictions of the human expert was compared to a machine learning system.

#### Interview

To gather information on how human experts dispatches issues at the telecom company, an interview was conducted with two employees, E1 and E2. Both employees, as a part of their employment performs issue dispatching, but for different projects at the company. Note that both projects are contained in dataset B, specifically, E1 is dispatching to the project represented in dataset A. The aim was to gain a general understanding of how issue dispatching is performed at the company, potential pitfalls and what information is important when dispatching. Because of the exploratory nature of the interview, the choice was made to conduct it as a unstructured interview.

The interview was then conducted over a conference call using voice and screen sharing, where both authors and both employees attended. The work was divided, such that one of the authors was focused on conducting the interview and the other on writing down the answers. Since it was a unstructured interview, no questions were prepared beforehand. Instead, the interview started with the employees walking through the user interface for submitting new issues, using screen sharing. Meanwhile, they also described what information they use when dispatching the issues.

The information used by the two projects was largely the same, but used quite differently. Both projects contained the same three tags, related to the physical location and technology used where the error occurred. These tags are from now on referred to as Tag1, Tag2 and Tag3. Both employees stated that these tags were used for dispatching, however, the project of E2 also contained a fourth tag, which was of higher importance. Furthermore, E1 stated that the tags are more important than the text for determining the team, even if both are used. In contrast, E2 uses the text more than the tags for determining the team. To conclude, the relative importance of text in the issues, when compared to tags, differs between projects.

Both employees also talked about how the teams are t-shaped, which means that a team has broad set of competences but also some distinct area of expertise. Causing the teams to have overlapping competence areas, which was already known to the authors. Therefore, the top-3 accuracy measure was described to the interview subjects and they were asked for their thoughts on the measure. They both agreed that this could be a good measure because of the overlap in team competences.
Experiment Design

The experiment was designed to see how a human expert, would dispatch different issues. As employee E1 dispatches issues for the project used to create dataset A, E1 was the natural choice for this experiment. Specifically, because the project chosen for dataset A had the advantage of many historical issues and a high rate of new issues, as described in Section 3.3. As employee E1 is the domain expert at dispatching the issues used in this experiment, E1 will be referred to as the human expert.

Two important considerations were taken into account when designing the experiment. Firstly, it was important that the test for the human expert is representative of how it works today. Therefore, the three tags Tag1, Tag2 and Tag3 were added to dataset A, as described in Section 3.3.1. Since, otherwise information which is commonly used by the human expert would not have been represented in the experiment, which would have violated the consideration. Secondly, it was also important to design an experiment such that the prediction of the human expert could be compared to the predictions of a machine learning system. By using data from the test set of dataset A for this experiment, it could be ensured that the model never had seen the issues beforehand. Hence, it could be evaluated on the prediction of these issues. Furthermore, it was decided the human expert would make a ranked list of three most likely teams for each issue. This allowed for comparing the top-3 predictions and, hence, the top-3 accuracy of the model and the human expert.

For this experiment the dataset A\textsubscript{p} was created; the dataset is a subset of 50 issues randomly sampled, without replacement, from the test set of dataset A. The information available to the human expert for these 50 issues were:

- Title
- Description
- Tag1
- Tag2
- Tag3

For each of the issues the human expert had the 30 teams, which has solved issues in the training set of dataset A, to choose from. Hence, this is the same teams as the model uses to make the prediction.

To reduce the risk that the way the information was displayed affect the results of the experiment, a system to generate images which looks similar to the interface used under normal circumstances was implemented. It worked by filling in custom text into a template screenshot of the Jira interface. Not only did this display information similarly to the normally used UI, but it also allowed high control of what information was displayed. Such that the issue only contains the information which is a part of the experiment and was present at the creation of the issue.

Since, the human expert was working at another location, a web server was created to host the experiment. This also allowed for easy creation of the interface using web technology. The landing page contained the list of issues with their respective test id, as shown in Figure 3.8. From the landing page the user could choose which test to do next and also get an overview of what was left to do.
3.7. Experiments

Figure 3.8: The landing page of the experiment performed by a human expert. White lines have not yet any prediction, yellow lines have some values on the predictions and green line has predictions for all three values. The gray row is the one currently hovered by the users mouse pointer. To the left the test id is shown and to the right are the currently made predictions or Not predicted if no prediction has been made.

Each issue had a prediction page where the generated image of the Jira interface was shown, only containing the allowed information. A fabricated example of how the information was displayed in a Jira interface image is shown in Figure 3.9. On the bottom of the page there are three searchable drop-down lists containing different teams, used for making the prediction, shown in Figure 3.10. Note that the team which solved the issue always existed in the list of teams, and were the same set of teams the machine learning models could choose between. An image of the prediction UI in its entirety is found in Appendix B.

Figure 3.9: The part of the UI used to display the information to the human expert during the experiment.
3.7. Experiments

(a) Choosing a team

(b) Team choosen

Figure 3.10: The part of the UI used to register the prediction of the expert, using three searchable drop-down menus. Predictions are then registered using the ok-button.

The experiment was performed over two weeks to allow the human expert to classify some issues between other work tasks. As a result, more issues could be classified than if the employee had to solve them all at once. Dedicating all the time needed at once would have been very impractical. Throughout the experiment continuous contact was held with the human expert, using e-mail. Finally, the result was stored as a csv-file, where each row contained the test-id and the three predictions made.

Comparison to Models

From the interview it was clear that the tags where an important part of how the issue dispatching is made today. This information was not incorporated into the models created so far. Therefore, a new neural network, called Tag NN, was designed which used only this information. The architecture of this neural network is shown in Figure 3.11 and consists of three channels, one for each tag, which is then combined for the final prediction. Embeddings were used to represent the nominal values of the tags. They were randomly initialized, using a mean of 0 and standard deviation of 0.05 and was then learned during training. All other layers were fully connected layers using the ReLU activation function, with exception for the output layer which uses softmax. Furthermore, the training was performed using the Adam optimizer with an initial learning rate of 0.001 and cross entropy loss. Finally, early stopping, with a patience of 10, was used to determine when to stop the training.

Each channel of the model has two fully connected layers and after concatenating the channels there are two more layers. The number of nodes in these layers, as well as the embedding sizes are found using random search, results can be seen in Appendix C. To reduce the search space, it was decided to have the same number of nodes in each layer in the channels. Differing from the other models, the parameter tuning of this model was performed on dataset A instead of dataset B. Because the tags are not universal and therefore does not exist in all projects of dataset B. When a set of parameters was found for Tag NN it was combined as a separate channel with the best CNN model from previous experiments, the character-level CNN. They where combined using multichannel architecture, similar to the one used to combine word- and character-level CNNs in Experiment C. It also uses the same set of parameters.
as for the multichannel architecture from Experiment C, with 380 overlap channels and 0.1 in dropout rate. This new multichannel model is referred to as multichannel CNN (characters + tags).

Figure 3.11: The architecture for the Tag NN model.
In this chapter the performance of the different CNN architectures, the baseline SVM model and the human expert is presented. The results are divided into two main sections; the first showing the results of the different machine learning models on dataset A, and the second comparing a multichannel CNN, using characters and tags as features, with a human expert based on the smaller dataset A_D. The results generated from the parameter tuning of the models can be found in Appendix C.

4.1 Dataset A

This section covers the performance of all evaluated models on the test set of dataset A. Before going in to the comparison of all model results, the overlap of the top-3 predictions for the word- and character-level CNNs are presented, which is related to RQ1. Then, the performance of all tested machine learning models in terms of the metrics chosen, see Section 3.6, is shown. Futhermore, the per class top-3 accuracy of the models are presented. Finally, the top-3 confusion matrix for the best model, multichannel CNN (characters + tags), are presented.
Figure 4.1: The overlap of the top-3 predictions of the word- and character-level CNN.

Figure 4.1 shows how 75% of all issues are correctly classified by both character- and word-level CNNs, in terms of top-3 predictions on the test set of dataset A. It also shows that 7% and 5% are only classified correctly by the character-level CNN and word-level CNN, respectively.

The performance of all models based on the chosen metrics are shown in Figure 4.2. Recall that the SVM is from Experiment A, the word-level and character-level CNNs are from Experiment B, the multichannel CNN for characters and words are from Experiment C and the last two models, Tag NN and the multichannel of characters and tags, are from Experiment D. The multichannel CNN (characters + tags) shows the best results in terms of all performance metrics, except for accuracy. The Tag NN has the second best performance with slightly lower scores for all metrics, except accuracy where it is the best neural network model; overall, the SVM had the highest accuracy. In general the performance of the models are quite similar, especially in the sense of the top-3 accuracy.
4.1. Dataset A

Figure 4.2: The performance of all models evaluated in this thesis; in terms of top-3 accuracy, accuracy and macro F1-score. The y-axis value correspond to the value of the different metrics, denoted by the different colors.

The per team top-3 accuracy are presented in Figure 4.3. In each sub-figure the model which produced the result and the experiment the model was originally designed for is written. However, all of the results are for the entirety of the test set of dataset A, even for the models originally create Experiment D. Multichannel CNN (characters + tags), Figure 4.3 (f), captures more teams than any of the other models, in the sense that it has correctly classified any ticket for those teams. These images were created to investigate the bias of the models, by comparing the per team prediction performance to the dataset distribution. The models seems to, in general, have a high performance on the same teams when inspecting the graphs. Furthermore, teams for which the models performed the worst also seem to, in general, be the same. Specifically, team 10 and team 22 has no correct classification by any of the models.
4.1. Dataset A

Figure 4.3: The per team top-3 accuracy of the models.
**Figure 4.4:** The top-3 confusion matrix of the Multichannel CNN (characters + tags), created for Experiment D. Note that because it is for the top-3 predictions, the maximum value possible is 0.33 and the sum of each row is one.

In Figure 4.4, the top-3 confusion matrix of the models is presented. As the explained in Section 3.6, the maximum value in the matrix is 0.33, visualized as a white cell. In the optimal case, each cell where the true and predicted team would be white, corresponding to the model having a top-3 accuracy of 1 (same as 100%). The true team is the team which actually solved a issue and the predicted team is the prediction made by the model, in this representation aggregated over the entirety of the test set of dataset A. The two team axes differs because, not all teams are present in the test set and not teams were predicted. The choice was made to not add teams without predictions to the horizontal-axis and not add teams not represented in the data to the vertical-axis. The top-3 confusion matrix is used to understand the predictive performance of the model for the different teams. For example, in the figure the value corresponding to team three being the true and predicted team is white; this shows the model always classified it correctly. It is also interesting with appearance of vertical lines which indicates a team is commonly predicted, even when it did not solve the issue. The
4.2 Dataset $A_D$

In this section the performance of the human expert and the multichannel CNN (characters + tags) on dataset $A_D$ is presented. Recall that dataset $A_D$ is a dataset created for Experiment D and is a subset of 50 issues from the test set of dataset $A$. The structure of this section is similar to the previous one. Firstly, the overlap in terms of correct issue classifications by the human expert and the model is presented. Secondly, the performance metrics are presented, followed by the per class top-3 accuracy. Finally, the top-3 confusion matrices, for both the model and human expert, are given.

![Confusion Matrix](image)

**Figure 4.5:** The overlap of the top-3 predictions of the multichannel CNN (characters + tags) and the human expert.

Figure 4.5 shows the overlap of the top-3 predictions for the human expert and the multichannel CNN using characters and tags. It shows a 70% overlap in the predictions and how the model and human expert correctly classified 10% of issues which the other one misclassified. There is also 10% of issues which is not correctly classified by either of them.

All the metrics described in Section 3.6 were calculated for the predictions on dataset $A_D$, for both the model and human expert. The values of the metrics are presented in Figure 4.6, showing how both had a top-3 accuracy of 80%. Looking at the rest of the metrics, the model outperformed the human expert in all cases.
4.2. Dataset A

Figure 4.6: Comparison between the best model and human expert performance on dataset A_D.

The top-3 accuracy in Figure 4.7, shows how both the human expert and the model have five teams which are not given any issues. However, they slightly differ on which teams are not given a issue. For the model, team eight is not given any issues and for the human expert, team 18 is not given any issues. The remaining four teams which had no correct classification were the same for both.

Figure 4.7: The per team top-3 accuracy for the human expert and the multichannel CNN (characters + tags) model.

Figure 4.8 and Figure 4.9 show the top-3 confusion matrix for the multichannel CNN (characters + tags) and the human expert, respectively. The difference between the vertical- and horizontal-axes are for the same reason as described for Figure 4.4 in Section 4.1. However, in Figure 4.9 there is also a predicted team labeled as -1. This comes from the human expert.
being allowed to only give one prediction if he was very certain. Because when only one prediction is made, the other two are labeled as -1. As a result, the value of the cells in the -1 column can reach 0.67, while the values of the cells in all others columns have a maximum of 0.33. Similar to the top-3 confusion matrix in Figure 4.4, these graphs also have two vertical lines for team one and team four.

Figure 4.8: The top-3 confusion matrix of the multichannel CNN (characters + tags).
Figure 4.9: Top-3 confusion matrix for the human expert
5 Discussion

This chapter is introduced by discussing the results of this thesis in relation to the research questions, imbalance in the datasets, and another study. Thereafter, the method used and the sources which provide the basis for this master’s thesis are discussed. Finally, the work of this master’s thesis is put into a wider context; what are the implications of designing and implementing a system that could be used for automatic issue dispatching?

5.1 Results

This section is divided into three main areas. Firstly, the results of the experiments are discussed in relation to the research questions, with the three first headings corresponding to one research question each. Secondly, it is discussed how the imbalance in the datasets might have impacted the results. Thirdly, the work in this thesis is compared to the work of Jonsson et al. [12], as their research lies close to the work of this thesis.

5.1.1 Word- and Character-Level CNNs

When investigating how issues were classified by the word- and character-level CNNs of Experiment B, we found that most of the issues were classified correctly by both models, as shown in Figure 4.1. However, some issues were distinctively classified by the two models; 7% for the character-level and 5% for the word-level. One interpretation of this result is that the word- and character-level CNNs were able to learn unique descriptors of the issue’s text descriptions and titles. If these descriptors could be used in perfect unison, there would be an increase in top-3 accuracy by 5% from the character-level CNN.

In order to evaluate if combining the word- and character-level CNNs could lead to a performance increase in practice, this thesis used the approach presented by Zhang et al. [7]. In their study, they combined a character-level and a word-level CNN, both based on the architecture presented by Kim [5], into a two-channel network. This architecture was the one used in this thesis, for the word- and character-level multichannel CNN, with the addition of a dropout layer right before the final fully connected layer. Similarly to the results of Zhang et al., the multichannel CNN beat the word-level CNN in all performance metrics presented.
in Figure 4.2. On the contrary, the multichannel CNN did not beat the character-level CNN in terms of accuracy and top-3 accuracy. An argument could still be made that the multichannel CNN is better than the separate word- and character-level CNNs, since the macro F1-score was improved over these two models. In comparison, the loss in accuracy and top-3 accuracy was less than 0.5 percentage points, while the gain in macro F1-score was greater than 1 percentage point. In other words, the multichannel model was somewhat better at recognizing smaller teams. It should be noted, however, that these scores are very close, and could therefore change somewhat given new data. By this reasoning, the multichannel CNN with word and character channels could be deemed to overall be equal to the character-level CNN.

To further point out the similarities and differences between the text-based CNN networks, the results can be studied on a per-class level. If Figure 4.3 is studied, where the per-class top-3 accuracy is shown, it is clear that the character-level and the multichannel word/character CNN are similar. Both models predict the largest teams correctly, and fail at the same four (23, 22, 17 and 10), while the mid to small sized teams have mixed results between the two models. The word-level CNN is worse than the other two also in this case, failing to classify two additional classes. In addition, the per-class top-3 accuracy drops off much quicker.

If only the character-level and word-level CNN are compared, it is apparent that the former was slightly better than the latter in terms of accuracy and top-3 accuracy, as shown in Figure 4.2. Why are characters a better type of input to a CNN than words in this case? The beginning of an answer to this question stems from the results of Zhang et al. [7]. In their study they evaluated a character-level and a word-level CNN, with the same architectures as used in this thesis, but in the context of sentiment analysis for tweets of multiple languages. The two models were evaluated in terms of accuracy, and the results showed that the character-level CNN performed better than the word-level CNN for most of the languages tested. One of the reasons suggested was that the character-level CNN was better at capturing different forms of a word, which depend on different grammatical rules for different languages. A way to mitigate this advantage is to use lemmatization for word-level representations of the text. However, resources are often lacking in terms of support for other languages than the most common ones, such as English. Another problem arises when the dataset contains mixed languages which makes lemmatization much harder. For example, words might mean one thing in one language, but another thing in another. The dataset used in this thesis is a mix of multiple languages, terminology specific for the telecom company and computer generated logs. This means that lemmatization is hard, and there will be a mix of grammatical rules and words carrying no meaning except within the company. By the reasoning above, this sets up the character-level to perform better than the word-level CNN, which also was the case. This mirrors the results of Zhang et al. [7] but in the context of issue classification.

A final observation is that the word-level CNN reaches a higher macro F1-score than the character-level CNN. This score stems from a model’s ability to classify the correct team as its best guess. As discussed the method, Section 3.6, top-3 accuracy is the preferred method of measuring the performance of the models in this thesis. Therefore, the advantage is shifted over to the character-level CNN as the best model out of the two, for the scope of this thesis. The main reason being that results based purely on the best guess of a model do not take into consideration that multiple teams may be able to solve the same issue.

### 5.1.2 Convolutional Neural Network or Support Vector Machines?

An SVM was compared to three CNNs; a word-level, a character-level and a multichannel (word + character) CNN through Experiments A-C. All of these models used only the text description and the title of each issue, in other words, only the text-based fields. In Figure 4.2
5.1. Results

A comparison in terms of top-3 accuracy, accuracy and macro F1-score is shown. The main results to take away from this figure is firstly that the SVM tied with the character-level CNN, and beat the other models, in terms of top-3 accuracy. Secondly, when it comes to accuracy and macro F1-score, the SVM outperformed the other models. One advantage of using an SVM over the suggested CNN models is, therefore, that the overall performance in terms of these three metrics is slightly better.

Another advantage of using the SVM, is that the training time was very short in comparison to the CNN models. For all test cases, the SVM took under a minute to train, whereas the CNN models could take hours or days. Furthermore, the CNNs had more hyperparameters to tune, which meant that the overall time for getting the models ready to perform classification, was much longer than for the SVM. However, having more hyperparameters can also be seen as an advantage for the CNNs. For example, the best text-only model was the character-level CNN for which the kernel width and number of feature maps were tuned. Beyond these two hyperparameters, there is the possibility of also tuning for example the learning rate or the embedding size, in order to improve the results further. For the SVM, there are fewer hyperparameters specific to the model. In the case of this study, there were only two hyperparameters that could be tuned for the SVM, and that were not related to the training settings, such as loss function or optimization technique1. As tuning the hyperparameters of any model could lead to improved results, the advantage of greatest unexplored potential lies with the CNNs. In addition, other CNN architectures than the ones explored in this thesis could be used, possibly improving the results even further.

Furthermore, the SVM and the CNNs were compared in terms of top-3 accuracy per class, see Figure 4.3. As was previously mentioned, the SVM and the character-level CNN had the highest top-3 accuracy. On the other hand, when looking at the accuracies per class, the SVM have one more team that is never correctly classified, than the character-level and the multichannel CNN; team number 12. This could be interpreted as the CNN models starting to learn distinguishing features for team 12, and as previously mentioned, this might be improved with further hyperparameter tuning. The SVM shows no such indication, and therefore, the CNN models might be regarded better than the SVM in this case, or at least, having more potential.

In conclusion, the SVM model was deemed useful as a tool when investigating the possibility of issue dispatching at a company. This stems from good results in the three main metrics used, ease of implementation (as it is readily available in several packages), and how quick it is to train. However, the CNN models are worth investigating to potentially increase the performance of an automatic issue dispatching system, by tuning of hyperparameters and exploring different architectures.

5.1.3 Use of Model in Real-World Applications

In order to determine the usefulness of a system automatically classifying issues, the approach was to first investigate the system for issue dispatching present at the telecom company, and measure it’s performance. In this case, this resulted in testing a human expert, and based on the description, title and three project specific tags, evaluate the expert’s ability to classify a set of issues to the correct team. The results of this experiment (Experiment D) are shown in Figure 4.6 in terms of accuracy, top-3 accuracy and macro F1 score. The human expert was able to reach a top-3 accuracy of 80% on dataset A_D. Second, a multichannel network was implemented which could use the same input as the human expert; text and tags. Specifically, the multichannel network combined two networks; a character-level CNN and a Tag NN, as described in Section 3.7.4. This multichannel network was able to match the

5.1. Results

human performance in terms of top-3 accuracy, while exceeding the performance in accuracy and macro F1-score.

Another interesting comparison between the multichannel network and the human expert comes from the two confusion matrices in Figure 4.8 and 4.9. Both the model and the human expert were able to classify teams 0-2, 4-7, 15 and 20 with good top-3 accuracy, showing that both found issues belonging to these teams easier to classify. In addition, both seem to include team 1 and 4 in many top-3 predictions. In the figures, this is shown by the two vertical lines going up from the team 1 and 4 labels on the x-axis. Why are these two teams common for predictions? One important fact to consider is that team 1 and 4 are the largest two teams in number of issues solved. The reason as to why these teams appear for many predictions might be that when the prediction is uncertain, guessing the largest teams gives the highest chance of a correct prediction. However, it might also be the case that these teams have a wide area of competence, which could be the reason why these teams have many issues solved to begin with. If so, these classifications might have been correct in practice. Going back to comparing the confusion matrices, both models failed to predict the correct team for the issues solved by team 22. However, an interesting observation is that they both predicted 11 and 16 in this case, which could be an indication that these teams could have solved the issue as well. Another similarity is that both the model and the human expert have predicted team 4 and team 1 for issues solved by team 4, and the other predictions are spread out across several teams, indicated by the horizontal red “lines” in both figures. Finally, both the human expert and the multichannel model did not correctly classify the issues belonging to team 17, 12 and 9. The predictions did not match for the model and the human expert in these cases, indicating that these issues might be hard to classify correctly, or that a multitude of teams could have solved these issues.

In addition to the similarities between the human expert and the multichannel network, there were also some differences. The human expert seemed to predict team 21 and 23 more often than the multichannel network, as is clear when comparing Figure 4.8 and 4.9. Instead, the multichannel network seemed to be more inclined to predict team 20 than the human expert. Furthermore, the human expert managed to correctly classify the only issue belonging to team 8, while the model did not. For team 18, which also had only one issue, it was the other way around; the multichannel was correct, while the human was not. As these issues were single issues belonging to these two teams, it is hard to say that one or the other was better in general when it comes to these two classes.

The human expert and the multichannel network were also compared in terms of how many issues that were distinctively classified, shown in Figure 4.5. The figure indicates that they managed to classify 70% of the issues correctly, and in the same way. By comparison, only 20% of the issues were classified correctly by the human expert or the multichannel network alone, with an equal split between them. Potentially, if the human expert could make use of the multichannel predictions, the total top-3 accuracy could be increased given that the human can recognize the errors of the model. One way of doing this is through a recommender system, where the multichannel network shows its top predictions along with the issue, and the expert has the opportunity to change the predictions. In this case, the speed of the model is sacrificed to allow for potentially greater accuracy. Another way is to use the model to dispatch the issue to multiple teams, and then let the teams pick whether or not they can solve it. Then the speed of the model is preserved, and the accuracy would be similar to that of a human expert, as shown in Figure 4.6. However, the downside would be that there would be slightly more misclassified issues than in the first approach, and more time would go towards finding and dispatch these issues again to another team.
In conclusion, as dataset $A_D$ is somewhat small, it is hard to draw any solid conclusions from these results discussed above alone. The multichannel network based on characters and tags did perform on a human level, but given more data it is hard to know if the expert or the model would perform best. However, as discussed in [12], the model might not have to reach human performance in order to be of use, since it has the added benefit of being quick when it comes to classification. There is a balance between performance in terms of accuracy and speed, since as long as the model is good enough, the saved time might save the company money. The saved money can come from saved work hours that would have been used manually dispatching the issues. If this argument is weighed in, even if the model would perform somewhat worse than a human, given more data, it could still prove to be useful.

5.1.4 Bias in the Datasets

At the telecom company the teams are of different sizes, which impact how many issues they are able to solve. In addition, the issues might be of varying difficulty, requiring more or less time, and also require different competences, which might lead to an unfair load balancing. As a result, the datasets which stem from the distribution of solved issues across the teams, is imbalanced. In other words, some teams have solved more issues than others. More specifically, distributions of the training sets for dataset A and B are shown in Appendix A, Figures A.1 and A.3. These distributions show that the teams 1, 6, 0, 4 and 5 are significantly more represented than the other teams. How such imbalances can be handled is discussed further in Section 5.2.1, whereas this section presents how it might have impacted the results.

The investigated models were better at predicting the larger teams, in terms number of issues solved, while being worse for the smaller ones. This is shown in Figure ?? and 4.3, where the accuracies and top-3 accuracies of each model are presented. In terms of top-3 accuracy, all models are above 85% for the five largest teams. This might be a result of the skewed datasets, since the correlation between common teams and the top-3 accuracy is high. If the training datasets had been balanced, the model might have been better at finding the smaller teams, but the result would probably suffer. These five teams cover 62% of the validation set, and reduced performance for these teams, in order to find more teams, might be negative in terms of total issues correctly classified. However, this needs to be investigated further, before any conclusions can be drawn.

5.1.5 Comparison to Other Studies

Jonsson et al. [12] compared research related to automated bug assignment to the correct team in a proprietary setting. As stated by Jonsson et al., this kind of task is very uncommon, with the authors finding no similar studies in this context. In any case, the research of Jonsson et al. is interesting to compare to the work of this thesis, since they

- work on bug reports in a proprietary setting,
- separate their test sets from training and validation based on creation date of the issues,
- use similar sized datasets and number of classes to this thesis,
- and, classify bug reports on a team-level.

To describe the bug reports in the study, they used the text title and description, who submitted the bug report, where it came from, revision and priority. In addition, the bug reports could come from external sources, not just from within the company. In the case of this study, the issues are of support nature, and only produced and solved internally within the company. Similarly, the text title and description of the issues are used in this thesis. However,
the additional features differ to [12] as the three tags used in this thesis represent physical location and technology used.

Jonsson et al. [12] used an ensemble method called stacked generalization (SG) and evaluated their model on five datasets. The best accuracy, 64% was reached on the smallest dataset, with only 17 teams. The closest datasets to this study in the number of teams contained 28 and 36 classes respectively. The SG model got 47% accuracy on the dataset with 28 teams, and 20.96% on the other. In the case of this thesis, the models were selected based on their top-3 accuracy, with the multichannel network using tags and characters reaching the highest score. In terms of top-1 accuracy, the multichannel model reached 54% on dataset A, which is worse than most other models tested in this thesis. Since both this thesis and the work by Jonsson et al. use closed datasets, it is hard to compare side-by-side. However, the top-1 accuracy of the best model of this thesis lies in the upper half of the range of results from the study by Jonsson et al.

5.2 Method

This section covers the discussion on the methods used for this study. Firstly, it comments on the preprocessing and the model parameter tuning. Secondly, it covers the replicability, reliability and validity of the study. Finally, the sources used as a base for this study are discussed.

5.2.1 Preprocessing

As described in Section 3.4.1, the raw feature reduction for words is based on both the SVM with TF-IDF and word-level CNN. In contrast, the raw feature reduction of characters is based on only an SVM with TF-IDF representation of the characters. The reasoning as to why only the SVM was used in the character case, as described in Section 3.4.1, is based mainly on one factor; the character-level CNN had too long training time. However, at least in the word-level case, the SVM and the CNN benefited from the same type of preprocessing. This part of the method is a weak point of this thesis, especially as TF-IDF for characters has no support in the literature. However, the performance of this model, while still worse, was extremely close to its word counterpart. This gave us some confidence in the approach, which, combined with this being the only possible way to test different sets of preprocessing for characters, resulted in this method. Finally, the only effect of properly tuning the preprocessing for the character-level CNN would be increased performance. Given the fact that, in the end, this was still the best text-based model, tuning the preprocessing settings using the character-level CNN would not have had a major impact on the conclusions of this thesis.

The dataset is very imbalanced, see Figure 3.3, where some teams solve significantly more issues than others. The choice to remove teams with less than 50 issues, was a simplification made to reduce some of the most extreme cases. The reasoning was that these teams were deemed inactive and misclassification of the issues solved by these teams would have a minimal impact on the results. If these issues were included in the dataset the results would, even if it is a minor change, most likely be worse. The models are obviously biased towards the larger classes, as discussed earlier in Section 5.1, which is a known problem in machine learning. In the case of this thesis, if the model bias could have been reduced, the model might have spread the workload more evenly across the teams. However, a more evenly spread workload might not be desirable, as different teams solve different amounts of issues.
5.2. Parameter Tuning

Almost all models had some amount of parameter tuning, but each model did not get the same amount of iterations. This is especially clear when inspecting the detailed results in Appendix C. The reason for this is the large difference in training time for different models. Instead, a similar amount of time was dedicated to the parameter tuning of each model, with some exceptions. The SVM and Tag NN had, in comparison to the other models, a low amount of time for tuning. For these two models, the training time was so short that a sufficient amount of tuning was reached much faster. The only model which had no amount of tuning is the multichannel of Tag NN and character-level CNN, which was used for the comparison to human performance. It is still the best performing model and further tuning could give it even more of an edge. The same is true for the character-level CNN, given more time for parameter tuning, even better hyperparameters might be discovered. Furthermore, there are additional hyperparameters which could benefit from tuning, to possibly further improve the results. One example is the parameters of the optimizer, see Section 2.3.4, which includes the learning rate.

The majority of the parameter tuning was performed using the random search algorithm. In the other cases, which were the parameter tuning of the SVM, and the last step of the tuning for the word-level CNN, grid search was used instead. Both of these tuning algorithms are covered in Section 2.3.4, with their respective pros and cons. Random search was the main algorithm used because, according to [31], it generally finds good parameters faster than grid search. Grid search was used for the final step of the tuning for the word-level CNN because a small enough search space had been found. It ensured that each combination only were used once, making it faster for that task. By using similar reasoning, grid search was also selected for the parameter tuning of the SVM, as only one parameter was tuned for that model.

For most models, the tuning of the parameters was done through multiple searches. The first search was to see if ranges of parameters providing good results could be found. The search space could then be narrowed for following searches, to focus the search on these specific ranges. The search space could also be increased, if the best results were obtained using parameters at the edges of their respective ranges. This was a way of mitigating the risk that the best choice of parameters could be outside the search space. The changing of the ranges does come with the risk of moving away from the optimal value, which was mitigated by waiting to change the parameters until a trend in the results could be identified.

Dataset B was used for the parameter tuning of all models, except for Tag NN, but the evaluation was performed on dataset A. As a result it is likely that the parameters chosen are not the optimal parameters for dataset A. The choice was made to find model parameters which were more robust and making the model easier to transfer to other projects within the company. Tag NN was tuned on dataset A because the tags were not universal and therefore not available for all projects of dataset B. The higher result of Tag NN, when compared to the other models, might be an effect of it being tuned for dataset A directly. Therefore, we would argue that the result of that model is not fully comparable to the other models. To prove it is truly better than the purely text-based models, all of the text-based models must be properly tuned directly on dataset A. However, even if Tag NN would be the best model, it has quite a narrow use because of the dependence of the project specific tags.

5.2.3 Replicability

The use of a proprietary dataset makes it impossible to fully replicate the study, without making the dataset public. However, an attempt has been made to give as much data specific information as possible; the goal is that future studies can, at least to some extent, compare their data to ours. The comparative dataset factors are information about the size, distribution
and descriptions of the contents of the issues. To only enable comparison through these factors is, of course, not enough for replicability, but it is as much as we can do. Furthermore, the models, preprocessing and how they were chosen is described in detail to allow for re-implementation of the system. The SVM with TF-IDF was also chosen to allow for other studies to compare the result of the model on their dataset. The deterministic training in combination with the easier tuning makes the SVM a good model for this purpose.

Multiple studies have used similar, open-source, issues for different classification tasks [33], [34], [41]. However, even when the data is from the same source, they differ in the extraction process. An alternative to trying to follow the extraction process of a single study, to get replicable results, is to use open datasets containing issues. One such open dataset was created by Lamkanfi et al. [48], with issues from the open-source projects Eclipse and Firefox. It is not certain, however, that good performance on a set of open-source issues necessarily translates to good results on the proprietary dataset, and vice versa. The interest of this thesis was also to study the use of issue dispatching to teams, but in the open datasets the issues are dispatched to individuals. In the end, the choice was made to focus on the possible use of issue dispatching inside a company and comparing it to the performance of the human expert, rather than evaluating on an open dataset.

5.2.4 Reliability

To start with, the stochastic process of running a random search makes it an unreliable process. Therefore, it is unlikely to end up with the same set of tested hyperparameter combinations, if the experiment would be conducted again. As a result, in Appendix C, the parameters used and the result they gave is shown, which makes it possible to try these settings again.

Can the models of the study be re-run with the same settings, and still get the same results? For the SVM with TF-IDF features, this is the case given the same parameters and the same data. The same can not be said for the neural network models. If the same neural network model were to be trained and evaluated two times in succession, the results between the two runs might differ. This comes from the use of stochastic gradient descent and the randomly initialized parameters of the neural network. As a result, if the tuning of the models would be re-run, it would be possible that other hyperparameters for the neural networks would have had the best performance. For most models, when inspecting the best results, there are usually several sets of hyperparameters similar to the best set at the top. This at least suggests that the hyperparameters are in a good area of the search space. The Tag NN model, however, does not have an obvious trend. In the case of that model, changes to the hyperparameters seem to have a relatively low impact. Therefore, even if another set of hyperparameters would be used, it would probably give similar results, since the score between the best and the worst set of hyperparameters only differs in 0.013 points in top-3 accuracy.

5.2.5 Validity

The evaluation of the models were mainly performed on the test set of dataset A. However, the multichannel of Tag NN and character-level CNN were also evaluated on dataset $A_{D_1}$, a subset of 50 issues from dataset A. This dataset was used to compare the predictions of the human expert and the multichannel network.

Top-3 accuracy, described in Section 3.6, has been the main metric used to evaluate models. The choice was confirmed by the human expert, from Experiment D, as a suitable metric for this task. Furthermore, it is also used in other literature; in [41] where the task was bug assignment, up to top-5 accuracy was reported for the models. On the other hand, there are also studies arguing against top-k metrics for bug assignment. For example, Jonsson et al. argue for the use of top-1 accuracy instead:
“However, our reasons for a top-1 approach are three-fold: First, for fully automatic bug assignment a top-1 approach is the only reasonable choice, since an automatic system would not send a bug report to more than one team. Second, a top-1 approach is a conservative choice in the sense that the classification results would only improve with a top-k approach. The third motivation is technical; to ensure high quality evaluations we have chosen to use the built-in mechanisms in the well-established Weka. Unfortunately, Weka does not support a top-k approach in its evaluation framework for classifiers.” [12, pp. 1552-1553]

As a way of motivating the use of top-3 accuracy, these three arguments are met, in the following three paragraphs, with the main reasons why top-3 accuracy was selected as the main metric.

The first argument is correct in the sense that, indeed only one prediction can be made from a fully automatic system. However, since there is an overlap in the competences of the teams, it is possible that the team that solved a certain issue, is not the only team that could have solved it. If the model is able to learn these competence overlaps, it is likely that the model does not always suggests the team that solved a certain issue, but another team with the same competence. Still, the team that solved the issue should be among the best few predictions of the model. Therefore, the top-3 accuracy might better reflect how the model would perform in a real-world scenario, rather than just using the best prediction. Furthermore, depending on how the model is used, it might be possible to predict multiple teams. For example, if the predictions are added as tags to the issue, it can be filtered into multiple teams backlogs and then one of the teams can pick it up.

The second argument, in the quote from Jonsson et al., ties into the first one, as it is true that top-1 accuracy is a more conservative metric. However, only using top-1 accuracy might be a too conservative measure, in the case where multiple teams are viable options. Still, since top-3 accuracy might give inflated results, other metrics are also presented in this thesis and should be taken into consideration when deciding on whether or not use an automated system. Another problem with the accuracy measure is that the models bias against large classes is not taken into account. This bias can be demonstrated in the per class accuracies, Figure ?? and 4.3. In addition, it is also indicated by the lower F1-score, and possible strategies to mitigate this bias on the model level have been discussed in Section 5.2.1.

Finally, the third argument of Jonsson et al. is not applicable for our case, as the model certainty for all classes is available for all libraries used in this thesis. By using the model certainties, it is trivial to get the top-3 predictions (the three teams with the highest certainty).

A common practice when evaluating and tuning machine learning models is to use k-fold cross-validation. Without going into the details, it involves evaluating and training on different parts of a dataset several times and averaging across the results. In short, the purpose is to get more accurate results, than by relying on a single evaluation. In this thesis, the choice was made not to use this method, for different reasons regarding the tuning and the evaluation respectively. One disadvantage of using k-fold cross-validation, is that it takes significantly longer time to get results, compared to a single evaluation. Since the neural network models already had a very long training time, using k-fold cross-validation would have reduced the search space for the hyperparameters significantly, due to time constraints, possibly leading to worse settings. In the case of evaluation, k-fold cross-validation was avoided to be able to better represent the real-world performance of the models. Using k-fold cross-validation would result in the model being trained partly on issues with later creation dates than the issues used for evaluation. However, this is not possible in a real-world application of the model, and would therefore make the results less accurate. Instead, the choice was made
to create a test set for evaluation, consisting of issues from later dates than the issues of the training and validation sets. In addition, Jonsson et al. [12] showed that for their dataset, the performance when using 10-fold cross-validation was higher than the performance measured when evaluating on issues chronologically. Therefore, using cross-validation could possibly also have inflated the performance of the models, compared to real-world performance.

5.3 Source Criticism

The information used for this thesis has mostly been collected from papers in different scientific databases and published books on the subjects. However, some information about the tools used is taken from their respective documentation, and for the work in a wider context section, news articles were also used.

The fact that an academic source is well-known and used by many researchers, could be an indication that the source is trustworthy. Therefore, the number of citations for each source is presented in this section, and is according to Google scholar as of 2019-05-20.

The theory, Chapter 2, has some sources which were more important than others. One of the main sources of the theory chapter is the book *Deep Learning* by Goodfellow et al. [17], where chapters five to nine and chapter eleven are used for this thesis. It is a well-established resource for deep learning, with over 7800 citations. For the NLP theory an important source is the book *Speech and Language Processing*, by Jurafsky and Martin [10], with 400+ citations. Another source for the NLP section is the paper *A Study on the Impact of Pre-Processing Techniques in Spanish and English Text Classification over Short and Large Text Documents*, by Orellana et al. [9], which does not have any citations. However, it was published by IEEE in 2018 *International Conference on Information Systems and Computer Science*, and many of its claims are also supported by Jurafsky and Martin [10].

For this thesis the sources on team-based issue dispatching within the context of companies have been of high importance. Firstly, it includes the paper *Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts* by Jonsson et al. [12], which was published in *Empirical Software Engineering* and which has 40+ citations. The paper by Jonsson et al. has been important for this thesis, as it used many different classifiers and covered many aspects of automated issues dispatching to teams. Secondly, the issue dispatching sources also include two master’s theses, *Tuning of machine learning algorithms for automatic bug assignment*, by Artchounin [46] and *Automating Text Categorization with Machine Learning: Error Responsibility in a multi-layer hierarchy* by Helén and Persson [44]. The two works have been included since both cover the same domain as this thesis; team-based issue dispatching in a company setting. However, since a master’s thesis is considered a weaker source compared to, for example, a published paper, the two works have only been used to show similar work in the area and not as a foundation for the research.

The news articles used in the work in a wider context section, are not used as reliable sources regarding the technical aspects of the research. Rather, they are used to provide insight into the impact of AI and machine learning on society, for the purpose of putting this thesis into a context. In this regard, they are treated as viable sources, and used accordingly. Worth noting is that in all cases where such a source is used, there is an additional source providing similar information.

---

2https://scholar.google.se/
5.4 The Work in a Wider Context

Throughout history, there has been several advancements in technology that have had major impacts on society. For example, machines in factories replaced the work of different types of artisans during the industrial revolution. There were many who resisted this change, since initially, many people lost their jobs as machines replaced them. In the end, the industrial revolution was a significant step towards modern society, with better living conditions and more affordable goods and wares. New jobs were created, as people were needed in the new factories, maintaining and working with and in between the machines, along with executing steps that could not be automated [49].

Today, computers have in many ways revolutionized the way we work, similar to the machines of the industrial revolution. Most easily work that are of a rule-based, and repetitive nature have been replaced by computers; for example the cashiers at fast food chains [50], [51]. However, with recent advancements in machine learning, computers have been put to use in more challenging cognitive tasks. Given sufficient computational resources and enough data, computers can be used to, for example, drive cars autonomously [52], [53]. With jobs being replaced, and more challenging tasks slowly being automated, a couple of important questions arises. Will new job opportunities arise, as in the case of the industrial revolution? Or, will society move towards being independent of human labor?

In the case of this thesis, machine learning was used to read the text description and title of issues, and deduct which team in a team-based project should solve it. As such, it contributes to the computerization of non-routine cognitive tasks. Generally, there are several advantages to the outcome of such solutions; avoiding human bias, saving time and money to name a few. Specifically, the machine learning model suggested in this thesis classifies an issue thousands of times faster than a human expert, with a similar level of accuracy. As this could save the telecom company time, and by extension money, by reducing the amount of manual labour, it also affects the humans who are experts at this task. Since manual labour would be reduced, it could potentially lead to them losing their jobs. However, the thesis work was partly suggested by people involved with this task, indicating that a reduction in workload rather would have positive effects not just for the company, but for the experts as well. At the telecom company, issue dispatching is only one of many different tasks that these experts are tasked with, and by reducing the time it takes to dispatch, they would have more time over for their other assignments. In addition, in the case that automatic issue dispatching would lead to fewer jobs concerning manual dispatching, it could at the same time create new ones. There would have to be infrastructure that can handle the training of the machine learning models, as the models would probably have to be updated regularly [12]. This infrastructure has to be designed, built and maintained which provides new job opportunities. In addition, people designing, implementing, maintaining and monitoring the models are needed, to sustain the use of the model.

5.4.1 Difficulties With Automating Issue Dispatching

Automating issue dispatching to some extent might produce new challenges. As was shown by the results, all machine learning models tested were better at predicting the larger teams than the smaller ones. Potentially, this could lead to load balancing issues if not properly monitored, as smaller teams could be without any issues to solve, while the larger teams are tasked to solve even more issues. However, the same type of bias can be observed in the human predictions, indicating that this problem is present no matter if a model or humans are used for issue dispatching. Another challenge is to learn how the models reason. If a human assigns the wrong team an issue, we could always ask him how he reasoned. This way, it is possible understand why an issue was wrongly classified, and strategies for avoiding this
in the future can be developed. For neural networks, this is a much more difficult problem. Sometimes neural network models are even referred to as black boxes; input is sent into the “box” and the corresponding output is obtained at the other end, without anyone knowing how it came to that conclusion [54], [55]. Since this is a common problem in several applications of machine learning, there have been research aimed towards better understanding the reasoning process of neural networks. For text-based problems, such as text-based issue classification, one way of understanding the reasoning of a model could be to use layer-wise relevance propagation. This approach provides a way of finding the usefulness of each word in predicting a certain class, as demonstrated by Arras et. al [56]. When using this approach, it is possible for humans to get at least a feeling for what the model has learned, and if any specific terms stick out as being wrong. Of course, when the features are at character-level, it complicates things further, and more research seems to be needed in this regard.
Conclusion

This master’s thesis aimed to investigate the use of convolutional neural networks (CNNs) for automatically determining which team should solve which issue in a team-based organization. Specifically, the aim was divided into three research questions for which the major conclusions are found in the first part of this chapter. The final part of this chapter is dedicated towards providing suggestions on how to continue the work beyond this master’s thesis.

6.1 Research Questions

This section repeats the research questions from Section 1.3, and summarizes the main findings related to each of them:

RQ1: Do word- and character-level CNNs, trained on issues, produce complementary features?

Two CNNs based on the same architecture were tuned and evaluated, one based on character-level and the other on word-level text input. From the results, it was apparent that most of the time the issues were classified similarly by the two models. However, there was a subset of issues which only either the character- or word-level CNN classified correctly. This indicates that the two CNNs have the potential to complement each other, given that it is possible to determine which one is right regarding these issues. To determine if this could be done in practice, a multichannel CNN was implemented which used the character-level CNN in one channel and the word-level CNN in the other. Surprisingly, while the multichannel network outperformed the word-level CNN, it only reached similar results as the character-level CNN. Given these results, no final conclusion could be drawn, as the results contradicted each other. Consequently, further investigation is needed to determine if complementary features of the word- and character-level CNNs exist, and how they can be used.

However, it is worth noting that the character-level CNN performed better than the word-level CNN, in terms of accuracy and top-3 accuracy. One possible reason for this result is that this master’s thesis used a dataset containing mixed languages and company-specific terminology. The character-level CNN might be better at capturing the different grammatical rules and variations influencing the words used in the issues’ descriptions and titles.
6.1. Research Questions

RQ2: How do the CNN models (of RQ1) compare to the commonly used SVM with TF-IDF model?

The work of this master’s thesis is based on a closed dataset. As such, it is hard to compare the results of our study to other research. Therefore, a widely used baseline model in the context of text classification, an SVM model with TF-IDF features, was implemented and compared to the CNN models; giving some indication towards the difficulty of the problem.

The results of the text-only CNN models and the SVM were very close. In terms of top-3 accuracy, the best text-only CNN, the character-level CNN, and the SVM performed the same. In terms of the other metrics, the SVM slightly outperformed the character-level CNN. This indicates that CNNs are a decent approach for this task. Furthermore, the CNNs have even more parameters that can be tuned, or the architecture can be reworked, in order to potentially further increase the performance over the SVM. However, in terms of training time, tuning and ease of implementation, the SVM is much better. As a result, if a company is interested in applying machine learning to their corresponding issue dispatching process, our recommendation would be to try the SVM first. At least to indicate how more advanced models might perform.

RQ3: How do humans with expert domain knowledge compare to an automatic system when dispatching issues across multiple teams?

The purpose of this research question was to provide insight into how close machine learning models are to the performance of humans in the role of dispatching issues. In order to do so, a human expert working with manual issue dispatching was tasked with labeling 50 issues with the correct team. The results were then compared to a multichannel network, using character-level text features in one channel, and project-specific tags as features in another. In summation, the multichannel network and the human expert reached the same top-3 accuracy, while the multichannel network reached higher scores in the remaining metrics. As the dataset used was very small, it makes the margin of error more significant, making it harder to draw conclusion from this result alone, since the model might perform worse than the human given more data. However, by also considering the fact that the multichannel network is much faster at classifying the issues, it might not even be necessary to reach human performance in order to be useful.

6.1.1 Future Work

In this thesis the use of CNNs for dispatching of issues to teams has been investigated. There are many ways in which this research could be continued, and in this section, some of the possibilities are described. This includes gaining understanding into model predictions, further model development, mitigation against data bias and finding good metrics for evaluating models used for these kinds of datasets.

All models evaluated, in general, showed similar results for the issues investigated in this thesis. However, the choice of model has many considerations, such as the robustness of the model. Therefore, future work includes building an understanding of what parts of the text the models use and how it differs between the models. Because if a model learns bad features it might not perform well over time. Gaining insight about what the model learns, can also make the predictions easier to explain when errors occur. Which, in turn, might also ease the adoptions of automatic dispatching of issues.

The multichannel architectures used for Experiment C and Experiment D both used only a single layer after concatenation of the channels. The multichannel of character- and word-level CNNs, also performed slightly worse than the character-level CNN on its own in terms of top-3 accuracy. Therefore, further investigation of how the channels can be combined to
get the maximum amount of gain is of interest for future work. This includes the addition of more fully connected layers, other methods of combining the vector outputs of the channels or even the use another model, like a SVM, on the combined representation. The goal is to get as much information out of the text as is possible.

It is clear that the model has a bias towards the large teams. Future work therefore includes evaluation of strategies such as re-sampling and cost-sensitive learning which can be used to reduce bias towards the original distribution of the dataset. However, it is important that the bias mitigation strategies do not reduce the performance on the larger teams too much, as this would result in a significant amount of misclassification.

A problem encountered in this thesis is the problem of measuring the performance of the model, when teams have a large overlap in competences. More generally, this can be thought of as classes which, by definition, are not entirely separable, and where multiple classes can be said to be correct. Therefore, there exists a need for research into suitable performance measures for these conditions. In this thesis the choice was top-3 accuracy which, as discussed in Section 5.2, has drawbacks when it comes to imbalanced datasets. For the common task of classification, multiple suitable choices of metrics are available: precision, recall and F1-score. However, for this use case there is, to the best of our knowledge, no recommendations on proper metrics to use and neither is any other metrics than the ones used in this thesis reported in other work on issue classification.


Appendices
Here the distribution of issues for different teams in the training, validation and test sets are covered for dataset A and B respectively. The distribution across teams is also shown for the small dataset A_D, which is 50 issues sampled from the test set of dataset A. This specific dataset was used for evaluation of the human expert in Experiment D, see Section 3.7.4.
Figure A.1: Distribution of issues across the teams in the subsets of dataset A.
Figure A.2: The distribution of issues over the different teams, for dataset $A_D$. 
Figure A.3: Distribution of issues across the teams in the subsets of dataset B.
In Figure B.1 the entire UI used for making the predictions of a single issue in the human user test is shown. The example is entirely fabricated because of company confidentiality, however it still reflects how the issues look in the experiment.
Figure B.1: Fabricated example of the entire page for prediction in the human expert experiments. With a image containing the information and the three boxes for assigning teams.
In this appendix all results from the parameter tuning of the different models are shown. The chapter is split into sections based on the models. All tables have been sorted based on the top-3 accuracy, as this was the main metric used for model selection. See Chapter 3 and 5 for more details on the metric and why it was used.

### C.1 Experiment A - SVM

The results for the SVM model, where words are represented as TF-IDF vectors of one and two grams. For the parameter $C$, which is a parameter for how the error is penalized, grid search was used to tune the parameter. The parameter set used was $C \in 0.3, 0.6, \ldots, 2.1$. The result of all iterations of the grid search is shown in Table C.1.

<table>
<thead>
<tr>
<th>$C$</th>
<th>Top-3 Accuracy</th>
<th>Accuracy</th>
<th>Macro F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6</td>
<td>0.77801</td>
<td>0.58853</td>
<td>0.43651</td>
</tr>
<tr>
<td>0.3</td>
<td>0.77745</td>
<td>0.58379</td>
<td>0.42411</td>
</tr>
<tr>
<td>0.9</td>
<td>0.77592</td>
<td>0.58881</td>
<td>0.44026</td>
</tr>
<tr>
<td>1.2</td>
<td>0.77368</td>
<td>0.58741</td>
<td>0.44186</td>
</tr>
<tr>
<td>1.5</td>
<td>0.77006</td>
<td>0.58728</td>
<td>0.44299</td>
</tr>
<tr>
<td>1.8</td>
<td>0.77006</td>
<td>0.58630</td>
<td>0.44327</td>
</tr>
<tr>
<td>2.1</td>
<td>0.76922</td>
<td>0.58588</td>
<td>0.44348</td>
</tr>
</tbody>
</table>

**Table C.1:** The results from all iterations of the grid search over the $C$ parameter of the SVM model, where $C \in 0.3, 0.6, \ldots, 2.1$. The results are ordered based on top-3 accuracy.

### C.2 Experiment B - Word-level CNN

To determine the hyperparameters *kernel height* and *number of kernels* both random search and grid search was used. First, 41 iterations of random search were run using the ranges:

- *Kernel Size* $\in \{1, 2, \ldots, 11\}$
Second, the results from the first set of iteration, shown in Table C.2, was used to change the parameters to:

- **Kernel Size** ∈ \{1, 2, \ldots, 5\}
- **Number of Kernels** ∈ \{20, 40, \ldots, 400\}

Third, the result of 18 more iterations of random search using the new parameters, Table C.3, is used to set up the parameters for a grid search over the parameters:

- **Kernel Size** ∈ \{1, 2, 3\}
- **Number of Kernels** ∈ \{180, 190, \ldots, 250\}

Finally, this resulted in 24 more results, Table C.4, which where combined with the previous results to determine the final hyperparameter values.
<table>
<thead>
<tr>
<th>Kernel Height</th>
<th>Number of Kernels</th>
<th>Top-3 Accuracy</th>
<th>Accuracy</th>
<th>Macro F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>280</td>
<td>0.77592</td>
<td>0.56314</td>
<td>0.37427</td>
</tr>
<tr>
<td>3</td>
<td>170</td>
<td>0.77592</td>
<td>0.56662</td>
<td>0.37245</td>
</tr>
<tr>
<td>2</td>
<td>220</td>
<td>0.77550</td>
<td>0.56774</td>
<td>0.38936</td>
</tr>
<tr>
<td>5</td>
<td>290</td>
<td>0.77341</td>
<td>0.56230</td>
<td>0.36306</td>
</tr>
<tr>
<td>4</td>
<td>300</td>
<td>0.77285</td>
<td>0.56621</td>
<td>0.38040</td>
</tr>
<tr>
<td>4</td>
<td>210</td>
<td>0.77020</td>
<td>0.56607</td>
<td>0.37297</td>
</tr>
<tr>
<td>3</td>
<td>150</td>
<td>0.76838</td>
<td>0.56021</td>
<td>0.36059</td>
</tr>
<tr>
<td>4</td>
<td>150</td>
<td>0.76838</td>
<td>0.56007</td>
<td>0.36997</td>
</tr>
<tr>
<td>1</td>
<td>240</td>
<td>0.76727</td>
<td>0.55825</td>
<td>0.36850</td>
</tr>
<tr>
<td>5</td>
<td>270</td>
<td>0.76699</td>
<td>0.55574</td>
<td>0.35993</td>
</tr>
<tr>
<td>5</td>
<td>180</td>
<td>0.76685</td>
<td>0.55839</td>
<td>0.35120</td>
</tr>
<tr>
<td>1</td>
<td>190</td>
<td>0.76671</td>
<td>0.55560</td>
<td>0.36186</td>
</tr>
<tr>
<td>4</td>
<td>140</td>
<td>0.76615</td>
<td>0.55895</td>
<td>0.35714</td>
</tr>
<tr>
<td>1</td>
<td>170</td>
<td>0.76545</td>
<td>0.55337</td>
<td>0.34935</td>
</tr>
<tr>
<td>4</td>
<td>170</td>
<td>0.76392</td>
<td>0.55630</td>
<td>0.35812</td>
</tr>
<tr>
<td>4</td>
<td>110</td>
<td>0.76336</td>
<td>0.55532</td>
<td>0.33745</td>
</tr>
<tr>
<td>1</td>
<td>110</td>
<td>0.76210</td>
<td>0.55142</td>
<td>0.35376</td>
</tr>
<tr>
<td>7</td>
<td>180</td>
<td>0.75903</td>
<td>0.55058</td>
<td>0.34311</td>
</tr>
<tr>
<td>8</td>
<td>290</td>
<td>0.75862</td>
<td>0.55142</td>
<td>0.35502</td>
</tr>
<tr>
<td>7</td>
<td>160</td>
<td>0.75736</td>
<td>0.55393</td>
<td>0.33994</td>
</tr>
<tr>
<td>7</td>
<td>140</td>
<td>0.75652</td>
<td>0.54444</td>
<td>0.33091</td>
</tr>
<tr>
<td>5</td>
<td>80</td>
<td>0.75652</td>
<td>0.54625</td>
<td>0.33555</td>
</tr>
<tr>
<td>6</td>
<td>100</td>
<td>0.75471</td>
<td>0.55058</td>
<td>0.34382</td>
</tr>
<tr>
<td>1</td>
<td>80</td>
<td>0.75429</td>
<td>0.54611</td>
<td>0.34244</td>
</tr>
<tr>
<td>7</td>
<td>110</td>
<td>0.75373</td>
<td>0.54542</td>
<td>0.33876</td>
</tr>
<tr>
<td>5</td>
<td>70</td>
<td>0.75192</td>
<td>0.54667</td>
<td>0.33498</td>
</tr>
<tr>
<td>7</td>
<td>130</td>
<td>0.75010</td>
<td>0.54653</td>
<td>0.32100</td>
</tr>
<tr>
<td>8</td>
<td>100</td>
<td>0.74857</td>
<td>0.54318</td>
<td>0.33212</td>
</tr>
<tr>
<td>11</td>
<td>150</td>
<td>0.74815</td>
<td>0.53914</td>
<td>0.33181</td>
</tr>
<tr>
<td>1</td>
<td>90</td>
<td>0.74745</td>
<td>0.54318</td>
<td>0.33420</td>
</tr>
<tr>
<td>8</td>
<td>100</td>
<td>0.74731</td>
<td>0.54151</td>
<td>0.34046</td>
</tr>
<tr>
<td>3</td>
<td>40</td>
<td>0.74508</td>
<td>0.54458</td>
<td>0.31317</td>
</tr>
<tr>
<td>9</td>
<td>90</td>
<td>0.74383</td>
<td>0.53467</td>
<td>0.32360</td>
</tr>
<tr>
<td>10</td>
<td>120</td>
<td>0.74062</td>
<td>0.53244</td>
<td>0.30783</td>
</tr>
<tr>
<td>11</td>
<td>120</td>
<td>0.73992</td>
<td>0.52853</td>
<td>0.32071</td>
</tr>
<tr>
<td>11</td>
<td>130</td>
<td>0.73908</td>
<td>0.53886</td>
<td>0.32986</td>
</tr>
<tr>
<td>11</td>
<td>100</td>
<td>0.73713</td>
<td>0.53091</td>
<td>0.31866</td>
</tr>
<tr>
<td>8</td>
<td>60</td>
<td>0.73545</td>
<td>0.53328</td>
<td>0.30588</td>
</tr>
<tr>
<td>9</td>
<td>40</td>
<td>0.72848</td>
<td>0.52449</td>
<td>0.28776</td>
</tr>
<tr>
<td>6</td>
<td>30</td>
<td>0.72234</td>
<td>0.51807</td>
<td>0.26434</td>
</tr>
<tr>
<td>11</td>
<td>50</td>
<td>0.71257</td>
<td>0.51193</td>
<td>0.27663</td>
</tr>
</tbody>
</table>

Table C.2: Results from running the first 41 iterations of the random search for parameter tuning of word-level CNN. Parameters was Kernel Size $\in \{1, 2, \ldots, 11\}$ and Number of Kernels $\in \{10, 20, \ldots, 300\}$
## Table C.3: Results from running the random search a second time for the word-level CNN with some minor tweaks to the possible values of the parameters. The parameters was Kernel Size ∈ {1, 2, . . . , 5} and Number of Kernels ∈ {20, 40, . . . , 400}.

<table>
<thead>
<tr>
<th>Kernel Height</th>
<th>Number of Kernels</th>
<th>Top-3 Accuracy</th>
<th>Accuracy</th>
<th>Macro F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>300</td>
<td>0.77843</td>
<td>0.56746</td>
<td>0.36796</td>
</tr>
<tr>
<td>2</td>
<td>400</td>
<td>0.77801</td>
<td>0.56621</td>
<td>0.37305</td>
</tr>
<tr>
<td>3</td>
<td>380</td>
<td>0.77536</td>
<td>0.56481</td>
<td>0.37055</td>
</tr>
<tr>
<td>3</td>
<td>180</td>
<td>0.77271</td>
<td>0.55797</td>
<td>0.35964</td>
</tr>
<tr>
<td>2</td>
<td>400</td>
<td>0.77243</td>
<td>0.56425</td>
<td>0.37542</td>
</tr>
<tr>
<td>5</td>
<td>380</td>
<td>0.77159</td>
<td>0.56216</td>
<td>0.38347</td>
</tr>
<tr>
<td>4</td>
<td>160</td>
<td>0.77048</td>
<td>0.56104</td>
<td>0.36212</td>
</tr>
<tr>
<td>1</td>
<td>320</td>
<td>0.76978</td>
<td>0.56104</td>
<td>0.36409</td>
</tr>
<tr>
<td>1</td>
<td>300</td>
<td>0.76964</td>
<td>0.55909</td>
<td>0.36608</td>
</tr>
<tr>
<td>2</td>
<td>120</td>
<td>0.76964</td>
<td>0.55476</td>
<td>0.34785</td>
</tr>
<tr>
<td>5</td>
<td>320</td>
<td>0.76950</td>
<td>0.55769</td>
<td>0.36246</td>
</tr>
<tr>
<td>4</td>
<td>180</td>
<td>0.76936</td>
<td>0.56104</td>
<td>0.36629</td>
</tr>
<tr>
<td>5</td>
<td>320</td>
<td>0.76587</td>
<td>0.55937</td>
<td>0.35752</td>
</tr>
<tr>
<td>4</td>
<td>160</td>
<td>0.76503</td>
<td>0.56049</td>
<td>0.36266</td>
</tr>
<tr>
<td>5</td>
<td>160</td>
<td>0.76476</td>
<td>0.55853</td>
<td>0.35080</td>
</tr>
<tr>
<td>1</td>
<td>200</td>
<td>0.75987</td>
<td>0.56021</td>
<td>0.36161</td>
</tr>
<tr>
<td>1</td>
<td>160</td>
<td>0.75987</td>
<td>0.55435</td>
<td>0.35887</td>
</tr>
<tr>
<td>1</td>
<td>140</td>
<td>0.75513</td>
<td>0.55142</td>
<td>0.33822</td>
</tr>
</tbody>
</table>
Table C.4: Grid search over the parameters Kernel Size \( \in \{1, 2, 3\} \) and Number of Kernels \( \in \{180, 190, \ldots, 250\} \), for parameter tuning of the word-level CNN. The parameter ranges are based on results from the two earlier random searches.

C.3 Experiment B - Character-Level CNN

To determine the hyperparameters kernel height and number of kernels for the character-level CNN two runs of the random search algorithm was used. The first run was 18 iterations of random search and was performed over the parameters:

- Kernel Size \( \in \{1, 2, \ldots, 11\} \),
- Number of Kernels \( \in \{20, 40, \ldots, 300\} \)

The result of the first run is shown in Table C.5, and these results where used to narrow the parameter range for the second run of random search. The second run, consisting of six more iterations, where run using the following parameter ranges:

- Kernel Size \( \in \{3, 5, 6, 7, 8, 9\} \),
- Number of Kernels \( \in \{120, 140, \ldots, 300\} \)

The additional results are shown in Table C.6, which together with the results of Table C.5 were used to decide on the final parameter values used for the model.
Table C.5: Results from the first random search made to find the hyperparameter for the character-level CNN, sorted based on top-3 accuracy. The parameter used for the search where Kernel Size \( P \in \{1, 2, \ldots, 11\} \) and Number of Kernels \( P \in \{20, 40, \ldots, 300\} \).

Table C.6: Result from the second run of random search for the character-level CNN, sorted based on top-3 accuracy. Using the parameter ranges Kernel Size \( P \in \{3, 5, 6, 7, 8, 9\} \) and Number of Kernels \( P \in \{120, 140, \ldots, 300\} \), chosen based on results from the first run.

C.4 Experiment C - Multichannel (Characters + Words)

The first run of random search was run for 26 iterations and the parameter ranges used was:

- **Batch Size** \( P \in \{1, 2, 4, 8, 16, 32, 64\} \)
- **Overlap Nodes** \( P \in \{100, 110, \ldots, 300\} \)
- **Dropout** \( P \in \{0.3, 0.35, \ldots, 0.85\} \)

The results of the first run are show in Figure C.7. This was followed by 29 more iterations of random search, results in Figure C.8, using the parameters:

- **Batch Size** \( P \in \{4, 8, 16, 32, 64\} \)
C.4. Experiment C - Multichannel (Characters + Words)

- **Overlap Nodes** ∈ \{100, 110, \ldots, 300\}
- **Dropout** ∈ \{0, 0.1, 0.2, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65\}

Finally, a third run of random search, with 29 iterations, were run using the parameters:

- **Batch Size** ∈ \{4, 8, 16, 32, 64\}
- **Overlap Nodes** ∈ \{300, 310, \ldots, 600\}
- **Dropout** ∈ \{0, 0.1, 0.2, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55\}

The results for the third run are show in Figure C.9.

<table>
<thead>
<tr>
<th>Batch Size</th>
<th>Overlap Nodes</th>
<th>Dropout Rate</th>
<th>Top-3 Accuracy</th>
<th>Accuracy</th>
<th>Macro F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>150</td>
<td>0.55</td>
<td>0.74480</td>
<td>0.52839</td>
<td>0.31342</td>
</tr>
<tr>
<td>4</td>
<td>270</td>
<td>0.4</td>
<td>0.74452</td>
<td>0.52351</td>
<td>0.32664</td>
</tr>
<tr>
<td>32</td>
<td>220</td>
<td>0.4</td>
<td>0.74410</td>
<td>0.53244</td>
<td>0.30085</td>
</tr>
<tr>
<td>32</td>
<td>190</td>
<td>0.55</td>
<td>0.74117</td>
<td>0.53844</td>
<td>0.30868</td>
</tr>
<tr>
<td>4</td>
<td>200</td>
<td>0.5</td>
<td>0.74048</td>
<td>0.52072</td>
<td>0.30082</td>
</tr>
<tr>
<td>4</td>
<td>180</td>
<td>0.5</td>
<td>0.74034</td>
<td>0.52435</td>
<td>0.30352</td>
</tr>
<tr>
<td>16</td>
<td>130</td>
<td>0.45</td>
<td>0.74034</td>
<td>0.53523</td>
<td>0.28774</td>
</tr>
<tr>
<td>64</td>
<td>290</td>
<td>0.45</td>
<td>0.73950</td>
<td>0.53914</td>
<td>0.31348</td>
</tr>
<tr>
<td>32</td>
<td>230</td>
<td>0.5</td>
<td>0.73922</td>
<td>0.53105</td>
<td>0.28794</td>
</tr>
<tr>
<td>4</td>
<td>110</td>
<td>0.35</td>
<td>0.73922</td>
<td>0.52867</td>
<td>0.29363</td>
</tr>
<tr>
<td>32</td>
<td>210</td>
<td>0.7</td>
<td>0.73783</td>
<td>0.52212</td>
<td>0.29132</td>
</tr>
<tr>
<td>32</td>
<td>180</td>
<td>0.3</td>
<td>0.73713</td>
<td>0.52770</td>
<td>0.30290</td>
</tr>
<tr>
<td>8</td>
<td>220</td>
<td>0.7</td>
<td>0.73685</td>
<td>0.53118</td>
<td>0.29524</td>
</tr>
<tr>
<td>4</td>
<td>150</td>
<td>0.5</td>
<td>0.73629</td>
<td>0.52616</td>
<td>0.29902</td>
</tr>
<tr>
<td>4</td>
<td>280</td>
<td>0.6</td>
<td>0.73559</td>
<td>0.51988</td>
<td>0.29177</td>
</tr>
<tr>
<td>64</td>
<td>210</td>
<td>0.6</td>
<td>0.73545</td>
<td>0.53858</td>
<td>0.29411</td>
</tr>
<tr>
<td>64</td>
<td>240</td>
<td>0.3</td>
<td>0.73518</td>
<td>0.52853</td>
<td>0.28501</td>
</tr>
<tr>
<td>4</td>
<td>220</td>
<td>0.4</td>
<td>0.73448</td>
<td>0.51765</td>
<td>0.26998</td>
</tr>
<tr>
<td>32</td>
<td>290</td>
<td>0.4</td>
<td>0.73420</td>
<td>0.52198</td>
<td>0.29474</td>
</tr>
<tr>
<td>16</td>
<td>250</td>
<td>0.8</td>
<td>0.73057</td>
<td>0.52630</td>
<td>0.27037</td>
</tr>
<tr>
<td>2</td>
<td>190</td>
<td>0.35</td>
<td>0.72150</td>
<td>0.50802</td>
<td>0.26137</td>
</tr>
<tr>
<td>2</td>
<td>270</td>
<td>0.4</td>
<td>0.72025</td>
<td>0.50886</td>
<td>0.27083</td>
</tr>
<tr>
<td>2</td>
<td>280</td>
<td>0.45</td>
<td>0.71550</td>
<td>0.50802</td>
<td>0.24851</td>
</tr>
<tr>
<td>4</td>
<td>280</td>
<td>0.75</td>
<td>0.71439</td>
<td>0.50105</td>
<td>0.24900</td>
</tr>
<tr>
<td>2</td>
<td>270</td>
<td>0.5</td>
<td>0.70211</td>
<td>0.49156</td>
<td>0.21874</td>
</tr>
<tr>
<td>2</td>
<td>230</td>
<td>0.55</td>
<td>0.69973</td>
<td>0.49170</td>
<td>0.20882</td>
</tr>
<tr>
<td>1</td>
<td>270</td>
<td>0.35</td>
<td>0.67895</td>
<td>0.47202</td>
<td>0.20860</td>
</tr>
<tr>
<td>2</td>
<td>150</td>
<td>0.65</td>
<td>0.67825</td>
<td>0.46728</td>
<td>0.19070</td>
</tr>
<tr>
<td>1</td>
<td>200</td>
<td>0.7</td>
<td>0.59411</td>
<td>0.40100</td>
<td>0.12500</td>
</tr>
</tbody>
</table>

**Table C.7:** Results from the first random search made to find the hyperparameter for the multichannel CNN, sorted based on top-3 accuracy. The parameter used for the search where **Batch Size** ∈ \{1, 2, 4, 8, 16, 32, 64\}, **Overlap Nodes** ∈ \{100, 110, \ldots, 300\} and **Dropout** ∈ \{0.3, 0.35, \ldots, 0.85\}.
<table>
<thead>
<tr>
<th>Batch Size</th>
<th>Overlap Nodes</th>
<th>Dropout Rate</th>
<th>Top-3 Accuracy</th>
<th>Accuracy</th>
<th>Macro F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>290</td>
<td>0.45</td>
<td>0.75624</td>
<td>0.52993</td>
<td>0.31496</td>
</tr>
<tr>
<td>16</td>
<td>380</td>
<td>0.1</td>
<td>0.75541</td>
<td>0.54290</td>
<td>0.34319</td>
</tr>
<tr>
<td>16</td>
<td>390</td>
<td>0.3</td>
<td>0.75415</td>
<td>0.54304</td>
<td>0.31399</td>
</tr>
<tr>
<td>16</td>
<td>310</td>
<td>0.4</td>
<td>0.75387</td>
<td>0.54081</td>
<td>0.30749</td>
</tr>
<tr>
<td>8</td>
<td>120</td>
<td>0.3</td>
<td>0.75373</td>
<td>0.53816</td>
<td>0.32765</td>
</tr>
<tr>
<td>8</td>
<td>180</td>
<td>0.2</td>
<td>0.75262</td>
<td>0.53942</td>
<td>0.32894</td>
</tr>
<tr>
<td>16</td>
<td>280</td>
<td>0.2</td>
<td>0.75220</td>
<td>0.53886</td>
<td>0.33988</td>
</tr>
<tr>
<td>4</td>
<td>350</td>
<td>0.2</td>
<td>0.75066</td>
<td>0.53314</td>
<td>0.30601</td>
</tr>
<tr>
<td>8</td>
<td>150</td>
<td>0.1</td>
<td>0.75010</td>
<td>0.53732</td>
<td>0.32795</td>
</tr>
<tr>
<td>16</td>
<td>390</td>
<td>0.5</td>
<td>0.75010</td>
<td>0.53509</td>
<td>0.31658</td>
</tr>
<tr>
<td>8</td>
<td>170</td>
<td>0.3</td>
<td>0.74941</td>
<td>0.53356</td>
<td>0.31143</td>
</tr>
<tr>
<td>16</td>
<td>300</td>
<td>0.1</td>
<td>0.74927</td>
<td>0.53886</td>
<td>0.34406</td>
</tr>
<tr>
<td>64</td>
<td>300</td>
<td>0</td>
<td>0.74676</td>
<td>0.53523</td>
<td>0.29519</td>
</tr>
<tr>
<td>4</td>
<td>200</td>
<td>0.2</td>
<td>0.74480</td>
<td>0.52881</td>
<td>0.34073</td>
</tr>
<tr>
<td>32</td>
<td>240</td>
<td>0.3</td>
<td>0.74480</td>
<td>0.53802</td>
<td>0.31117</td>
</tr>
<tr>
<td>4</td>
<td>250</td>
<td>0</td>
<td>0.74369</td>
<td>0.53188</td>
<td>0.34182</td>
</tr>
<tr>
<td>4</td>
<td>330</td>
<td>0.3</td>
<td>0.74299</td>
<td>0.52937</td>
<td>0.30502</td>
</tr>
<tr>
<td>4</td>
<td>320</td>
<td>0</td>
<td>0.74159</td>
<td>0.51709</td>
<td>0.30468</td>
</tr>
<tr>
<td>4</td>
<td>390</td>
<td>0.4</td>
<td>0.74159</td>
<td>0.52505</td>
<td>0.28559</td>
</tr>
<tr>
<td>32</td>
<td>200</td>
<td>0.35</td>
<td>0.74090</td>
<td>0.53132</td>
<td>0.29441</td>
</tr>
<tr>
<td>32</td>
<td>140</td>
<td>0.6</td>
<td>0.74048</td>
<td>0.53928</td>
<td>0.29437</td>
</tr>
<tr>
<td>64</td>
<td>220</td>
<td>0.3</td>
<td>0.73852</td>
<td>0.53663</td>
<td>0.27692</td>
</tr>
<tr>
<td>4</td>
<td>290</td>
<td>0.55</td>
<td>0.73811</td>
<td>0.51779</td>
<td>0.28700</td>
</tr>
<tr>
<td>4</td>
<td>120</td>
<td>0.1</td>
<td>0.73769</td>
<td>0.52965</td>
<td>0.29364</td>
</tr>
<tr>
<td>32</td>
<td>310</td>
<td>0.6</td>
<td>0.73629</td>
<td>0.52895</td>
<td>0.27592</td>
</tr>
<tr>
<td>8</td>
<td>190</td>
<td>0.6</td>
<td>0.73490</td>
<td>0.52100</td>
<td>0.27618</td>
</tr>
<tr>
<td>64</td>
<td>170</td>
<td>0.1</td>
<td>0.73462</td>
<td>0.52742</td>
<td>0.29481</td>
</tr>
<tr>
<td>32</td>
<td>120</td>
<td>0.55</td>
<td>0.73280</td>
<td>0.52979</td>
<td>0.28721</td>
</tr>
<tr>
<td>4</td>
<td>140</td>
<td>0.65</td>
<td>0.71746</td>
<td>0.51179</td>
<td>0.25449</td>
</tr>
</tbody>
</table>

Table C.8: Results from the second random search made to find the hyperparameter for the multichannel CNN, sorted based on top-3 accuracy. Using the parameters Batch Size ∈ {4, 8, 16, 32, 64}, Overlap Nodes ∈ {100, 110, … , 300} and Dropout ∈ {0, 0.1, 0.2, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65}, based on the results of the first run.
<table>
<thead>
<tr>
<th>Batch Size</th>
<th>Overlap Nodes</th>
<th>Dropout Rate</th>
<th>Top-3 Accuracy</th>
<th>Accuracy</th>
<th>Macro F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>490</td>
<td>0.2</td>
<td>0.75527</td>
<td>0.54500</td>
<td>0.33373</td>
</tr>
<tr>
<td>8</td>
<td>410</td>
<td>0</td>
<td>0.75373</td>
<td>0.54751</td>
<td>0.34648</td>
</tr>
<tr>
<td>16</td>
<td>570</td>
<td>0.4</td>
<td>0.75359</td>
<td>0.54723</td>
<td>0.32779</td>
</tr>
<tr>
<td>8</td>
<td>460</td>
<td>0.3</td>
<td>0.75276</td>
<td>0.54388</td>
<td>0.33805</td>
</tr>
<tr>
<td>32</td>
<td>560</td>
<td>0.3</td>
<td>0.75164</td>
<td>0.54025</td>
<td>0.31484</td>
</tr>
<tr>
<td>32</td>
<td>500</td>
<td>0.35</td>
<td>0.75108</td>
<td>0.54472</td>
<td>0.31792</td>
</tr>
<tr>
<td>16</td>
<td>390</td>
<td>0.5</td>
<td>0.75024</td>
<td>0.53746</td>
<td>0.31130</td>
</tr>
<tr>
<td>64</td>
<td>530</td>
<td>0</td>
<td>0.74997</td>
<td>0.54095</td>
<td>0.31501</td>
</tr>
<tr>
<td>64</td>
<td>470</td>
<td>0.3</td>
<td>0.74704</td>
<td>0.53802</td>
<td>0.30437</td>
</tr>
<tr>
<td>4</td>
<td>510</td>
<td>0.3</td>
<td>0.74606</td>
<td>0.53286</td>
<td>0.33158</td>
</tr>
<tr>
<td>32</td>
<td>450</td>
<td>0</td>
<td>0.74550</td>
<td>0.54081</td>
<td>0.32227</td>
</tr>
<tr>
<td>32</td>
<td>390</td>
<td>0.5</td>
<td>0.74466</td>
<td>0.53649</td>
<td>0.29649</td>
</tr>
<tr>
<td>64</td>
<td>580</td>
<td>0.4</td>
<td>0.74466</td>
<td>0.53774</td>
<td>0.29804</td>
</tr>
<tr>
<td>4</td>
<td>460</td>
<td>0.3</td>
<td>0.74424</td>
<td>0.53398</td>
<td>0.32151</td>
</tr>
<tr>
<td>32</td>
<td>300</td>
<td>0.2</td>
<td>0.74355</td>
<td>0.54416</td>
<td>0.31034</td>
</tr>
<tr>
<td>4</td>
<td>450</td>
<td>0.35</td>
<td>0.74341</td>
<td>0.52100</td>
<td>0.31316</td>
</tr>
<tr>
<td>64</td>
<td>580</td>
<td>0.4</td>
<td>0.73908</td>
<td>0.53788</td>
<td>0.30342</td>
</tr>
<tr>
<td>64</td>
<td>430</td>
<td>0.2</td>
<td>0.73783</td>
<td>0.53732</td>
<td>0.30012</td>
</tr>
</tbody>
</table>

Table C.9: Results from the third random search made to find the hyperparameter for the multichannel CNN, sorted based on top-3 accuracy. Using the parameters $\text{Batch Size} \in \{4, 8, 16, 32, 64\}$, $\text{Overlap Nodes} \in \{300, 310, \ldots, 600\}$ and $\text{Dropout} \in \{0, 0.1, 0.2, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55\}$, based on the results of the first and second run.

### C.5 Experiment D - Tag Neural Network

The Tag NN was only tuned using one iteration of of random search over the parameter ranges:

- $\text{Tag1, Tag2, Tag3} \in \{5, 10, 20, 30, 40\}$
- $\text{L1, L2, L3, L4} \in \{50, 60, 70, 80, 90, 100, 150, 200\}$

L1 and L2 are multiple layers which were train using the same value for the number of nodes to reduce the search space. Tag1, Tag2, Tag3 and L3 and L4 are just single layers. The architecture is described in detail in Section 3.7.4. The results of the random search are shown in Figure C.10.
Table C.10: Results from the random search over the parameters of Tag NN, sorted on top-3 Accuracy. Using the parameters Tag1, Tag2, Tag3 ∈ \{5, 10, 20, 30, 40\}, L1, L2, L3, L4 ∈ 50, 60, 70, 80, 90, 100, 150, 200. Where the columns named Tag are the embedding size of respective tag variable. The L columns are the number of nodes in the corresponding layer. The layers L1 and L2 are duplicated for each tag, as they are treated as different channels to start with. L3 is then used to join the tag channels together and L4 is the last hidden layer.