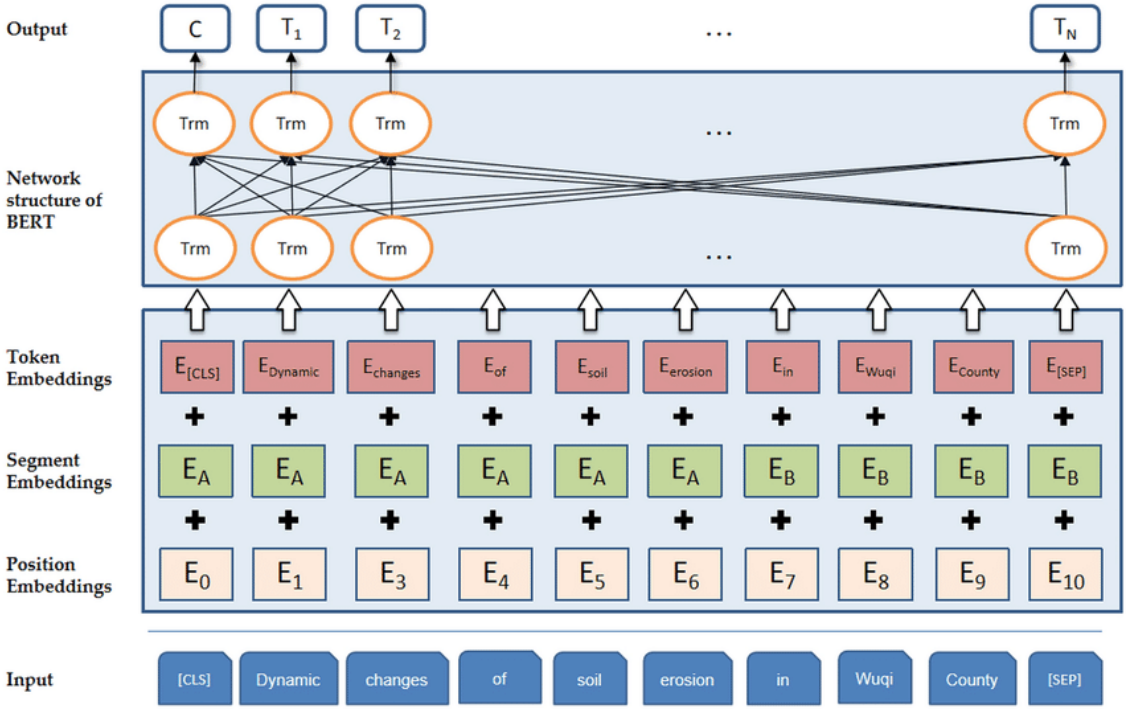
CHALMERS
UNIVERSITY OF TECHNOLOGY



Methods for Reducing Costs of Running Large-Scale Machine Learning Models

A study in methods of different ways of reducing cost of running a BERT model for predicting bias in text

Master's thesis in Complex Adaptive Systems

ANTON SANDBERG

DEPARTMENT OF PHYSICS

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

Methods for Reducing Costs of Running Large-Scale Machine Learning Models

A study in methods of different ways of reducing cost of running a BERT model for predicting bias in text

ANTON SANDBERG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2023

A study in methods of different ways of reducing cost of running a BERT model for predicting bias in text
ANTON SANDBERG

Supervisors: Ewa Tusien, MACHINE LEARNING DEVELOPER, Substorm
Sergio Liberman Bronfman, MACHINE LEARNING DEVELOPER, Substorm
Examiner: Giovanni Volpe, Professor and Group Leader, Soft Matter Lab, Physics Department, Gothenburg University

Master's Thesis 2023
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The overall structure of a BERT model. Image source [1]

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Methods for Reducing Costs of Running Large-Scale Machine Learning Models
A study in methods of different ways of reducing cost of running a BERT model for predicting bias in text
Anton Sandberg
Department of Physics
Chalmers University of Technology

Abstract

Large Language Models have taken over our world with our biggest contributor to that being chat GPT but many more companies such as Facebook are launching their own versions to be able to keep up in the race. The model footprints are increasingly large and therefore also the cost associated with running them. The company, Substorm, has a transformer model of a type called BERT which is today used to be able to classify male and female bias in text. They are interested in looking into different ways of reducing cost of said model as well as models further into the future.
In this Master Thesis you will be introduced to methods for both faster loading of Transformer models but also methods for reducing it's model byte-size footprint. The methods are tested on both a smaller Fully Connected network trained and tested on the MNIST data set as well as Google's highly competitive BERT model, used first and foremost for the classification of text in different ways. The model is trained on the PANDORA data set which is consisting of a large sum of comments compiled from reddit as well as a large sum of them being gender labeled. For the loading part of the project a speedup of 99% is shown when cold loading the model. For the model minimisation part, three different variants of the model are presented, one quantized model, one pruned model as well as one both quantized and pruned model. The modified models are then tested towards it's original counterparts on the PANDORA test set to be able to determine their viability. For the quantized model, no accuracy loss was detected while being able to reduce the model footprint by 60%. For the 75% pruned model an accuracy loss of only 2% is shown while being able to theoretically decrease the model weight footprint by 50%. For the both quantized and 75% pruned model an accuracy loss of only 2.2% while being not being able to theoretically decrease the model footprint but increase the model weights by 25% in comparison to the quantized model.
These size decreases means that Substorm have the possibility of decreasing their server costs in at least half since the servers usually come in multiples of two from each other. This also means faster computational time associated with running the model in comparison to its original state while maintaining competitive accuracy results when the model is being used.

Keywords: machine learning, neural networks, natural language processing, model minimisation

Acknowledgements

Thank you Ewa Tusień for being my supervisor in the beginning of this Master's Thesis and a very helping hand in the starting stages of the project. Thank you Sergio Liberman Bronfman for being overall as solid as a rock and always being firm with me to make sure that the project holds up to a certain standard. Thank you also for being kind in periods were maybe not as much work had been done as were expected from me, you're the main reason this project went over the finish line in the end and that I'm eternally grateful for. Thank you Giovanni Volpe for taking on the role as examiner for the project giving me the opportunity to do this. A big thank you also goes out to Marta-Lena Antti, Professor at LTU, who has helped me out in proof reading the report as well as giving me tips and tricks when I needed them.

Anton Sandberg, Gothenburg, November 2023

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis:

AI	Artificial Intelligence
AUC	Area Under ROC Curve
BERT	Bidirectional Encoder Representations from Transformers
CNN	Convolutional Neural Network
EC2	Elastic Compute Cloud 2
FN	False Negative
FP	False Positive
FPR	False Positive Rate
LLM	Large Language Model
LSTM	Long Short-Term Memory
ML	Machine Learning
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
TN	True Negative
TP	True Positive
TPR	True Positive Rate

Contents

List of Figures

List of Tables

1
Introduction

Large Machine Learning models such as chat GPT-3.5, BERT and DALL-E, and many more, have shown impressive performance on a wide range of natural language processing tasks as well as large and complex classification tasks. Deep learning methods have consistently outperformed traditional methods for object detection. Furthermore, the world is at a never-before-seen turnstile since the latest version of chat GPT can provide complex answers to complex problems as well as provide well-written code for software engineers on demand as long as the right prompt is provided to the model [2]. However, training and deploying these models can be costly regarding computational resources, memory and energy consumption [3], [4], [5], this makes the large models incapable of being stored in further mobile environments such as cars or mobile phones. Further larger models also beg the question as to being over-parameterised, and even though they increasingly get better and better at performing different tasks raises the question as to what can be done to decrease model sizes and still be able to perform the tasks they are assigned to at an acceptable or even similar rate.

The long startup times of these models make them difficult to use in real-time or with on-demand applications [3]. Additionally, it is hard to scale up services in real time using auto-scalers even though there is significant progress being made on the subject [6]. These challenges have primarily been faced by large corporations and research institutions that have the resources to train and deploy these models. However, with the increasing availability and accessibility of Machine Learning tools and techniques, as well as the heightened positive results from implementing Machine Learning and AI in companies of all sizes, it's more and more common that smaller organisations are also interested in utilising the capabilities of these models. This also means that more and more different markets are increasingly interested in the development of artificial intelligence and machine learning [7].

For the project in question, the company, Substorm has a BERT model trained on gender specific data where the model is supposed to predict gender bias in some text being inserted by the user. The problem mainly lies in that deployment of very large deep learning models is associated with high costs for computing power. This might not be the problem that large and leading corporations face since this cost might be negligible for them but this will be an area in which can be taken advantage of to still be able to look at solutions to be able to decrease server and running cost for the model in question. In the case of large models that are used often, one way to solve this would be to use on-spot machines and load the models

as quickly as possible to reduce response time. The model in its current state has a high relative cost and it's in Substorm.ai's interest to decrease this cost. The company is hosting four different containers containing two BERT models and two XGBoosts, two for each since there is one for Swedish and one for English input. The four containers reside in two EC2 instances of type m5.large on Amazon Web Services and each of them cost roughly at least 0.096 USD per hour or 1.0176 SEK per hour. This results in a yearly rate of roughly 1682 USD or 17829 SEK per year. Roughly because there are costs for the volumes or disks that get created for each instance, network traffic, additional resources, auto-scaling and most importantly, taxes. Substorm.ai also has interest in using this on other large Natural Language Processing models.

There's quite a substantial amount of work being done on minimising BERT models, both DistillBERT [8] and TinyBERT [9] uses different types of distillation methods in order to decrease model size and and these are also the techniques being grounded in the method. Q8BERT [10], BinaryBERT [11] and I-BERT [12] uses different forms of quantization in order to minimise size and keeping performance metrics reasonable. oBERT [13] uses pruning in order to minimise model size as well as limit accuracy drop to a minimum. These are models directly modifying the original BERT model with great results. These methods are greatly efficient and are seen as starting points in the project as well as discussions with the project's supervisor in order to find a good starting point. Some of these models requires a lot of time and effort, for example with re-training the BERT model. The methods seen as viable for this task, where it's mandatory that the methods are able to be used post training as well as to be able to be used on models in general, due to the fact that Substorm require the methods to be viable on models further into the future. Therefore both quantization and pruning are to be seen as great starting points.

The questions that then were intended to be answered during the project were as follows:

1. Is it possible to decrease the loading time for the model?
2. Is it possible to decrease the model's byte-size footprint?
3. Is it possible to decrease the model's byte-size footprint while still keeping competitive performance metric of the model?

2

Theory

Here quick summaries of different subjects taken under consideration during the project thought to be better inserted into a separate chapter than to be further explained in the text.

2.1 Machine Learning Background

People have been trying to achieve mathematical ways to be able to create machines that in some way recreate the functions of a human brain, to learn with provided information and data [14]. Neural network algorithms for machine learning, started off as ways to recreate the architecture and dynamics of neurons in the brain [15]. The typical challenge involves training the mentioned network with a data set, enabling it to predict new data, typically derived from a test data set, before deployment for real-world use. In the case of supervised learning where a supervised model is a function parameterised by weights that maps an input space given by a defined vector space specific to a task and outputs a value that can be a) a value in case of regression and b) a class for a classification problem.

There is also both semi-supervised learning and unsupervised learning. Semi supervised learning can be of good use when the data provided isn't fully labelled. Unsupervised learning contains quite different challenges where the data doesn't contain labels at all, this has seen a lot of success for example trying to create bots in games where the human touch can create scores for different positions, actions and how the game currently looks in comparison to what the user feels is the optimal way of doing things. This is usually very efficient since the model doesn't take into consideration how things have previously looked and can then create completely new strategies in comparison to what we as humans have previously seen. A few examples of this are in the game Alpha GO where Reinforcement Learning (RL) models have been created to beat the then current world champion in the game and succeeded [16]. In chess where two of the biggest and most successful models are named AlphaZero [17]. These two models are currently vastly superior in comparison to human players. As can be seen, Machine Learning models succeed the most in areas where they can process information at a much quicker pace than humans as long as the data is there to train it up, making them great for tackling simple or more complex problems that can be broken down into more mathematical problems.

2.1.1 Different Neural Network types

There are a lot of different types of neural networks to cover when talking about the subject of Artificial Intelligence (AI) and Machine Learning (ML). This will be a brief summary of the most popular types of models and also be given an introduction to the model that this text will mostly be about, the BERT Model provided by Google, which is a Transformer-based model.

2.1.1.1 Fully Connected Neural Network

Fully Connected Neural Networks (FCNN), are dense networks consisting of different layers of matrices, as well as bias that takes some flattened input (in the shape of a vector). It then feeds it through the network multiplying the input vector with the weight matrix, multiplying that output with the next matrix, giving us an output in the shape of the output layer. The simplest form of a network is only consisting of one layer, but there's also higher order networks which has multiple layers, which means that there might be a input layer, some amounts of layers in the middle called hidden layer followed by an output layer. An activation function is usually also present in between each layer in the network, as well as a threshold If the output were to noted i and the input neuron j, where the output meaning just the layer following the input layer, the output becomes:

$$O_i = g(b_i), b_i = \sum_j w_{ij} x_j - \theta_i \tag{2.1}$$

where w_{ij} is the weight, x_j the input value, θ_i the bias and g the activation function chosen in between the layers. A number of activation functions can be chosen for this task, the most common one is relu [18], but leaky relu or others are also in common use. Relu is defined as follows:

$$g(x) = max(0, x) \tag{2.2}$$

Depending on the loss function of the network it then gets updated using a back propagation algorithm [19] based on a loss function, for example gradient descent [20] updating the layer weights and the layer biases to minimise the loss function. To be able to limit the speed of the learning of the weights a learning rate η is also chosen. This means that for a weight w_{ij} the appropriate adjustment δw_{ij} is:

$$\delta w_{ij} = -\eta \frac{\partial H}{\partial w_{ij}} \tag{2.3}$$

where H denotes the chosen loss function. A higher learning rate means quicker increase or decrease in the value on the weight which speeds up training but also means potential misses of either local or global minimum in the loss function. A learning rate can either be static or dynamic. A 10x10x10x2 fully connected network can be seen in figure 2.1.

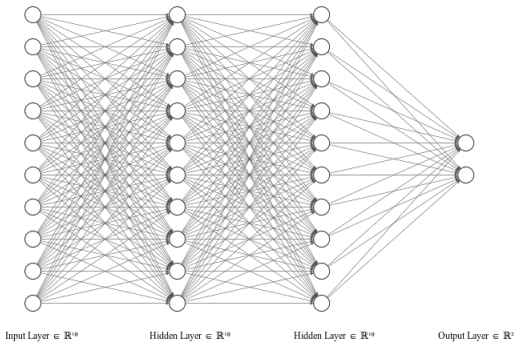


Figure 2.1: A simple 10x10x10x2 Fully Connected Network

2.1.1.2 Recurrent Neural Network

Recurrent Neural Networks (RNN) are inherently a different type of network as they don't share the same feed-forward structure as previously mentioned. A feed-forward network maps fixed-size inputs to fixed-size output, this is, as mentioned previously done by feeding the network and image in its raw data form or another type of input being represented in a set of numbers to different probabilities of the classes defined. More traditional feed forward networks however are not good at storing information throughout the data set, recurrent neural networks on the other hand are better at solving this issue due to having loops inside the network and better storing information previously given [21]. They have seen great success in speech recognition, language modelling, image captioning to only name a few [22]. See Figure 2.2 for a visualisation of how the core principles of an RNN are built. However, especially in text, a standard RNN might have some problems with predicting what is commonly known as "long-term dependencies". This can be explained as information that might have come a lot further back and might be useful in trying to predict something in its current state. This however can be improved by using what is a so-called Long Short-Term Memory network.

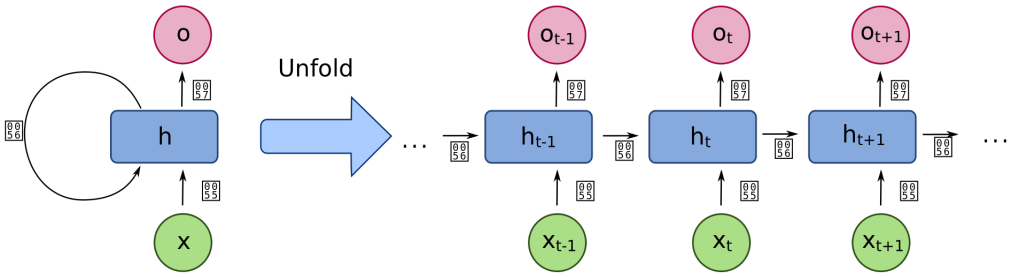


Figure 2.2: Visualization of a RNN, image source [23]

2.1.1.3 Long Short-Term Memory

Long Short-Term Memory Networks (LSTM) were first introduced in 1997 by Sepp Hochreiter and Jürgen Schmidhuber [24]. The LSTM network works in a way where it stores information from the input that might be relevant in the future. For a text, it might be for example the gender of a subject to be able to remember to be able to use the correct pronunciation, a name to be able to identify or some other arbitrary information that might be of importance further down the line in the time series data. In 2.3 a visualisation of an LSTM network can be seen.

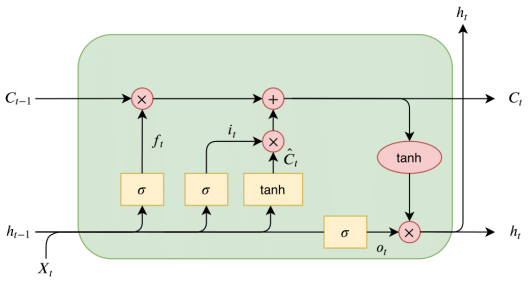


Figure 2.3: Visualisation of an LSTM, image source [25]

2.1.1.4 Transformer

We're now getting to the more interesting networks of the introduction and also one of, if not the most successful neural network architecture ever created. These are the models that have taken the Natural Language Processing area by storm and are also what the leading companies use for solving simpler as well as more complicated text-based tasks such as google's BERT Transformer [26], Open.ai's chat GPT [27], as well as their DALL.E/DALL.E 2 [28]/[29] and explained in more detail here [30]. The first two work solely with text, where BERT is used mostly as a classifier, for example to classify gender in text, and chat GPT is more of a jack of all trades that can be used more like your own search engine. There are also even newer models such as Facebook's Llama [31], which is launched more or less as a competitor to Open.ai's counterpart. Transformers are based on what is called attention and were first introduced here [32] which then led Google to further explore the subject in

this paper [33], this helps majorly with computing relationships between words in a sentence fed to the network. A fairly simple explanation of what Transformer-based models are capable can be seen in 2.4.

Figure 2.4: Examples of what a transformer can be used for where the transformer to the left shows a translator example and the transformer to the right shows a text-to-image transformer, image source [34]

Transformers are also typically built using what is called encoder and decoder, where there are usually multiple of both in what is called a stack of either encoders or decoders. The encoder contains an attention layer followed by a feed-forward layer and the decoder contains an attention layer, followed by its own mini encoder-decoder followed by a feed-forward layer. Both the encoder and decoder contain their own set of weights assigned to them.
There are of course many different variations of transformers being developed but this is usually the standard structure.

2.1.1.4.1 Attention As stated earlier attention is an integral part of why the transformer performs so well, especially on text. The reason it was developed was firstly due to the problem of "long dependencies", we've covered this problem briefly but it can be described as the difficulty of remembering an important detail stated in the input sequence. This is tackled with an attention layer which scores each word in comparison to each other word to find out which words are in relation to each other, or in other words, which word/words are important in relation to another word. A simple example are these two sentences:

- The **dog** ate the food because *it* was hungry
- The dog ate the **food** because *it* tasted good

Here you can clearly see in these two sentences that the "it" in both these sentences are referenced to two different other words e.g. the dog in the first one and the food in the second. This is what the attention layer excels at and it's used to feed the input through the layer to be able to identify words that associate with each other [35]. An example of what scores for the word "it" for the first sentence could look like can be seen in 2.1 (Higher score means more associated with each other):

Table 2.1: Example values of what scores for "it" in the first sentence can look like

the	cat	ate	the	food	because	it	was	hungry
0.2	0.8	0.02	0.05	0.15	0.001	0.1	0.03	0.04

2.1.1.4.2 BERT Bidirectional Encoder Representations from Transformers or BERT for short is a new language representation model released by Google in 2018 [26]. In comparison to a regular Transformer model BERT is only encoders used for supervised learning tasks. The model at its release in 2018 saw state-of-the-art improvements on some of the most used data sets for evaluating benchmarks on natural language processing tasks such as:

- General Language Understanding Evaluation
- Stanford Q/A dataset SQuAD v1.1 and v2.0
- Situation With Adversarial Generations

The BERT Model comes delivered in two different sizes, they are called $BERT_{BASE}$ and $BERT_{LARGE}$. They are structurally very similar, as the name of the models suggests, there is one smaller model and one larger model. The larger model consists of more layers in the encoder stack, larger feed-forward layers (which enables the larger network to feed larger sentences through the network) and more attention heads. Down below can a more detailed view be made of the difference between the two networks. Both of the two networks have been pre-trained using data set of books as well as Wikipedia articles and the aim of the two models is to predict the masked word in a sentence [36], [26]. The user then needs to attach a fully connected layer at the end with hidden size of the number of classifications that are needed and fine-tune the data on a specific data set of the user's need (in this project case it's the Pandora data set with two different labels possible, male and female, which is further explained in Chapter 3.4.1.1.1).

A more detailed description of the difference between the two different BERT models can be seen in Figure 2.1.

Table 2.2: Details of the different BERT networks

Model	Size [MB]	Parameters	H.L size	Att. Heads	Encoder blocks
$BERT_{BASE}$	450MB	110M	768	12	12
$BERT_{LARGE}$	1.2GB	340M	1024	24	16

For an even more detailed explanation the paper from Google [26] as well as `https://huggingface.co/docs/transformers/model_doc/bert` can be accessed. The latter, both BERT models are accessible as well as further explanation of the details of the architecture.

2.2 Quantization

The overall idea of quantization is to re-scale the variables inside the neural network in order to be represented on a smaller byte size than the original neural network had. This means converting a floating point representation to either another floating point in smaller byte size or down to an integer point representation. There are a few different ways to do this, and here dynamic quantization will be briefly explained.

2.2.1 Dynamic Quantization

This method represents the most straightforward and adaptable approach to quantizing a neural network. It involves pre-quantizing only the weights, while the activations are dynamically quantized in real-time during inference, explaining the use of the term "dynamic." This is used where the model execution time is dominated by the loading of the weights rather than computing the matrix multiplication such as LSTMs and Transformer model types [37].

2.3 Pruning

Pruning in terms of neural network means removing or setting values to zero mainly in the larger dense layers as well as in the bias to increase the models efficiency. Here both random pruning as well as L1 pruning will be briefly explained.

2.3.1 Random Pruning

The first method of pruning that was decided to take a look at is random pruning, random pruning basically means exactly the way it sounds. After the training, random weights inside each weight matrix are set to zero according to ratio of pruning specified in the function call. If 50% pruning is decided as close to 50% as possible of the available weights in each weight matrix is set to zero. A simple demonstration of what the pruning can look like can be seen in the matrix demonstration down below.

$$\begin{bmatrix} 0.2 & 0.5 & 0.3 \\ 0.5 & 0.7 & 0.1 \\ 0.6 & 0.9 & 0.4 \end{bmatrix} \rightarrow \begin{bmatrix} 0.2 & 0 & 0 \\ 0.5 & 0.7 & 0 \\ 0 & 0.9 & 0.4 \end{bmatrix}$$

2.3.2 L1 Pruning

L1 Pruning is fairly similar to random pruning. The goal is still to null a certain amount of weights in each linear layer of the neural network, however this time the strategy is to null the lowest values in each layer. The purpose of this is due to the fact that they most likely contribute the least to the result and therefor makes the most sense to be removed in comparison to the higher valued weights, as well as lower magnitude weights won't be as affected by the change of nullifying them.

$$\begin{bmatrix} 0.2 & 0.5 & 0.3 \\ 0.5 & 0.7 & 0.1 \\ 0.6 & 0.9 & 0.4 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0.5 & 0 \\ 0.5 & 0.7 & 0 \\ 0.6 & 0.9 & 0 \end{bmatrix}$$

2.4 Performance metrics

To be able to evaluate the performance of different models, performance metrics are an essential part in the evaluation. In this thesis, accuracy, recall, precision, F-score

as well as Receiver operating characteristic has been used to determine the quality of the model, both before any modification have been made as well as after. In all the metrics below both positives as well as Negatives are mentioned frequently, a Positive in this thesis' case is a male having written the text, where as a negative in this thesis' case is seen as a female having written the text.

2.4.1 Accuracy

Accuracy is the fraction of predictions our model model got right. Formally has the following definition:

$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{2.4}$$

For binary classification, a more precise equation can be calculated in terms of positive as well as negatives:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.5}$$

Where TP = True Positives, TN = True Negatives, FP = False Positives and FN = False Negatives [38].

2.4.2 Recall

Recall or sensitivity as it's often referred to in psychology is the proportion of Real Positive cases that are actually predicted as Real Positives. Mathematically this leads to the following equation:

$$Recall = \frac{TP}{TP + FN} \tag{2.6}$$

Where TP = True Positives and FN = False Negatives [39], [38].

2.4.3 Precision

Precision or Confidence as it's usually called in data mining denotes the proportion of predicted Positive cases that are correctly Real Positives. This is primarily what Machine Learning, Data Mining and Information Retrieval focus on, but this metric is completely ignored in ROC analysis. In other words it can be identified as the measure of accuracy of Predicted Positives in contrast with the rate of discovery of Real Positives [39]. Mathematically this is summed up as:

$$Precision = \frac{TP}{TP + FP} \tag{2.7}$$

Where TP = True Positives and FP = False Positives [38].

2.4.4 F1 score

F1 score, also known as balanced F score or F measure. It can be interpreted as the harmonic mean of precision and recall, where a perfect F1 score reached value 1 and the worst score reaching 0. The relative contribution between precision and recall are equal.

$$F1 = 2 * \frac{precision * recall}{precision + recall} = \frac{2 * TP}{2 * TP + FP + FN} \tag{2.8}$$

Where TP = True Positives, FP = False Positives and FN = False Negatives [40].

2.4.5 Confusion Matrix

In the field of machine learning a confusion matrix is known as a table layout that allows for visualised performance on how a model performs, where the method is most common used in supervised learning tasks. The matrix visualises all the different outcomes of a classification task for all classifications. By definition a confusion matrix A is such that $A_{x,y}$ is equal to the number of observation known to be in group x and predicted to be in group y. In a binary classification problem this means that the count of True Positives is $A_{0,0}$, False Negatives is $A_{1,0}$, True Negatives are $A_{1,1}$ and False Positives are $A_{0,1}$. In this thesis' case this is also visualised in a quota instead of an absolute number to get more of a grasp of how the two different classes are able to be classified by the different models [41]. An example of a confusion matrix of the $BERT_{BASE}$ model fine tuned by Substorm can be seen in Figure 2.5, where Positives in this case is a text written by a male and Negative corresponds to a text written by a female.

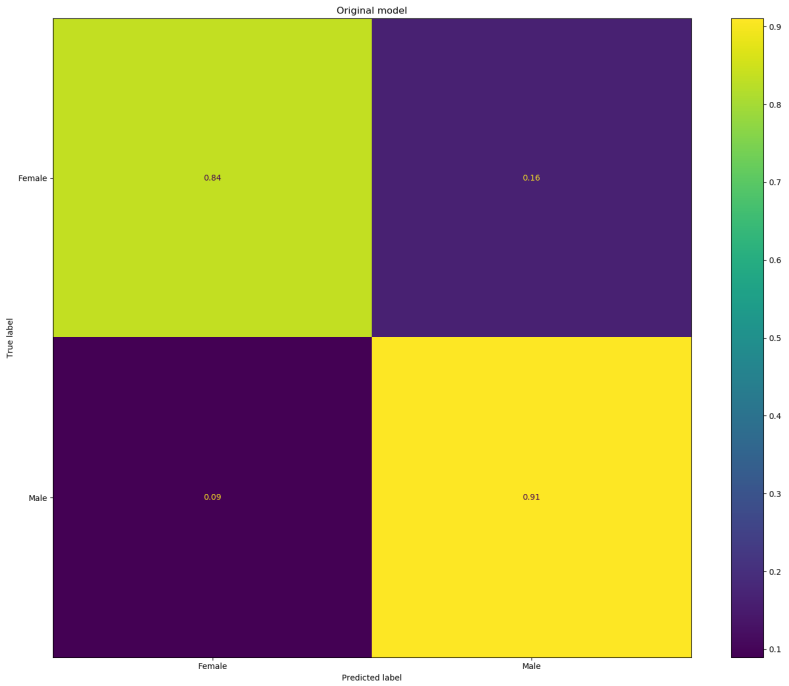


Figure 2.5: Confusion matrix for the original $BERT_{BASE}$ network, used as an example to explain a confusion matrix

2.4.6 Receiver Operating Characteristic

Receiver Operating Characteristic (ROC), analyses the rate of True Positives against the rate of False Positives at all classification thresholds. True Positive Rate (TPR) is defined as follows:

$$\text{True Positive Rate} = \frac{TP}{TP + FN} \tag{2.9}$$

and False Positive Rate (FPR) is defined as follows:

$$\text{False Positive Rate} = \frac{FP}{FP + TN} \tag{2.10}$$

where TP = True Positive, FP = False Positive, TN = True Negative, and FN = False Negative. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. A perfect good model would have a steep rise at the beginning of the evaluation, following the x-axis of the plot coming to a halt at the top of the graph following the ceiling of the graph enabling a lot of space under the plotted line. A bad, random or singular classifying model would instead follow the diagonal of the window. A totally mislabelling model would follow the inverse of the very good model and instead follow the y-axis of the plot window followed by a steep incline, following the east side of the window. This would mean very little space under the plotted line [42].

2.4.6.1 Area Under the ROC Curve

Area Under the ROC Curve (AUC), provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as the

probability that the model ranks a random positive example more highly than a random negative example [42]. AUC ranges in value from 0 to 1. A model that is classifying 100% wrong achieves an AUC score of 0 while a model that is perfect in predicting achieves an AUC score of 1. The reason AUC is desireable is due to two reasons. The first is because the score is scale-invariant. It measures how well predictions are ranked instead of their absolute value. The other is due to the fact that the AUC score is classification-scale-invariant, it measures the quality of the model's predictions irrespective of what threshold is chosen. The equation being used to calculate the area is as follows:

$$AUC = \sum_{i}^{n-1} 0.5 * (TPR_{i+1} + TPR_i) * (FPR_i - FPR_{i+1}) \tag{2.11}$$

Where n = number of thresholds, TPR = True Positive Rate and FPR = False Positive rate. This method is commonly called the trapezoid rule.

3
Methods

3.1 General methodology

To have some kind of structure during the work it's decided to follow a general working methodology. First, investigate the current solution for both the loading or the minimization of the model. Then, look at what is currently done to optimise the problem. Then implement a solution based on the research in question and analyse the results. A more detailed description can be seen below:

1. Familiarise myself with the current solution
2. Implement enhancement suggested by research papers
3. Reflect on implemented improvements
4. Extract data and results

3.2 Loading

To be able to get a grasp on how the current method holds up in comparison to other regular loading techniques it was decided to as well as testing the current loading method, which is the pytorch.lightning developed method load_from_checkpoint, with some modification to be able to acquire even more data. As well as the one currently in use it was decided to also look at other simple or more standard loading methods, for example, Pytorch's own method for loading to be able to identify competitors loading.

After this step, research was made and a method found that claims 340x speedup for machine learning models, it uses two functions described down below in Chapter 3.2.1. It also makes use of an instance initiated by the user that saves the model in what can be described is a place more accessible to the user, in a place called Plasma. Ray is an open source system for building high-performance distributed systems. What's interesting here is its main memory object store, called Plasma, which uses shared memory to pass objects between processes on each machine in a Ray cluster. If a Ray task needs to read a NumPy array from Plasma, the task can access the array's data without copying any data into its local heap. That means it's possible to store the weights of a PyTorch model onto Plasma, as long as they are converted to NumPy array before doing so, without actually making any copies and hence also speed up the process. The last thing that is done is to reconnect the weights onto the model, which means doing the reverse process of what's been done

before and convert the NumPy arrays to tensor objects. The standard way to do so creates a new Tensor object and copying over the arrays into those. But PyTorch has an alternative way of doing so which involves the torch.as_tensor() method instead of creating a new Tensor objects from scratch [43]. The steps used to make the faster loading happen are described below, which consists of two separate series of steps:

1. Load the model from the disk
2. Separate the original PyTorch model into its weights and its graph of operations
3. Convert the weights into NumPy arrays
4. Upload the NumPy arrays and the model (minus weights) to Plasma

When the model and the weights are in object storage, it then becomes possible to do a zero-copy of the model, this can be described below:

1. Deserialize the model (minus weights) from Plasma
2. Extract the weights from Plasma (without copying the data)
3. Wrap the weights in PyTorch tensor object (without copying yet again)
4. Reconnect the weight tensors into the reconstructed model

3.2.1 Functions

The two functions needed to make the process a reality is the method for extracting the tensors from the model, called extract_tensors as well as the method for restoring the tensors onto the original model, called replace_tensors [43].

```
def extract_tensors(m: torch.nn.Module)
-> Tuple[torch.nn.Module, List[Dict]]:
    """
    Remove the tensors from a PyTorch model,
    convert them to NumPy
    arrays, and return the stripped model and tensors.
    """
    tensors = []
    for _, module in m.named_modules():
        # Store the tensors in Python dictionaries
        params = {
            name: torch.clone(param).detach().numpy()
            for name, param in
            module.named_parameters(recurse=False)
        }
        buffers = {
            name: torch.clone(buf).detach().numpy()
            for name, buf in
            module.named_buffers(recurse=False)
        }
        tensors.append({"params": params,
        "buffers": buffers})
```

```
    # Make a copy of the original model and strip
    # all tensors and
    # buffers out of the copy.
    m_copy = copy.deepcopy(m)
    for _, module in m_copy.named_modules():
        for name in ([name for name, _ in
        module.named_parameters(recurse=False)]
                     + [name for name, _ in
                     module.named_buffers(recurse=False)]):
            setattr(module, name, None)

    # Make sure the copy is configured for inference.
    m_copy.train(False)
    return m_copy, tensors



def replace_tensors(m: torch.nn.Module, tensors: List[Dict]):
    """
    Restore the tensors that extract_tensors()
    stripped out of a
    PyTorch model.
    :param no_parameters_objects: Skip
    wrapping tensors in
     ``torch.nn.Parameters`` objects
     (~20% speedup, may impact
     some models)
    """

    modules = [module for _, module in m.named_modules()]
    for module, tensor_dict in zip(modules, tensors):
        # There are separate APIs
        # to set parameters and buffers.
        for name, array in tensor_dict["params"].items():
            module.register_parameter(name,
                torch.nn.Parameter(torch.as_tensor(array)))
        for name, array in tensor_dict["buffers"].items():
            module.register_buffer(name,
            torch.as_tensor(array))
```

To then be able to test the methods for loading the BERT Model and making sure that the method was improving the loading time both for smaller amounts of loading as well as larger number of loading iterations, timings for [10, 25, 50, 100] loading sequences was made, making sure that cache wasn't contributing to the results in each of them, as well as see how different amount of loading affects the timing.

3.3 Model Minimisation

Here can be seen all the work that's been done on minimising the byte-size footprint of the BERT model. It starts off with a smaller test phase where a simpler and smaller model has been worked on to quickly identify in which direction to further take the solution to test on the larger BERT model which is the model of interest.

3.3.1 Early Work

To gain a better understanding of various methods for minimizing the model, the decision was made to initially begin with something simple. This approach allows for a quicker determination of loosely which methods prove more effective, facilitating more efficient testing of solutions on a larger scale. This choice stems from the acknowledgment that testing these methods on the BERT model would be time-consuming. The aim is to avoid investing time in solutions that might not be of interest or fail to deliver results comparable to the current model. The "test" model that would be looked at would therefore be a simple fully connected network being trained on the MNIST dataset. The MNIST dataset is a large data set containing handwritten digits in the range of 0-9, where this more simple model would try to predict the digits, this would as mentioned earlier just be looked at as indications of the solutions that would be of interest to bring forward on the actual model at work, the BERT Model.

3.3.1.1 Fully Connected Network

The fully connected network is a very simple one constructed of four layers, the first one with a size of 784x1200, the second one with a size of 1200x840, the third one with a size of 840x300 and the output layer with a size of 300x10, between the first three layers standard Relu [18] activation function is used and for the last one a softmax [44] activation function is used. The loss function is CrossEntropyLoss [45] and the optimiser used is Adam [46] with a learning rate of 0.001. The model is then trained on the training data set for a total of 10 epochs with a batch size of 100. A visualisation of the test model can be seen in Figure 3.1:

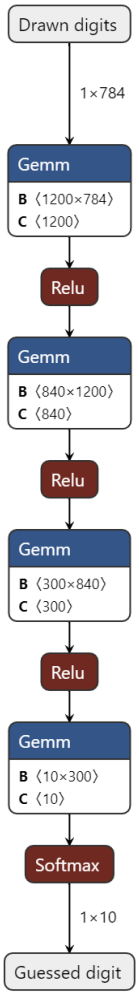


Figure 3.1: Visualisation of the early test model

To be able to replicate this on the actual model of interest this is all the training that's been considered since the BERT model won't be retrained on and only be modified after its training already have been done. This is due to the fact that the training in itself would take too long and only the testing of the actual model for the whole test data set takes almost a day in duration.
The general methodology through out the work is to only look

3.3.1.1.1 Data set The data set used as mentioned earlier is the MNIST data set. The data consist of 70000 images in 28x28 arrays displaying pixels of the hand drawn images. The data set is split in 60000 images for the training set and 10000 images for the test set. The batch size of the training data is 200 while the batch size for the test data is 100. Down below in Figure 3.2 a quick demonstration of how some of the images look like can be seen.

The training is then done under 10 epochs running where the model is improved during each of those towards the final model which then is ready for minimisation techniques.

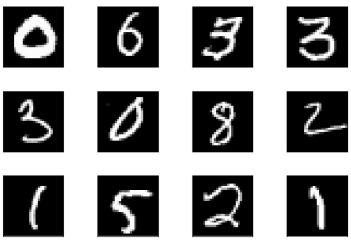


Figure 3.2: Visualisation of some of the images in grey scale

3.3.1.2 Quantization

Quantization was as mentioned in Chapter 1 considered a very viable method for reducing the memory footprint of a neural network model. This section will briefly describe the action taken to apply the method on the fully connected network to be able to get a grasp on the viability of the method. The method used is called dynamic quantization [47] from the torch.ao.quantization library in Pytorch. The method re-scales the weights on the network from its current byte size, float32, down to the smaller byte size, int8, which if we only take a look at the memory footprints of the weights would result in a 4-time decrease on size. To be noted is that this method can decrease the byte size to some larger values as well, however, an 8-bit integer is as low as the method can go. This in reality isn't really the case since there are some other values in the neural network that remain static in size. This isn't really noticeable in the fully connected neural network but will be more noticeable when the BERT model is evaluated using the same method, due to the fully connected network being almost exclusively made up of weights in terms of size. The method also gives us the convenience of only processing the model after its training duration, which as stated earlier is a criterion. A further explanation of what dynamic quantization is can be found in Chapter 2.2. There are other quantization methods at hand as well but some of them require the user to retrain the model and some just aren't suitable or available for use on transformer models with text input, as well as time being a factor.

3.3.1.3 Pruning

To further investigate methods it was decided that pruning was a method that might suit this objective adequately [3]. This paired with the conversion of the weight matrices into sparse matrices would decrease the model size depending on the pruning amount, as well as the overall structure of the model. It was decided that as well as for the quantization method to look for existing packages to do the load of the work for us and therefore Pytorch's own library for pruning [48] was decided to be used for this task. This would also work great for post-training processing where the weight matrices would be processed after training and then tested on the MNIST test data set.

3.3.1.3.1 Random Pruning The first method that decided to be investigated in viability was random pruning. Random Pruning is further explained in chapter 2.3.1. The method that has been used for this is both random_unstructured as well

as random_structured from the torch.nn.utils.prune library. Both are fairly simple to use where the strategy has been to only prune the dense layers in the network. It then creates a new layer on top of the original one identical to the layer before except for the pruned variables in the layer now being set to 0. After this is done the user can remove the old layer leaving us only with the pruned dense layer. This is done for all layers in the network for both methods.

3.3.1.3.2 L1-Pruning The second method that was decided to be investigated in viability was L1 pruning. The method that has been used for this is both l1_unstructured from the torch.nn.utils.prune library. This method works practically identically to the method described in Chapter 3.3.1.3.1 and is further explained in chapter 2.3. Once again the method can prune all different structures containing numbers in the network, mostly the weight matrices as well as the bias but it was decided to only prune the dense layers. The method then creates a new dense layer on top of the old one, which is identical except for the variables decided to be pruned set to 0. The user then can remove the old not pruned layer leaving us with only the dense pruned layer. This is then done on all linear layers in the network.

3.3.1.4 Testing

Since this is just a test to gauge how well the methods will work on the real BERT model, it's opted to stick with basic accuracy checks for the modified fully connected models. The testing is done on the set of the data not used in the training. For the original model as well as the quantized model only accuracy checks are taken but for the pruned as well as the pruned and quantized network loops over different pruning amounts are done to be able to identify potential breaking in the networks at higher pruning amounts. Only pruning of the weights are done here as consistent with later pruning method.

3.3.2 Later Work

It inherently isn't written in stone for the project to actually use the current BERT model as the model under investigation. However since Substorm currently doesn't have another model deployed, as well as the project not being drowned in training time as well as testing time if it were to be decided to use another bigger model, the base model for later work is decided to be the BERT Model in it's fine tuned state, which was also recemmended by Substorm. For model minimisation techniques currently having been done on the model it's identified that none has been performed. The only thing that can be seen as minimisation is the fact that Substorm has decided to fine-tune the smaller of the two BERT alternatives, the BASE model, in comparison to the LARGE model. That gives us free reins in identifying methods to be able to decrease the models memory footprint in order to be able to later on decrease server as well as running costs.

3.3.2.1 Data set

As mentioned in the introduction the data set that the BERT model Substorm uses is the PANDORA data set. The data set is composed of comments from the web page reddit.com, where the data comes with a number of different labels. The data set that has been used to test the network is a large fully balanced data set with 35200 text entries written by females and 35200 text entries written by males. This data set is unfortunately a subset of the training data but was discovered too late to run larger simulation of the test portion of the data set. This was mostly due to the case of bad communication in the earlier stages of the project. The data set ranges from really short prompt to rather large ones. The interesting one in this Thesis is the gender label which the model has been trained to predict, where the data will be used to test the quality and robustness of the models developed. A few examples of the test data can be viewed down below:

- blah blah blah! | Female
- Haha...I think this is possibly the stupidest band name I've thought of for this thing. It's quite possible. lol. Anyways...I don't remember the poll either. And I've gotten over my problem with not being the only girl on here...so I think she should stay! lol Larry's funny. Anyways. Life is pretty boring. Everyone's sick. It's dreadful. I hate being sick.SQEE! | Male
- This is the greatest website ever... bask in the greatness of urlLink Rate my Kitten.com ! PS I really wish I was being an ironic hipster with this one, but anyone who's seen me around a cat, or heck just my desk at work, knows that I just love cats... | Male
- I have to keep breathing. Because tomorrow the sun will rise. Who knows what the tide will bring?"" Tom Hanks - Castaway | Female
- Economy | ""So often we rob tomorrow's memories by today's economies."" – John Mason Brown | Male

To be able to ensure the model developed are able to produce decent results on new data as well a smaller test set not previously seen by the model also originating from the PANDORA data set was used. That's a much smaller set and contains similar samples as the data set mentioned here. The results from those tests can be seen in Appendix 1 and follows the exact same structure as the methodology that follows in chapter in the rest of chapter 3.4.2.

3.3.2.2 Quantization

The same methodology is taken here as in the MNIST network, take the fine-tuned model that has been trained on the PANDORA training set. And convert all the weight matrices in the network from 32-bit variables into 8-bit variables, made using the $quantize_dynamic$ method in the torch.ao.quantization library. The method works straight out of the box on the fine tuned BERT model that's been trained on the PANDORA data set, hence all we need to provide the method is the model that is intended to be quantized, and what datatype to quantize the weights into, which is the qint8, which is pytorch own int8 data representation which stores the quantized value as an 8-bit integer as well as a scale and a zero-point, as previously

described.

3.3.2.3 Pruning

Even though there are several pruning methods available and random pruning also presented and tested in the early work, it was early on quite clear that the one method that would perform the best out of the two, as well as was shown in the result for the MNIST Network was the more calculated pruning metrics where only the smaller weights was removed instead of the random. Because of the random nature of the pruning process, the decision was made to exclusively employ L1 pruning in this context. In order to selectively prune only the dense layers within the BERT network while preserving other components such as the LayerNorm layers, the dense layers were intentionally grouped into a list and iterated over to ensure exclusive pruning for each specified amount. The pruning amount that was decided to be tested over was 60%-95% pruning over the dense layers. This resulted in an overall pruning percentage of the whole network between 46%-73.15% due to not all the components in the network being pruned for each iteration of pruning amount.

3.3.2.4 Pruning and Quantization

Due to curiosity as well as the fact that it would be possible it was also determined to prune as well as quantize the same amount of networks being evaluated in the pruning section. That was made in order to determine if it was possible to further minimise the model size, the same methodology was made here as in the pruned models, except for the fact that it was firstly pruned and lastly quantized using l1_unstructured from torch.nn.utils.prune library as well as quantize_dynamic from the torch.ao.quantization library. The same pruning amount was considered here as for the pruned only models, 60%-95% pruning amount for all dense layers in the network.

3.3.2.5 Testing

To be able to determine the quality of the three main methods that has been developed there needs to be adequate testing on them to assure their robustness in comparison to the original BERT model. Since the quantized model only comes in one configuration, there's no need to test for different parameters. However, for both the pruned as well as the pruned and quantized model there needs to be pruning values determined to be able to quicker find out how far it's possible to push the pruning amount without breaking the model. It was therefore decided to first only test the pruned model on a subset of the data set while varying the pruning amount from 0, 100 percent in 20 steps to be able to determine which lower value on the pruning amount as well it's higher value. The metric for this step was accuracy. This is primarily due to the fact that one simulation of running through the test data on the original model takes almost a full day (due to the size of the data set) in which there's not enough time available, as well as pruning isn't as much of a worthwhile method for lower pruning amounts since the variables that aren't pruned in each layer needs to have their indices added to the weight matrices. The batch

size decided upon in the testing phase was 100.

The pruned models, encompassing various pruning amounts, and the pruned and quantized models with varying amounts (ranging from 60% to 95%), representing the chosen minimum and maximum pruning extents, were subjected to measurement using metrics that exclusively consider the predictions of the models described in Chapter 2.4. These metrics evaluate the models' performance towards the correct labels from the data set. The models in the range of 60-85% would also have their outputs from the model saved to be able to create more performance metrics, which then would be used to also create ROC curves for those models as well as compute AUC scores. This was also done for the original model as well as the quantized model. The reason that the two highest amounts of pruning (90%, 95%) was decided to be left out was for one their really poor performance on the earlier performance metrics as well as the fact that these values were collected after the initial batch of runs already had been done, which would mean that some time was saved in the testing phase.

The second to last thing that needs to be done is to compute the size difference compared to the original $BERT_{base}$ model, for the quantized model that's easy due to the fact that the size decrease is already implemented. For the pruned variant that gets a bit more tricky, due to the fact that the models in their current states aren't actually decreased but they are available to be as long as a sparse implementation is implemented. A sparse implementation instead of keeping all variables in a dense layer only keeps the none-zero ones and instead keeps track of the indices in which the variables used to have. This can be for example the package sparselinear in pytorch, which contains tool for doing these sort of operation while also keeping the necessary components to be able to run a neural network the same way as a dense layer would. To be able to calculate this, first, it's looked at how much overall decrease in variables have been made for different pruning amounts, since not all variables are pruned but only the variables contained in dense layers. After that, the indices are assumed to have int16 representation due to the fact that not all dense layers have lengths small enough to be able to fit in a int8 space. It's then possible to look at how much space the dense layers would have in comparison to the original 110M variable model. The pruned and quantized model follows the same strategy however this time it's compared against the quantized model. This means that the variables inside the dense layers will be counted as being of int8 representation, while the indices for the weights remaining after pruning will still be of int16 representation. Those will then be compared against the 110M variable quantized model.

One final test was made after all other simulations was done in order to be able to further determien the developed models viability. Substorm has another dataset with gender labeled texts, this data set however was early on discarded since the accuracy performance on even the $BERT_{BASE}$ model was very low (below 60%). However due to the models needing some further evidence it was decided that the data set could still be of use if it's looked how the developed models are predicting

in comparison to the $BERT_{BASE}$ model. The first 1000 text samples in the data set was compared against.

4

Results

4.1 Loading

The results for average loading of Pytorch's load method, Substorm's current method, Huggingface's load_pretrained, Ray's loading method both with the initialisation of the Ray instance as well as without it. The loading times can be seen in Figures 4.1, 4.2, 4.3 and 4.4 where the iterations are [10, 25, 50, 100] as specified in the captions.

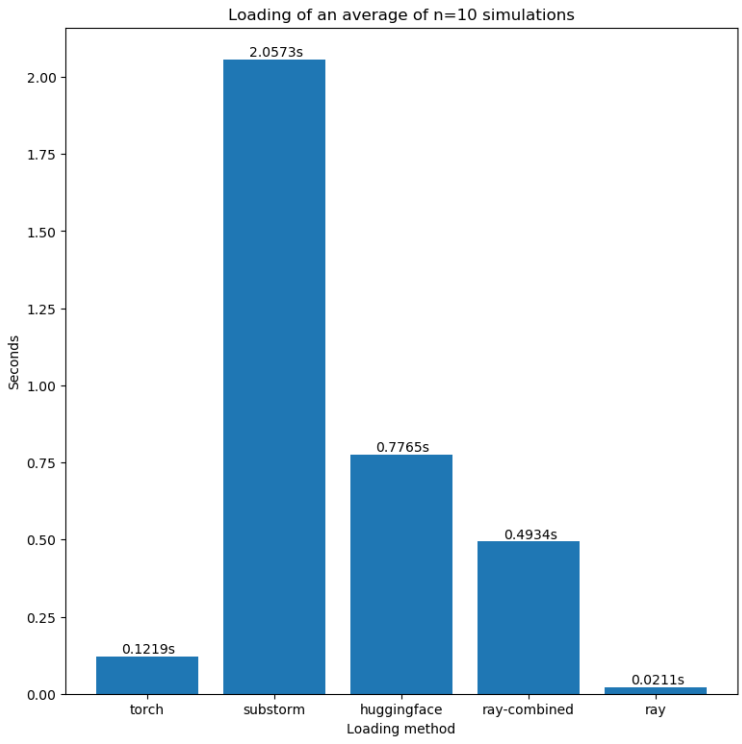


Figure 4.1: Loading of BERT model for different methods an average of 10 times

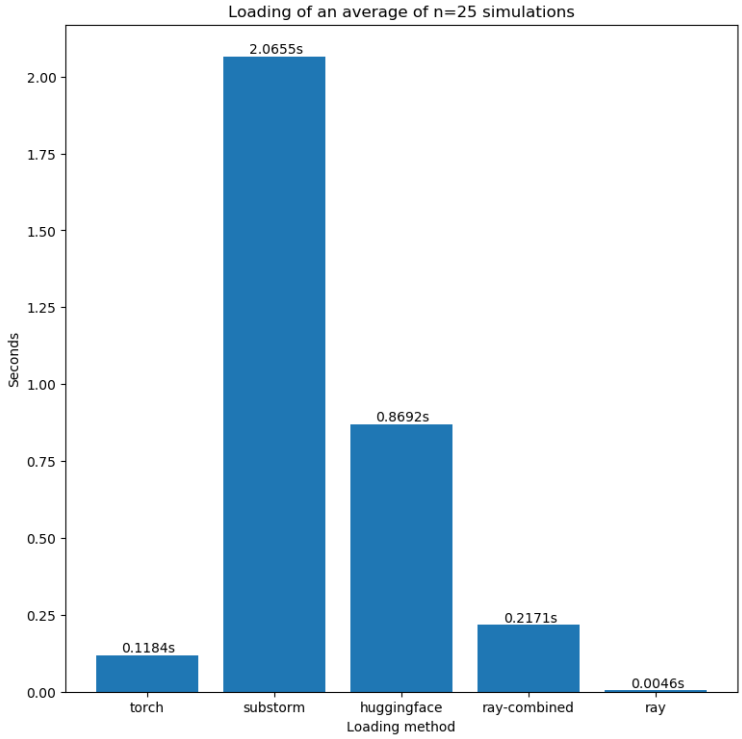


Figure 4.2: Loading of BERT model for different methods an average of 25 times

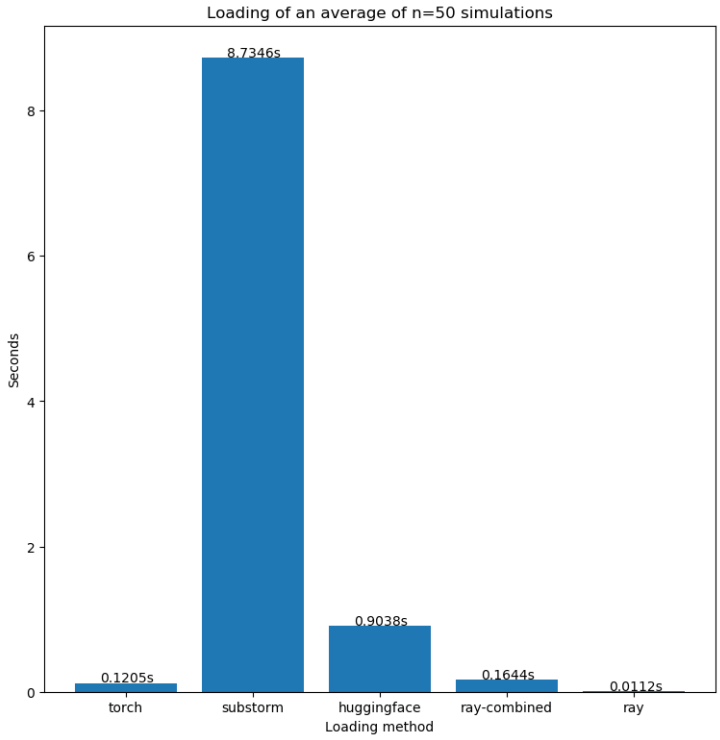


Figure 4.3: Loading of BERT model for different methods an average of 50 times

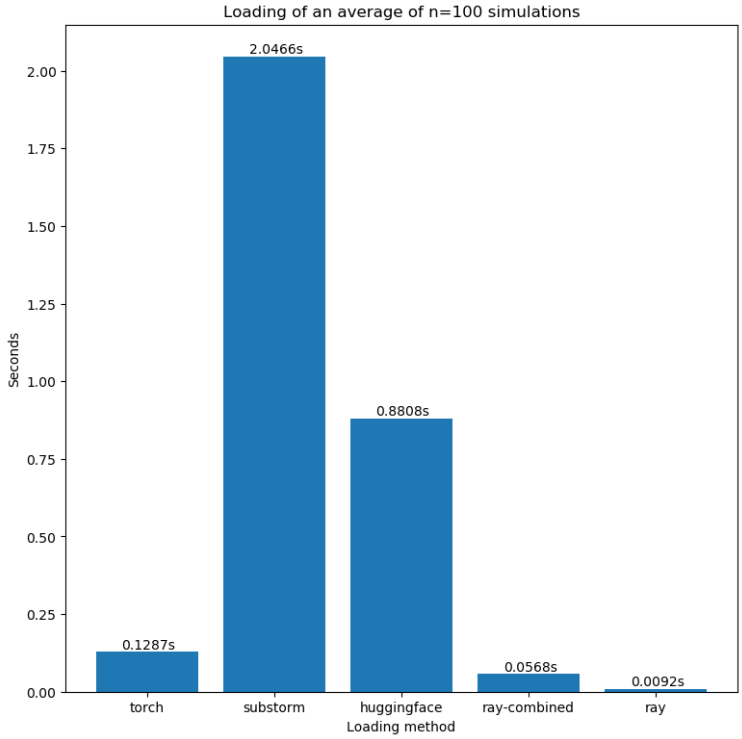


Figure 4.4: Loading of BERT model for different methods 100 times

4.2 Model Minimisation

4.2.1 Early Work

To be able to determine how much the early methods contributed to both the accuracy as well as the minimisation of the MNIST Network first it had to be determine how accuracy as well as how big the original model was, that can be seen in Table ??.

Table 4.1: Results for original Fully Connected MNIST Network: Both the accuracy as well as the current size of the MNIST network post training and tested on the test data can be seen

Network	Accuracy	Size [MB]
MNIST-net	95.29%	8.826865

4.2.1.1 Quantization

The quantized MNIST-Network takes the same approach as the original network, after training of the network, quantize the network using quantize_dynamic in the pytorch.ao.quantization library and test the network on the whole MNIST test data set. All the results can be seen down below in Table 4.2.

Table 4.2: Results for dynamic quantized Fully Connected MNIST Network: Both accuracy as well as the current size of the modified network can be seen in

Network	Accuracy	Size [MB]
MNIST-net dyn quant	95.32%	2.217465

4.2.1.2 Pruning

As mentioned in chapter 3, it was determined to construct pruned networks for both random unstructured pruning, random structured pruning as well as L1 unstructured pruning. That was done through looping through different pruning amounts, 10-90% while testing the pruned models on the whole test data set. This was as again mentioned in chapter 3 to be able to determine which methods to move forward with on the much larger BERT model in order to save time on testing those networks later on. The results for random unstructured pruning on the MNIST network can be seen in 4.5 as well as Table 4.3 where the accuracy numbers are delivered in a bit more easy to read fashion.

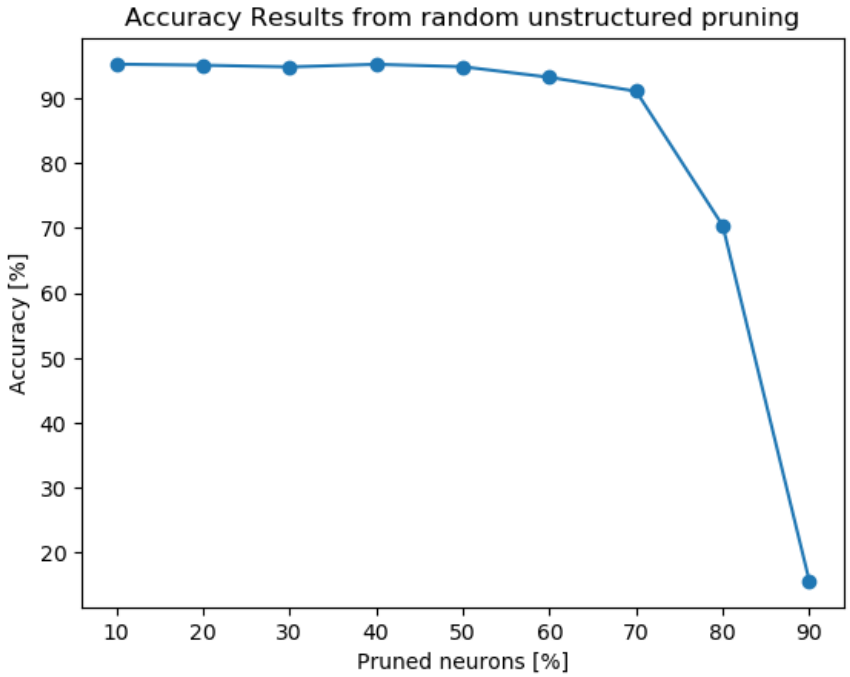


Figure 4.5: Accuracy results for pruned Fully Connected MNIST Network with different pruning amounts for random unstructured pruning

Table 4.3: Accuracy scores for random unstructured pruning of MNIST Network

Pruned Amount:	Accuracy
10%	95.36%
20%	95.08%
30%	95.51%
40%	94.94%
50%	95.09%
60%	94.33%
70%	91.45%
80%	61.46%
90%	9.95%

Since the method was available to us, it was also decided to test the method random_structured pruning to be able to identify it's viability moving forward. The result for that simulation which took the same strategy as the random_unstructured one where it was decided to loop through different pruning amounts between 10-90%. The results for that simulation can be seen in Figure 4.6 as well as Table 4.4 where the specific values can be seen a bit more easily.

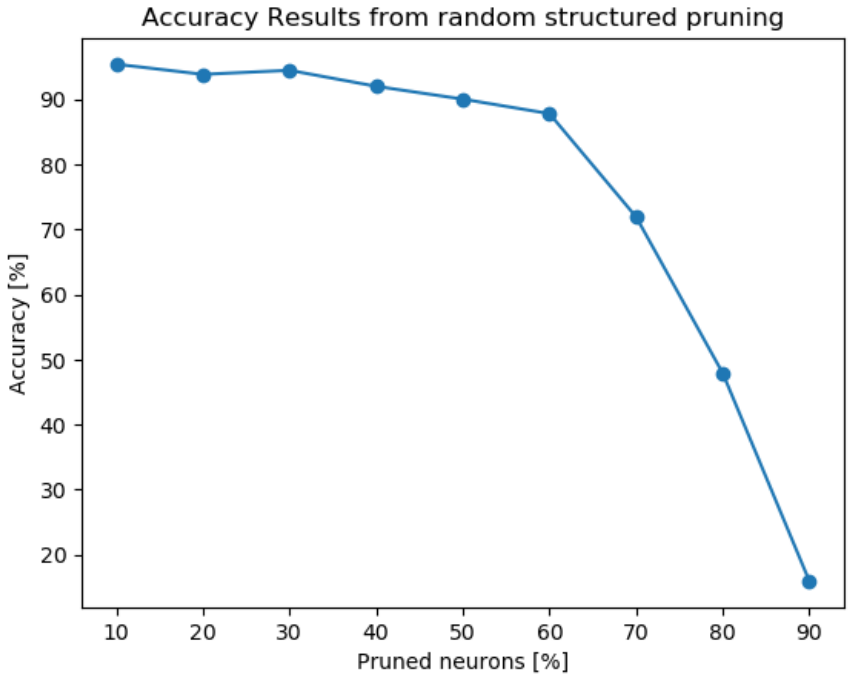


Figure 4.6: Accuracy results for pruned Fully Connected MNIST Network with different pruning amounts for random structured pruning

Table 4.4: Accuracy scores for random structured pruning of MNIST Network

Pruned Amount:	Accuracy [%]
10%	95.06
20%	94.57
30%	93.86
40%	89.90
50%	90.52
60%	86.65
70%	74.10
80%	51.91
90%	11.29

Lastly the final method that was tested on the MNIST Network was L1 unstructured pruning, as was described in Chapter 3 removes the smallest values in each weight matrix to be able to remove what can be described as lesser important values in each matrix to be able to ensure a better result. Once again the same strategy was made as the two earlier pruning methods, loop through different pruning amounts between 10-90%, test the accuracy on the whole test data set. The results can be seen in Figure 4.7 as well as Table 4.5 where the accuracy values can be seen a lot more clearly.

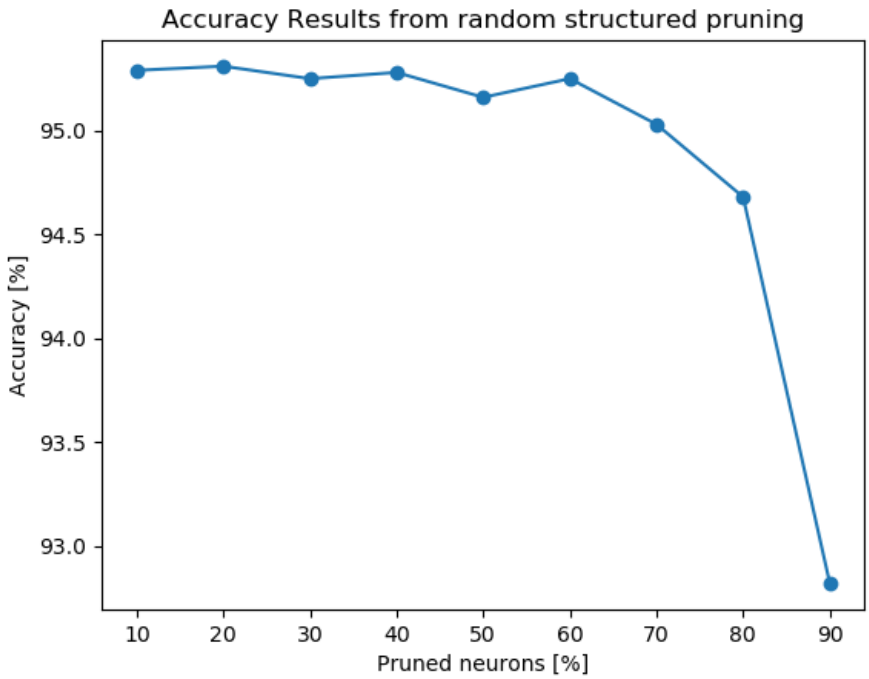


Figure 4.7: Accuracy results for pruned Fully Connected MNIST Network with different pruning amounts for L1-unstructured pruning, to note is that the limit on the y-axis is different compared to Figure 4.5, 4.6.

Table 4.5: Accuracy scores for random unstructured pruning of MNIST Network

Pruned Amount:	Accuracy [%]
10%	95.29
20%	95.31
30%	95.25
40%	95.28
50%	95.16
60%	95.25
70%	95.03
80%	94.68
90%	92.82

4.2.2 Later Work

Almost an identical strategy has been taken for the process of testing the methods on the original model as in the earlier work. Implement the methods on the model in question (the BERT model), which has been tested on a limited amount of pruning ratios due to time constraints (a whole simulation of testing on the original model takes almost a full day), which means only test values a bit more closely towards where it can be assumed the model breaks in quality. But, to be able to determine the quality of the methods of the it must first be determined how the original BERT model performs, that can be seen in table 4.6.

Table 4.6: Performance metrics calculated on the $BERT_{BASE}$ network tested on the complete data set

Model:	Accuracy [%]	Precision [%]	Recall [%]	F1 score [%]	Size [MB]
$BERT_{BASE}$	87.27	84.67	91.03	87.73	438.004169

As well as the metrics listed above, it was also decided to compute a confusion matrix. That can be seen in Figure 4.8.

Finally, a ROC curve was constructed as well as an AUC score was computed. The ROC curve tells us how certain the model is at classifying the input text and can therefore be used as another metric for determine the quality of the network. The result from those can be seen in Figure 4.9 as well as Table 4.7.

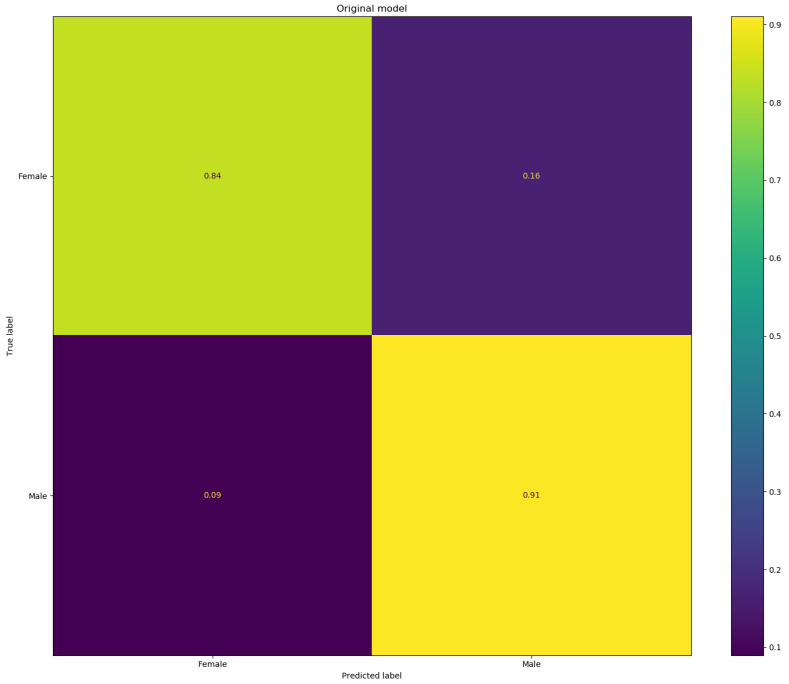


Figure 4.8: Confusion matrix for the original $BERT_{base}$ network

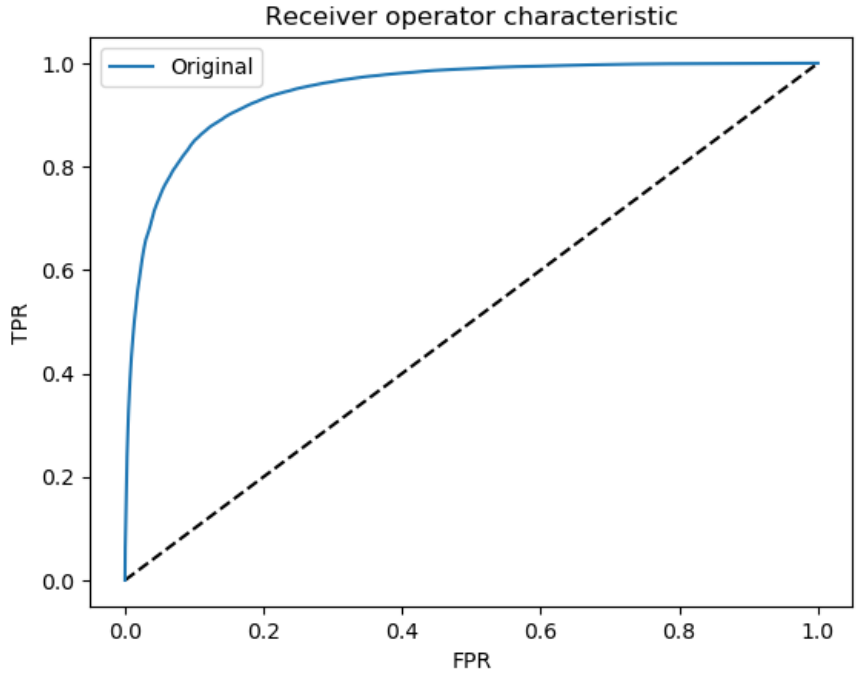


Figure 4.9: ROC curve for $BERT_{BASE}$ network. The diagonal dashed line is to symbolise a network that is completely guessing and or in our case, only guesses one of the labels since it's a balanced data set

Table 4.7: Area Under Curve for the ROC curve that can be seen in Figure 4.9

Model:	AUC
$BERT_{BASE}$	0.94766

Further on in this chapter the dynamic quantized network will be noted as BERT quant and the pruned variant will have the percentage of which was pruned for each model in the figure and/or table description as well as a notation specifying whether or not it's only a pruned network or both a quantized and pruned network.

4.2.2.1 Quantization

Here can be seen all the evaluation done on the dynamic quantized BERT network. It starts off with the same performance metrics measured as on the original $BERT_{BASE}$ network that has been fine-tuned on the PANDORA training data set. The summary of that can be seen in Table 4.8.

Table 4.8: Performance metrics calculated on the quantized BERT network tested on the complete test data set

Model:	Accuracy [%]	Precision [%]	Recall [%]	F1 score [%]	Size [MB]
BERT quant	87.24	84.59	91.06	87.71	181.487013

Following the performance metrics, again similar to the $BERT_{BASE}$ network, a confusion matrix was constructed to give further indication on where the strength and weaknesses of the quantized BERT network lies. That can be seen in Figure 4.10.

Lastly, both a ROC curve was constructed as well as AUC was calculated to be able to determine the quantized networks robustness, this gives us even more data to be able to determine the quantized networks' robustness in comparison to the original $BERT_{BASE}$ network. Those can be viewed in Figure 4.11 and Table 4.9 respectively.

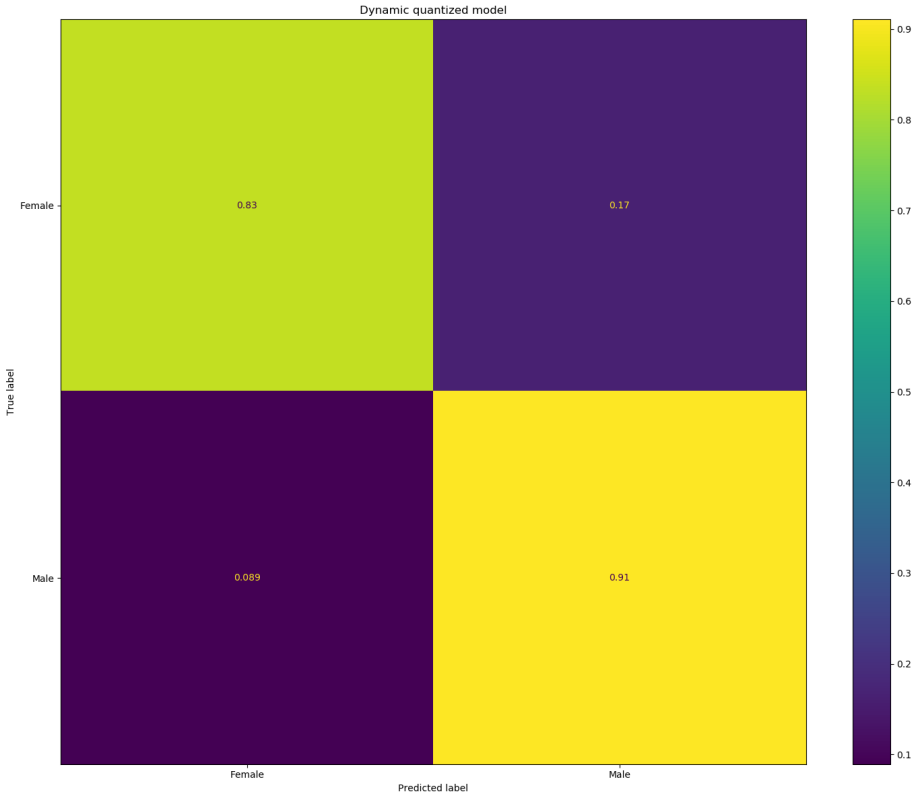


Figure 4.10: Confusion matrix for the quantized BERT network

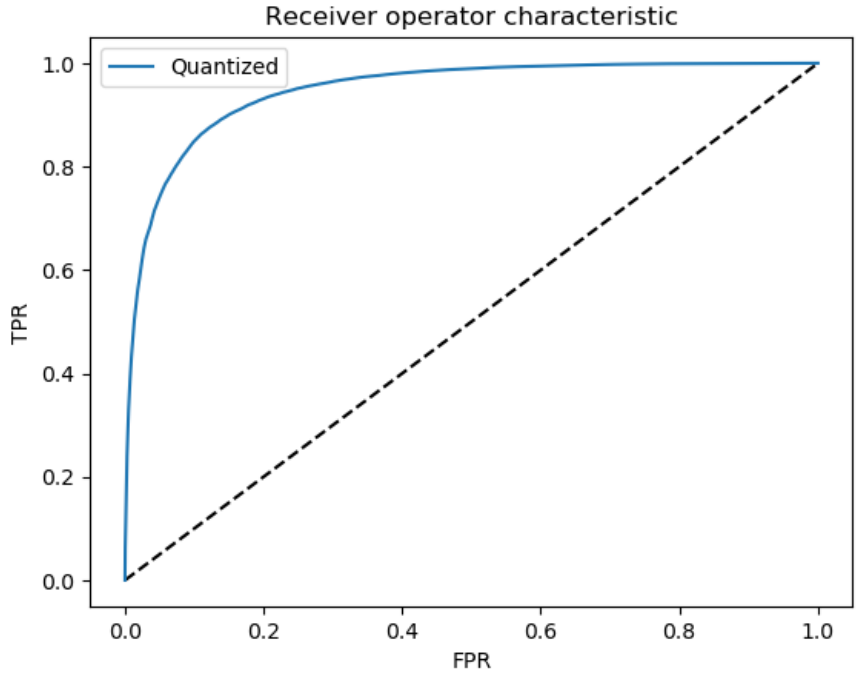


Figure 4.11: ROC curve for quantized $BERT_{BASE}$ network. The diagonal dashed line is to symbolise a network that is completely guessing and or in our case, only guesses one of the labels since it's a balanced data set

Table 4.9: Area Under Curve for the ROC curve that can be seen in Figure 4.11

Model:	AUC
BERT quant	0.94763

4.2.2.2 Pruning

As mentioned in Chapter 3, it was determined to first loop over a subset of the test data and vary the pruning amounts in this section to be able to identify breaking points in the amount where the model couldn't handle the amount of pruned variables in the dense layers. This was again done in order to not having to run simulations on pruning values lower than necessary in order to save time in the testing phase. The results of that simulation can be seen in Figure 4.12, as well as details of the simulation in the caption to said figure.

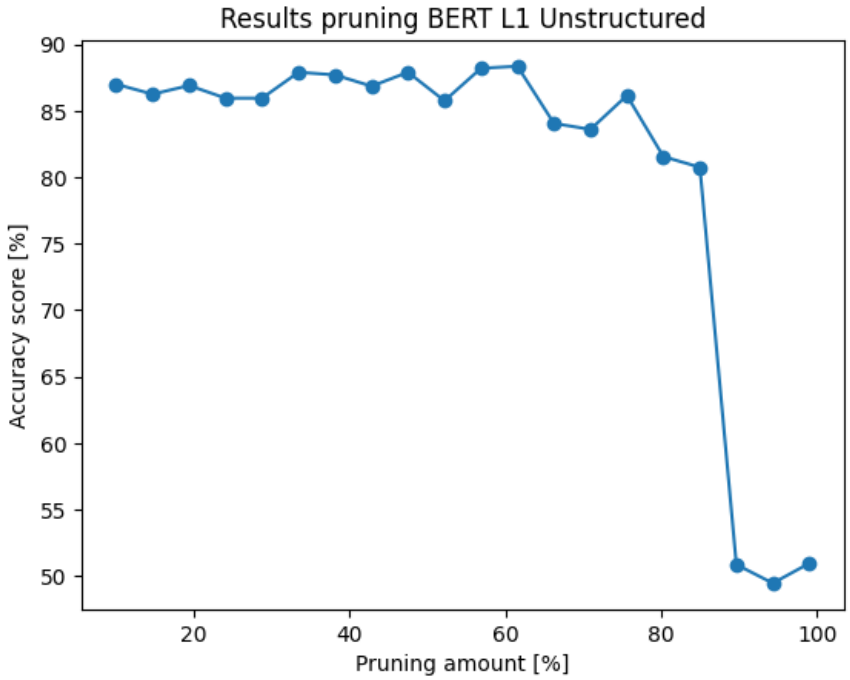


Figure 4.12: Results of the test when looping through different pruning values of pruned BERT networks. This test was made in order to find a reasonable starting point for the pruning values in order to save some time in the testing phase, due to the data set being fairly large. The test amount was 2000 samples from the data and the pruning amount was varied from 10%-99% with 20 steps chosen linearly in that span.

After looping through a sub set of values, 2000 samples in the test data set, the values decided to prune over the BERT network was 60%-95% with an increment of 5% in each step. This was decided from the data collected in Figure 4.12 as well some more space taken before the breaking point of the pruning amount on both sides to confirm that it wasn't just a deviation in the smaller amount of testing

samples in said figure. The results of that simulation as well as performance metrics taken can be seen in Table 4.10.

Table 4.10: Performance metrics for the pruned BERT models

Prune amount:	Accuracy [%]	Precision [%]	Recall [%]	F1 [%]
60%	85.53	92.99	76.86	84.15
65%	84.70	93.48	74.60	82.98
70%	87.35	86.45	88.57	87.50
75%	78.84	94.56	61.19	74.30
80%	78.92	94.56	61.36	74.43
85%	51.28	99.13	2.58	5.02
90%	50	0	0	0
95%	50	0	0	0

To be consistent with previous tests, confusion matrices were constructed for all pruned models evaluated on the entire data set, again with pruning amounts ranging from 60%-95% with 5% increment for each model. That can be seen in Figure 4.13.

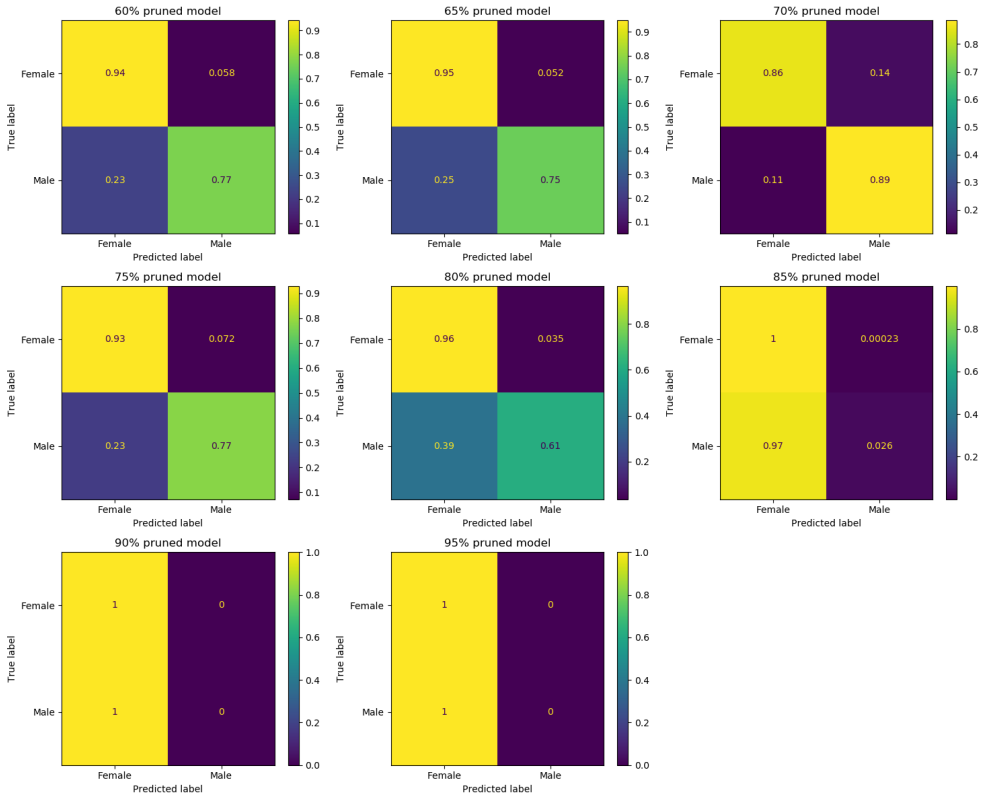


Figure 4.13: Confusion matrices for all pruned models evaluated on the entire PANDORA test data set

Again to be consistent with previous tests as well as confirming the robustness of the modified models, ROC curves for most pruning amounts were calculated, as well as AUC scores for the models given ROC curves. The two models with the highest

pruning amounts, the model with 90% as well as 95% pruned dense layers were decided to be discarded from this due to their poor performance in the previous performance metrics evaluations, which can be seen both in Table 4.10 as well as Figure 4.13. The result from that simulation can be seen in Figure 4.14 as well as Table A.6.

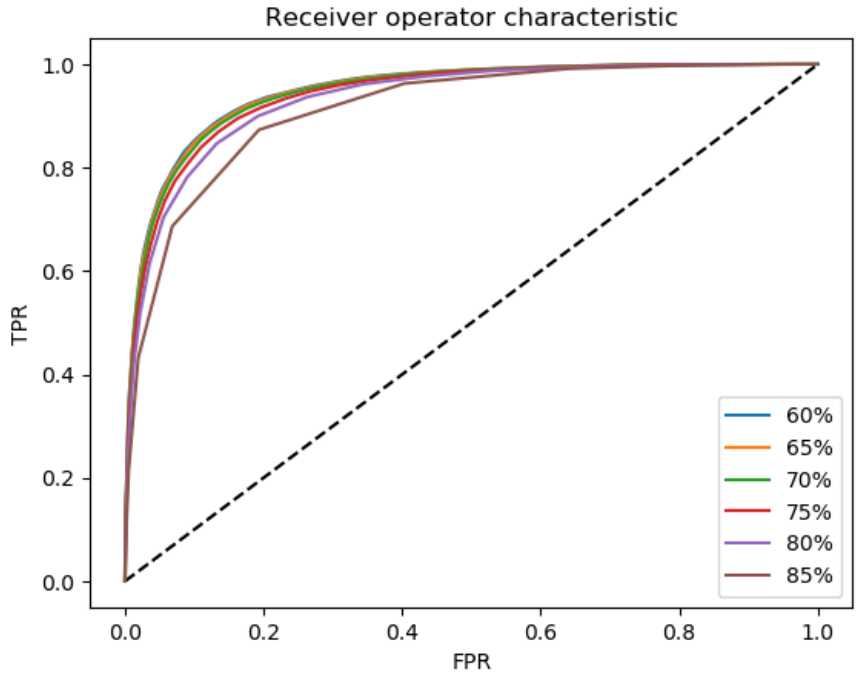


Figure 4.14: Receiver operating characteristics for pruned model variants. The different pruning amounts are denoted by it's percentage value in the labels in the figure, ranging from 60%-85% in steps of 5%. The diagonal dashed line is to symbolise a network that is completely guessing and/or or in our case, only guesses one of the labels since it's a balanced data set.

Table 4.11: AUC scores for different pruning values of the BERT model

Model	AUC score
60% pruned	0.94838
65% pruned	0.94707
70% pruned	0.94486
75% pruned	0.94021
80% pruned	0.93264
85% pruned	0.91311

Lastly, a table showing how the weight sizes in the pruned models differ depending on the amount of pruning done of the $BERT_{BASE}$ model, that can be seen in Table 4.12.

Table 4.12: Overall weight size decrease in comparison to the original model. If the number is above 100% that means the weight size is increased compared to the model it's compared to

Prune amount:	Weight size [%]
60%	80
65%	70
70%	60
75%	50
80%	40
85%	30
90%	20
95%	10

4.2.2.3 Pruning and Quantization

Lastly, here are all the results from the pruned as well as quantized model variants. The models follow the exact same structure as the pruned model variants where firstly tests on smaller samples of the data set were made in order to find critical values where the model couldn't handle the amount of pruning in the dense layers. The results from the smaller simulation can be seen in Figure 4.15.

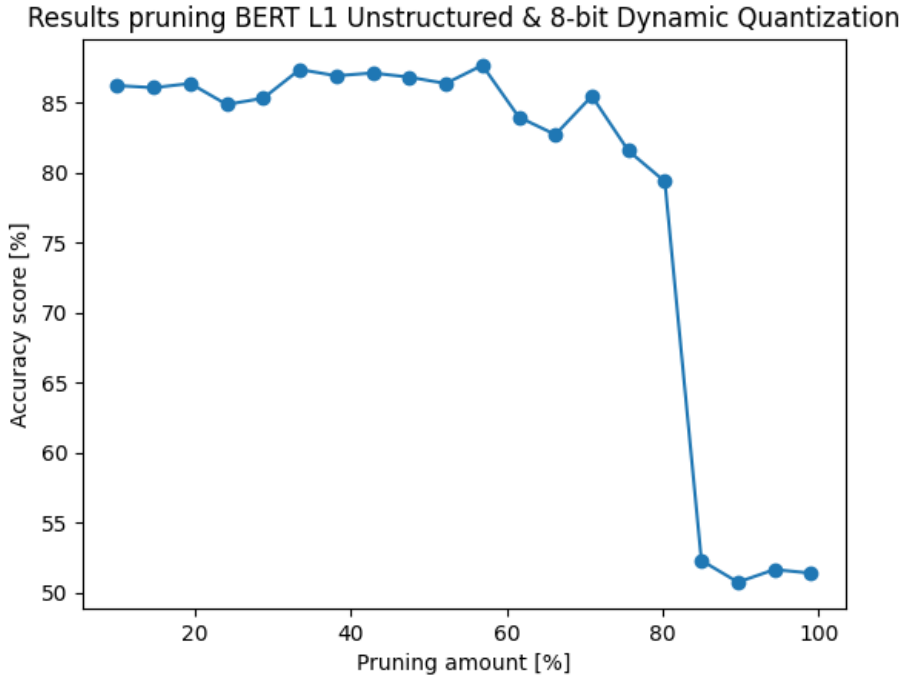


Figure 4.15: Results of the test when looping through different pruning values of pruned and quantized BERT networks with varying pruning amount. This test was made in order to find a fitting starting point for the pruning values in order to save some time in the testing phase. The test amount was 2000 test samples from the test data and the pruning amount was varied from 10%-99% with 20 steps chosen linearly in that span.

After the smaller testing phase on a larger variant of pruning amounts, it was decided to move forward with the same pruning span as in Chapter 4.2.2.2, which means running the whole PANDORA data set on 60%-95% pruned dense layers in the BERT network. The same performance metrics was taken as in the previous sections of Chapter 4.2.2. The results from that can be seen in Table 4.13.

Table 4.13: Performance metrics for pruned and quantized BERT models

Prune amount:	Accuracy [%]	Precision [%]	Recall [%]	F1 [%]
60%	85.31	93.23	76.14	83.83
65%	84.47	93.66	73.94	82.64
70%	87.37	86.67	88.34	87.50
75%	84.93	91.68	76.84	83.61
80%	78.84	94.56	61.19	74.30
85%	51.30	99.14	2.62	5.01
90%	50	0	0	0
95%	50	0	0	0

Further more, keeping consistent with previous test in Chapter 4.2.2, a confusion matrix was to be constructed for all the models tested in Table 4.13. The results from that can be seen in Figure 4.16.

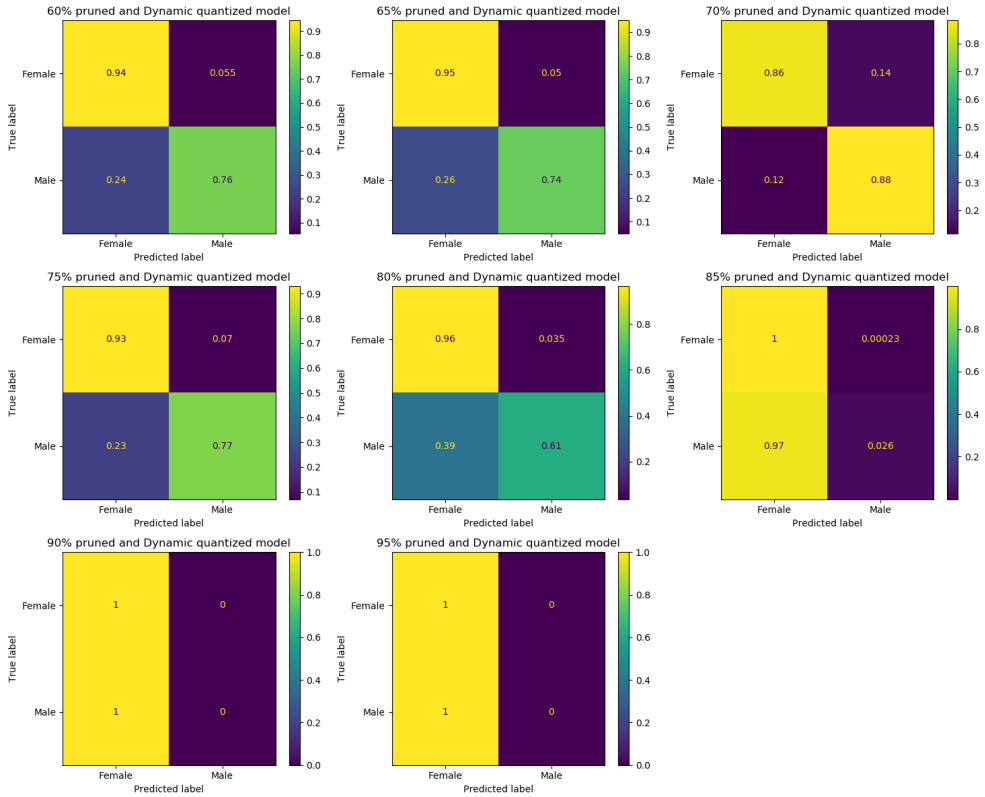


Figure 4.16: Confusion matrices for all pruned and quantized models evaluated on the entire PANDORA test data set

Moving on keeping consistency with earlier work in chapter 4.2.2, ROC curves for most of the models evaluated in Table 4.13 was constructed, discarding the two highest pruning amounts due to very poor performance metrics in Table 4.13. The results from those can be seen in Figure 4.17.

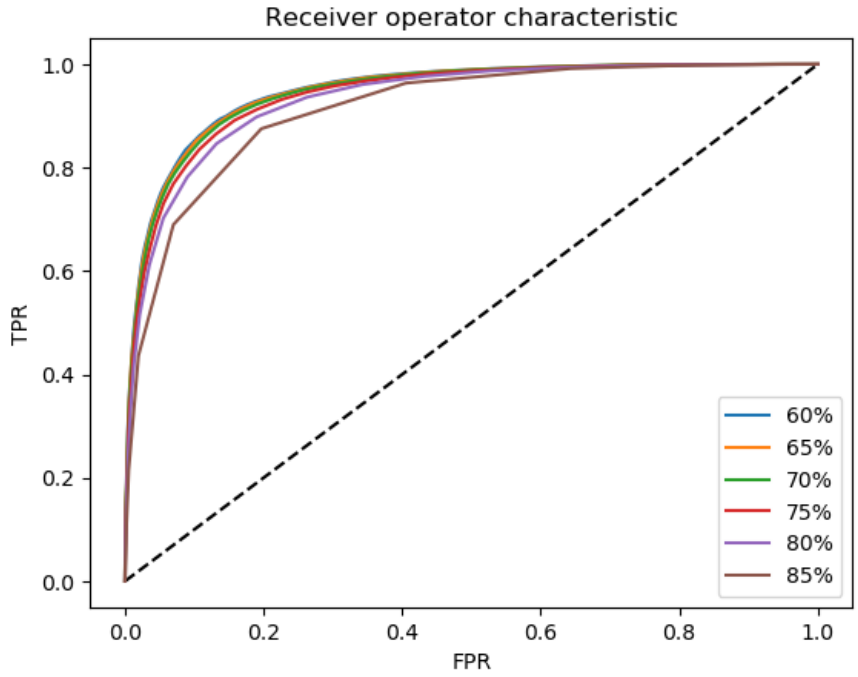


Figure 4.17: Receiver operating characteristics for pruned and quantized model variants

Lastly, AUC scores for all the models evaluated in Figure 4.17 and the results from those can be seen in Table 4.14.

Table 4.14: AUC score for different pruning values of a pruned and quantized BERT model

Prune amount:	AUC score
60%	0.94837
65%	0.94706
70%	0.94482
75%	0.94018
80%	0.93260
85%	0.91299

Lastly, to be able to see if these methods are actually of value overall it's necessary to know with how much the pruning can affect the model size. The overall size decrease of the weights in the models can be seen below in Table 4.15, the weights in a $BERT_{BASE}$ model accounts for approximately 77.83% of it's total size, which means in the weights in a quantized $BERT_{BASE}$ would account to roughly 1/4 of that value for a quantized model due to the weights being int8 instead of float32.

Table 4.15: Overall weight size decrease in comparison to the quantized model. If the number is above 100% that means the weight size is increased compared to the model it's compared to

Prune amount:	Weight size [%]
60%	200
65%	175
70%	150
75%	125
80%	100
85%	75
90%	50
95%	25

5

Discussion

For the results seen in Figure 4.1-4.4 it's determine as a major success in order to be able to minimise loading time to be able to load in a model from storage. The reason for there being two different staples for the Ray version is as described in chapter **??** there needing to be an initial initialising of the Plasma server to be able to store the tensors in an efficient way. But still, even with that as can be seen in ray-combined instead of simply ray, where ray and the creation of the instance Plasma being used have been combined, there's a massive difference in loading time of the model compared to the original solution labeled substorm. Depending on where the model has been stored initially it also might be necessary to use the original loading method once in order to be able to access the model, but all other times the model then is loaded to the user it should be able to use the faster ray sequence, this isn't accounted for in the test due it being a negligible difference when the user is loading in the model multiple times. The model is also confirmed to keep it's original structure post fast loading which enables it to be used exactly like the model would have in case it was loaded any other traditional way.

Let's discuss the issue of using training data to validate the models instead of test data as can be described in Chapter 3.3.2.1. It was done mainly due to the fact that the data that was provided for me by Substorm was very small in terms of test data (500 rows of text), as well as a lack of communication from the author part when simulating the tests. The different methods, as well as all the performance metrics has been tested towards the much smaller testing set as well and the information provided similar results, those can be seen as previously mentioned in Appendix A. The only major difference being in figure A.8, where the ROC scores and subsequently the AUC scores are following the diagonal for all models evaluated. The other difference being the overall scores across all performance metrics for all models, including the original BERT model, dropping about 7% points in overall accuracy but the model minimisation techniques being used show similar values as the larger set being used which makes it competitive to the result also being represented. It also isn't as big of a problem as one might initially think. The problem we are trying to investigate isn't to optimise a model and inherently validate it towards test data never before seen by the network. The reason it's important to test the networks being developed is to test them towards the original model in terms of the performance metrics holding up. It's realised that it isn't in best practice to test on data previously seen by the network in the training phase but the results should still be seen as valid due to the fact that as long as the networks developed holds up it's performance towards the original model.

For the results seen in Chapter 4.2.2.1 we can see that the dynamic quantized network follows an impressive similarity to the original BERT in every single metrics which can be seen in Table 4.8, 4.9 as well as Figures 4.10, 4.11 in comparison to all metrics being presented for the $BERT_{BASE}$ in the first part of 4.2.2. This in contrast to the fact that the quantized model only contains a byte-size footprint of 181MB in comparison the original model which contains a byte-size footprint of 438MB leads us to a decrease in size of 60% while still being functional close to identical to the original model.

For the pruned models the performance metrics holds up fairly well for the lesser pruned models of 60-75% where the recall score as well as the F1 scores is dropping a bit more than the other scores as can be seen in Table 4.10. It can clearly be seen as early as 85% that the model also can't keep up with the pruning amount and the models only predicting female authors of the texts provided, leaving us with very high precision scores while very low recall score. This is also very easily detected in figure 4.13, where it's clearly visible. However, both the ROC scores as well as the AUC scores holds up for the models provided to it, 60-85% which is fairly good news and something to take away from it all. For the smaller test set the performance metrics there keeps the same closeness to the original model, which can be seen in Table A.5, Figure A.5, 4.12 and Table A.6. All the values there are lower than the one provided for the larger PANDORA data set used however when compared to the original model they seem to follow the same trend. The model weight size are 80-30% of it's original size respectively for pruning amounts of 60-85%, with 75% having the most pruning while still holding up overall average, with 50% decrease in size of the weights, these details can all be seen in Table 4.12.

For the pruned and quantized models the pattern follows the pruned models very closely. The models 60-75% holds fairly competitive performance metrics as can be seen in Table 4.12 as well as Figure 4.16. The models passed the pruning amount of 75% contains the same problem as the pruned models of similar pruning values, the models start to increasingly predict women as authors of the texts provided in the data set and the precision score increases to very high values while the recall score plummets. This can as before also be seen in the confusion matrix for the models provided in Figure 4.16. The models that have been evaluated in Figure 4.17 looks to be holding up fairly well but when looking at the same models being evaluated on the smaller PANDORA data set which can be seen in Figure A.8, it seems that the models really can't keep up with the model minimisation being performed on them, regardless of the pruning amount. The model weight size also are increased by 25% in comparison to only the quantized model. Due to the fact that these models compared to only the quantized model is in need of having extremely high pruning amounts in order to actually decrease the size of the model, these models aren't really seen as useful in comparison to only the pruned models or only the quantized model which holds up across all performance metrics compared to the original model both when looking at the larger PANDORA data set as well as the smaller one.

For the last performance measurement on the models, Table A.9, where the developed models are compared to how similarly they predict the text samples in comparison to the $BERT_{BASE}$ model. These show similar results to what's previously shown, the quantization method holds up well, pruning as well as pruning and quantization seems to also hold up up until the 80% mark for both where the similarity to the original model start to fall off.

As mentioned briefly in the last part of the Chapter 1, there are staggering advances being made in terms of model minimisation and pushing the boundaries of what can be in terms of optimising a models byte-size footprint while still keeping competitive performance in comparison to the original models they are in which they are developed from. However these results are not to be seen as negligible due to that fact. For one, the company isn't using this BERT model anymore and it was more to be seen as a general model in which to investigate methods to be able to use for newer models that Substorm might use in the future, which still makes the results seen here as useful in terms of moving forward for the company. Two, these methods can be used without having any retraining of the model, which was a mandatory due to the fact that the time it would take to fine-tune all of these models from scratch would take way too long. Just one testing simulation of the larger PANDORA data set takes roughly 15h to compute, which means re-training of all the models would take magnitudes longer in time.

Another point of discussion is what this means on other much larger models in terms of cost savings. One model that has been mentioned as of interest for Substorm is to make use of the open source models that Facebook been launching out into the world the Llama 2 transformer models, which is free for both personal and commercial use. They come in variable amounts of [7, 13, 70]B parameters as well as model footprints of [13.5, 25, 129]GB. These supposedly comes in variables specified in 16-bit floating points numbers [49]. Which means at least the quantization method would be highly competitive even here to reduce the size of the model footprint and the pruning method as well might be able to compress size as well depending on how the model holds up at the higher pruning amounts. If it's not only looked into the cost as a size-decreasing variable but also the fact that these types of servers are really costly from an environmental standpoint as well where the larger the model becomes the more absolute space can most likely be decreased (due to the model being bigger, the ratio would be the same) and therefore both money and environmental costs of running the model becomes substantially smaller than before.

If instead of looking at the ways it's possible to decrease the model to be able to achieve competitive accuracy results one can look into why this is really necessary. When is an accuracy loss acceptable when trying to create and use a model that's trying to predict something as complex as bias in text. The model is part of a research project called NoBias which was created to be able to be used as a tool for example companies when they want to create text that are as gender neutral as possible. This can be when sending emails, creating blog posts or creating

any sort of social media statement. This would then in turn generate revenue for Substorm under the assumption that the service would be something that can be paid for by outstanding companies. It's really hard to then saying when delivering a product to a company that maybe not are of interest of the model minimisation part that the model could have been of more effective use, but the company wanted to save money. In reality it's really hard to tell where the threshold goes for an efficient model contra an accessible and cheaper to maintain model, and while these are interesting points of discussion, the thesis was to investigate methods to reduce costs.

The discussion can also be made that the scores achieved from the Loading part of the project would further enhanced by the optimisation part of the project, right now they are dealt as separate parts of the project and aren't interacting with each other. The larger the model size reduction, the more pronounced the reduction in loading times for the model can be, this is something that hasn't been explored in this project but is something that definitely would see an even further decrease in loading time due to the decreased size of the models evaluated.

6
Conclusion

When looking at the research questions, methods for both quickly loading the model as well as minimise the model footprint have been identified and vigorously tested.

6.1 Loading

As seen in the report the new loading sequence has managed to improve the time of loading in the model by a substantial amount. For further work, it can be looked into a faster way to initially get a hold of the model fro mthe storage.

6.2 Model Minimisation

Multiple methods have been identified and implemented to be able to decrease the model size without completely damaging the accuracy result of the model. As stated, some methods are ready to be implemented straight away, but the pruning method and pruned in combination with quantization method need some more work to be able to actually decrease the size and not just state a theoretical value. Overall, for a size-decreased BERT model to maintain accuracy in predicting the gender of certain texts, both the quantization method and the pruning method are considered successful. However, the pruned and quantization method cannot be deemed an overall success due to its high pruning amount required for size reduction, when comparing it to the quantized model. Additionally, the ROC and AUC scores associated with this method are very ambiguous. In terms of economic cost for at least the quantized model it would mean that the model having under half it's original size, would be able to be stored in a smaller space and there for also lowering eventual server costs. For the pruned models this would also be the case for the higher pruned models 70-80% where the 75% pruned model definitely would be the best model in terms of both quality and size decrease. If it's instead looked at through an environmental lens, the decrease in server size would also mean less energy being emitted and a cost in that sense as well. For further work, implementing the size decrease for the pruned as well as the pruned and quantized models is a great first step. One might also look into methods that require retraining of the model where more methods become available.

Bibliography

[1] The overall structure of the bert model. `https://www.researchgate.net/figure/The-overall-structure-of-the-BERT-model_fig1_359301499`. Accessed: 2023-10-30.

[2] Mingyu Zong and Bhaskar Krishnamachari. a survey on gpt-3, 2022.

[3] Jiasi Chen and Xukan Ran. Deep learning with edge computing: A review. *Proceedings of the IEEE*, 107(8):1655–1674, 2019.

[4] Etienne Mueller, Julius Hansjakob, Daniel Auge, and Alois Knoll. Minimizing inference time: Optimization methods for converted deep spiking neural networks. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021.

[5] Ziheng Wang. Sparsednn: Fast sparse deep learning inference on cpus. *CoRR*, abs/2101.07948, 2021.

[6] Parminder Singh, Pooja Gupta, Kiran Jyoti, and Anand Nayyar. Research on auto-scaling of web applications in cloud: Survey, trends and future directions. *Scalable Computing: Practice and Experience*, 20:399–432, 05 2019.

[7] Raffaele Pugliese, Stefano Regondi, and Riccardo Marini. Machine learning-based approach: global trends, research directions, and regulatory standpoints. *Data Science and Management*, 4, 2021.

[8] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

[9] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding, 2020.

[10] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8bert: Quantized 8bit BERT. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*. IEEE, dec 2019.

[11] Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jing Jin, Xin Jiang, Qun Liu, Michael Lyu, and Irwin King. Binarybert: Pushing the limit of bert quantization, 2021.

[12] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. I-bert: Integer-only bert quantization, 2021.

[13] Eldar Kurtic, Daniel Campos, Tuan Nguyen, Elias Frantar, Mark Kurtz, Benjamin Fineran, Michael Goin, and Dan Alistarh. The optimal bert surgeon: Scalable and accurate second-order pruning for large language models, 2022.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[15] Bernhard Mehlig. Artificial neural networks. *CoRR*, abs/1901.05639, 2019.

[16] Alphago. https://deepmind.google/technologies/alphago/. Accessed: 2023-10-30.

[17] Alphazero: Shedding new light on chess, shogi, and go. https://deepmind.google/discover/blog/alphazero-shedding-new-light-on-chess-shogi-and-go/. Accessed: 2023-10-30.

[18] Relu. https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html. Accessed: 2023-10-18.

[19] Brent Scarff. Understanind backpropagation. https://towardsdatascience.com/understanding-backpropagation-abcc509ca9d0. Accessed: 2023-09-18.

[20] Daksh Trehan. Gradient descent explained. https://towardsdatascience.com/gradient-descent-explained-9b953fc0d2c. Accessed: 2023-09-18.

[21] Nikita Moriakov Jonas Teuwen. *Handbook of Medical Image Computing and Computer Assisted Intervention*. Academic Press, 2020.

[22] The unreasonable effectiveness of recurrent neural networks. https://karpathy.github.io/2015/05/21/rnn-effectiveness/. Accessed: 2023-10-30.

[23] Rnn image. https://upload.wikimedia.org/wikipedia/commons/thumb/b/b5/Recurrent_neural_network_unfold.svg/800px-Recurrent_neural_network_unfold.svg.png?20170619114714. Accessed: 2023-10-30.

[24] Jürgen Schmidhuber Sepp Hochreiter.

[25] Lstm image. https://thorirmar.com/post/insight_into_lstm/. Accessed: 2023-10-30.

[26] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[27] OpenAI. Gpt-4 technical report, 2023.

[28] Dall · e: Creating images from text. https://openai.com/research/dall-e. Accessed: 2023-09-12.

[29] Dall · e 2. https://openai.com/dall-e-2. Accessed: 2023-09-12.

[30] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation, 2021.

[31] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[32] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[34] Cat image. `https://hips.hearstapps.com/hmg-prod/images/russian-blue-royalty-free-image-1658451809.jpg`. Accessed: 2023-10-30.

[35] Ketan Doshi. Transformers explained visually (part 1): Overview of functionality. `https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452`, 2020. Accessed: 2023-09-13.

[36] Explanation of bert model – nlp. `https://www.geeksforgeeks.org/explanation-of-bert-model-nlp/`. Acessed: 2023-09-18.

[37] Quantization. https://pytorch.org/docs/stable/quantization.html. Accessed: 2023-09-28.

[38] Accuracy. `https://developers.google.com/machine-learning/crash-course/classification/accuracy`. Accessed: 2023-10-27.

[39] David M. W. Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation, 2020.

[40] F1 score. `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html`. Accessed: 2023-10-27.

[41] Confusion matrix. `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html`. Accessed: 2023-10-30.

[42] Classification: Roc curve and auc. `https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc`. Accessed: 2023-10-27.

[43] Fred Reiss. How to load pytorch models 340 times faster with ray. `https://medium.com/ibm-data-ai/how-to-load-pytorch-models-340-times-faster-with-ray-8be751a6944c`.

[44] Softmax. `https://pytorch.org/docs/stable/generated/torch.nn.Softmax.html`. Accessed: 2023-10-18.

[45] Crossentropyloss. `https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html`. Accessed: 2023-10-18.

[46] Adam. `https://pytorch.org/docs/stable/generated/torch.optim.Adam.html`. Accessed: 2023-10-18.

[47] Quantize dynamic. https://pytorch.org/docs/stable/generated/torch.ao.quantization.quantize_dynamic.html. Accessed: 2023-10-18.

[48] Pruning tutorial. https://pytorch.org/tutorials/intermediate/pruning_tutorial.html. Accessed: 2023-09-28.

[49] Llama2. https://huggingface.co/docs/transformers/model_doc/llama2. Accessed: 2023-10-25.

A

Appendix 1

Her are the compiled performance metrics for the simulations done on the much smaller test data not previously seen by the network. The data set contains 500 samples instead of the much larger data set used in Chapter 4.2.2, and is also part of the PANDORA data set previously used. The data has 252 samples being texts written by males, 248 texts written by females. The simulations follow the exact same structure as the work done in Chapter 4.2.2 so details won't be given as to what is looked at only the results in their raw form.

Base model

Table A.1: Performance metrics calculated on the $BERT_{base}$ network tested on the smaller PANDORA test data set

Model:	Accuracy [%]	Precision [%]	Recall [%]	F1 score [%]	Size [MB]
$BERT_{base}$	80.00	76.43	86.29	81.06	438.00417

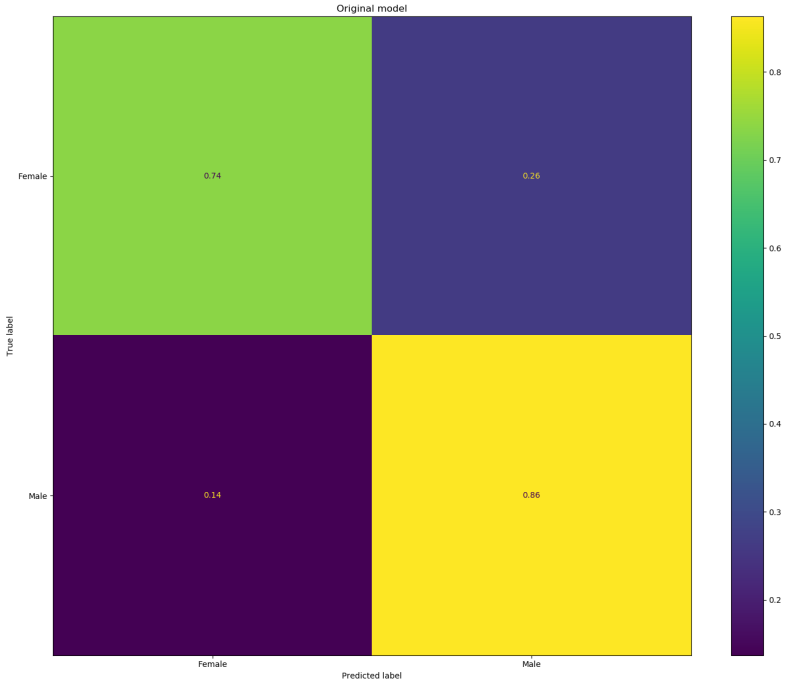


Figure A.1: Confusion matrix for the original $BERT_{base}$ network

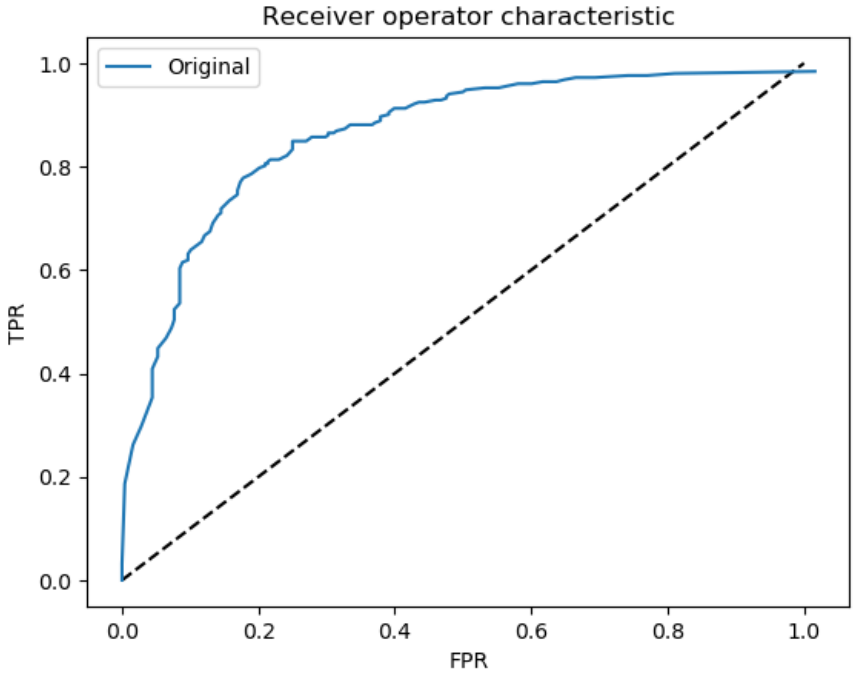


Figure A.2: ROC curve for $BERT_{base}$ network. The diagonal dashed line is to symbolise a network that is completely guessing and or in our case, only guesses one of the labels since it's a almost completely balanced data set

Table A.2: Area Under Curve for the ROC curve that can be seen in figure 4.9

Model:	AUC
$BERT_{base}$	0.87856

Quantization

Table A.3: Performance metrics calculated on the quantized BERT network tested on the complete test data set

Model:	Accuracy [%]	Precision [%]	Recall [%]	F1 score [%]	Size [MB]
BERT quant	80.00	76.24	86.69	81.13	181.487013

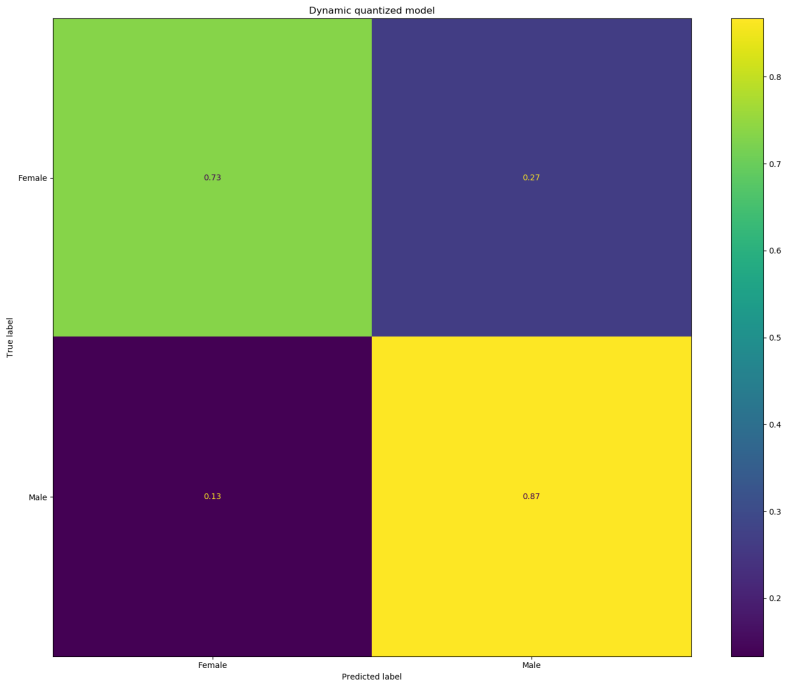


Figure A.3: Confusion matrix for the quantized BERT network

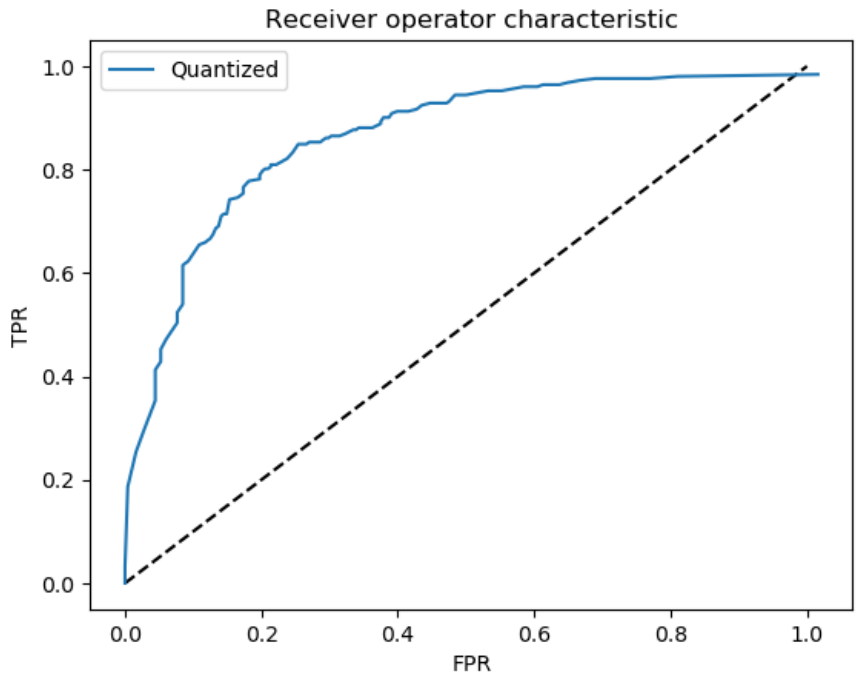


Figure A.4: ROC curve for quantized $BERT_{base}$ network. The diagonal dashed line is to symbolise a network that is completely guessing and or in our case, only guesses one of the labels since it's a so close to a balanced data set

Table A.4: Area Under Curve for the ROC curve that can be seen in figure 4.11

Model:	AUC
BERT quant	0.87888

Pruning

Table A.5: Performance metrics for pruned BERT models

Prune amount:	Accuracy [%]	Precision [%]	Recall [%]	F1 [%]
60%	78.60	86.15	67.74	75.85
65%	78.00	87.10	65.32	87.01
70%	80.60	78.71	83.47	81.02
75%	77.00	82.44	68.15	74.61
80%	74.80	87.65	57.26	69.27
85%	50.60	100	0.40	0.80
90%	50.40	0	0	0
95%	50.40	0	0	0

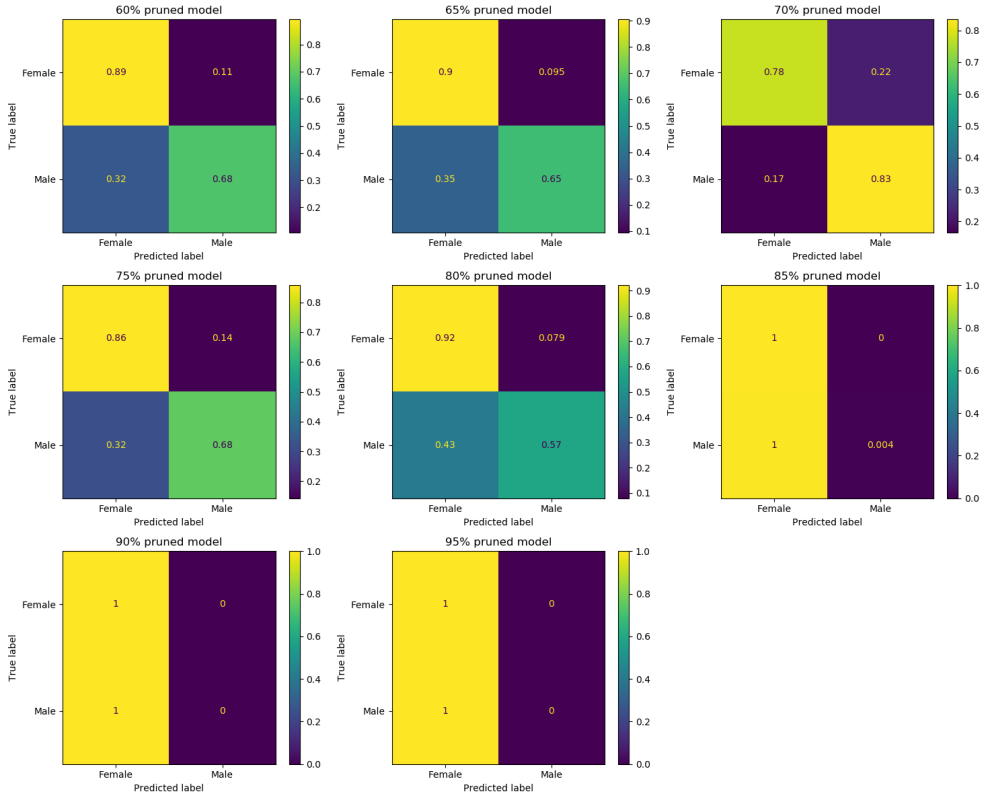


Figure A.5: Confusion matrices for all pruned models evaluated on the smaller PANDORA test set not previously seen by the model

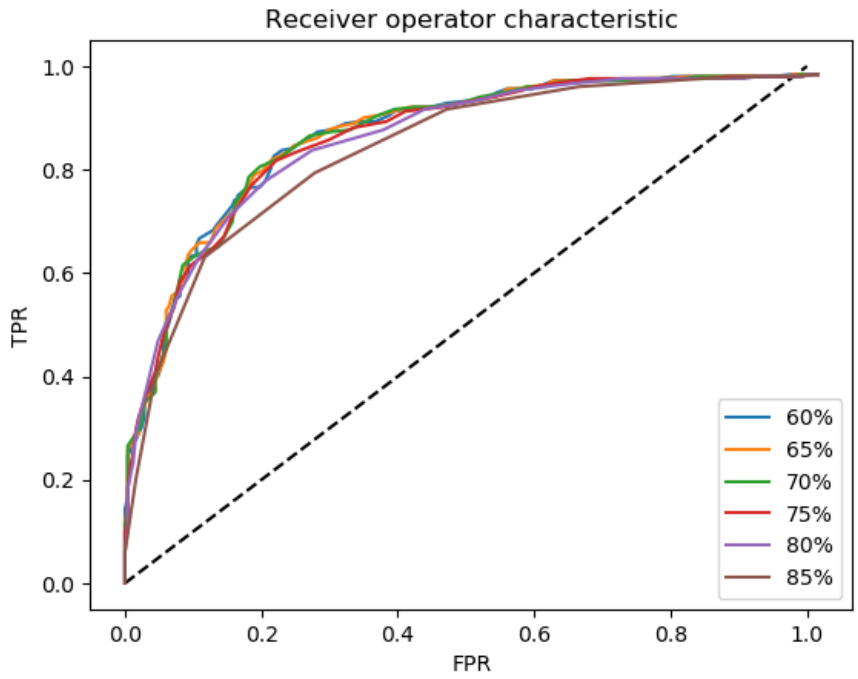


Figure A.6: Receiver operating characteristics for pruned model variants. The different pruning amounts are denoted by it's percentage value in the labels in the figure, ranging from 60%-85% in steps of 5%. The diagonal dashed line is to symbolise a network that is completely guessing and or in our case, only guesses one of the labels since it's a balanced data set

Table A.6: AUC scores for different pruning values of the BERT model

Prune amount:	AUC score
60	0.88149
65	0.88196
70	0.88030
75	0.87675
80	0.87290
85	0.85162

Pruning and quantization

Table A.7: Performance metrics for pruned and quantized BERT models

Model:	Accuracy [%]	Precision [%]	Recall [%]	F1 [%]
60%	78.60	86.15	67.74	75.85
65%	78.00	87.10	65.32	74.65
70%	80.60	78.71	83.47	81.02
75%	77.00	82.44	68.15	74.80
80%	74.80	87.65	57.26	69.27
85%	50.60	4.03	0	0
90%	50.40	0	0	0
95%	50.40	0	0	0

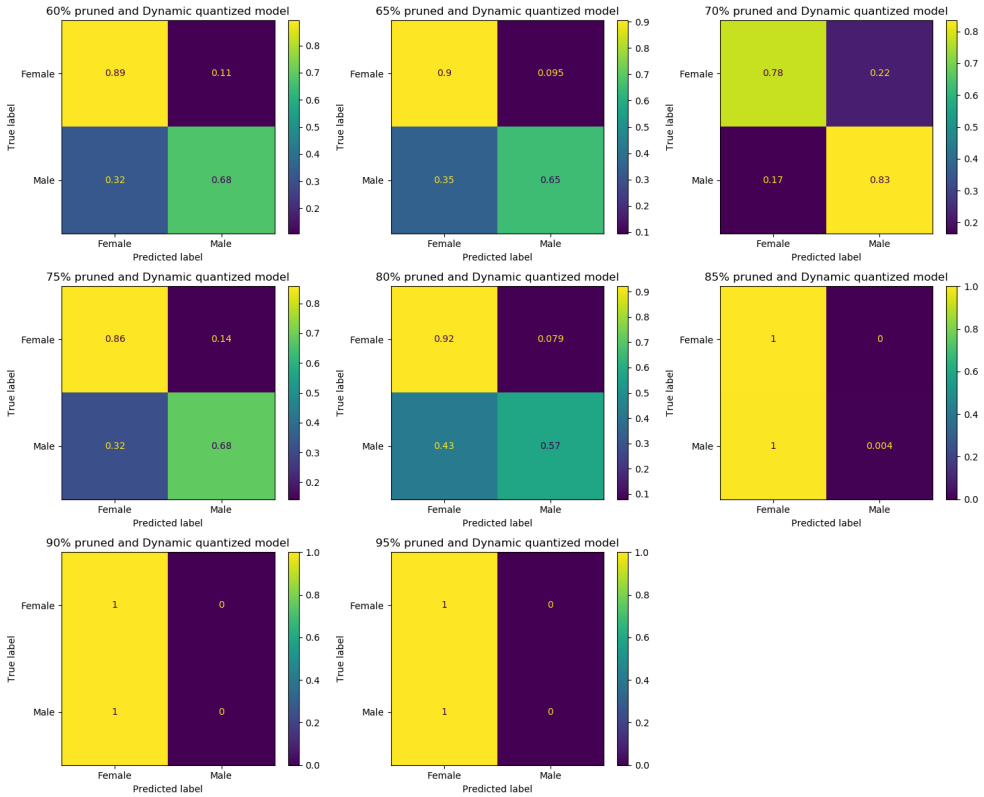


Figure A.7: Confusion matrices for all pruned and quantized models evaluated on the smaller PANDORA test set not previously seen by the model

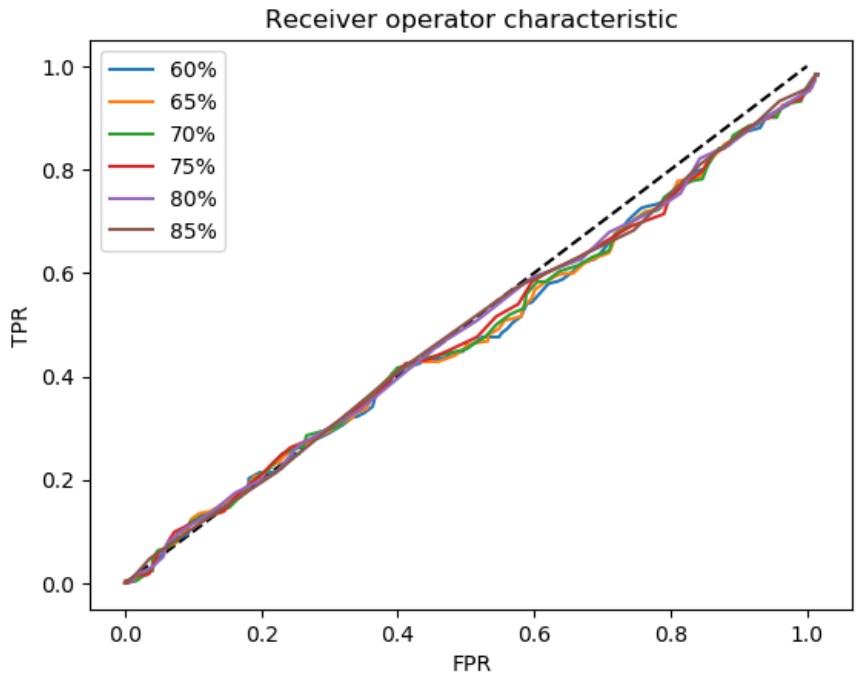


Figure A.8: Receiver operating characteristic for quant and pruned model variants. The diagonal dashed line is to symbolise a network that is completely guessing and or in our case, only guesses one of the labels since it's a so close to a balanced data set

Table A.8: AUC score for different pruning values of a quantized BERT model

Prune amount:	AUC score
60%	0.49017
65%	0.49213
70%	0.49403
75%	0.49670
80%	0.50033
85%	0.50164

Comparison to $BERT_{base}$ model prediction

Table A.9: Final comparison between the developed models and $BERT_{BASE}$. The comparison is made on another data set in Substorm's possession which also is gender labeled. The test is done on the first 1000 text samples in the data set and what's shown is how similar the models predict versus the $BERT_{BASE}$ model.

Model:	Prediction similarity [%]
quant	99.4
60% pruned	85.1
65% pruned	82.8
70% pruned	91.1
75% pruned	84.2
80% pruned	73.9
85% pruned	45.0
90% pruned	38.7
95% pruned	38.7
60% pruned and quant	84.7
65% pruned and quant	82.5
70% pruned and quant	91.0
75% pruned and quant	83.9
80% pruned and quant	73.5
85% pruned and quant	44.9
90% pruned and quant	38.7
95% pruned and quant	38.7

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY