Classifying Medical Text using State-of-the-art Natural Language Processing Techniques

Master's Thesis

Sandro Luck
sluck@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Pascual Ortiz Damian
Prof. Dr. Roger Wattenhofer

November 12, 2020

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisor, Pascual Ortiz Damian, for his support, help, expertise, and the time he dedicated to me. I thank Prof. Dr. Roger Wattenhofer, the head of the Distributed Computing Group at the Eidgenössische Technische Hochschule Zürich, for making this project possible.

Abstract

Interactions of patients with medical personal produce various documents in text form. The task of ICD coding is using the International Classification of Diseases system(ICD) to assign a label to those medical documents. Currently, this process is time-consuming, necessary, and needs to be performed by well educated medical personal.
Utilizing machine-learning-based approaches to solve this task automatically has shown great promise in recent years. This master thesis proposes a machine-learning-based model for automatic ICD coding utilizing the recent improvements in the natural language processing technique BERT. Additionally, we will utilize modern neural network interpretability methods to ensure that a decision made by our system is not just correct but also explainable, both to medical personal and patients. We focus on the biggest publicly accessible benchmark dataset Mimic3 to ensure the results are realistic and of high quality.

Contents

Chapter 1

Introduction

The demand for medical services has been increasing for several decades. Especially due to the COVID-19 pandemic, practitioners and hospitals worldwide are challenged in 2020. The technologies and complexity of medical systems and the possibility of treatment have also been expanding rapidly. The national and international costs for health care have constantly been increasing. To save costs and improve service quality, medical workers increasingly utilize automatic systems, both in treatment and administration, to improve quality and efficiency and serve their patients better [1].

Automatic systems have in recent times increasingly been combined with machine learning and deep learning related techniques and thus allowed for new applications. Deep learning related systems are especially promising in medical domains where previous rule-based systems failed since they can learn these rules directly from the data. Given the medical field's complexity, different treatment methods, and increasingly diverse options for treating diseases, machine-learning-based systems could also potentially incorporate the newest findings faster.

The challenges of machine-learning-based systems are manifold. One of the most challenging aspects is the amount of available data. Since machine-learning-based systems learn from the data presented to them, they can usually deliver better results when trained on more data. The conventional idea is that more data beats better algorithms. However, this poses a problem for the medical domain [2]. Since medical datasets are especially sensitive datasets, they usually are small and contain between 1'000 and 100'000 samples [3, 4]. The amount of hospital stays in the USA exceeded 35'000'000 in 2016 alone, and therefore there would exist enough data points [5]. However, most of this data is not publicly available. Outside the USA, such data is rarely available as a public dataset, the main reason being that generating such datasets is time and resource consuming. If data is available for public research, it is heavily anonymized, which is desirable but poses additional challenges. This anonymization process is costly and has not been clearly standardized, which again adds to the scarcity of datasets.

Another common problem for machine-learning-based approaches is data quality. Data quality is essential for successful learning, and improving the data quality can often help the system learn better, even when no architectural changes were

made. In the medical domain working with data that has not directly been collected for a specific purpose can also lead to quality issues [6].
One also has to consider that a dataset is often from one data source or closely connected data sources where data is available. In other words, the treating doctors are often from one hospital and the patients from one specific geographic location. For example, the Mimic3 dataset, one of the biggest medical datasets, contains only samples from the Beth Israel Deaconess Medical Center, Massachusetts.
The data format of medical data is also often specific and not commonly used in other domains. We will mostly focus on the textual data, which has gotten more attention in recent years due to NLP-based methods' progress. The textual representation of medical data poses specific problems, as we will show in more detail later. Problems include a domain-specific multilingual vocabulary, illnesses, and body parts, may be mentioned in Latin, English, and potentially local slang terms. Its vocabulary includes many abbreviations and measurements specific to the medical domain and can only partially be transfer learned. The modern state of the art NLP deep learning models relie on a lot of data, which may only partially be available for the medical domain. For example, GPT-3, a recent NLP model from OpenAI, has gained widespread attention by being trained on almost 1 trillion words [7]. In general however, the medical literature is still vast and can be used to transfer learn the medical vocabulary and context without having massive datasets.
A decision made in the medical domain is sensitive; the decision can have a long-lasting impact and may not always be easily reversible. This justifies building systems that evolve past optimizing a certain score and additionally explain themselves to the user. Such an explanation may be given by highlighting what data source the respective decision was most influenced by. For example, when using gender-related information for a prediction, the model should indicate that this was considered when making the prediction or decision. While such features as age contain valuable information, it should be made clear to the using specialist that this information was considered. We may consider a patient with an occupation like "security(-guard)" where night shifts are frequent and then is labeled to have sleeping problems, but he has normal working hours and no sleeping problems. While it may be beneficial for the model to assume that all shift workers have sleeping problems, it is not always correct.
In the following work, we aim at developing an assistance system for clinical practitioners that helps them in their administrative work and explains itself. While our system would be more useful to help practitioners in the administrative setting, the underlying interpretability usage could also be extended to the predictive settings.
In the following thesis, we will first explain the broader field this work operates in. We will then try to give an intuitive overview of the different systems that we used and developed. It will be followed by explaining the components of the two subsystems and the respective design choices made for their development.

Later on, we will present the system's evaluation and interpretation and explain the mathematical foundation it is built upon. We will then conclude with a discussion and possibilities for future work.

Chapter 2

Related Work

This work is mostly operating at the intersection of medicine and Natural Language Processing(NLP). We will use medical text to predict so-called ICD codes. The International Classification of Diseases (ICD) is a system for classifying diseases [8]. The codes are alphanumeric, and there are several variations of this system ranging from ICD-1 to ICD-10. In this work, we will be using ICD-9, which has been used between 1979 and 1998 [8].
In total, ICD-9 has around 15'000 codes where the exact amount may change depending on the source USA, World Health Organization(WHO), etc.[9]. This is significantly less than the number of codes in the newer version ICD-10, which contains over 140'000 codes. While this is a respectable difference, one can generally assume that only a small part of these illnesses are actually used in practice. These ICD codes have different uses ranging from billing to predictive modeling of patient states and aim at covering all illnesses and general conditions from "Cholera, 001.9" to "Unemployment, V62.0".

2.1 Automatic ICD Coding

Automatic ICD Coding refers to automatically predicting the ICD codes from the data available in the health facility's data sources. The task of ICD Coding, in general, is prevalent and extensively studied [10, 9, 11, 12, 13, 14, 15, 16, 17]. Today, this task is mostly conducted by hand, which takes a long time, is error-prone, and has to be conducted by skilled medical personnel and is therefore expensive. Therefore automatic ICD Coding has been studied since at least 1998 and would improve the healthcare system's efficiency [10].
The main usage of ICD Coding is to describe to the health insurance companies which services were conducted and how much the treating entity is paid. ICD codes are also used to aggregate and create statistics on the health of citizens [18]. Most works in the field try to infer the codes exclusively from medical free-text, which is also the main data source clinical practitioners use today to manually conduct this task [11]. Doing so is challenging for 3 reasons 1) The labeling space is very sparse 2) Medical text is complicated and contains many domain-specific

languages and redundant information 3) The texts tend to be very long. In recent years specifically, the Mimic3 dataset has become increasingly popular. We will describe the specific challenges more closely in the section 2.2.
Additionally, the ICD codes and labels are not always applied thoroughly, meaning that even though the information is present in the text, the respective label is not set. Since ICD codes are hierarchical and ambiguous, one doctor may label a specific headache as "Headache, from the neck" or "Headache, mixed," which leads to a certain ambiguity. This is more common in categories that are relatively cheap to treat [18].

2.2 Mimic3

The Mimic3 dataset was released in 2016 [3]. Mimic3 is a large database of medical data in various forms. The dataset has 53'423 distinct hospital admissions and is anonymized. For this work, we focus on extracting the knowledge mainly from the discharge summaries.
Discharge summaries are medical documents created by doctors at the end of a stay in a medical facility. They summarize the diagnoses, diagnostic procedures performed, therapy received while hospitalized, clinical course during hospitalization, prognosis, and plan of action for the follow-up [19]. Discharge summaries are a common choice for the task of ICD Coding in the Mimic3 dataset [9, 15, 16, 17]. Their main advantage is that they contain all information necessary to identify all labels. In contrast, a nursing/radiologist note may only contain selective information on a patient's state at a specific point in time.
One discharge summary refers to one stay in the hospital. Thus, one has to be careful to split the dataset by patients and not simply by samples. We model the task as a multilabel classification task since a sample/discharge summary can simultaneously have multiple active illnesses. In fact, a discharge summary has 13.15 ICD labels on average associated with it.
In the Mimic3 dataset, only 8'921 of the around 15'000 ICD-9 codes appear. The average amount of samples per ICD code in the dataset is 100.29, and the median is 6. We can see the distribution in the figure 2.1. We will mostly focus on the 50 most frequent ICD codes in the Mimic3 dataset, due to sparseness. This subtask is very common in the literature and covered in most prominent works on ICD-Coding [9, 15, 16, 17].
When building a recommender system for clinical practitioners, it is also important to get the most common codes right, and additionally, this makes interpretability-related visualization easier. The distribution of the top 50 most frequent ICD-9 codes is also significantly more favorable for training neural networks since some labels occur below 10 times otherwise. We can see the distribution of the top 50 codes in the figure 2.2. More specifically, the entire list and frequency can be seen in the table 2.1.
We may notice this distribution has a very long tail and is heavily concentrated

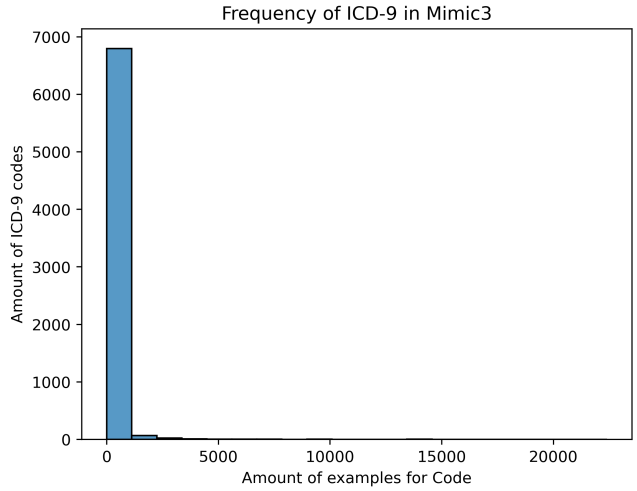

Figure 2.1: ICD-9 Code distribution in Mimic3

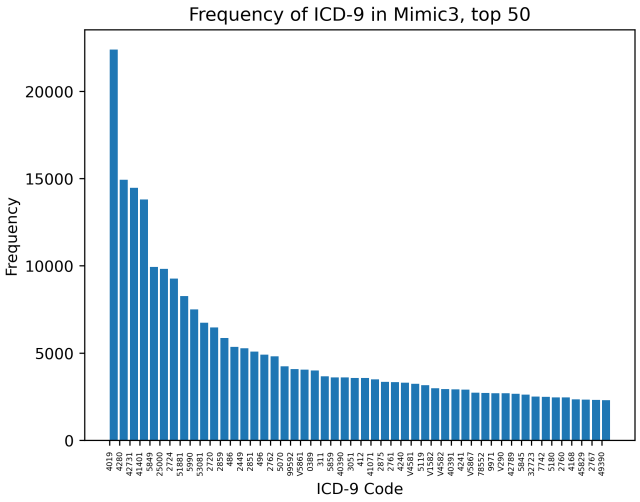

Figure 2.2: ICD-9 Code distribution in Mimic3, Top 50

in the 4 most frequent diseases, all of which are cardiovascular diseases. We will only include samples that include at least one of the 50 most common ICD codes, which happens in 49'091 of 52'726 samples, i.e. we discarded only 6.9% for the runs with the bigger dataset. We kept the train, test, validation split at 90%,7%, and 3% as Mullenbach et al. [9].

ICD-9 Code	Present in % of samples	Description
4019	39.3%	Hypertension NOS
4280	27.4%	CHF NOS
42731	26.7%	Atrial fibrillation
41401	24.5%	Crnry athrscl natve vssl
2724	18.2%	Hyperlipidemia NEC/NOS
5849	17.5%	Acute kidney failure NOS
25000	16.6%	DMII wo cmp nt st uncntr
51881	16.0%	Acute respiratry failure
5990	13.9%	Urin tract infection NOS
53081	11.4%	Esophageal reflux
486	11.4%	Pneumonia - organism NOS
2720	11.1%	Pure hypercholesterolem
2859	10.7%	Anemia NOS
2449	10.4%	Hypothyroidism NOS
2762	9.74%	Acidosis
5070	9.6%	Food/vomit pneumonitis
2851	8.72%	Ac posthemorrhag anemia
496	8.30%	Chr airway obstruct NEC
V5861	8.24%	Long-term use anticoagul
40390	7.77%	Hy kid NOS w cr kid I-IV
0389	7.7%	Septicemia NOS
V4581	7.63%	Aortocoronary bypass
99592	7.29%	Severe sepsis
4241	7.14%	Aortic valve disorder
5859	6.74%	Chronic kidney dis NOS
V1582	6.68%	History of tobacco use
2875	6.61%	Thrombocytopenia NOS
41071	6.61%	Subendo infarct - initial
412	6.54%	Old myocardial infarct
V4582	5.93%	Status-post ptca
5119	5.79%	Pleural effusion NOS
311	5.65%	Depressive disorder NEC
3051	5.59%	Tobacco use disorder
V290	5.59%	NB obsrv suspct infect
V5867	5.52%	Long-term use of insulin
78552	5.52%	Septic shock
4240	5.52%	Mitral valve disorder
49390	5.38%	Asthma NOS
9971	4.9%	Surg compl-heart
2761	4.84%	Hyposmolality
40391	4.7%	Hyp kid NOS w cr kid V
5845	4.63%	Ac kidny fail - tubr necr
42789	4.43%	Cardiac dysrhythmias NEC
5180	4.43%	Pulmonary collapse
45829	4.29%	Iatrogenc hypotnsion NEC
32723	4.29%	Obstructive sleep apnea
4168	4.22%	Chr pulmon heart dis NEC
7742	4.16%	Neonat jaund preterm del
2767	4.16%	Hyperpotassemia
2760	3.61%	Hyperosmolality

Table 2.1: Top-50, ICD-9 Codes and distribution in Mimic3

2.3 Preprocessing

The task of ICD Coding using Mimic3 discharge summaries has been extensively studied [10, 9, 11, 12, 13, 14, 15, 16, 17], and many different preprocessing forms have been used. When working with text in machine learning, it is often beneficial to clean the text before processing it, also referred to as preprocessing. Most newer works in the Mimic3 ICD Coding task use the preprocessing form proposed by Mullenbach et al., which includes the most recent and state-of-the-art model from Vu et al. [9, 17, 20, 16].

- Convert the text to lowercase
- Remove Numbers
- Removing words that occur infrequently
- Truncate the text

We use a modified version of this preprocessing form, we will discuss this variation and justify it in more detail in section 4.3.

2.4 Models for ICD Coding

The task of ICD Coding using text has been tackled over the years using several different techniques. Traditional machine learning methods have been used to solve this task, [21, 22] but more recently, the focus has shifted towards more deep learning related methods [9, 11, 12, 13, 14, 15, 16, 17].
The deep learning related methods often relied on convolution [9, 20], The most recent and best works in the area, have been using LSTMs and labeling attention to learn their embeddings [17]. Also, incorporating new text data for diseases from the World Health Organization(WHO) is a common approach to solve the problem of the label space's sparsity [9]. The most recent successes in the Mimic3 dataset have been achieved using convolution, LSTMS, and word embeddings [9, 11, 12, 13, 14, 15, 16, 17]. On a high level, their approach is to find an embedding for each word. Each document/sample can then be represented as a list of n word embeddings. Then they use various forms of recurrent layers such as LSTMs, RNNs, and their bidirectional counterparts [9, 17, 16]. Here the approaches diverge but for example, Vu et al. use the output of this information to learn a vector representing the document for each label, i.e. they will then have l label vectors representing a document. Attention is often stressed to play an important role in handling the information coming from the embeddings and making the final predictions [20, 17].
Traditional deep learning methods being better than BERT seems surprising

since the BERT architecture has shown great promise in NLP tasks in the Biological and medical domain in recent years [23, 24, 25]. Specifically, however, for the Mimic3 dataset, researchers have failed to achieve state-of-the-art performance using BERT like architectures. In a very similar work to ours, Chen et al. finetuned BERT on the Mimic3 dataset. They found that their model's biggest challenges were that the discharge summaries were too long [26]. While they achieved a relatively good score, their performance falls short of beating the state-of-the-art performance. While Zhang et al. recently showed that using BERT in ICD Coding is beneficial, they pointed out that working on the Mimic3 dataset is especially difficult due to the low amount of data [27] and instead used a total of 7.5 million notes from the Anonymous Institution EHR system. Their dataset is not publicly available. They state that in their dataset "We found that all BERT based models far outperform non-transformer based models"[27]. They additionally state that the amount of data in the Mimic3 dataset is small and the content constrained to critical care units, which seems to be why they did not compare their model's performance to other models.

2.5 BERT

In recent years, transformer-based architectures [28] such as BERT [29] have shown great promise in various NLP tasks. In NLP tasks, transformer-based architectures were used successfully for summarization, translation, generation, and classification [30]. The major improvement is the self-attention mechanism that can be efficiently parallelized on modern GPUs [31].
Inspired by the usage and success across other domains, BERT architectures are increasingly used in the medical domain. To recommend medication Shang et al. used a BERT-like architecture via embedding learning for ICD codes [32]. Clinical BERT has been developed by training on the Mimic3 dataset, both the discharge summaries and notes [25]. This BERT variation can then be used for further downstream prediction tasks. BioBERT has been developed by pretraining on biological articles and publications. Its goal is to be useful in tasks related to biology [33]. It has also been successfully used for ICD annotation on experiments with animals by Sänger et al. [34]. The most recent and most related publicly available BERT architecture to the task of ICD Coding has been proposed by Gu et al. from Microsoft [23]. This model has been trained on abstracts from PubMed and full articles from PubMedCentral. Using this procedure, they achieved state-of-the-art performance on several medical NLP tasks and are also state-of-the-art in the Biomedical Language Understanding and Reasoning Benchmark [23].
One of the main obstacles of using BERT related model is that they need big amounts of training data and training time to perform competitively [7]. The emergence of the excellent Huggingface environment [30] has made working with BERT like architectures significantly easier. The pretraining phase can be skipped,

and a comparison between flavors of BERT is simplified. It allows for the possibility to download and use models that have been trained for several weeks for various fields such as biology and medicine. We will show an evaluation of the models available on Huggingface in the section 4.2 that were pretrained on medical or biological data. All previously mentioned BERT variations are available for comparison in section 4.2 with the exception of clinical BERT, which was pretrained on Mimic3 and should therefore not be used for our task.

2.6 Interpretability of Neural Networks

Neural networks are complex models and combine a variety of different transformation layers and activation functions. While predicting with an already trained network is fast, understand how this prediction was made by the model is hard. Therefore neural networks are also often referred to as black-box models [35].
Interpretability is a need in settings where testing all possible input combinations/risks is practically impossible. In situations like this, Interpretability can help in the analysis of neural networks [36].
From a societal perspective, understanding how the predictions are made is essential to avoid generalization, which is not accurate. Further, the European Union mandated that "Among other rights, the GDPR guarantees individuals the right to have a decision based solely on automated processing (an algorithm) be made or reviewed by a natural person instead of a computer" [37] which indicated that interpretability would be necessary to explain neural networks that base their decision on multiple 1'000s of variables[36].
In the medical field, interpretability is of special interest due to decisions having far-reaching implications and often not being reversible.

Chapter 3

Overview

The task of predicting the 50 most common ICD codes in the Mimic3 dataset is challenging. Due to limitations, we will mention in the section 4.4 we will work for the most part with a two-model architecture. In the first step, we will convert the discharge summaries to an embedding. Then we will use this embedding to predict the final ICD codes.
An embedding in this work is a vector $x \in \mathbb{R}^n$, which captures some part of the information present in a document. This embedding is constructed such that it represents the underlying discharge summary or a part of it. The final embedding size varies by architecture, but we can think of the final embedding as containing between 1'000 and 200'000 variables.
Once we constructed such an embedding, we used a second model to work on top of it to extract the final prediction. We decided to use such an architecture since it allows us to utilize more information and take a more global view of a sample. Additionally, we can use combinations of the same model with different transformation methods and then concatenate them into a new embedding. This is necessary since if we would only use a normal BERT architecture, we would only be able to see roughly $\frac{1}{8}$ of the text due to reasons we will discuss in more detail in the section 4.4.

3.1 Embedder Overview

To create an embedding, we choose the successful BERT architecture described in more detail in chapter 4. We use the very convenient and optimized library FastBert to train our Embedder since it allows for several useful hardware optimizations such as halfprecision training, multi-GPU training, etc. [38]. Which results in a more rapid development time. This is especially important to us since our architecture is extremely time-consuming and takes a long time to train, which is normal for BERT based architectures [30]. We will discuss the hardware constraints later in the section 4.5.
The basic concept is that we finetune an already pretrained BERT version. This

pretraining was done on excessive amounts of data and over a long time span, often over several weeks on several GPUs.
The process of finetuning a BERT model involves loading a pretrained BERT and adding a layer between the last layer and the BERT-pooler. We then train this new construct (Embedder) using the binary cross-entropy loss with logits to predict the illnesses' probability in the last layer. After training this Embedder, we can generate an embedding for a sample by taking the last layer's output after the forward pass. This process is visualized in the figure 3.1. The white fields are the Classifier we train (aka Embedder), the grey box is the embedding we can generate using this Classifier. We differentiated here its two functions while training it, it is a Classifier, and when we use it after training, it is an Embedder. They are the same thing.
As illustrated in white, the Classifier is already a completely usable solution to the ICD problem. We can gain additional insights by combining their results, as we will show in the following chapters. As we can see, the Classifier is very similar to a BERT architecture [29]. The 12 BERT-layers are identical. The main difference is visible after the BertPooler, which is the last layer both have in common. The last classification layer is basically a simple linear layer that is "packed on top" of a BERT architecture and can use BERTs information for our classification task.

3.2 Embedder Training

The final embedding is an input produced by one or several Embedders. Combining the output of these Embedders will yield a new and more informative embedding. An overview of how this process functions on a high level can be seen in the figure 3.2.
As we can see, we take a document and transform it into several new variations of the document, each containing a different "view" on the document. In one variation, we may look at all lung related text, and in another, we may look at all history related text.
Once this transformation has been conducted, we are left with a new dataset "A." After finetuning now a BERT on this dataset, we are left with an Embedder "A." To create the final embedding, we concatenate several of these embedding, which we can see in the figure 3.3.

3.3 Embedding Construction and Predictor

The Predictors task is it to predict the 50 illness given the m embeddings. On a high level, we create m embeddings given $n <= m$ Embedders as described previously. One Embedder may create several embeddings by been given different

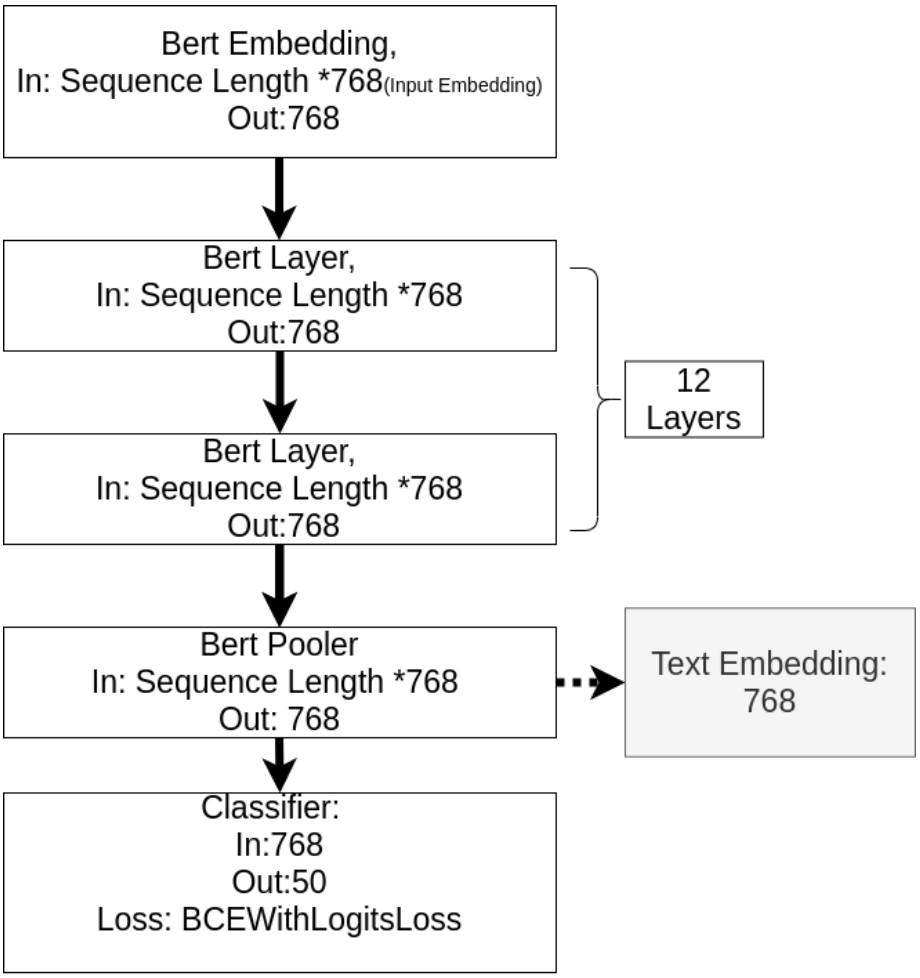

Figure 3.1: Simplified Embedder.

inputs for one sample. We can see a visualization of this process in the figure 3.3. As we can see, a sample is transformed using the m transformations. This results in m distinct views on the sample. Using the Embedder specialized in working on the specific transformation, we can now generate m embeddings. After generating these m embedding, we now create new bigger, and more informative embedding. The Predictor then gets this embedding to generate the final predictions.

3.4 Evaluation

Like most other works in the field, we will focus on the F1 scores. Since F1 is the harmonic mean of the precision and recall, selecting model based on the F1 score is sensible. Given $|L|$ classes, the F1 macro and micro score are given by,

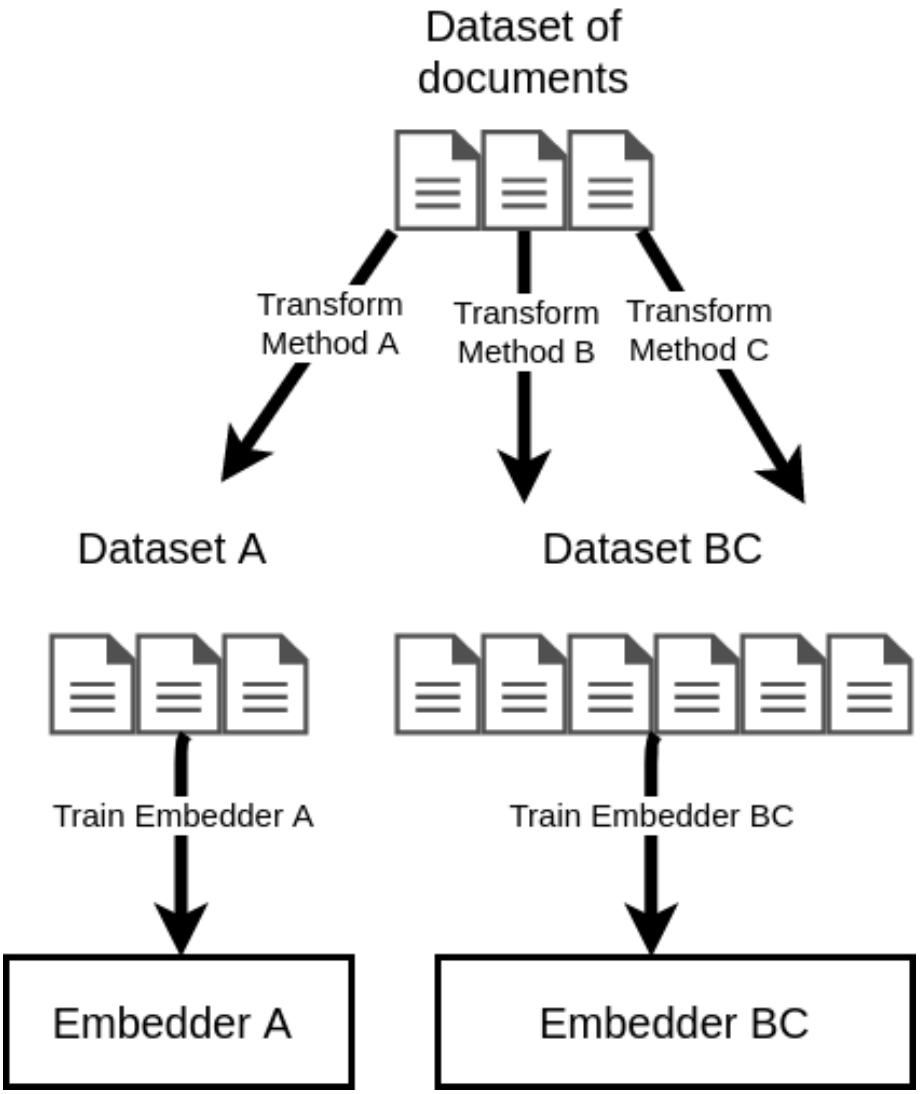

Figure 3.2: Showing the process of creating the Embedder.

$$F1_{micro} = \frac{TP}{TP+0.5*(FP+FN)}$$
$$F1_{macro} = \frac{1}{|L|}\sum_{l=1}^{|L|}\frac{TP_l}{TP_l+0.5*(FP_l+FN_l)}$$

Where TP are the True Positives, FP are the False Positives, and FN are the False Negatives and lower case l their respective value for class l. We particularly focused on the micro F1 score. Focusing on the micro score as opposed to the macro score was an early decision. Since a recommendation system is supposed to assist the clinical practitioners, it is more important for their efficiency that the most frequent codes are predicted correctly. Also, we found that the macro and micro F1-score tend to move in the same direction in our implementation, and the

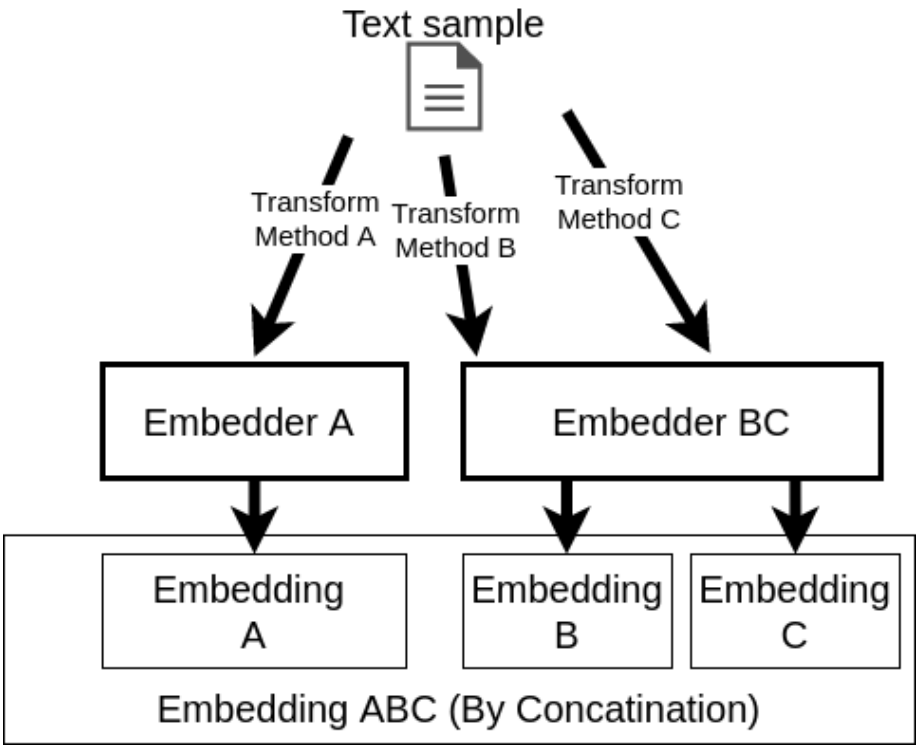

Figure 3.3: Showing how final Embedding is created.

micro F1 score is mentioned in all recent works [9, 11, 12, 13, 14, 15, 16, 17]. This is why we will, in most illustrative plots, focus on the F1 micro score and provide a more thorough investigation in the chapter 6, including the macro scores.

Chapter 4

Embedder

The Embedder is a function learned to transform the discharge summarie text into a vector representing this text in a beneficial way for ICD Coding. We learn these functions using a pretrained BERT architecture and adding a dense layer on top. We then train the entire BERT to predict illnesses in the last layer with all its layers again. This process will be called finetuning in the following sections. In the following sections, we will discuss different ways to train such an Embedder. We will also discuss the important technologies and design decisions that were involved.
On a high level, we download a pretrained model using Huggingface. This pretrained model is then finetuned using FastBert. The finetuning happens by using transformed text. The transformation is the main difference between most Embedders. Only two embeddings have been generated differently to this procedure and will be described in sections 4.13 and 4.14.

4.1 Data

To train and evaluate the models in the sections of this chapter we used the preprocessed discharge summaries of the Mimic3 dataset. The preprocessing described in the section 4.3, if not stated otherwise, has been used to generate the following evaluations. We focused on the 50 most common ICD codes, as described in section 2.2. We only used discharge summaries, including at least one of the 50 most frequent ICD codes. This results in a dataset with 49'091 samples. Of these, we used 44'175 for training, 1'468 for validation, and 3'448 for the test set. We split the data in such a way that each patient (which may be in several samples) can only be in one of the three sets. The following evaluations have been conducted on the validation set. If a discharge summary was longer than 1024 tokens, the length of the text was automatically truncated. For the distribution-related plots, the training set has been used. To compare different architectures, the validation set was used, the test set was used to evaluate the Predictor as we will later describe in the chapter 6.

4.2 Huggingface

The excellent library Huggingface [30] has several pretrained version of BERT and various subversions such as ROBERTA available. Due to the medical nature of Mimic3, we considered for this work mostly models pretrained on large datasets of medical and or biological data. We also tested some other models that showed promise in long texts but found that they performed largely inferior. We assume, as also pointed out by Zhang et al., that the medical vocabulary is very different from the ordinary English vocabulary and thus needs a specialized BERT [27] Specifically, we tested these pretrained variations of BERT:

- biobert_v1.1_pubmed_squad_v2
- biobert_v1.1_pubmed_nli_sts
- biobert-nli
- oubiobert-base-uncased
- BiomedNLP-PubMedBERT-base-uncased-abstract-fulltext

Each of these models has been pretrained for a long time on different medical or biological datasets. We loaded each of these models into FastBert and trained them using the preprocessed dataset we will describe in the section 4.3. Once a model has been trained, we used our validation dataset to measure its performance and decide which BERT model is the most useful for our purpose. As we can see in the figure, 4.1 the recently release BiomedNLP-PubMedBERT-base-uncased-abstract-fulltext from Microsoft [23] is clearly superior for our task. It has been released very late into the thesis and was only used in the last 2 months of the thesis, which is why we will later in the chapter 6 have a difference between a big and a small dataset.

4.2.1 Hyperparameters for comparison

The evaluation for the model was based on the hyperparameters in the table. 4.1

Hyperparameter	Value
Optimizer	Lambda
Learning Rate	0.0005
Scheduler	Warmup Cosine
Batch Size	8
Precision	Full Precision

Table 4.1: Hyperparameters used for selection of pretrained BERT versions

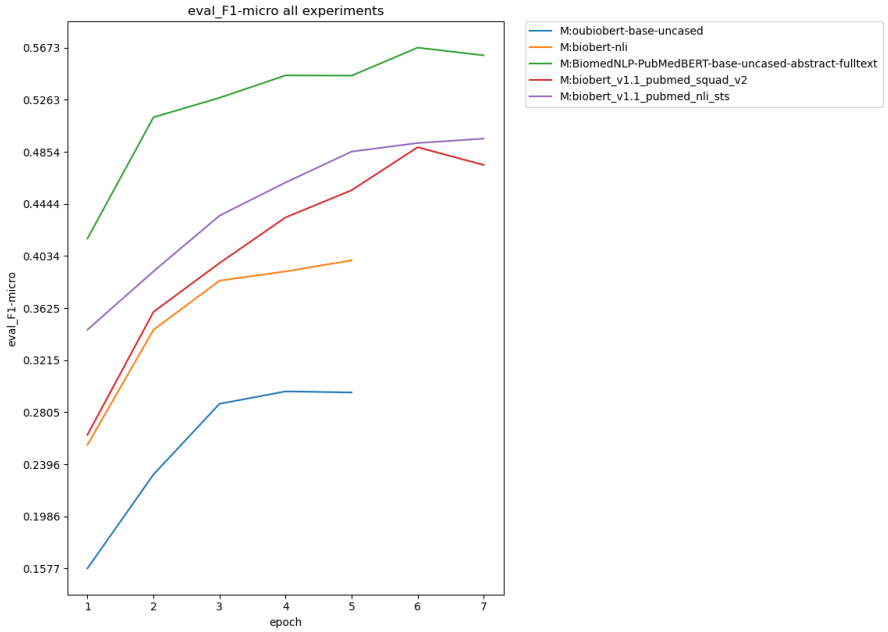

Figure 4.1: Comparison of the different BERT models, data as described in section 4.1

4.3 Preprocessing

As already discussed in the section, 2.3 we used a variation of the preprocessing form proposed by Mullenbach et al. [9]. While we theoretically have the same task at hand, the differences in used technologies justify using preprocessing variations. We also converted the entire text to lower case since it was beneficial for our training process.
Truncating the text was not explicitly done in our implementation, but it happened implicitly when training. Mullenbach et al. found that after 2500 words, the gain is insignificant [9]. We were limited to 1024 tokens, which can be thought of as roughly $\sim$ 600 words. The reason for this will be discussed in section 4.4. We did not remove infrequent terms as Mullenbach et al., since it did not improve our predictions. We assume that BERT's use of the WordPiece algorithm, which handles infrequent words by splitting them helped here. WordPiece splits them in such a manner that the subwords again are known tokens for BERT [29].

4.3.1 Numbers

Especially numbers seem to be of great importance for the success of the training process. We tested several different variations of preprocessing to discover the most effective one. In particular, we tested the variants shown in the table 4.2.

As we can see in the figure 4.2 the Mullebach variation is the most suitable for

Identifier	Example Before	Example After using Preprocessing
No Editing	[[Dr. 7]] treated patient with 25mg and was given 7 pills.	[[Dr. 7]] treated patient with 25mg. and was given 7 pills.
Removing Numbers Everywhere	[[Dr. 7]] treated patient with 25mg. and was given 7 pills.	[[Dr.]] treated patient with mg. and was given pills.
Removing Numbers and Replacing the brackets	[[Dr. 7]] treated patient with 25mg. and was given 7 pills.	Dr. treated patient with 25mg. and was given 7 pills.
Removing Numbers, except inside the brackets	[[Dr. 7]] treated patient with 25mg. and was given 7 pills.	[[Dr. 7]] treated patient with mg. and was given pills.
Mullenbach Variation	[[Dr. 7]] treated patient with 25mg. and was given 7 pills.	[[Dr. 7]] treated patient with 25mg. and was given pills.

Table 4.2: Examples of the different number processing forms.

our purposes. While the difference is not significant, we considered that using the same preprocessing as other works would improve comparability [9, 17]. Therefore from here on, we will mostly focus on this form of preprocessing.

Hyperparameters for comparison

We used the hyperparameters listed in the table 4.3 for this comparison.

Hyperparameter	Value
Optimizer	Lambda
Learning Rate	0.0001
Scheduler	Warmup Cosine
Batch Size	16
Precision	Half Precision

Table 4.3: Hyperparameters used for selection of preprocessing variation

4.4 Sequence Length

One of the most important hyperparameters in BERT models is the sequence length. The sequence length describes how many tokens can be consumed by the

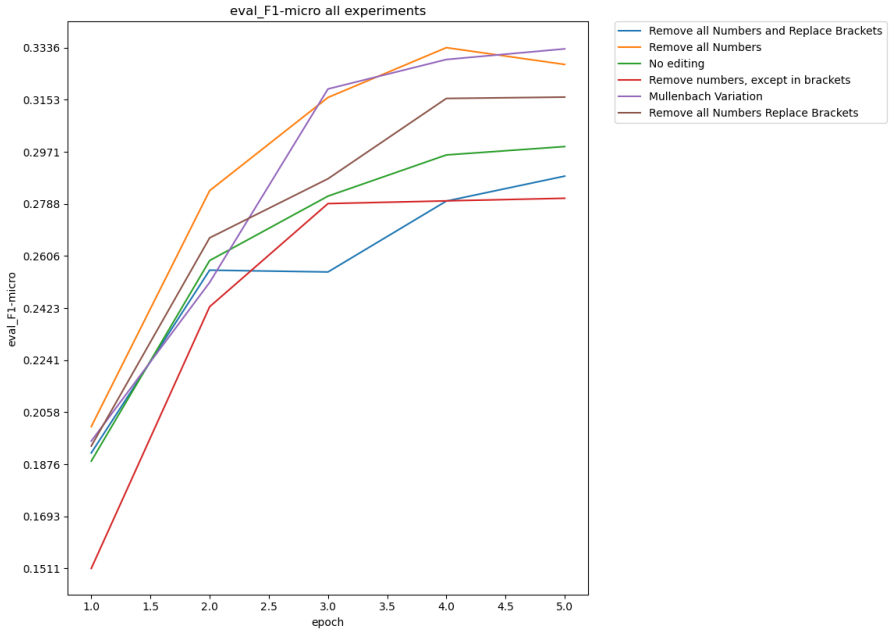

Figure 4.2: Comparison of the different preprocessing methods, data as described in section 4.1, preprocessing as described in section 4.3

BERT model. A token is one entry in a BERT vocabulary.
Tokenization describes the process of converting text to a list of tokens. This process is done such that if a word exists exactly in the vocabulary, it will be converted to one token covering the entire word. If the word is not present in the vocabulary, the word is split into smaller pieces and converted to more than one token. For example, assume that our vocabulary contains ["cardio," "vascular"], and the word cardiovascular will be represented as the combination of the two tokens ["cardio," "##vascular"]. This process is powered by the previously described algorithm called WordPiecing [29].
This particularity of the BERT model is especially important for us due to the relatively long texts in the Mimic3 discharge summaries. In the figure 4.3, we can see the distribution of word counts in our dataset. In the figure 4.4, we can see the distribution of token counts in our dataset. In the table 4.4, we can see the distribution of words and tokens per sample.

We compared the influence of the sequence length on the micro F1 score. In the figure 4.5, we can clearly see that the performance increases as we increase the sequence length. Ideally, the sequence length would be increased even further. However, on GPUs available to us in Oct.2020, we can not train with bigger sequence lengths than 1024. As we can see in the figure 4.5 the improvement from 512 to 1024 is significant but not as huge as one might expect. We assume the reason for this is, as also pointed out by Chen et al. [26] that a lot of

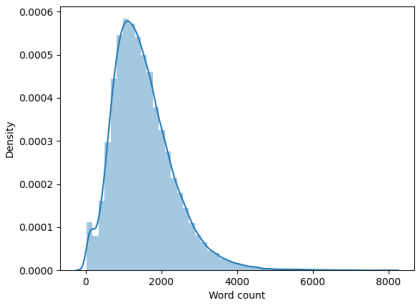

Figure 4.3: Word count in mimic3 discharge summaries.

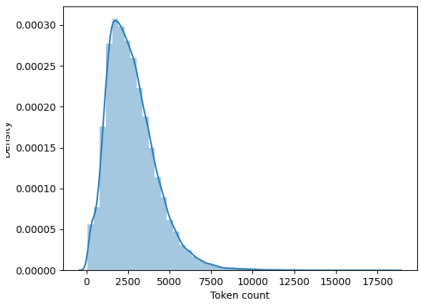

Figure 4.4: Token count in mimic3 discharge summaries.

Metric	Word Distribution	Token Distribution [23]
mean	1519.50	2740.12
std	800.54	1492.80
min	21.00	78.00
Q1	957.00	1657.00
median	1394.00	2500.00
Q3	1948.00	3547.00
max	7980.00	18429.00

Table 4.4: Comparison of the word and token distribution in mimic3.

information is concentrated in the beginning.

4.5 Training Time and Hyperparameters

BERT architectures are known to be costly to train. While we found additional training time to be beneficial, we often constrained ourselves to around 2 days of training time. This allowed our model and process to be evaluated biweekly. Our longest model was trained for roughly 8 days, and the training time mainly depended on the amount of data.
While halfprecision decreased the epoch time significantly, it also decreased the F1 score of the model's in our experiments. We mostly used halfprecision training for parameter tuning and prototyping.
Unless otherwise noted, the hyperparameters in the table 4.5 were used to train our most competitive models. These hyperparameters were found using grid-search but were only reevaluated seldomly, since testing one setting took 2 GPU days.

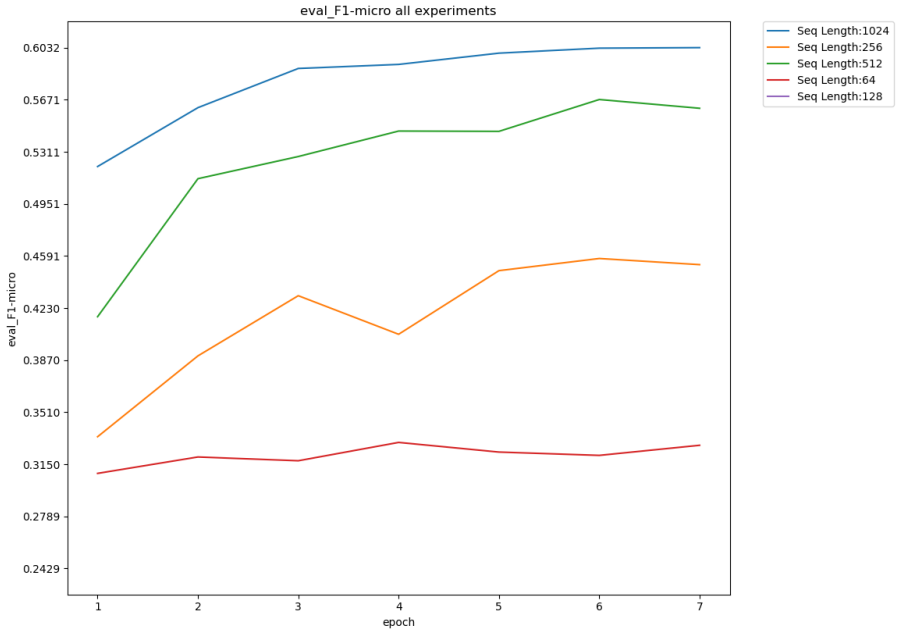

Figure 4.5: Illustration highlighting the impact of the sequence length on the F1 score, data as described in section 4.1

Hyperparameter	Value
Optimizer	Lambda
Learning Rate	0.0005
Scheduler	Warmup Cosine
Batch Size	2
Precision	Full Precision
Sequece Length	1024

Table 4.5: Hyperparameters used for all 1024 sequence length models.

4.6 Training Direction

Transforming the text such that BERT is trained from different views is desirable if the text length exceeds the maximum possible length trainable with BERT. The issue was discussed in the section 4.4, namely that we can only train with 1024 tokens.
Different directions in training yield good results as Sun et al. [39] already found, we decided to evaluate them. The 3 basic different directions that we tried are Front, Back, and Mixed. Assuming a sequence length of 1024, the Front is the first 1024 Tokens. The Back is the last 1024 tokens. Mixed is the combination of the first 512 Tokens and the Last 512 tokens of the text. We evaluated the

differences in performance between the 3 versions. We can see the results in the figure 4.6. As we can see, the mixed transformation is clearly superior to the

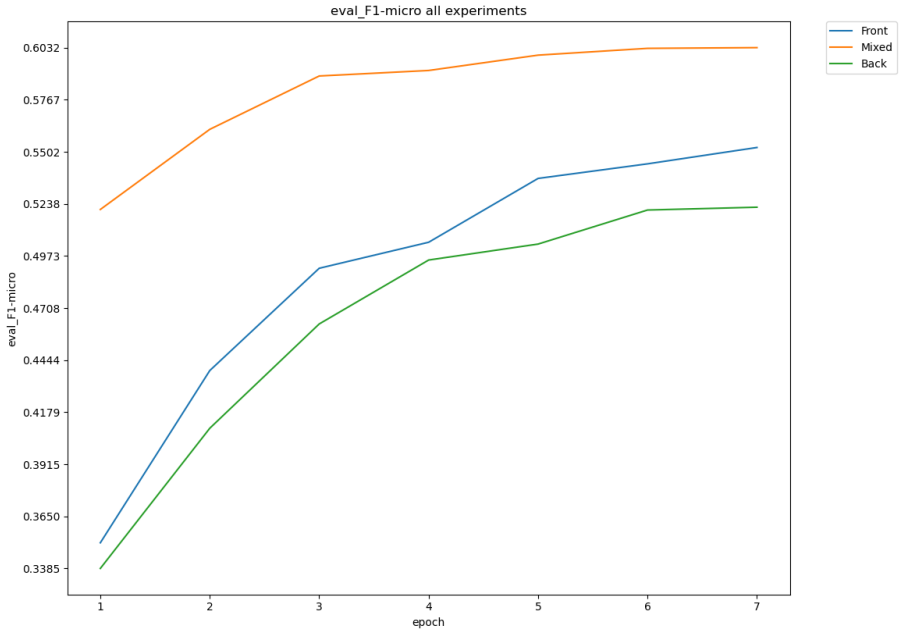

Figure 4.6: Comparison of the impact of training direction on the F1 score, data as described in section 4.1, text split as described in section 4.6

other two transformation methods. The Front version is the default behavior of most models and also the default used by Huggingface [30]. Surprisingly in our case, Mixed would be a far better choice if one was limited to only one method. We assume that especially since the addendum and the discharge medication and steps are often documented in the end, having a certain amount of Back information is valuable.

4.7 Paragraph Splitting

After observing the samples closely, we found that the discharge summaries expose paragraphs. We define paragraphs as pieces of texts that start at the beginning of a new line and are a few words, followed by a ":". An example of paragraphs may be telephone and fax in the following example.

Example:
telephone: daily dose of X increased.
fax: daily dose of Y reduced.

We can see a complete list of the 200 most common paragraphs in the table 4.2, and the figure 4.7 shows their frequency. 100% Frequency means that the paragraph exists in 100% of the samples. The reason for one paragraph occurring in 180% is that it is a special paragraph. When it occurs in a text, it usually occurs multiple times. The respective paragraph is called "disp." and is a medical abbreviation describing medicines the patient is given, e.g., "disp. 30 pills X per month".

Paragraph	Occurs in %	Paragraph	Occurs in %	Paragraph	Occurs in %	Paragraph	Occurs in %
disp	188.5%	physical examin	14.6%	micro	4.0%	lives with	2.5%
		chest	13.6%	conclusions	4.0%	psych	2.5%
admission date	99.9%	pulse	13.5%	pulmonary	4.0%	respiratory	2.5%
service	99.7%	cardiac	13.2%	sensation	3.9%	pe	2.5%
history of pres	92.8%	discharge diagn	13.0%	ii	3.9%	hpi	2.5%
past medical hi	90.7%	condition on di	12.6%	hematology	3.7%	contraindicatio	2.4%
allergies	87.8%	sig	10.4%	xii	3.6%	id	2.3%
date of birth	86.7%	heart	9.9%	care and recomm	3.6%	language	2.3%
social history	81.9%	medications	9.4%	gi	3.6%	discharge exam	2.2%
discharge medic	78.1%	phone	9.4%	height	3.6%	physical examin	2.2%
discharge dispo	77.6%	primary	9.1%	viii	3.6%	doppler	2.2%
medications on	76.1%	admission labs	9.0%	secondary diagn	3.5%	admission exam	2.2%
discharge diagn	75.9%	pulm	9.0%	reflexes	3.5%	gastrointestina	2.2%
family history	75.8%	gu	8.7%	psh	3.4%	admission physi	2.1%
chief complaint	75.6%	imaging	8.5%	final diagnosis	3.4%	rectal	2.1%
discharge condi	73.5%	secondary	8.2%	mitral valve	3.3%	cardiologist	2.1%
physical exam	73.1%	neurologic	8.2%	aortic valve	3.3%	back	2.1%
attending	72.3%	on admission	7.7%	tobacco	3.3%	cards	2.1%
discharge instr	72.0%	name	7.6%	left ventricle	3.3%	carotid bruit r	2.1%
brief hospital	71.1%	indication	7.6%	right ventricle	3.3%	varicosities	2.0%
major surgical	70.9%	reason	7.3%	studies	3.3%	labs on dischar	2.0%
pertinent resul	69.5%	reason for this	7.3%	dp right	3.3%	the following c	2.0%
followup instru	68.9%	cardiovascular	7.2%	radial right	3.3%	cc	2.0%
impression	66.5%	follows	6.8%	femoral right	3.3%	unit no	2.0%
heent	56.4%	review of syste	6.6%	incision	3.2%	primary pediatr	1.9%
facility	46.1%	history	6.6%	for days	3.2%	vital signs	1.9%
neck	41.5%	past surgical h	6.5%	tricuspid valve	3.2%	laboratories on	1.9%
completed on	39.2%	resp	6.4%	microbiology	3.2%	name of primary	1.8%
neuro	39.0%	discharge labs	6.1%	disposition	3.2%	appointment	1.8%
ext	36.5%	discharge statu	6.1%	aorta	3.1%	primary diagnos	1.8%
cv	33.2%	on discharge	5.9%	infectious dise	3.1%	echo	1.8%
lungs	32.8%	motor	5.7%	occupation	3.1%	medication chan	1.8%
abd	30.8%	laboratory data	5.6%	labs on admissi	3.0%	medications on	1.8%
general	30.7%	location	5.3%	secondary diagn	3.0%	note	1.8%
abdomen	29.7%	pmh	5.3%	contrast	3.0%	immunizations r	1.8%
gen	29.6%	address	5.2%	source	2.9%	sex	1.8%
dictated by	27.7%	primary diagnos	5.2%	transitional is	2.9%	discharge date	1.7%
provider	26.6%	cxr	5.0%	physical examin	2.8%	psychosocial	1.7%
mental status	24.8%	comparison	4.8%	left atrium	2.8%	recommendations	1.7%
skin	24.2%	sensitivities	4.8%	coordination	2.8%	diagnosis	1.7%
extremities	22.9%	ros	4.6%	extr	2.7%	specialty	1.6%
level of consci	19.6%	etoh	4.6%	sensory	2.7%	neurological ex	1.6%
activity status	19.5%	cranial nerves	4.5%	pericardium	2.7%	pt	1.6%
findings	19.1%	left	4.3%	addendum	2.7%	cvs	1.6%
hospital course	17.5%	ekg	4.2%	approved	2.7%	abdominal	1.6%
department	17.2%	right	4.2%	general comment	2.7%	interpretation	1.6%
vs	16.6%	nebulization si	4.1%	comments	2.7%	test type	1.6%
building	16.1%	medications on	4.0%	cor	2.6%	we made the fol	1.6%
campus	16.1%	other labs	4.0%	orientation	2.6%	medications at	1.6%
vitals	15.5%	technique	4.0%	surgeon	2.5%	hospital course	1.6%

Table 4.6: List of the 200 most common paragraphs

We split each text into these paragraphs. After this, we counted the frequency of each paragraph type and ordered them by frequency. We decided to take the 200 most frequent paragraphs since, after this, they become very sparse.

As we notice, there is a big chunk of very common paragraphs that are also the most influential factors, as we will later show. However, certain rare paragraphs are still valuable since they are informative for less common diseases.

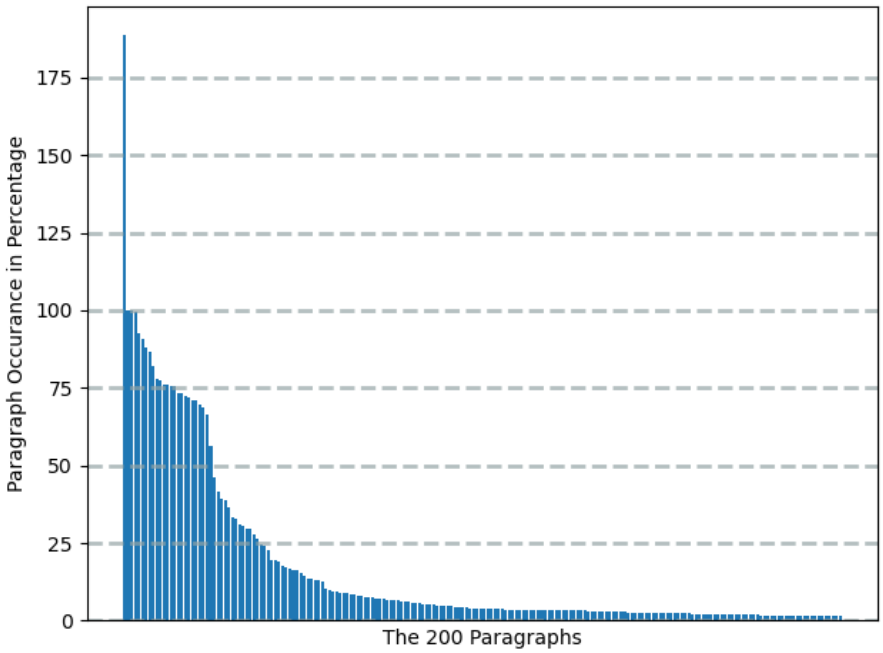

Figure 4.7: Distribution of the paragraphs, data as described in section 4.1

To extract these paragraphs, we additionally had to preprocess the paragraphs. The reason for this was that they are often not identical. For the more common paragraphs like "Sex" "History of present illness," the writing was often identical, but this was rarely the case for the less common words. For example, the paragraph "lung" occurred in several styles, e.g. ["lung," "lungs," "Lungs," "LUNGS," "Lung"]. To harmonize these styles, we had to process them by using the steps.

- Casting to lower case
- Stemming (Snowball stemming)
- Removing stopping words (in case the paragraph is multiple words)

After using this procedure, we were left with 25'909 unique paragraphs. Out of these, roughly 17'000 occur in this specific combination only once.

4.8 Paragraph Embedder

Using the paragraph splitting method, we discovered two different possible embeddings. We will call them Paragraph embedding and Rest Text embedding.

These two are generated from the same transformation, but their respective training set is created differently.
The Paragraph Embedder is trained on the augmented paragraph dataset. The augmented paragraph dataset is generated by applying the paragraph splitting method described in the section 4.7. In detail, this works by applying this splitting/transformation method to each sample. This could theoretically generate up to 200 new samples. In practice, most text samples have between 10 and 45 paragraphs, as shown in the figure 4.8. Our dataset for the bigger training set

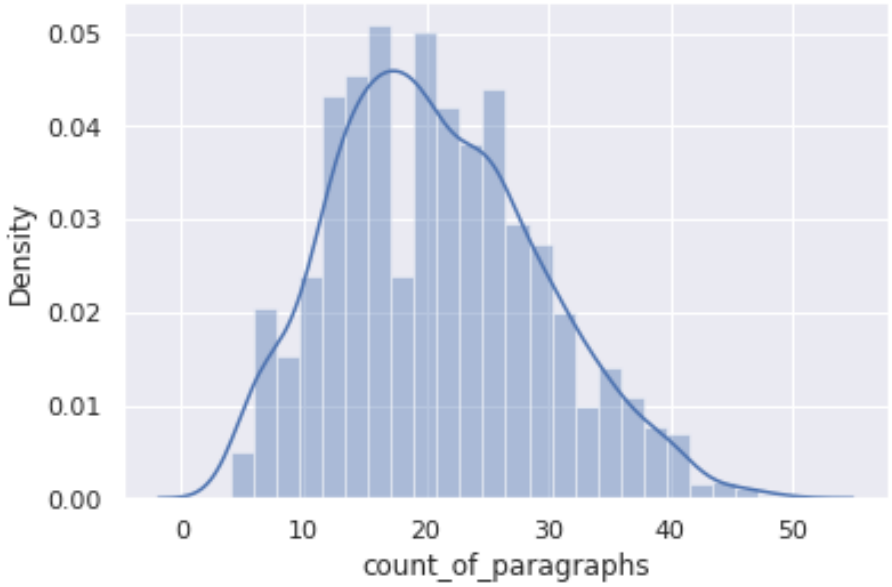

Figure 4.8: Distribution of paragraphs found per sample, data as described in section 4.1

with 44'175 samples in total will increase to 898'616, roughly by a factor of 20. The numbers are similar for the validation and test set.
We consider the tuple (augmented sample, original labels) as a new training example in paragraph dataset. The length of these new paragraphs is significantly shorter than the original discharge summaries' length, as we can see in the figure 4.9. The 75% Percentile is at 52, which indicates that the texts are overwhelmingly concise. The reason for this is that many of the paragraphs that occur frequently have concise texts. For example, the paragraph "physical exam" is in over 70% of cases answered with a phrase like "OK."
We trained this system with a sequence length of 256. We did so since it improved the score and only increased the training time. In general, however, 128 would have been enough to obtain most of the information.
Once we trained this system, we obtained the Paragraph Embedder. To produce the embedding, we now converted each sample into the paragraphs it contained.

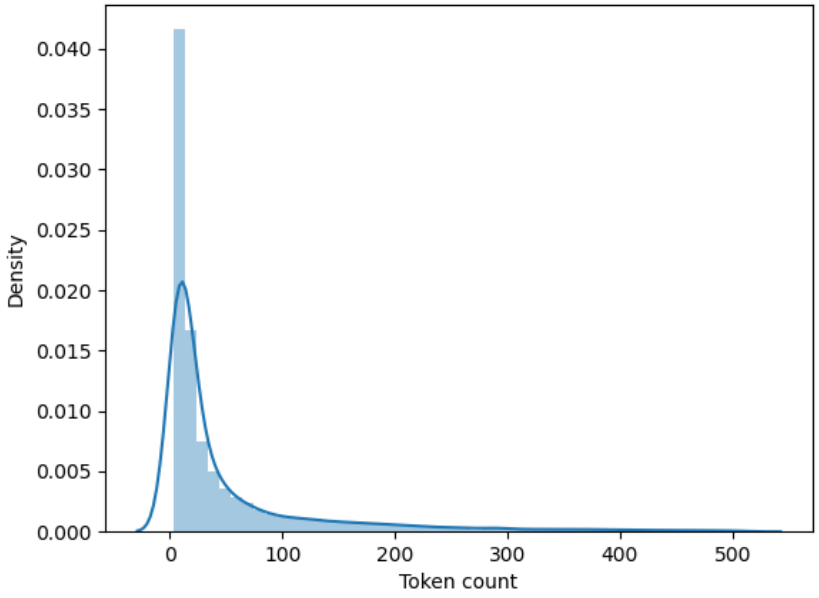

Figure 4.9: Distribution of token length in the paragraph dataset, data as described in 4.1

Of these paragraphs, all that was in the 200 most common paragraphs were embedded using the Paragraph Embedder to obtain the vector $x_i \in \mathbb{R}^{768}$.
If paragraph p does not occur in the sample, it is defined to be the zero vector. The paragraph Embedding is now defined as the concatenation of the $p = 200$ paragraph embeddings.

$$ParagraphEmbedding = [x_0, x_1, ...x_p]$$

$$x_p = \begin{cases} ParagraphEmbedder(paragraph) & \text{if paragraph p in text} \\ 0^{768}, & \text{if paragraph p not in text} \end{cases}$$

This makes a total of $200 * 768 = 153'600$ variables, where a significant amount of these variables is 0.

Relevant Details

Before we embedded the text using the ParagraphEmbedder, we add the paragraph text in-front of it. We do so due to potential downsides the preprocessing steps listed in the section 4.7 had. For example, if one text contained "allergie" singular and the other "allergies" in the plural, then the model would still produce slightly different embeddings but be in the same position in the embedding

vector.
In the special case that one paragraph is included multiple times in a sample, we concatenate the texts. From our experience, this almost exclusively happens with the "disp." paragraph. The concatenated texts are then embedded jointly by Paragraph Embedder.

Example:
p:"disp.: 30pills X"
p:"disp.: 5mg. Y"
p_0 :"disp.: 30pills X disp.: 5mg.Y"
$embedding_{disp} = ParagraphEmbedder$("disp.:30 pills X disp.: 5mg.Y")

4.9 Rest Text

Only the 200 most common paragraphs are embedded. All other detected paragraphs that are not in the 200 most common are concatenated and added to the so-called Rest Text. We do so since paragraphs are very common, as we have described in the section 4.7, but a significant amount of them can't be categorized into the 200 top-paragraphs. This Rest Text includes a significant portion of the information that would be lost if not embedded here.

Example:
p:"telephone/fax 1: daily dose increased."
p:"telephone/fax 2: daily dose reduced."
p_{rest} :"telephone/fax 1: daily dose increased. telephone/fax 2: daily dose reduced."

The length of the rest text is significantly different from the paragraph embedding. This justifies processing this paragraph collection separately. The distribution of tokens can be seen observed in the figure 4.10. The 75% percentile is at 923, which justifies training it with a sequence length of 1024 instead of 512. We train the Rest Text Embedder with the same hyperparameters as the front, mixed, and back as described in the table 4.5.

4.10 Chunked Embedding

A chunked dataset can be generated by taking discharge summaries and splitting them into smaller chunks. One needs two hyperparameters the Word Count Per Chunk(WPC) and the Chunk Count(cc). We generated a chunked dataset by

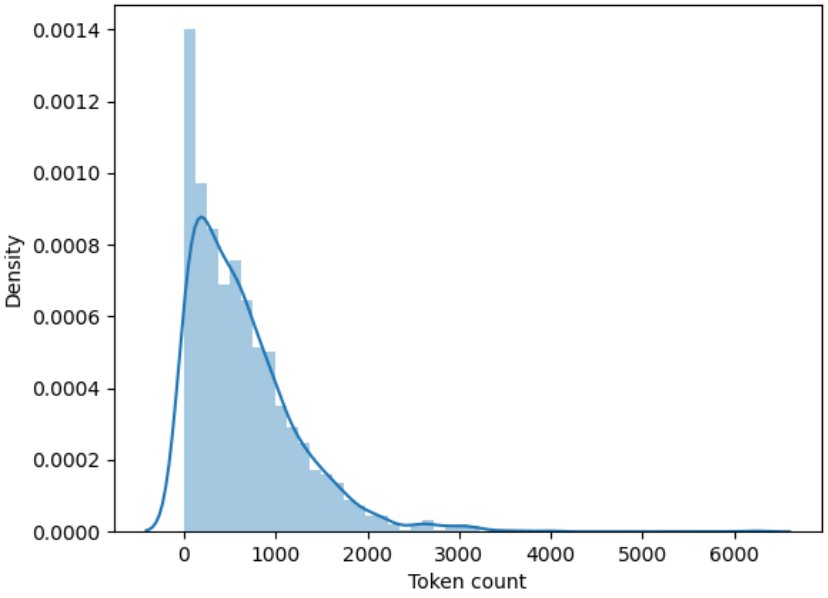

Figure 4.10: Distribution of token length in the rest text dataset, data as described in 4.1

splitting each sample of length X into as many sequential chunks/pieces of text as possible. Such that the chunks are direct neighbors and have no overlap. The Chunked Embedder was then trained on $samples{*}average\_chunks\_per\_sample$ samples, where each sample is trained with the original labels.
To generate a Chunked Embedding, we now split each text into its chunked parts, embedd each part, and concatenate these embeddings into a $cc * 768$ long embedding. If the text is too short to do so, we fill the missing chunks by 0's. We tested this Embedding form and found that it unperformed in comparison with Paragraph Embedding. We assume that the Paragraph Embedder outperforming the Chunked Embedder is that the paragraphs are aligned while the chunks are not. Since the chunked embedding did not improve the predictions, we did not continue to explore it in this form.

4.11 Random Embedding

Similar to the Chunked Embedding, one can also find the chunks by splitting them randomly. We found that this works better than the chunked embedding. The main advantage is that using this way; we can sample theoretically a nearly infinite amount. To generate a random dataset, one defines two hyperparameters.

One is the number of random draws(r), and the other is the number of random embeddings(b). Using a sequence length of 1024, we can now split a sample with length l into r new samples. We do so by drawing r random points $p \in [0, l-1024]$. For each of these points p, we now add the text from token p until token $p+1024$ to the dataset for random embeddings.
We now train the Random Embedder on these texts. In total, there will be $new_{samples} <= b * samples$ samples since some rare texts have less than 1024 tokens. The main advantage is here that one can augment the number of samples in the dataset significantly. However, the resulting set is also less homogenous.
Once the Random Embedder has been trained, we can create an embedding by drawing b samples. These samples are then embedded using the trained Random Embedder. These outputs are then concatenated, which results in the Random Embedding with $768 * b$ variables. In case the text has less than 1024 tokens, the sample will be 0 padded until $768 * b$.

4.12 Sentence Embedding

Sentence Embedding is a special form of embedding. To generate a Sentence Embedding, we tried to gather more information about each illness. Our general goal is to extract a list of k terms for each illness, with relative importance for the specific illness compared to all other illnesses. This list is then used to separate the sentences, which include the words in those lists from those that don't. The hope is that we can reduce the induced noise while training by being sure that the respective sentence contains the respective label. If we trained on all sentences with all labels, the training process would not work since a sentence usually contains at most one ICD code.
To achieve this, we scan the document collection using term frequency–inverse document frequency (TF-IDF). TF-IDF generates given our dataset, preprocessed as described in the section 4.3, a matrix $M \in \mathbb{R}^{d \times w}$ d is the d-th document (discharge summaries), and w is the w-th word, excluding stop words. The entry M_{ij} can be interpreted as the relative importance of word j in document i. We defined then for each illness, a vector using the following procedure.
To find the relative importance of a word, we generate for each illness x two vectors. One is the mean of all documents d, which include the illness x. We call this vector x_{is}. The other one is the mean of all documents that do not include the illness, named x_{isnot}. To generate now a vector which includes at each position the relative importance of word w, we define $y = x_{is} - x_{isnot}, y \in \mathbb{R}^w$. The top-k-words can now be generated by taking the k words with the highest value in y. This is the list L_i the list of predictive words for illness i.
To generate the training set Sentence, we now, for each sample, split the discharge summary into its sentences. A sentence is defined by either '.' or a 'new line.' We now define the new sample's labels as 1 for illness z if illness z is in the sample labels, and any of the words contained in the sentence are in the list

L_z. In practice, we chose $k = 20$ since the training set becomes too big and includes too many sentences if chosen bigger. We can now define the training set Sentence, which includes all sentences with the respective labels. We can now train the Sentence Embedder on this Sentence dataset.
Once this Sentence Embedder has been generated, we have discovered two possibilities to define a sentence embedding for discharge summaries. In one, we encode all sentences. In the other, we encode only the sentences with any of the words in any of the lists L_i in it.
We found that we can constrain the sentence embedding to 200 sentences, since this is enough to encode most discharge summaries, and 50 for the case, we encode only the sentences containing any of the words. For each of the 200 sentences, we generate a sentence embedding $a \in \mathbb{R}^{768}$.
To generate the embedding for the discharge summaries, we now concatenate the 200 sentence embeddings. If a discharge summary has less than 200 sentences, we append 0's, such that a sentence embedding for one discharge summary is now defined to be c, $c \in \mathbb{R}^{200}$ or $c \in \mathbb{R}^{50}$

4.13 Illness List Embedding

A beneficial method we used was the Illness List Embedding. The general idea of this approach was that after we trained Paragraph Embedder, we could define for each paragraph how predictive it was for specific illnesses. We would define this by using the Paragraph Embedder directly for predicting the illnesses from the respective paragraph.
The idea is now that once we know how predictive paragraph i is for illness j, we could define a list of size k consisting of the most predictive paragraphs for the illness. The average of the non-zero paragraphs that were both existing in the sample and the list for illness i did then defined a new embedding called I_i. Since we had 50 illnesses, this new embedding was of size $I \in \mathbb{R}^{50\times768}$.
To defined the mentioned predictiveness of paragraph i for illness j we then predicted for each sample in the validation data, the illnesses and compared them to the true labels. A prediction made by the Paragraph Embedder for an illness was considered to be 1 if the probability threshold of 0.25 was surpassed, i.e. $ParagraphEmbedder(x)[i] > 0.25$. We then filled a matrix M_{ij} were each illness i, and each paragraph j had a defined F1 score. We can see the resulting matrix in the figure. 4.11 In practice, we calculate the entry M_{ij} by using all samples with both a paragraph j and an illness i.
We defined each illness list, which consisted of the 50 most predictive paragraphs for this illness. We only considered combinations with an F1 score of above 0.25 since some illnesses have no predictive paragraphs. The addition embedding for each illness is now the average of these non-zero paragraphs.

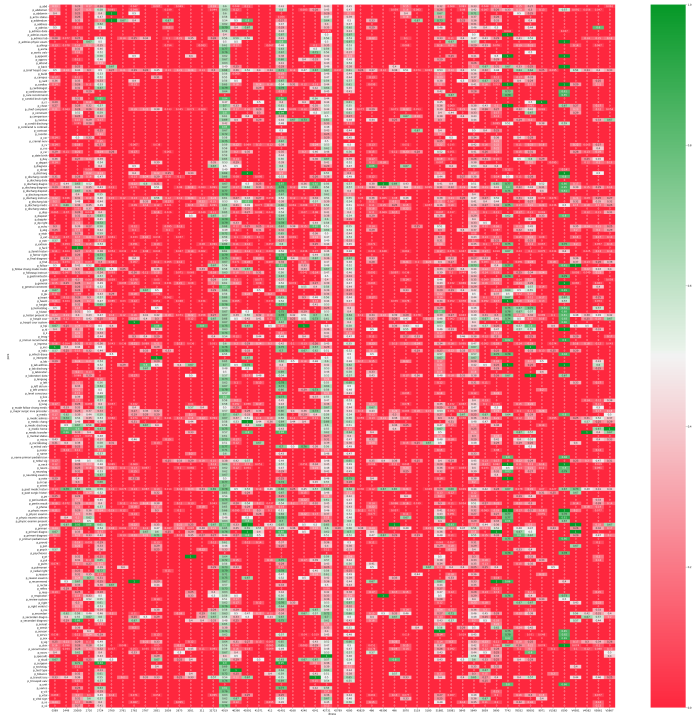

Figure 4.11: Decision matrix for Illness List Embedding, data as described in section 4.1,

4.14 Additional Variables

To the Embedding, we add variables to encode non-textual information. We generate these variables directly from the text. Additional variables include,

- Gender
- Age
- Stay Time
- Error

We parsed these variables from the text by looking for indicators. The Error column would be 1 if we did not manage to extract the other variables from the

text. However, this was the case in only 7% of all discharge summaries. We normalized each of these variables (except Error).

Chapter 5

Predictor

The Predictor operates on the output of the Embedder(s). Taking as an input their last layer weights in the form of a vector. We tested this process on several architectures we are going to describe in the following sections. In contrast to the Embedder model, which was realized using Hugginface and FastBert, the following architectures have been build using PyTorch Lightning [40].

5.1 General Architecture

We had several common aspects to all our Predictor models. The input was always of the form $x \in \mathbb{R}^{N \times e \times 768}$, N refers here to the batch size e to the number of embeddings.
At the output level, we always had to produce a variable $y \in \mathbb{R}^{N \times 50}$. Each of the 50 values can be interpreted as the probabilities that the respective illness is present in the sample. As an activation function in the last layer, we always used a sigmoid function.
We initially used a binary cross-entropy loss to train the neural networks. Later on we switched to use the binary cross entropy loss with logits. This loss combines the binary cross entropy loss with a sigmoid and makes it numerically more stable. The loss is defined as,

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top,$$
$$l_n = -w_n \left[y_n \cdot \log \sigma(x_n) + (1 - y_n) \cdot \log \sigma(x_n) \right)] \text{ [40]}$$

Where N is the batch size.

5.2 Simple Dense Architecture

We tried several variations of this architecture. In general, when not tuning the hyperparameters heavily, this architecture tends to be one of the best. When

we refer to a simple dense architecture, we mean an architecture containing only fully connected linear layers. We found that using the combination,

- Linear Layer
- Layer Normalization
- PRelu
- Dropout

To work best, thus we will refer to this combination as a linear layer from now on. While we tried different normalization forms, we found layer-wise to work best. We assume this is due to the input being somewhat normalized as they are the output of a BERT model itself. We found using the popular PRelu beneficial, which is a form of Relu or Leaky Relu, where the steepness of the leaky part of the function itself is learned. We used at least one hidden layer but found using two to work better. After this, increasing the number of linear layers seemed to bring no benefit. We can see an overview of this architecture in the figure 5.1.

5.3 Simple Dense Architecture, Force Through Smaller

One of the clear issues with the simple dense architecture is that when we are working with more than 200 embeddings, we have more than 150'000 variables. This makes it hard for the neural network to learn the right variables to pay attention to. To reduce this challenge, we tried to force each embedding of size 768 through a smaller layer, with a size $o \in [5, 50]$. This effectively reduces the amount of parameter later on, from e*768 to e*o. An overview can be seen in the figure 5.2. As we will later show in the section 6.4 this approach was mostly costly and did not help.

5.4 Aggregating Techniques

Initially, we discovered that a faster process for aggregating embeddings exists. One can trade-off F1 score for training time. This training time can then again be traded for more F1 scores in the grid-search process. We tested in our implementation several variations of this process at several points in the architecture. We found that especially aggregating the uncommon paragraphs was beneficial and significantly reduced the Predictors' training time. In general, we tried the following aggregation techniques.

- Average

- Median after weighting through Neural Network
- Sum
- Average after weighting through Neural Network

We used these aggregation techniques exclusively with the 200 paragraph embeddings and the 50 illness lists. As we showed in the table 4.6, the idea behind this is that most paragraphs are 0 for most samples and therefore aggregating them is sensible. We found that the median of the paragraphs yielded a good result. In most cases, it's performance was only slightly worse than the same architecture without the averaging. This was extremely useful in reducing excessive training time. The results achieved by averaging after weighing the vectors via neural networks were almost as good as not aggregating. Since it was slightly beneficial to weight the terms by a neural network, we did so.

5.5 Transformer Encoders

The transformer encoder layer is made up of self-attention and feed-forward networks. This encoder layer is based on the works of Vaswani et al. [28].
The main advantage of these layers is that they learn what parts of the input sequence the decision needs to attend to, i.e. what "importance" weighting needs to be assigned to each input embedding. In our implementation, we used it in two different ways. Initially, we used it to input each of our embeddings as a separate "token." This process worked very well but was trained for an excessively long time and was hard to tune well. Later on, we used the transformer architecture only for the paragraph part of the architecture to find the individual embeddings "importance" weighting. After the transform encoder, we then aggregated the resulting embeddings. This process showed to work just as well as the full transformer architecture but saved us roughly 70% of the training's time.

5.6 Best Model

Throughout all our experiments on the smaller dataset, the following architecture has yielded the best results. In the section 6.6, we will show which embeddings contributed how much to the final result. For this model, we used the embeddings,

- Front, Back and Mixed as described in section 4.6
- Paragraph Embedding as described in section 4.7
- Rest Text as described in 4.9

- Illness List as described in 4.13
- Additional Information as described in 4.14

We got in total $200 * 768$ variables from the paragraph embedding. Another $3 * 768$ variables from the front, back, and mixed embedding. $1 * 768$ from the rest embedding. $50 * 768$ variables from the illness list embedding. And 4 variables from the additional information embedding. This adds up to a sum of $(200 + 50 + 4) * 768 + 4 = 195'076$ input-variables in the Predictor step.
We realized that both the paragraph and illness list embedding have varying degrees of information depending on the individual sample. To deal with this, by teaching the model to learn when to weight which information how much, we feed these embeddings through a transformer encoder. For both, we later aggregate the results into 768 variables each.
The other variables, namely the rest, front, back, mixed, and additional embedding, are treated separately. We move them through a separate dense layer before finally merging them in the last 2 linear layers. We found bigger is better here.
After these 3 separate aggregations, we again aggregate the resulting variables through a dense layer. Finally, we predict the illnesses after applying a sigmoid activation function. We can see this process visualized in the figure 5.3. We generally found that this system works best with a large L_i layer. The biggest and best possible size was 2048. This is due to constraints on modern GPUs. We found 1024 to work best for the second layer. We, in general, considered [128,256,512,1024,2048]. Dropout was tested on both the first and the second layers, in steps of 10%. We found that 10% worked best. The impact of the machine precision in the layer normalization layer was marginal.

5.7 Other Notable Aspects

5.7.1 Optimizers

We tested several optimizers but found that the Lamb optimizer yielded the best (and fastest) results. The performance of Adam optimizer and AdamW were similar, but both took longer to compute the answer.

5.7.2 Early Stopping

We used early stopping to terminate the training process. Our early stopping stopped the optimization process after 10 epochs of patience if it detected that the evaluation loss did not decrease further.

5.7.3 Batch Size

In the Predictor, we generally used the biggest possible batch size. We never exceeded 256 since if we increased it further, there were too few batches for the smaller dataset. The biggest possible batch size was not always identical, depending on the GPU we used. It's worth noting that the performance slightly varied depending on the underlying GPU.

5.7.4 Learning Rate

To finetune the learning rate, we generally used the theoretical work of Smith et al. [41] called cyclical learning rates. This feature was implemented in PyTorch Lightning and helped to reduce the hyperparameter tuneing time significantly [40]. We generally found that the found learning rate was too low by roughly a factor of 10. So after finding the optimal learning rate using the cyclical learning rates, we multiplied it by a factor of 10.

5.7.5 Learning Rate Decay

We tested the most common forms of learning rate decays. We specifically tested exponential learning rate decay, epoch number based, and linear learning rate decay. We found that a constant learning rate was better than epoch number based and exponential learning rate decay. Using linear learning rate decay showed the best results. We Generally found that $decay = \frac{learningrate}{Nr.EpochsThisRunEndedLastTime}$ yielded the best results. This was then deducted after each epoch.

5.7.6 Weight Decay

As we added more parameters to the Predictor model, we discovered that weight decay began being useful. Initially, it showed to be useless or even harmful to the F1 score, but after around 170'000, we found weight decay to be useful. We mostly used the the value $1e-3$.

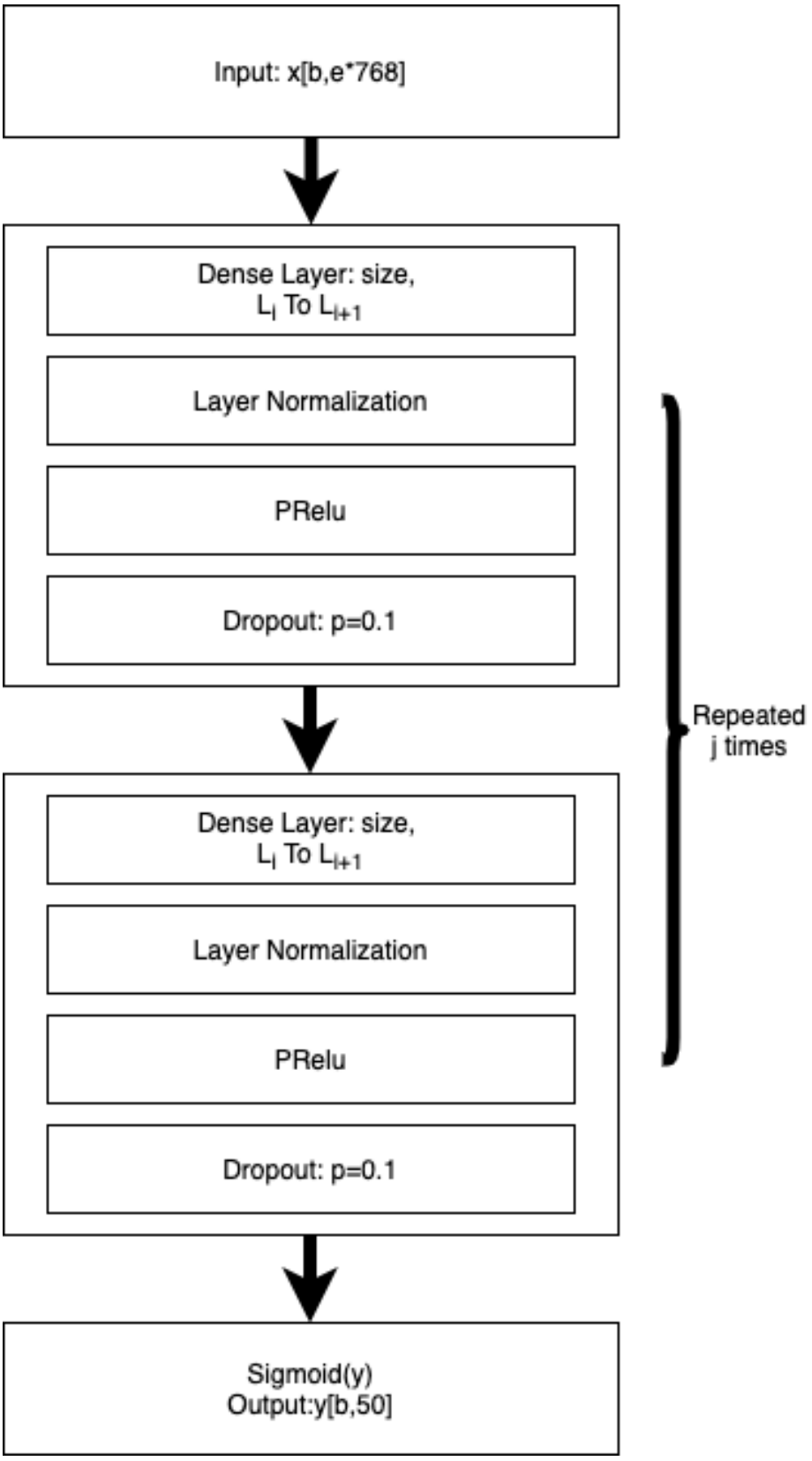

Figure 5.1: Simple Dense Architecture

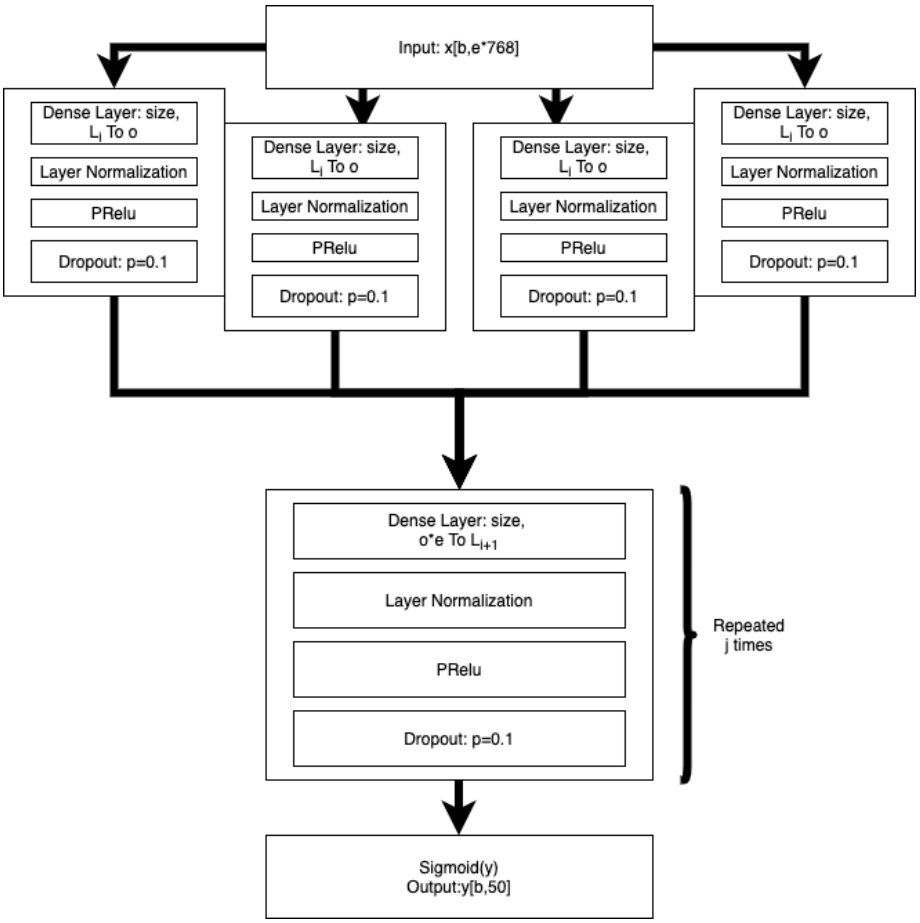

Figure 5.2: Simple Dense Architecture, Force Through Smaller

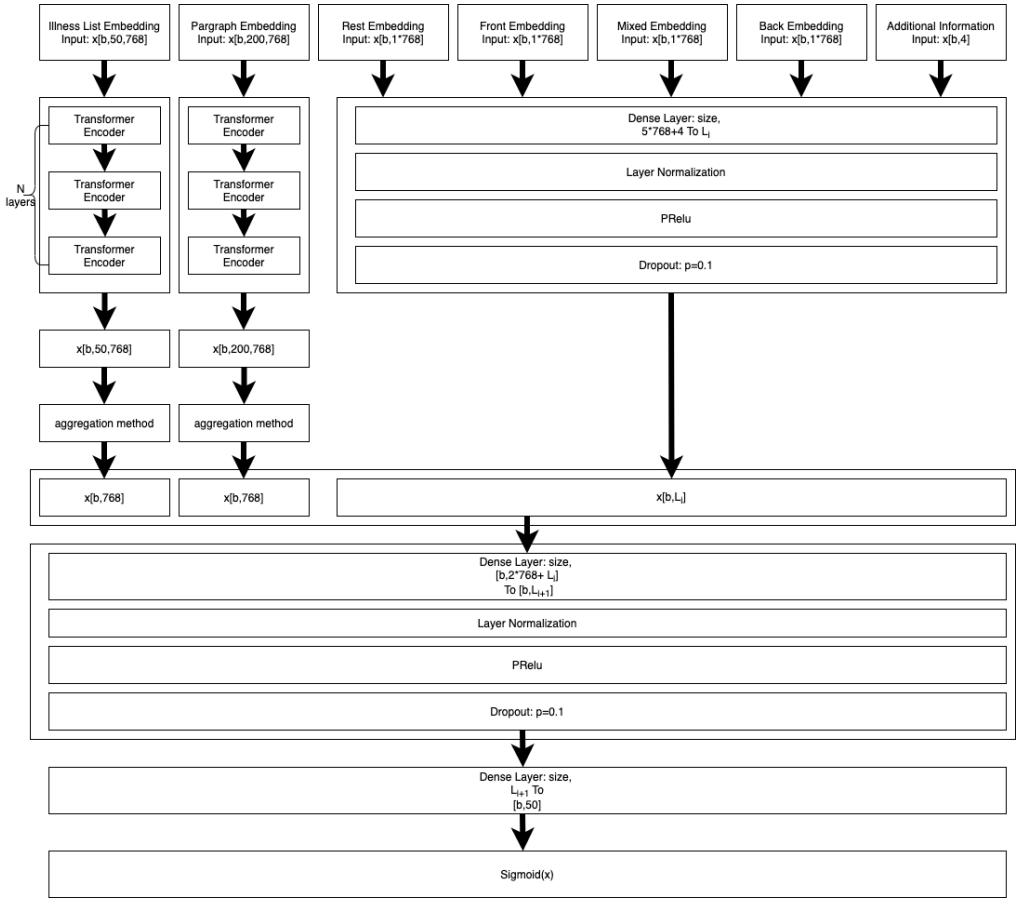

Figure 5.3: Best Model Architecture

Chapter 6

Evaluation & Interpretation

After training our Embedders and predicting each illness's final probabilities, we calculated its scores by comparing the F1 and AUC score. For the F1 score, we took the threshold at 0.25; e.g., when the model predicted $Model(x)_i >= 0.25$ for an illness i, we considered it to be "true." The reason is that this threshold works best and improves the F1 score.

6.1 Data

While we in total had 53'423 distinct samples, we only used a subset of those. We used one subset in this Predictor evaluation and one for the Embedder selection. We used the bigger dataset for the Embedder evaluation since we initially wanted to explore the task with all the data. For this comparison section, however, we use the smaller dataset to compare ourselves with other implementations of the 50 most common tasks. We will denote these datasets big and small in the following evaluation. In general, we only used the samples that contained at least one of the 50 most common ICD codes. For the predictions shown in comparison to other works, we exclusively used the smaller dataset, including for training the Embedder. For all tables/plots in this chapter, we will be using the test set. The two datasets are described in the table 6.1.

	Train	Validation	Test	Total
Small	8'067	1'574	1'730	11'371
Big	44'175	1'468	3'448	49'091

Table 6.1: Comparison of the datasets used

6.2 Sentence Style

In this section, we will show the evaluation of the two variations of using the sentence embedding. We described the sentence embedding in 4.12. The two ways of generating the sentence embedding are either taking all sentences for each sample or only the sentences, including a word included in the top-k-word lists. We will call the way that includes all sentences "all" and the other "reduced." We can see the results in the table 6.2. As we can see, the embedding version reduced

Version	F1 -Micro	F1-Macro	AUC-Micro	AUC-Macro
All	0.3197	0.1013	0.7358	0.6403
Reduced	0.3456	0.2062	0.7683	0.6821

Table 6.2: Comparison of sentence embedding on the small Mimic3 test set as described in 6.1

performs better. While version all has more information to make its predictions, the reduced version has fewer input variables. Another big difference here is that we used 50 Sentences as an upper limit for the reduced task and 200 sentences for the all task. The predictor architecture failed to make sense of the bigger context and preferred having a smaller, more informative selection. Overall, these two versions' predictions seem to be significantly worse than the predictions made by our other models. We did not conduct further experiments in this direction.

6.3 Dense Architecture

We will evaluate the difference between a simple flat architecture and a more complicated architecture, our best-found architecture. As described in the section 5.2, a dense Predictor model is already competitive. We can see the comparison of the results in the table 6.3. In comparison to our best model, as described in

Version	F1 -Micro	F1-Macro	AUC-Micro	AUC-Macro
Flatt	0.5860	0.4921	0.8896	0.8598
Best	0.6019	0.4879	0.9018	0.8713

Table 6.3: Comparison of architectures on the small Mimic3 test set as described in 6.1

section 5.6. As we can see, the best model outperforms the flat, dense architecture on the AUC, both macro and micro. However, we can also see that simple architecture remains competitive on this level.

6.4 Dense Architecture, Force Through Smaller

We evaluated the performance losses of learning how to reduce the number of variables. In particular, when forcing the embeddings who have a size of 768 through a smaller layer of size o, we would assume that the performance increases since the model should have learned how to condense the information. However, in practice, we could not find an architecture that would outperform our best model by reducing the amount of information in the first layer, as already discussed in section 5.3. We can see the results in the table 6.4. As we can

Version	F1 -Micro	F1-Macro	AUC-Micro	AUC-Macro
Best	0.6019	0.4879	0.9018	0.8713
Layer-5	0.5083	0.4297	0.8474	0.8205
Layer-25	0.5501	0.4636	0.8643	0.8385
Layer-50	0.5508	0.4535	0.8680	0.8420

Table 6.4: Comparison of the force through smaller architecture on the small Mimic3 test set as described in 6.1

see, the performance is better if we increase the number of nodes in each layer. Our best model outperforms the other by a significant margin. As we can see, 25 and 50 are very similar and have almost identical values, indicating that the performance decreases flattens. If we increase the parameter o further, we get closer to the best and simple dense architecture while being more expensive to train. Therefore this architecture is not desirable.

6.5 Aggregating Techniques

In this section, we evaluated the different aggregation techniques described in section 5.4 we generally found aggregation methods to be beneficial. We tested the different aggregation methods on the model described in the figure 5.3. We can see the results in the table 6.5 As we can see, the sum aggregation is too

Version	F1-Micro	F1-Macro	AUC-Micro	AUC-Macro
Average after NN	0.6019	0.4879	0.9018	0.8713
Average	0.5945	0.4647	0.9006	0.8683
Median after NN	0.5893	0.4882	0.8939	0.8621
Sum	0.5325	0.3841	0.8834	0.8467

Table 6.5: Comparison of the aggregation methods on the small Mimic3 test set as described in 6.1

simplistic and not beneficial. The median is relatively strong in this evaluation and would potentially, in some cases, be better than the average, but we found, in

general, the average to work better. The simple averaging technique is relatively strong but not as good as averaging the weighted embeddings through a neural network.

6.6 Individual Embedding Performance

This section will look at the relative importance/performance of the input-features. To keep this section brief, we will focus on the best architecture as shown in the figure 5.3.
The performance of the individual embeddings can be compared in the table 6.6. As we can see, the informative front, back and mixed (Long in table) embeddings as described in the section 4.6 have the best comparative performance. Since these embeddings have the most information achieved through their superior sequence length, this makes sense. The importance of the paragraph embedding is highly predictive as well. While the illness list seems to preform inferior, it adds value to the entire prediction, presumably, since it's information content is very different from the other features.

Version	F1 -Micro	F1-Macro	AUC-Micro	AUC-Macro
Only Long	0.5738	0.4927	0.8627	0.8389
Only Illness list	0.3258	0.4630	0.8332	0.7901
Only Paragraphs	0.5419	0.4661	0.8465	0.8209
All	0.6019	0.4879	0.9018	0.8713

Table 6.6: Comparison of the different embeddings on the small Mimic3 test set as described in 6.1

6.7 Comparison to Other Works

In comparison to the newest other work we underperform. Compared to other BERT-based works, we are competitive. BERT-based architectures' main problems in this task are that BERT-based architectures perform better with more data. The Mimic3 dataset is already relatively small with its 53'423 samples in general, for BERT based architectures.
Predicting the 50 most common ICD codes is generally evaluated on the smaller dataset with 11'371 samples, which further reduces the amount of data. The reason for this is to stay comparable to earlier works of Shi et al., which used only these 11'371 samples to make their predictions [11]. We generally are confident that our approach would improve in comparison when given more data, as our experiments in chapter 4 indicated, and since Zhang's work stated that in their significantly bigger dataset, BERT outperforms other approaches [27]. We can

see the comparison in the table 6.7. As we can see, our approach outperforms the earlier works by Prakash et al. and Shi et al. and are comparable to Mullenbach et al. with their CAML architecture. When comparing to the work of Vu et al., which is the most recent of the works and considered state-of-the-art, we underperform. When comparing our work to the only directly comparable

Version	AUC-Micro	AUC-Macro
Logistic Regression [9]	0.864	0.829
C-MemNN [42]		0.8330
Shi et al. [11]	0.900	
CAML [9]	0.9090	0.8750
DR-CAML [9]	0.9016	0.8804
Vu et al. [17]	0.9460	0.9205
Our	0.9018	0.8713

Table 6.7: Comparison of works on the small Mimic3 test set as described in 6.1

BERT-based architectures of Chen et al. [26], we can see that we outperform it. We do outperform both their $BERT_{large}$ and their $BERT_{base}$ version. Generally, we should compare our results to the $BERT_{base}$ version since this is what we trained with. It includes 110 million parameters as opposed to the 340 million parameters found in $BERT_{large}$. We can see the results of this comparison in the table 6.8. We can not compare the BERT-based architecture proposed by Zhang et al. since they do not provide their code or provide the respective values for the small Mimic3 test set.

Version	AUC-Micro	AUC-Macro
$BERT_{base}$ [26]	0.8640	0.8290
$BERT_{large}$ [26]	0.8890	0.8580
Our	0.9018	0.8713

Table 6.8: Comparison of BERT-based works on the small Mimic3 test set as described in 6.1

6.8 Interpretation

In recent years interpretation of neural networks has become increasingly popular. One of the main issues with predictions made by neural networks is that they do not explain themself. This may become increasingly problematic when working in areas where the model is taking an advisor's role. In other words, a role where a human still takes the final decision, and the model is assisting him. In such a case, it is not only important what the decision is but also which information the model considered to make this prediction.

To provide such an explanation, the field of neural network interpretability has been increasingly utilized. This work will mostly use the newly developed Captum environment [43], which offers several well-known interpretability methods. Captum is an interpretable environment which works in combination with PyTorch. Inside Captum, especially the two methods Integrated Gradient [44] and Deep Lift [45], seemed to provide value for our interpretability method, and explaining which paragraphs are important for the respective illness.

6.8.1 Deep Lift

Deep Lift is an attribution approach that back-propagates changes to the input, based on the differences between a reference(baseline) and the input [45]. Deep Lifts' goal is to explain the output difference by changing the input variable towards the reference variable. These changes are then mapped to the output changes to compare how important they are for the respective output prediction. Deep Lift uses so-called multipliers to blame specific neurons for differences in the output.

6.8.2 Integrated Gradient

Integrated gradients work using the integral of gradients concerning inputs along the path from a given reference to an input [44]. The integral can be approximated using the Gauss Legendre quadrature rule or a Riemann Sum.

6.8.3 Interpretation Designer

An Interpretation Designer(ID) is a function that takes as an input a Predictor $P(x)$ as described in the chapter 5 and outputs for each feature group their respective importance. As we described in the chapter 5, a Predictor takes as an input $x \in \mathbb{R}^{e \times 768}$ [1] and outputs $y \in \mathbb{R}^{50}$ where each entry y_i is interpreted as the probability of illness i being present in the sample.
An interpretation method $M(x, P(x), k, b)$ like Deep Lift and Integrated Gradient, generates given a sample from the test set x, a Predictor $P(x)$, a reference variable b and a target illness y_k and outputs $z \in \mathbb{R}^{e \times 768}$ with the same dimensionality as x. So $z \in \mathbb{R}^{e \times 768}$, the entry at position z_{ij} can now be interpreted as the impact that the variable x_{ij} had on the prediction of y_k compared to the baseline b. We used as a baseline b always the zero vector of size $b \in \mathbb{R}^{e \times 768}$. The value z_{ij} represents how much a unit change in x_{ij} affects the prediction of a specific y_k. These values may be positive or negative, depending on the relative position to the baseline, always 0 in our case. Since we are interested in the influence of variables on predictions, we will refer to this quantity's absolute value.

[1] For simplicity we will ignore the batch dimension in the following examples

To define the importance of embedding e, we have to aggregate 768 variables into a single value.

6.8.4 Aggregation of e

To aggregate e, we experimented with 3 approaches. The dot product between input and output, the mean of the absolute values, and the L_2 norm. We eventually used the dot product as an aggregation method since the results were presented the clearest in this way.

- $agg(x, z) := x \cdot z$
- $agg(x, z) := \|z\|$
- $agg(x, z) := \frac{1}{768} \sum_{i=0}^{768} |z_i|$

Using any of these methods on the vector x and z for each illness gave us an importance matrix $M \in \mathbb{R}^{e \times 50}$. One entry M_{ij} described how important the embedding at position i was for the prediction of illness j. This defined now per sample a matrix M.

6.8.5 Aggregation Over All Test Examples

For each sample in the test set, we now had a matrix M. We had to aggregate them to interpret the entire effects of the inputs for the entire population. To aggregate l such matrices M into a matrix I, we tried several aggregation methods. Two of the methods worked. Either we chose to take the sum of all matrices or take all M_{ij} where the input embedding i was not the 0 vector. We found that the second approach worked better since weighting the feature importance relative to their occurrence reduced the "frequency noise" generated by the interpretation methods.
While we tried other approaches, such as comparing the results of all examples or considering only the examples where illness j was predicted correctly by embedding i. We found that evaluating it in this way shows the observer better how it works.

6.9 Interpretation Results

We will present an example results of our interpretation pipeline. [2] While more resulting visualization was interesting, we found the following subset to be most

[2] Highresolution image location: https://gitlab.ethz.ch/disco-students/fs20/sluck_medicalnlp_clean/-/tree/master/model/predict_from_embedded/plot_choosen

informative. We will mostly focus on the dot product results since we found the results to be the clearest. We will also only present the weighted results and mostly focus on the paragraph related variables since they are easier to understand. In the figure 6.1, we can see the total importance aggregated over all illnesses and with all variables. We can see that in general, overall illness, the

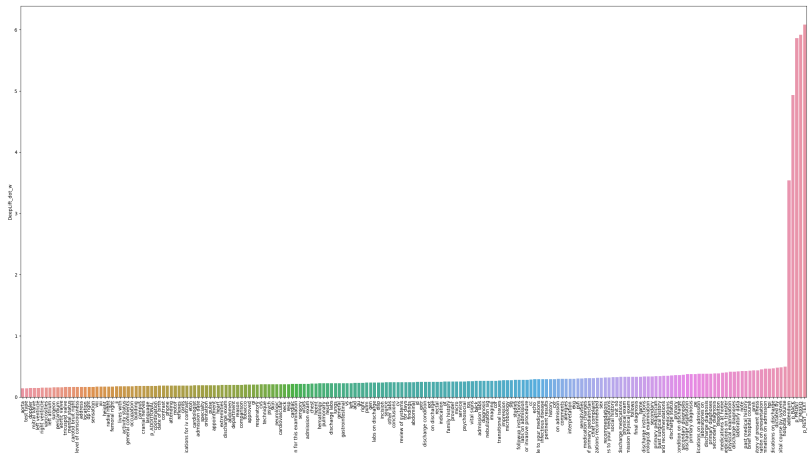

Figure 6.1: Barplot illnesses weighted on the small mimic3 test set as described in 6.1

most important embeddings in order of importance were

$$Front, Mixed, RestText, Back, Addendum, HospitalCourse, ..$$

This makes sense since the bigger embeddings were trained with more words and a bigger sequence length, as described in 4 and also supported by our results in section 6.6. The most important paragraphs seem to be the Hospital Course and the Addendum paragraph in this evaluation. These paragraphs are so important when aggregated over all illnesses since these 5 embeddings are present in most of the samples and generally filled with more context, as indicated in the section 4.8.

6.9.1 Paragraph Interpretation

When looking in more detail into the paragraphs, we can see how the model makes its predictions. As mentioned previously, weighing the paragraphs by the amount they occur shows a clearer picture. We do so to account for the case that we may have a very rare paragraph that is highly predictive of a certain illness in the above aggregations, but not common. In this case, the value for this paragraph treated as a sum or average will be shown as diminished, but after

weighting will show its true "if exists" importance. We see in the figure 6.2, the weighted results again aggregated over all illnesses. As we can clearly see here, the frequency should be considered before reading the interpretation. Intuitively

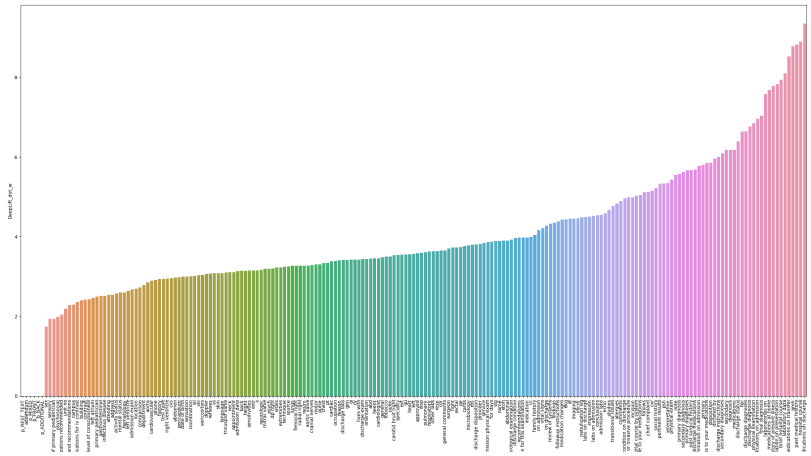

Figure 6.2: Overview most important paragraphs, weighted on the small Mimic3 test set as described in 6.1

we can understand the predictions by looking at, for example, the paragraph "respiratory" or "resp" (a common abbreviation) is present in the most important 10 paragraphs for the illnesses "Acute respiratory failure," "History of tobacco use," "Pulmonary collapse," and "Tobacco use disorder."
We can see a complete overview of the relative importance of paragraph i for illness j in the heatmap in the figure 6.3. As we can see, the cells which appear greener in the figure are the relatively more important input-variables for the model's predictions for illness j. This overview, often containing sparks, is good and shows that certain paragraphs well determine some diseases. Some diseases are predominantly red, which indicates that no paragraphs are highly predictive of this specific illness. We applied z-scores to make this visualization easier to interpret.
A problem with the shown visualization is induced noise. We assume stems from illnesses that often happen together or one causing the other, or in general illnesses that can be measured or detected by measuring non-directly related aspects of the body. For example, the "femoral right" paragraph is included in acute kidney disease. The femoral right is not directly related to chronic kidney disease. Still, it appears in "Acute kidney failure NOS," and indeed, we can find in the literature a connection by Hsu et al. [46]. Furthermore, acidosis and kidneys have a connection, in the sense that kidney problems can cause acidosis. However, a detailed evaluation of all possible interpretations is outside of this work's scope and unfeasible for nonmedical professionals. To look at

a complete, understandable example, "Obstructive Sleep Apnea" has the most common 10 paragraphs, "['comments', 'tobacco', 'department', 'mental status', 'location', 'disp', 'level of consciousness', 'campus', 'activity status', 'building']". Sleep apnea is a potentially serious sleep disorder in which breathing repeatedly stops and starts. If you snore loudly and feel tired even after a full night's sleep, you might have sleep apnea [47]. We can generally discover that "Obstructive Sleep Apnea" has a direct connection to smoking (as in tobacco), physical activity (activity status), may impact the mental status/level of consciousness [47], The variables campus/department/building/location all relate to a similar variable set describing where the patient was treated. When we look at the list of the most common paragraphs for "Obstructive Sleep Apnea" in the unnormalized list, we see the paragraphs "['past surgical history', 'medications on admission,' 'nebulization sig,' 'past medical history,' 'psh,' 'comments,' 'pmh,' 'medications at home,' 'brief hospital course,' 'final diagnosis']" and indeed when searching for the exact paragraph name in the test set of the 18 occurrences(found with the exact match "Sleep Apnea") we concluded that 12 were found in a section in the above-mentioned list, particularly the "Past Medical History" has appeared often. Of the other mentions, 2 were in a section called "Sleep Apnea," which was too uncommon to be included in the 200 most common illnesses. We conclude that finding 14/18 in the top-10 paragraphs as good.

In general, we hope that this interpretation helps understand the model's predictions more clearly to a medical professional and increase their efficiency while labeling. We also hope it increases the general trust by practitioners in the models and can help avoid errors that may results when blindly trusting a model.

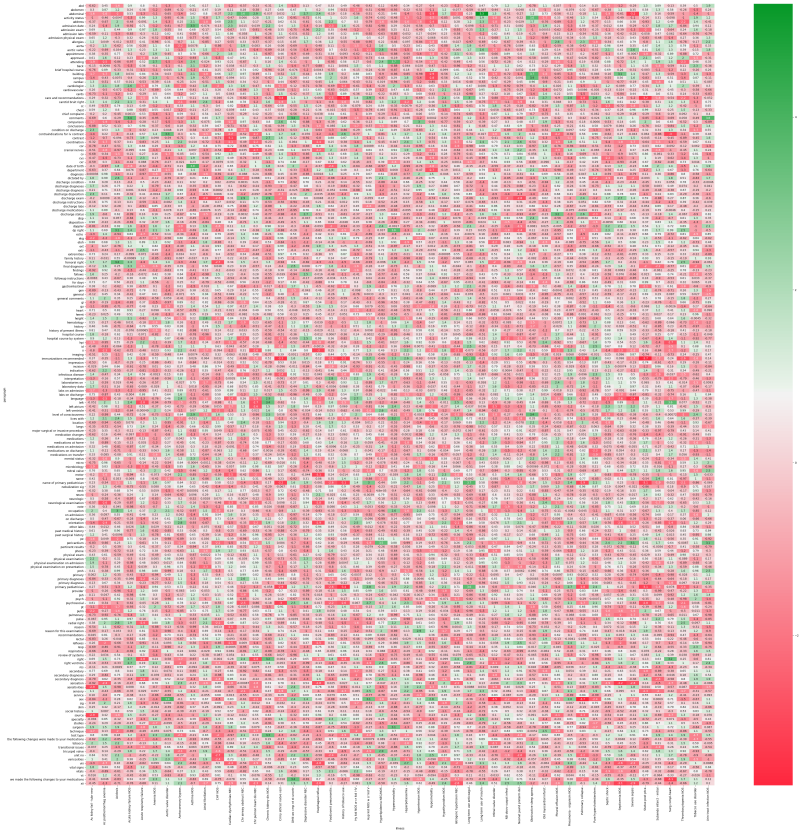

Figure 6.3: Overview most important paragraphs to Illness, weighted and zscore applied on the small Mimic3 test set as described in 6.1

Chapter 7

Conclusion

The various Predictors/Embedders and other models' implementation and design have been an educational and challenging task. The project's most demanding aspect was the usage of the relatively new NLP techniques and libraries like Captum, and it's combination with the field of medicine. The combination of various aspects of Computer Science, such as statistics, machine learning, and BERT systems, was fascinating and educational. The author thoroughly enjoyed developing and writing this master thesis and had a lot of fun doing so.

We have seen how modern NLP techniques can be used to predict illnesses from the free-floating text. While we could not outperform state-of-the-art for this specific task, we have shown that such techniques can be used successfully. The BERT architecture's main challenge in this task is that the texts' length greatly exceeds modern GPUs' memory possibilities. The amount of training data is unfavorable for such models. We are confident that new developments in long-context transformer architectures such as Longformer and BigBird or other developments may tackle this problem more successfully in the coming years.

Additionally, we have shown how to easily and intuitively explain machine-learning-text-based models to clinical practitioners, using modern interpretation methods. While at the current state of development, interpretation in neural networks is still a novel topic. We are confident that explainability will be very important to bring such a model from academia into practice in hospitals and other sensitive organizations.

In particular, splitting the information into paragraphs is possible in many form-based texts in many institutions. By splitting forms into such paragraphs before using them for machine-learning-based models, the interpretability part of the model can then show the person it is advising how it made its decision and, for example, highlight the respective paragraph such that the user can easily verify it.

7.1 Future Work

The task of ICD Coding remains challenging. While we can expect automatic or advisory systems based on machine learning to be deployed into production in hospitals worldwide, we assume that cultural and linguistical barriers, both regional and medical, will make a one size fits all approach infeasible. This encourages further work and development of datasets and models for the field accessible to researchers worldwide. We expect that some of our approaches remain useful and used in other models and domains. Interpretability methods based on text segments will be a good choice for neural network-based methods that should also provide their users with information on how the predictions were made. Such use cases will be found in several areas where form-based reporting is common. We could expect the usage in other governmental and management support systems, which commonly use such a form-based reporting system. Reports often contain big chunks of text preceded by a title and expose a natural way of splitting long information. Human interpreters can then look at which text passages the networks used to predict and understand how the network came to its conclusion. If an entity thinks the prediction is unjust or wrong, a knowledgeable domain expert can use this interpretation to verify and explain the prediction to the entity quickly.

Bibliography

[1] K. N. Vokinger, U. J. Mühlematter, A. S. Becker, A. Boss, M. A. Reutter, and T. D. Szucs, "Artificial intelligence und machine learning in der medizin: eine medizinische und rechtliche würdigung am beispiel der radiologie," *Jusletter*, no. 28.08. 2017, p. online, 2017.

[2] P. Domingos, "A few useful things to know about machine learning," *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012.

[3] A. E. Johnson, T. J. Pollard, L. Shen, H. L. Li-Wei, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. A. Celi, and R. G. Mark, "Mimic-iii, a freely accessible critical care database," *Scientific data*, vol. 3, no. 1, pp. 1–9, 2016.

[4] Y. Peng, S. Yan, and Z. Lu, "Transfer learning in biomedical natural language processing: An evaluation of bert and elmo on ten benchmarking datasets," *arXiv preprint arXiv:1906.05474*, 2019.

[5] W. Freeman, A. Weiss, and K. Heslin, "Overview of us hospital stays in 2016: variation by geographic region: statistical brief# 246," 2019.

[6] D. A. Grimes, "Epidemiologic research using administrative databases: garbage in, garbage out," *Obstetrics & Gynecology*, vol. 116, no. 5, pp. 1018–1019, 2010.

[7] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[8] "Icd-9." [Online]. Available: https://www.cdc.gov/nchs/icd/icd9.htm

[9] J. Mullenbach, S. Wiegreffe, J. Duke, J. Sun, and J. Eisenstein, "Explainable prediction of medical codes from clinical text," 2018.

[10] L. R. de Lima, A. H. Laender, and B. A. Ribeiro-Neto, "A hierarchical approach to the automatic categorization of medical documents," in *Proceedings of the seventh international conference on Information and knowledge management*, 1998, pp. 132–139.

[11] H. Shi, P. Xie, Z. Hu, M. Zhang, and E. P. Xing, "Towards automated icd coding using deep learning," 2017.

[12] S. Karimi, X. Dai, H. Hassanzadeh, and A. Nguyen, "Automatic diagnosis coding of radiology reports: a comparison of deep learning and conventional classification methods," in *BioNLP 2017*, 2017, pp. 328–332.

[13] T. Baumel, J. Nassour-Kassis, R. Cohen, M. Elhadad, and N. Elhadad, "Multi-label classification of patient notes a case study on icd code assignment," *arXiv preprint arXiv:1709.09587*, 2017.

[14] C. Song, S. Zhang, N. Sadoughi, P. Xie, and E. Xing, "Generalized zero-shot icd coding," *arXiv preprint arXiv:1909.13154*, 2019.

[15] G. Wang, C. Li, W. Wang, Y. Zhang, D. Shen, X. Zhang, R. Henao, and L. Carin, "Joint embedding of words and labels for text classification," *arXiv preprint arXiv:1805.04174*, 2018.

[16] F. Li and H. Yu, "Icd coding from clinical text using multi-filter residual convolutional neural network." in *AAAI*, 2020, pp. 8180–8187.

[17] T. Vu, D. Q. Nguyen, and A. Nguyen, "A label attention model for icd coding from clinical text," *arXiv preprint arXiv:2007.06351*, 2020.

[18] K. J. O'malley, K. F. Cook, M. D. Price, K. R. Wildes, J. F. Hurdle, and C. M. Ashton, "Measuring diagnoses: Icd code accuracy," *Health services research*, vol. 40, no. 5p2, pp. 1620–1639, 2005.

[19] "discharge summary." [Online]. Available: https://medical-dictionary.thefreedictionary.com/discharge+summary

[20] X. Xie, Y. Xiong, P. S. Yu, and Y. Zhu, "Ehr coding with multi-scale feature attention and structured knowledge graph propagation," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019, pp. 649–658.

[21] A. Perotte, R. Pivovarov, K. Natarajan, N. Weiskopf, F. Wood, and N. Elhadad, "Diagnosis code assignment: models and evaluation metrics," *Journal of the American Medical Informatics Association*, vol. 21, no. 2, pp. 231–237, 2014.

[22] B. Koopman, G. Zuccon, A. Nguyen, A. Bergheim, and N. Grayson, "Automatic icd-10 classification of cancers from free-text death certificates," *International journal of medical informatics*, vol. 84, no. 11, pp. 956–965, 2015.

[23] Y. Gu, R. Tinn, H. Cheng, M. Lucas, N. Usuyama, X. Liu, T. Naumann, J. Gao, and H. Poon, "Domain-specific language model pretraining for biomedical natural language processing," 2020.

[24] K. Trapeznikov, ""iobert_v1.1_pubmed_squad_v2"." [Online]. Available: "https://huggingface.co/ktrapeznikov/biobert_v1.1_pubmed_squad_v2"

[25] E. Alsentzer, J. R. Murphy, W. Boag, W.-H. Weng, D. Jin, T. Naumann, and M. McDermott, "Publicly available clinical bert embeddings," *arXiv preprint arXiv:1904.03323*, 2019.

[26] Y. Chen, "Predicting icd-9 codes from medical notes–does the magic of bert applies here?"

[27] Z. Zhang, J. Liu, and N. Razavian, "Bert-xml: Large scale automated icd coding using bert pretraining," *arXiv preprint arXiv:2006.03685*, 2020.

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[29] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "Bert pre-training of deep bidirectional transformers for language understanding," 2018.

[30] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *ArXiv*, pp. arXiv–1910, 2019.

[31] M. Zaheer, G. Guruganesh, A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, and A. Ahmed, "Big bird: Transformers for longer sequences," 2020.

[32] J. Shang, T. Ma, C. Xiao, and J. Sun, "Pre-training of graph augmented transformers for medication recommendation," *arXiv preprint arXiv:1906.00346*, 2019.

[33] J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C. H. So, and J. Kang, "Biobert: a pre-trained biomedical language representation model for biomedical text mining," *Bioinformatics*, vol. 36, no. 4, pp. 1234–1240, 2020.

[34] M. Sänger, L. Weber, M. Kittner, and U. Leser, "Classifying german animal experiment summaries with multi-lingual bert at clef ehealth 2019 task 1." in *CLEF (Working Notes)*, 2019.

[35] N. Poerner, B. Roth, and H. Schütze, "Evaluating neural network explanation methods using hybrid documents and morphological agreement," *arXiv preprint arXiv:1801.06422*, 2018.

[36] R. Hasani, "Interpretable recurrent neural networks in continuous-time control environments," Ph.D. dissertation, Wien, 2020.

[37] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (gdpr)," *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 2017.

[38] K. Trivedi, "Fast bert." [Online]. Available: https://github.com/kaushaltrivedi/fast-bert

[39] C. Sun, X. Qiu, Y. Xu, and X. Huang, "How to fine-tune bert for text classification?" in *China National Conference on Chinese Computational Linguistics*. Springer, 2019, pp. 194–206.

[40] W. Falcon, "Pytorch lightning," *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*. [Online]. Available: "https://pytorch.org/docs/master/generated/torch.nn.BCEWithLogitsLoss.html#torch.nn.BCEWithLogitsLoss"

[41] L. N. Smith, "Cyclical learning rates for training neural networks," 2017.

[42] A. Prakash, S. Zhao, S. A. Hasan, V. Datla, K. Lee, A. Qadir, J. Liu, and O. Farri, "Condensed memory networks for clinical diagnostic inferencing," 2017.

[43] PyTorch, "Captum," *GitHub. Note:https://github.com/pytorch/captum*. [Online]. Available: "https://github.com/pytorch/captum"

[44] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," 2017.

[45] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," *arXiv preprint arXiv:1704.02685*, 2017.

[46] S. Hsu, D. E. Rifkin, M. H. Criqui, N. C. Suder, P. Garimella, C. Ginsberg, A. M. Marasco, B. J. McQuaide, E. J. Barinas-Mitchell, M. A. Allison *et al.*, "Relationship of femoral artery ultrasound measures of atherosclerosis with chronic kidney disease," *Journal of vascular surgery*, vol. 67, no. 6, pp. 1855–1863, 2018.

[47] [Online]. Available: https://www.mayoclinic.org/diseases-conditions/sleep-apnea/symptoms-causes/syc-20377631

Appendix A

Summary of The Findings

We will describe here a summary of our findings for a quick reference. To develop an ICD coding system using BERT, it is beneficial to use preprocessing. The variation of Mullebach et al. is the most useful known to us, as shown in the section 4.3.

A.1 Embedder

When using a simple BERT classifier with a classification layer on top, it's most beneficial to use the mixed training direction as described in the chapter 4. Using a larger sequence length is beneficial. Paragraph splitting, as described in section 4.7 works and is beneficial for interpretability related methods. Combining various forms of embeddings generated via the process in chapter 4 works and is beneficial for the overall prediction score. Our general experience has been that these embeddings should ideally be aligned. The embeddings variation chunked and random did, for example, not work well. We assume they did not work well because the alignment was suboptimal. The hyperparameter we found to work best for the long embedders are,

Hyperparameter	Value
Optimizer	Lambda
Learning Rate	0.0005
Scheduler	Warmup Cosine
Batch Size	2
Precision	Full Precision
Sequece Length	1024

Table A.1: Hyperparameters used for all 1024 sequence length models.

The Embedder that worked best was "BiomedNLP-PubMedBERT-base-uncased-abstract-fulltext" and was downloaded through Huggingface. Training this setup for around 12 epochs was good, more gave diminishing results.

A.2 Predictor

We found the combination of the embeddings,

- Front, Back and Mixed as described in section 4.6
- Paragraph Embedding as described in section 4.7
- Rest Text as described in 4.9
- Illnesslist as described in 4.13
- Additional Information as described in 4.14

to work best. It was beneficial, where the number of paragraphs could potentially

Version	F1 -Micro	F1-Macro	AUC-Micro	AUC-Macro
Only Embeddings	0.5738	0.4927	0.8627	0.8389
Only Illness list	0.3258	0.4630	0.8332	0.7901
Only Paragraphs	0.5419	0.4661	0.8465	0.8209
All	0.6019	0.4879	0.9018	0.8713

Table A.2: Comparison of the different embeddings on the small mimic3 test set as described in 6.1

be reduced. Also, the illness lists could potentially be removed. We would advise keeping the other embeddings.
We generally found that using the Lambda optimizer was good. Training using early stopping was beneficial; patience 3 is suitable. Dense architectures are competitive and can be used for prototyping. The best architecture can be seen in the figure 5.3. Using a learning rate around $5e-4$ was good, may depend heavily on the model. Linear learning rate decay over the training was beneficial.

A.3 Interpretation

Using the Deep Lift variation showed the most useful results. We considered using the dot product better for overall usage. Weighting the aggregated results by the amount of non-zero input was essential. For cross illness/input variable comparison applying z-score is beneficial.