

Efficient Algorithms for Artificial Neural Networks and Explainable AI

Hassan Gharib Eshkiki

919165

Submitted to Swansea University in partial fulfilment
of the requirements for the Degree of Doctor of Philosophy



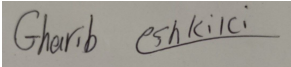
Swansea University
Prifysgol Abertawe

Department of Computer Science
Swansea University

20th July 2023

Declaration

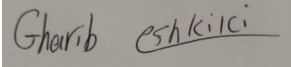
This work has not been previously accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed  (candidate)

Date 01/06/2023

Statement 1

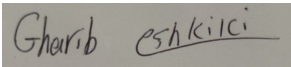
This work is the result of my own independent study/investigations, except where otherwise stated. Other sources are clearly acknowledged by giving explicit references. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure of this work and the degree examination as a whole.

Signed  (candidate)

Date 01/06/2023

Statement 2

I hereby give my consent for my work, if accepted, to be archived and available for reference use, and for the title and summary to be made available to outside organisations.

Signed  (candidate)

Date 01/06/2023

'A school bell that sounds annoying at 08:00am, sounds exciting at 04:00pm... It is a matter of time; you will get there.'

Osunsakin Adewale . . .

Abstract

Artificial neural networks have allowed some remarkable progress in fields such as pattern recognition and computer vision. However, the increasing complexity of artificial neural networks presents a challenge for efficient computation. In this thesis, we first introduce a novel matrix multiplication method to reduce the complexity of artificial neural networks, where we demonstrate its suitability to compress fully connected layers of artificial neural networks. Our method outperforms other state-of-the-art methods when tested on standard publicly available datasets.

This thesis then focuses on Explainable AI, which can be critical in fields like finance and medicine, as it can provide explanations for some decisions taken by sub-symbolic AI models behaving like a black box such as Artificial neural networks and transformation-based learning approaches. We have also developed a new framework that facilitates the use of Explainable AI with tabular datasets. Our new framework Exmed, enables non-expert users to prepare data, train models, and apply Explainable AI techniques effectively. Additionally, we propose a new algorithm that identifies the overall influence of input features and minimises the perturbations that alter the decision taken by a given model.

Overall, this thesis introduces innovative and comprehensive techniques to enhance the efficiency of fully connected layers in artificial neural networks and provide a new approach to explain their decisions. These methods have significant practical applications in various fields, including portable medical devices.

Acknowledgements

I would like to take this opportunity to express my gratitude to those who have assisted me in various ways. Over the past five years, from my Masters to my Ph.D, my supervisor Dr Benjamin Mora has supported me in every challenge. I am thankful for the effort and extra time he has contributed to my academic development, and I hope that I have met his expectations.

Another person I would like to thank is Professor Xiangua Xie for all his assistance and support during this Ph.D.

I am also very grateful to Dr A. Rahat and Dr A. Pauly for their suggestions on how to make the best decisions.

Last but not least, I want to express my gratitude to my friends in the Computational Foundry, including Dr Oleg Zaikin, Dr Indirajith Ananth, Dr M. White, Mr D. Greco, Mrs K. Pugh, Mr S. Oliver, and all the other wonderful individuals whom I have met there.

Contents

List of Tables	vi
List of Figures	viii
1 Introduction	1
1.1 Motivations	1
1.2 Overview	4
1.3 Publications and significant contributions arising from the work detailed in this thesis	4
1.4 Outline	6
2 Background	7
2.1 Introduction	7
2.2 Neural Network Models and their Compression and Acceleration	8
2.3 Explainable AI and Local Model-Independent Methods	16
3 Efficient Fully Connected Layers in ANN	23
3.1 Preliminaries	23
3.2 Related Work	24
3.3 Mediterranean Matrix Multiplication	32
3.4 Compressing Fully Connected Layers	43
3.5 Results	47
3.6 Conclusion	56
4 Explainable AI	59
4.1 Introduction	59

4.2 Literature review	60
4.3 ExMed Workflow	66
4.4 The Neighbour Migrating Generator Model	74
4.5 Conclusion	89
5 Conclusions	93
6 Future Work	97
Bibliography	99
Appendices	116
A Implementation of a Relevant Algorithm	117
B Supplementary Data	119

List of Tables

2.1 Comparing CART and ID3 for building a decision tree.	18
3.1 Reduced upper bounds for exact matrix multiplication algorithms.	28
3.2 Complexity bounds for rectangular matrix multiplication algorithms.	28
3.3 Comparison of the different practical factors influencing the performance of the signed matrix approximation [1] and our own Hamming distance kernel. (GPU: NVIDIA 1080Ti)	52
3.4 Comparison of the best compression rates with Binary-Connect [2] and XNOR-Net [3]. Our technique allows changing the compression rate to favour either compression or quality.	55
4.1 Non-pharmaceutical COVID Control Measures.	69

4.2	Prediction performance on the COVID dataset with four different classifiers with K-Fold Cross Validation ($k = 10$).	70
4.3	Each patient is described with 20 features.	73
4.4	Survival Time Feature Value Count	73
4.5	Prediction performance on the Lung Cancer dataset with four different classifiers with K-Fold Cross Validation ($k = 10$).	73
4.6	A description of the architecture of the models used for the three datasets. The F model is the original model that we want to study, while G is the NMG model used to migrate to decision boundaries.	82
4.7	Description of features in the Pima diabetes datasets.	84
4.8	Variation among features when changing instances. For each original class, our global NMG model returns the closest class available (left two columns). The results match what can be witnessed on Fig. 4.16. For the first two migrations, results show that the petal length is the only important feature to migrate class. For the Setosa class, all parameters have some influence, but the petals' length and width are still the most contributing factors.	84
4.9	Description of features in the Thyroid-disease datasets.	88
4.10	Description of features in the Thyroid diabetes datasets.	89
4.11	Variation among features when changing instances (local variant of the algorithm) for the Pima Indian Diabetes Dataset for 6 randomly selected samples. Units and standard ranges for glucose, blood pressure, insulin and Body Mass Index are given at the top, and samples outside the range are highlighted in red. The second value in cells, when present, indicates the variation generated by the migration model on some of the features. Unchanged values are not shown.	91
B.1	(a) MNIST without Data Augmentation	119
B.2	(b) MNIST without Data Augmentation	120
B.3	(c) MNIST with Data Augmentation	120
B.4	(d) MNIST with Data Augmentation	121
B.5	(e) CIFAR10	121
B.6	(f) CIFAR100	122
B.7	(g) CINIC10	122
B.8	Some additional results to table 4.11 in chapter 4	129

B.9	The table illustrates the weight vector impact on model G in the NMG. We run model G with three different values in the weight factor, non-frozen input ($\omega = 1$), freezing Age feature (cell correlated to age in ω equals 99), and Age and Sex are frozen. Each row shows either the input sample (top rows) or the output of model G.	130
-----	--	-----

List of Figures

2.1	Mathematical model of a neuron by McCulloch and Pitts. X_i refers to inputs, and W_i shows the strength of the neuron connection. θ is threshold for h (sum of the weights) to active output	9
2.2	The figure displays a Feed forward neural networks that receives a vector X with three dimensions as input. Each input dimension and the neurons in the fully connected layers are connected in completely connected layers.	10
2.3	This figure contains four activation features. An activation function will be given some inputs and neuronal weights, and it will decide how to respond based on the formula they contain.	11
2.4	Plotting a decision tree can show what feature has been selected and the criteria that lead a sample to leaves.	19
2.5	The red line indicates the boundary between two classes in a binary classification problem for a two-dimensional space. An example is depicted with a black circle enclosed by a larger blue circle. The blue area represents how far new data points can extend. The size of the area explained by a different linear model can vary.	20
3.1	Sub-matrices in strassen's algorithm	25
3.2	An illustration of Monte Carlo sampling of an angle between two angles which accumulates results from random tests (Algorithm 1). Left: Both points are on the same side of the hyperplane and test will return 0. Right: The hyperplane is separating the two points and the test will return +1.	34

3.3	Pipeline of operations for estimating the product $Y \approx WX$. Operations needed during inference are shown in blue. Note that the entries of the constant matrix E follow a normal distribution and do not need to be stored.	45
3.4	Comparison of error rates between M^3 method and [1] in approximating matrix AB , by varying the matrix sizes with $p = n$ or fixing n and changing the number of samples used. The chart also displays the final error rates of both methods after varying the μ parameter of the Gaussian random number generator, with $\sigma = 1$	48
3.5	Performance comparison between our method, $ASS^T B$ [1] (b, c & d), and a direct matrix multiplication of A and B (e & f). As we mentioned, to achieve the same level of error, M^3 necessitates about 2.46 times as many samples. . .	50
3.6	Breakdown of performance for the various kernels used in our GPU implementation. It shows the percentage of operations required by each group in separate columns.	51
3.7	Effect of replacing the standard matrix multiplication with the M^3 version for training the MNIST datasets in all but the last Fully Connected layer. The horizontal represents the number of epochs while the vertical axis represents the accuracy obtained on the testing dataset. We study forward-only replacement cases. The number of planes k used is the same for each layer and each pass in a single experiment (batch size used is 1024). Getting good results in the forward pass with M^3 requires a much larger number of planes.	53
3.8	Convergence of accuracy according to the number of hyperplanes used. The horizontal axis displays the compression obtained for just the internal Fully Connected layers (a) and for the neural network as a whole (b). The vertical axis displays the obtained accuracy of the network relative to that of the original, unmodified network – with 100% meaning that the compressed network has the same accuracy as the original one. All network models have been tested with 256, 512, 1024 and 2048 hyperplanes as represented with continuous or dashed lines.	54
4.1	ExMed Activities. ExMed provides the user with a sequence of simple actions, including loading, merging, and editing data, and creating prediction as well as explanation models. Various visualisation techniques are supported in several stages of this pipeline.	63

4.2	Working with biology photos is simple and quick using Fiji's many plugins. A biological image with a filter applied to it is shown in the figure.	64
4.3	Massive Online Analysis is a framework for running machine learning algorithms on data series databases.	65
4.4	The University of Waikato in New Zealand created the Waikato Environment for Knowledge Analysis (Weka) since it has a free software license.	65
4.5	ExMed interface for some of the main activities as describe in Fig. 4.1. Data from various supported file types is loaded, with the option to combine this data with other datasets.	67
4.6	Example of an Explanation computed with SHAP and LIME. For this instance, both explainers consider top measures contributing to this prediction being <i>Domestic Travel, Cafes and Restaurants Closure</i> and <i>Pubs and Bars Closure</i>	70
4.7	Global explanations generated using SHAP on our COVID dataset for the prediction whether $R_t \geq 1$. We see that closing down cafes and restaurants as well as pubs and bars are the most effective control measures. When their feature values are high (red), they have string negative impact to the prediction; whereas when their feature values are low (blue), they have strong positive impact to the prediction.	71
4.8	Local explanation on the Lung Cancer life expectancy data set for a patient instance. We see that the most impactful features amongst SHAP and LIME are the same: " <i>Grade</i> " <i>How the cancer cells act; the higher the grade the less normality the cell resembles, and it may act more aggressive</i> and " <i>M Best</i> " <i>Presence or Absence of Distant Metastatic Spread</i> , followed by a disagreement on age attribution.	74
4.9	We see that the largest impact towards the survival boundaries <i>greater than 1 year</i> and <i>less than 6 months</i> is the cancer grading - having direct impact on the longest and least time survived. This, followed by an associative relationship between height, weight, and the patient age determinants of body mass index (BMI), having high attribution towards each class. This, then followed by cancer specific traits such as " <i>M Best</i> " and laterality of the tumour.	75

4.10	Global explanation for feature attribution measured against the class <i>Survival time of less than 6 months</i> , where we see the cancer grade of higher value - indicative of cell abnormality and more aggressive, followed by "M Best" <i>Presence or Absence of Distant Metastatic Spread</i> , with the associative BMI attributes "height", "age" and "weight" following this.	76
4.11	Global explanation for feature attribution measured against the class <i>Survival time of greater than 12 months</i> , we see an inverse plot of cancer grade to that shown in Fig.4.3.2 (a), such that a lower grade and what seems to be a better controlled BMI and a lower "M Best" contributing to a longer survival time.	77
4.12	Global explanation for feature attribution measured against the class <i>Survival time between 6 and 12 months</i> , we see that a controlled BMI and lower cancer grade are attributive to this survival boundary, whilst the distributive "M Best", performance and cancer grade containing high values in both positive and negative impacts on the model are likely the reason for the central survival boundary.	78
4.13	The calculation of the Feature Importance weights I provides an explanation of the global impact of features for the different classes of the original model.	79
4.14	Loss function of Neighbour Migrating Generator	81
4.15	We demonstrate the effectiveness of our NMG model G by applying it to some random samples from an implicit 2D heart function dataset [4]. The dataset consists of blue dots representing class 0 samples and red dots representing class 1 samples. We also select some green dots at random from the dataset to transform using our NM Generator. The results, shown in Figure X, demonstrate that our NMG model is able to satisfactorily migrate samples to the decision boundary.	83
4.16	2D plotting the IRIS dataset. The graph indicates that using only two features (Petal Length and Petal Width) out of four is enough to separate most samples of this dataset. Note that Iris Versi color appears to be in between the two other classes.	86
4.17	shows the relationship between each type of species, and features are displayed.	87
4.18	2D plotting the Thyroid dataset. The graph indicates that using only two features ($TT4$ and TSH) out of nine is enough to separate most samples of this dataset.	88

B.1	Effect of replacing the standard matrix multiplication with the M^3 version for training the MNIST datasets in all but the last fully connected layer. The horizontal represents the number of epochs while the vertical axis represents the accuracy obtained on the testing dataset. We study backward-only, forward-only and backward-forward replacement cases. The number of planes k used is the same for each layer and each pass in a single experiment. The batch size used is 1024. The M^3 seems to be beneficial to back-propagation that shows a convergence similar to the use of a standard matrix multiplication with a sufficiently low number of planes.	123
B.2	Convergence of accuracy according to the number of hyperplanes used. The horizontal axis displays the compression obtained for just the internal fully-connected layers (a) and for the neural network as a whole (b). The vertical axis displays the obtained accuracy of the network relative to that of the original, unmodified network – with 100% meaning that the compressed network has the same accuracy as the original one. All network models have been tested with 256, 512, 1024 and 2048 hyperplanes as represented with continuous or dashed lines.	124
B.3	Error comparison between our method and [1]. Results are given by either varying the sizes of matrix A and B with the number of samples p equal to n ($n = p$) or by fixing n and varying the number of samples used for the approximation.	125
B.4	Data exploration tools in ExMed. (a) is the Data Editor that supports standard data editing functions. (b) and (c) are the Data Visualiser that supports different plot types such as Line, Scatter, Bar, Histogram, Violin Plots and Pie chart. For each plot type, various customisation options are implemented, including changing the axes, layout, and adding texts.	126
B.5	ExMed interface for some of the main activities as describe in Fig. 4.1. (b) Data is optionally pre-processed with some of the plugins available. (c) A model is created, with the option of reducing the number of features with a PCA algorithm and explanation generation with SHAP and LIME.	127

B.6 ExMed Operation Overview. This figure shows the flow of ExMed operations, along with the key features available in the interface. Once the "Application" is running, the first window "Data Dashboard" is shown. Black arrows denote event-driven actions that take the user to the next window in chronological order. Colours highlight the Key features for each window, along with a short description provided for each feature. The indentation of boxes represents a dependency between windows. For instance, the 'Feature Dashboard' window can lead to the 'Edit Table' window, which subsequently can open the 'Plot Viewer' window. The dotted line represents a database extension that is to be added in the future. 128

Chapter 1

Introduction

Artificial Intelligence (AI) can be broadly categorised into two primary branches: symbolic and sub-symbolic. Symbolic AI methods are categorised by the utilisation of explicit symbolic methods, such as formal methods and programming languages, and are commonly employed for deductive knowledge [5]. This branch of AI is typically associated with knowledge bases and expert systems [6]. Due to their reduced dependence on input data, symbolic techniques are better suited for abstract problems [7].

Sub-symbolic artificial intelligence establishes complex correlations between input and output variables. These relationships are often formalised through functions that map input data to an output. Sub-symbolic AI includes a range of learning methods, such as artificial neural networks algorithms. These methods are particularly well-suited for addressing complex problems and require less prior knowledge compared to Symbolic AI [7].

1.1 Motivations

Artificial Neural Networks (ANN) revolutionised the way we gather information and extract insights from data over the last few decades [8]. Artificial neural networks have proven to be incredibly effective across a wide range of applications, from computer vision [9] to time series analysis [10]. Thanks to improvements in computational power and the availability of increasingly sophisticated algorithms, ANN models have become more accurate and more efficient than ever before [11].

However, while the benefits of ANN are clear, there are also challenges that come with deploying these models. One major concern is the resources required to run a model

efficiently. With many devices having limited processing power and storage space, it is crucial to develop models that can operate effectively even under these constraints. Many approaches have been explored to make models more appropriate for those devices such as pruning [12] (reducing complexity by removing unnecessary parameters) or compressing models to achieve comparable accuracy with fewer parameters in mind [13].

Symbolic AI methods have the ability to explain and reason about their conclusions, and even their intermediate steps are often interpretable. This makes them easier to debug, explain, and control. For instance, Rule-based systems have the advantage of rule modularity, where rules can easily be added or removed from a knowledge base. They also provide knowledge interoperability, allowing for knowledge transfer between closely related applications [14].

However, with sub-symbolic AI, the lack of explanations in most models is an important limitation. In critical areas like healthcare, where decisions based on a model can have life-or-death consequences, it is essential to be able to explain how and why the model took a particular decision.

Explainable AI (ExAI) becomes useful when dealing with a black-box model. It tries to explain the behaviour of sub-symbolic AI models by providing tools [15] and techniques [16].

In this thesis, we tackle both of these challenges head-on. One primary goal is to develop a more efficient method for building ANN models on low-resource devices. To achieve this, we propose a new matrix multiplication algorithm based on Monte-Carlo sampling to estimate angles between vectors, which enables faster and simpler calculations. We also introduce a novel compression method for Fully Connected Layers, which reduces the number of parameters required while maintaining accuracy.

Therefore, we not only aim to make ANN models more efficient, but we also want to explain their behaviour. To that end, we provide a tool to use of Explainable AI in medical domains, where trust is especially critical. We also present a new technique for explaining an ANN model and its input space, which can help to explain how the model reached its conclusion. In a nutshell, this research aims to extend the boundaries of artificial neural networks, making these powerful tools more accessible, efficient, and trustworthy for everyone.

1.1.1 Compressing Artificial Neural Networks Model

The performance of most machine learning methods including artificial neural networks in various fields, from speech recognition to image segmentation, is widely recognised [17, 8, 18]. This success is largely due to their ability to create deep models with a high degree of complexity, a feat made possible by the impressive advancements in GPU computational speed. With these developments, models with millions of parameters can be trained on large datasets, such as AlexNet [19] with 61MB and VGG16 [20] with 128MB.

Nevertheless, many systems with constrained resources still find it difficult to manage these huge models, despite the impressive advancements achieved in computational power. Additionally, portable devices require the use of batteries power, which creates other difficulties. As a result, there is an increasing need to develop methods for reducing the size of neural network models so that they can be used on devices with limited resources.

In order to meet this need, this study offers a cutting-edge methods for compressing fully connected layers of artificial neural networks models. To be more precise, we present a novel matrix multiplication technique that enables us to reduce the quantity and complexity of operations that a model needs to perform. This method offers an efficient solution to the difficulties linked to deploying Artificial neural networks models on resource-limited devices, with the potential to greatly increase the speed and also improve the energy usage of portable devices. The specifics of this strategy are covered in Chapter 3.

1.1.2 Explainable AI

Artificial neural networks have brought a paradigm shift in the field of artificial intelligence. However, the inscrutability of ANNs in describing their decision-making processes has become a major impediment to their widespread adoption [21]. This lack of transparency is particularly relevant in real-world applications such as medical diagnosis, investment recommendation, and process optimization, where users demand increased transparency and accountability.

To address this issue, Explainable AI (ExAI) methods have garnered considerable attention from both academia and industry [22]. However, there are other existing methods for explaining models as white-box such as Fuzzy Logic and Bayesian Learning [23]. Explainable Artificial Intelligence (XAI) is beneficial to explain black-box models. ExAI originated from a project by the Defence Advanced Research Projects Agency (DARPA) [24].

In Chapter 4, we attempt to leverage the use of ExAI tools and techniques for artificial neural networks. This chapter presents a user-friendly framework for training and explaining models, enabling faster and more intuitive decision-making processes. The approach involves preparing data from expert systems for non-expert users and subsequently applying ExAI methods to enhance model explainability. Moreover, we propose a novel method for Explainable AI that offers great flexibility and versatility. In the same chapter, we introduce the Neighbour Migrating Generator, which leverages both global and local mechanisms to explain distinct aspects of a model, offering varying degrees of explainability power.

1.2 Overview

Chapter 3 introduces a novel technique for estimating matrix multiplication, which is a crucial step in compressing the Fully Connected Layers of ANNs. By using XOR and POP-COUNT operators, the proposed approach can perform most inference operations on binary streams while maintaining a tuneable degree of error tolerance. This method has been shown to perform as well as or better than previous approaches.

Chapter 4 offers a novel framework for Explainable AI (ExAI) in the medical database field, which is designed to be user-friendly and easy tool. The framework includes data preparation, analysis, and model training, as well as classic ExAI methods for trained models. The chapter also introduces the Neighbour Migrating Generator, a mechanism that enhances explainability by incorporating both global and local mechanisms to provide varying degrees of explainability power.

Subsequently, chapter 5 presents the conclusion of the contributions and work described in this thesis. The chapter 6 outlines potential enhancements and plans for extending the work discussed herein.

1.3 Publications and significant contributions arising from the work detailed in this thesis

- **Title:** Fully Connected Networks on a diet with the Mediterranean Matrix Multiplication
Published at: IEEE Transactions on Neural Networks and Learning Systems
Authors: Hassan Eshkiki; Benjamin Mora; Xianghua Xie

Abstract: The Mediterranean Matrix Multiplication is a newly proposed randomised algorithm that aims to approximate matrix multiplication in a simple and practical manner. The algorithm samples angles between the rows and columns of two matrices with sizes (m, n, p) to achieve this goal. The number of steps required for this approximation is $O(k(mn + np + mp))$, where k is a constant determined by the desired precision. The algorithm is mostly based on bitwise operators, making it suitable for a simplified processing architecture and compressed matrix weights. The study shows that the Mediterranean Matrix Multiplication outperforms the standard approximation using signed matrices in terms of size and number of operations. Additionally, the algorithm can compress fully connected layer weights by $30\times$ to $100\times$ with minimal impact on inference accuracy. This paper demonstrates the first application to ANN inference by showing that weights of Fully Connected Layers can be compressed between $30\times$ and $100\times$ with little to no loss in inference accuracy. The requirements for pure floating-point operations are also down as Mediterranean matrix multiplication relies mainly on simpler bitwise operators.

- **Title:** ExMed: An AI Tool for Experimenting Explainable AI Techniques on Medical Data Analytics
Published at: 2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)
Authors: Marcin Kapcia; Hassan Eshkiki; Jamie Duell; Xiuyi Fan; Shangming Zhou; Benjamin Mora
Abstract: The paper introduces ExMed, a tool that allows domain experts to perform ExAI data analytics without the need for programming skills. It includes various feature attribution algorithms to explain machine learning classifications and regressions. The tool's effectiveness is demonstrated through two medical case studies - one analysing COVID-19 control measures and the other estimating lung cancer patient life expectancy using the Simulacrum health dataset. The study concludes that ExMed is a flexible and transferable tool that provides deep insights to researchers and domain experts in medical sub-domains.
- **Title:** Neighbour Migrating Generator: Finding the closest possible neighbour with different classes
Accepted at: Study of Artificial Intelligence and Simulation of Behaviour (AISB) 2023

Authors: Hassan Eshkiki, Benjamin Mora.

Abstract: This paper presents the Neighbour Migrating Generator, a novel approach for identifying the nearest neighbour with a distinct class label. In essence, this method aims to locate decision boundaries that require minimal effort to transform a given input. The requirements for pure floating-point operations are also down as Mediterranean matrix multiplication relies mainly on simpler bitwise operators.

1.4 Outline

The remaining chapters of this work are outlined as follows:

- **Chapter 2:** Background
- **Chapter 3:** Efficient Fully Connected Layers in ANN
- **Chapter 4:** Explainable-AI
- **Chapter 5:** Conclusion
- **Chapter 6:** Future work

Chapter 2

Background

2.1 Introduction

Artificial intelligence (AI) is the term used to describe the capacity of a machine to carry out operations that traditionally call for human intelligence. AI is a broad field that includes many different techniques and approaches, one of which is machine learning. Machine learning (ML) involves developing algorithms that enable computers to learn from data without being explicitly programmed.

Arthur Samuel (1959) provides a succinct overview of machine learning as a "field of study that gives computers the ability to learn without being explicitly programmed". ML algorithms have a broad impact on computer science and have diverse relationships with data-intensive issues. Along with the availability of data and increased processing capacity, novel learning algorithms help address a variety of real-world problems. Many computationally intensive machine learning methods such as Artificial neural network have been improved due to modern computing advances [25]. Models with deeper designs (having more layers) can benefit the computation power to work with larger datasets [26, 27, 28, 20].

However, increasing the model size is not desirable for devices with lower computational power, particularly because prediction time and storage can be challenging. In section 2.2, after providing the basic principles of artificial neural networks and related background knowledge, we will cover compression methods for neural networks and present approaches that can address these difficulties (see section 2.2.2).

To get the most out of Sub-symbolic AI models, the end-user needs confidence and faith in the outcome. Most of the time, understanding the explanation behind a model's decision is difficult. Explainable AI covers a field of algorithms that allow explaining Sub-symbolic AI models for humans to comprehend and trust results obtained. We will review some Explainable AI techniques and discuss some of the approaches in each category in section 2.3. However, since there is no universal agreement on how to classify Explainable AI methods, we will mainly follow the Interpretable Machine Learning Book [16] categorisation.

2.2 Neural Network Models and their Compression and Acceleration

Frank Rosenblatt in "Principles of Neurodynamics. Perceptrons and the Theory of Brain Mechanisms" [29] introduced the fundamentals of neural networks, where he described a group of machine learning model called Artificial Neural Network (ANN) that are motivated by the structure and function of the human brain. These networks are made of layers of neurons, or nodes that are interconnected through weights.

In this section, we will review the fundamental concepts of neural networks and explore their technical aspects to provide the necessary information for the subsequent chapters, where we will focus on their theoretical foundations and practical applications.

2.2.1 Artificial Neural Networks (ANNs)

Artificial Neural Networks are a subtype of sub-symbolic Artificial Intelligence that consists of computer models inspired by the structure and function of the human brain. ANN may be used to carry out a variety of tasks since they are built to learn from data by spotting patterns and relationships.

Neural networks models come in a variety of forms, including perceptron, feedforward, radial basis function, recurrent, and modular neural networks. Each variety has distinctive qualities and scopes of application.

$$h = \sum_{n=1}^i X_i \cdot W_i \quad (2.1)$$

For instance, Perceptron was mathematically formulated by McCulloch and Pitts in 1943. Each input is coupled to neurons through a weight that indicates the relevance of that input,

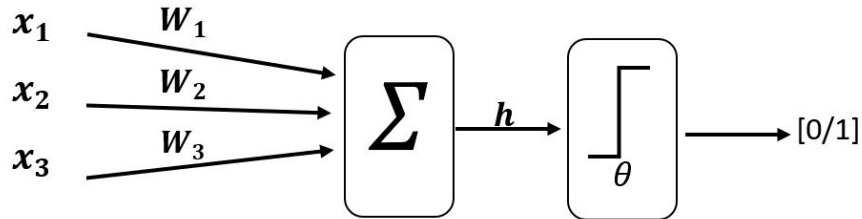


Figure 2.1: Mathematical model of a neuron by McCulloch and Pitts. X_i refers to inputs, and W_i shows the strength of the neuron connection. θ is threshold for h (sum of the weights) to active output

as seen in the figure 2.1. The sum of these inputs (see equation 2.1) will be transferred to the activation function, which will determine whether it should fire or not (equation 2.2). In addition, we have the option to add some bias value for equation 2.1.

$$output = \begin{cases} 1 & \text{if } h > \theta \\ 0 & \text{otherwise} \end{cases} \quad (2.2)$$

The presented perceptron network is a binary classification model where its lack of non-linearity prevents it from being applied to real-world problems.

Feedforward Neural Networks

A more robust type of neural network is a feedforward one, which has hidden layers between the input and output layers. Here the changes aren't simply about the number of layers. There are also different continuous activation functions that tackling non-linearity issues.

In feedforward neural networks, input dimensions are linked to neurons of the first layer as before. While we have more layers, all neurons in the previous layers are connected to the next layer of neurons.

The last layer usually has as many outputs as the number of classes in the dataset (in the case of classification). In the learning stage, the weight of each neuron is generated at random or fixed numbers. We then feed some input samples to the model and compare the results to the required outcome before modifying the weights. The fundamental weight initialization can be constant values like one, zero, or other real numbers. When a neural network is initialised with constant value, it is more likely to assume equal values

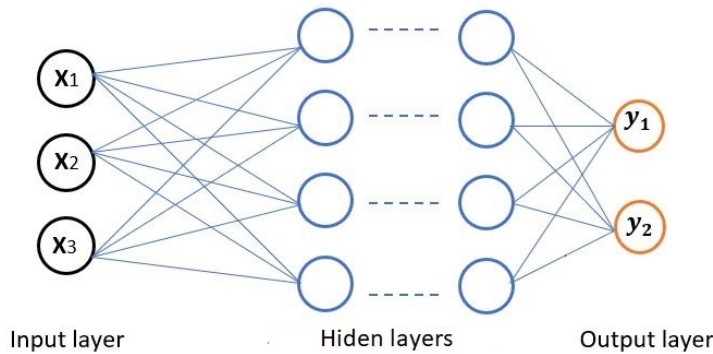


Figure 2.2: The figure displays a Feed forward neural networks that receives a vector X with three dimensions as input. Each input dimension and the neurons in the fully connected layers are connected in completely connected layers.

throughout training, which might lead to undesirable Symmetry breaking scenarios [30] the latter referring to the case when in training phase of the model, it is difficult or impossible for the weights to diverge. However, such values can be incorporated, but the model may require a different structure to prevent difficulties like those described by Blumenfeld et al [31]. Multiple different approaches to establishing the initial weight function have been developed, along with much research on their impact on the model, such as [32] and [33].

Activation Functions

An activation function is a mathematical function that maps the output of neurons to a specific range. These functions come in various shapes, each of which can add non-linearity to a model. In simple Perceptron networks, a step activation function is used, which produces a simple output of either zero or one. On the other hand, Multi-Layer Perceptrons (MLPs) use a different activation function. Figure 2.3 presents some examples of these functions.

$$f(x) = x \tag{2.3}$$

The linear function, in equation 2.3 performs a simple mapping of the input to the output. This is particularly useful when an MLP is designed to predict continuous values. For instance, in Chapter 4, the NMG model’s output is a real number, and the linear activation function is appropriate for this purpose. However, depending only on linear activation functions may restrict the neural network’s ability to deal with more complex

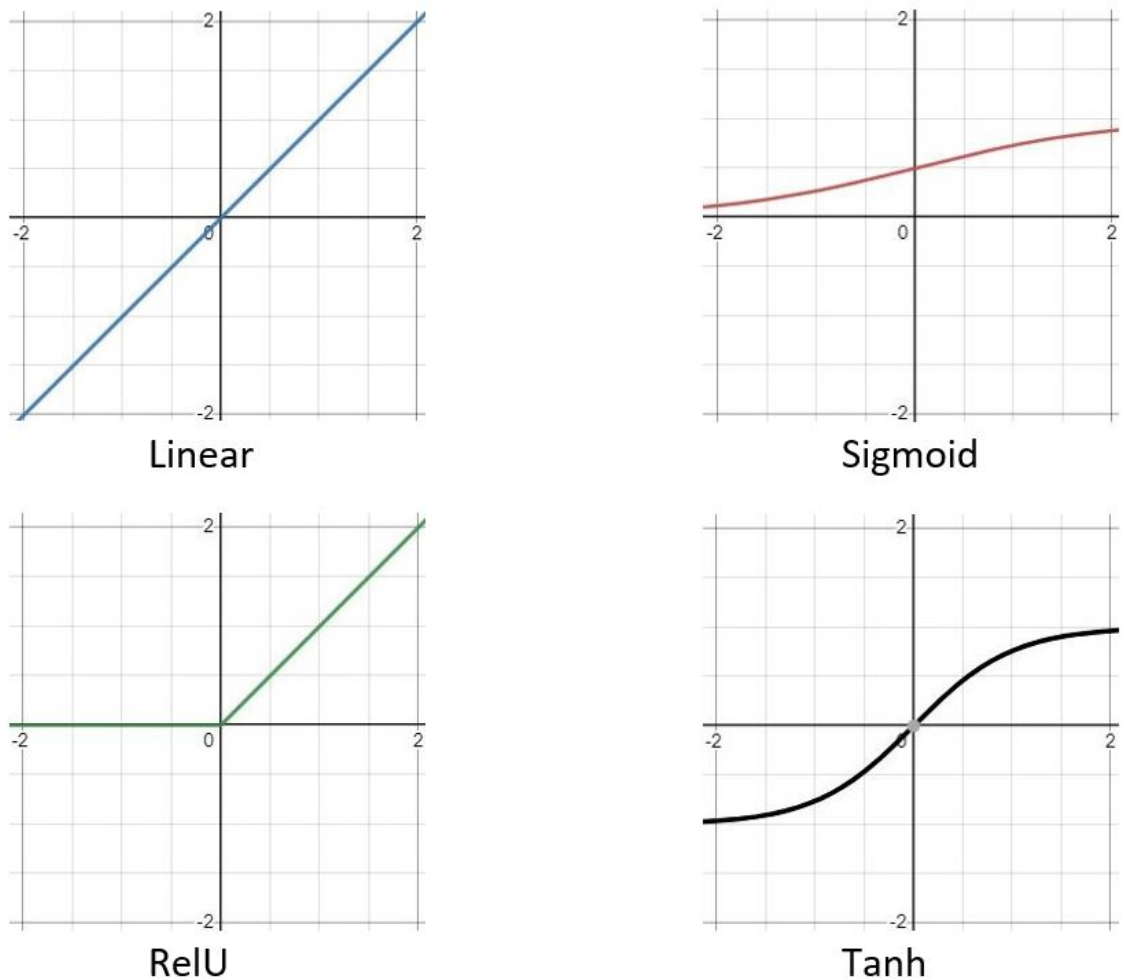


Figure 2.3: This figure contains four activation features. An activation function will be given some inputs and neuronal weights, and it will decide how to respond based on the formula they contain.

nonlinear patterns. The Hyperbolic Tangent Function (as indicated in equation 2.5) and the Sigmoid (see equation 2.4) are for instance two more activation functions that have attracted significant attention in the field of artificial neural networks.

The Sigmoid function, also referred to as a logistic function, can be employed when the output is a probability due to its output range of 0 to 1. However, this function is plagued by several issues, such as vanishing gradient and saturation, which can impede a neural network model from learning or can reduce its learning rate. The vanishing gradient problem arises when the gradient becomes negligible, and the weights in the lower layers

fails to change significantly. After several epochs, during which learning happens relatively quickly, the value of the linear component will be far from the centre of the Sigmoid function [34]. The Sigmoid activation function formula is shown in Equation 2.4.

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \quad (2.4)$$

The Hyperbolic Tangent Function (Tanh) outputs values within the range of -1 to 1 (Equation 2.5), which results in a normalised output in relation to its input. Typically, Tanh is employed in the middle layers of a neural network, while the Sigmoid function is used in the final layer. Tanh offers certain advantages over Sigmoid, but it too is susceptible to the vanishing gradient problem [35, 36].

$$\text{Tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.5)$$

The Rectified Linear Unit (ReLU) activation function serves to remove negative values from its inputs and allow only positive values to pass through (Equation 2.6). In contrast to both the Tanh and Sigmoid functions, the ReLU function does not suffer from vanishing gradients, yet it is prone to a "dying ReLU" issue, which arises when the neuron is inactive, and the ReLU gradient becomes zero, rendering training through gradient descent impractical [37].

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \quad (2.6)$$

Other variants of the ReLU activation function, such as Leaky ReLU [38] and QReLU [39], aim to enhance efficiency or tackle the dying ReLU problem. For example, the leaky ReLU function assigns a small slope to negative inputs, which allows the derivative to have a non-zero value for negative inputs. This small slope ensures that the dying ReLU problem is prevented because the neurons with negative input can still learn. The equation for Leaky ReLU is shown in Equation 2.7.

$$\text{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases} \quad (2.7)$$

The Sigmoid function can create output values ranging from 0 to 1, which is beneficial for probabilities, but each output is independent of the others. Therefore, this would prevent it from being used in multiple classification problems. To determine the relative

probability of the classes, a SoftMax activation function [40] is commonly deployed in the last layer of neural networks. The SoftMax activation function formula is given in Equation 2.8. It is sometimes called the normalised exponential function as each output after exponentiation is normalised by the sum of all its inputs. This allows obtaining probability values in the range [0..1] with the output vector x having a norm of 1.

$$SoftMax(x) = \frac{e^{x_i}}{\sum_j e^{x_j}} \quad (2.8)$$

Back-propagation

In neural networks, the initial weights do not inherently produce the desired output, and as such, they need to be updated to achieve a good result. The most common algorithm used to enable training is back-propagation [41].

To train a model, the first step involves defining the loss function. This function provides a measure of how well the model is achieving its objective. One such loss function that is commonly used is the cross-entropy loss function. It calculates the difference between two probability distributions as shown in Equation 2.9.

$$L(y, y') = \frac{-1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i)) \quad (2.9)$$

Where variables N , y , y' , and $p(y)$ in this formula represent the number of samples, true label, predicted probability, and positive class, respectively. In order to determine the loss value, the average negative log-likelihood of the real labels is taken into account along with the anticipated probabilities.

After the forward pass, where inputs are fed into the model, the difference between the desired value in the loss function and the model's output must be minimised. This process is known as back-propagation, which uses a chain rule to calculate derivatives of neural network parameters layer by layer. Equation 2.10 shows the partial derivative of each neuron with respect to its associated weights:

$$\Delta\omega_{i,j} = -\alpha \frac{\partial E}{\partial \omega_{i,j}} \quad (2.10)$$

An optimiser determines how to adjust the model's parameters after back-propagation has calculated an estimation of the gradient of the gradient of the loss function at each model weights. For instance, gradient descent [42], Adam [43], and AdaGrade [44] are some

of the optimisers that can be used. Each optimiser may have different hyperparameters; for instance, the learning rate is a hyperparameter in the optimiser that indicates how much the model's parameters should be adjusted, e.g., how much the weights have to be moved in the gradient direction. Some optimisers also take previous past steps by incorporating the previous gradients obtained at these iterations into the calculation.

Training neural networks is a complex process, and the distribution of each layer's input may change as a result of changes in the previous layer's parameters. Training the model in batches can help to reduce the duration of the training process, which can often be lengthy [45]. Additionally, computer equipment such as GPUs perform better when performing multiple operations in parallel. Therefore, using larger batches may help to fully leverage their capabilities.

2.2.2 Compressing Neural Networks

In Artificial Neural Network, measuring the accuracy and other relevant metrics of a model is a common practice to evaluate its performance. However, other characteristics such as computational cost and memory utilisation can also have a significant impact on the model's applicability, especially for low-memory devices. Therefore, it is advantageous to have strategies for reducing a model's complexity while maintaining its performance. In this section, we examine several general approaches for achieving this goal.

Pruning and Quantization of Parameters

Pruning and quantization techniques are widely used to eliminate unnecessary and irrelevant parameters that do not contribute to a model's performance. These approaches can be applied to different layers of neural networks and can be used during training or with pre-trained models. The techniques can be divided into three categories: Quantization and Binarization, Network Pruning, and Structural Matrix [46].

Quantization and binarization reduce the precision of weights and activations, which reduces their size and allows more operations to be run on computational devices, and reduces memory usage on these devices [47, 48, 49]. By using simpler operations such as XOR and POPCOUNT, the number of operations that can be run simultaneously on a device is increased, reducing the time complexity [3, 2, 50].

Pruning eliminates connections between neurons using different approaches such as loss functions or accuracy checks. Optimal Brain Surgeon [51] uses the Hessian of the

loss function to remove redundant connections and quantizes the weights. Several of these algorithms require retraining after pruning. For example, Chen et al. [52] employed a simple regularisation method based on soft weight-sharing and retrained the model. However, not all approaches require further training. Alqahtani et al. [53] present an effective strategy for simultaneously identifying important neurons and pruning the model during training without the need for any pre-training or fine-tuning operations. They reduced the size of models trained on the CIFAR10 dataset by 79% and the MNIST dataset by 91%.

Low-rank factorisation

Low-rank factorisation is a technique that uses matrix decomposition to estimate informative parameters. These methods are simple to apply and can be used with a variety of strategies, including pre-trained models. Typically, smaller matrices are generated whose multiplication produces a matrix that is similar to the original matrix. Low-rank decomposition aims to enhance inference speed, memory requirements, and initialization reliability.

Singular value decomposition (SVD) is a common technique for matrix decomposition in this context. For example, Lu et al. [54] used shortened SVD to obtain a small version of designs. Other decompositions, such as [55], have been introduced and used in neural network compression as well. Low-rank decomposition is ideal for speech recognition because the final layer has a large number of neurons. Switchboard, for example, is a corpus with 300 hours of transcribed speech and 9,300 output targets [56]. Therefore, investing in this field is interesting, as Sainath et al. [57] offered a technique for voice recognition that applies to the model's last layer.

Distillation knowledge

Distillation of knowledge involves training a compact neural network by transferring knowledge from a larger model. This process purifies the knowledge contained in the larger model, allowing it to be applied more efficiently to the smaller model. However, the effectiveness of this method is highly dependent on the application and network structure, and it cannot be used with pre-trained models. In addition, distillation of knowledge may introduce bias, which could reduce the generality of the model and negatively impact its accuracy [46] Typically, fully connected layers are used as a non-linear transform, with $f(x, w) = \phi(x, w)$ representing the input of the fully connected layer, where x is

the input dataset or the output of another layer, and w represents the weights inside the fully connected layer. A non-linear function ϕ applies element-wise non-linearity to the output. The complexity of this function is easily calculable, with the storage complexity of the equation being $\mathcal{O}(np)$ for a matrix $w_{n \times p}$.

Distillation of knowledge methods propose a new way to calculate the matrix multiplication in function f in order to reduce memory usage and storage complexity or decrease calculation time.

In the exploration of parameter redundancy in deep networks with circulant projections presented by Cheng et al. [58], the conventional linear projection in fully connected layers were replaced with the circulant projection in order to reduce parameters. The Adaptive Fastfood transform was used to reparametrize the matrix-vector multiplication [59]. With a series of matrices, after transformation, the computational cost was reduced to $\mathcal{O}(n \log d)$ and the storage cost to $\mathcal{O}(n)$.

2.3 Explainable AI and Local Model-Independent Methods

The complexity of neural networks, which can contain millions of parameters, makes it challenging to comprehend how they reached a given decisions. Explainable Artificial Intelligence (ExAI) aims to explain the behaviour of these models. To address this challenge, numerous ExAI models using different methods have been developed to explain various aspects of a model, such as the global perspective or a local view. In contrast, some models, such as Decision trees, are self-explanatory.

One technique for ExAI is to visualise the decision-making process of Sub-symbolic-based AI models. This method provides a graphical representation of how the model arrives at its decisions, making it easier to understand. Another technique is to use feature importance measures to identify which features or inputs are most influential in the decision-making process. This approach helps to highlight which inputs or features of the model is focusing on when making decisions.

In summary, several techniques have been developed in Explainable AI to explain different aspects of a machine learning models, such as the global and local perspectives. These techniques are based on different methods and have been used in numerous ExAI models. The following paragraphs will discuss some of these techniques. (See [60] for further reference).

2.3.1 Interpretable Machine learning Models

Interpretable models refer to models that can be easily explained and understood by humans. These models are particularly important in the context of explaining a model, as they can provide transparent insights into how the model makes predictions or decisions. Linear Regression and Decision Trees are two popular interpretable models that are commonly used in various domains, such as finance, healthcare, and social sciences.

The following section outlines two models that possess the characteristics of self-explanatory or interpretable models.

- **Linear Regression:** Linear regression is widely used in various fields, such as economics [61], social sciences [62], and other to make predictions or to analyse the relationship between variables. Linear regression uses a linear approach for modelling the relationship between a dependent variable and one or more explanatory factors (independent variables).

Linear regression assumes the prediction is made via a weighted sum of the feature inputs (see the equation 2.11). It is a common statistical method used to model the relationship between the predictors and the response variable. In addition, it assumes that the relationship can be expressed as a linear combination of the predictor variables and is often the simplest and most widely used model form in regression analysis [63].

$$y = \epsilon + \beta_0 + \sum_{i=0}^{i=n} \beta_i x_i \quad (2.11)$$

This weight (β , coefficients) can be used to interpret the decision. Mostly higher value is the most influencing the results [64, 65]. For example, the importance of the input features for a regression model is linked to the absolute values of the estimated weights scaled with their standard error (t-estimate) [66]. Some researchers have also demonstrated the use of plots to explain a model ([67], [68]).

In linear regression, drawing plots like the Weight Plot and the Effect Plot, which respectively illustrate how a feature affects the results predicted, and demonstrate how much the weight-feature combination contributes to the predictions in the data, can also provide understanding.

- **Decision tree:** A decision tree is a machine learning technique that is suitable for both linear and non-linear problems. It belongs to class of supervised learning

2. Background

Algorithms	CART	ID3
Type	Classification and Regression trees	Classification trees
Most common dividing criteria	Gini impurity or the mean squared error	Information gain or gain ratio
Missing data	Handle missing data	Needs imputing
Performance	Computational effective	Less Computational effective (large datasets or trees with many branches)

Table 2.1: Comparing CART and ID3 for building a decision tree.

algorithms where the data has been labelled in advance. The decision tree can be defined as a set of nodes in which each node divides the input data into several segments based on the values of a given feature. The decision tree splits the data into new subsets at each node, and this process continues recursively until it reaches a leaf where it can determine that the sample used belongs to a particular class. Some of the criteria to prevent a tree from growing include specifying the maximum depth of a tree or ensuring that every item in a node belongs to the same group.

There are numerous metrics available to determine which feature is best in order to split data at a given node. Using these metrics, each feature is given a score. Scores can be defined based on different criteria such as information gain ratio [69] (the information gain metric for the number of branches that result from splitting on a given feature) or Relief [70] (the relevance of a feature by comparing the values of neighbouring instances with the same target class). For instance, some of the measures used to calculate the relevance of a feature are the Mean Decrease Impurity (MDI) [71] or the Gini index [72].

Two of the most well-known decision tree algorithms are "Classification and Regression Trees" (CART) [73], and "Iterative Dichotomiser 3" (ID3), which was developed by Quinlan [74]. Table 2.1 provides a comparison of some of the characteristics of these algorithms.

Decision trees are also inherently interpretable, meaning that the decision-making process of the model can be easily visualized and understood. The features that have been selected each time to split data can be used to determine the feature's overall importance [75]. As shown in Figure 2.4, plotting a decision tree can show which features have been selected to split the data and the criteria that lead a sample to the

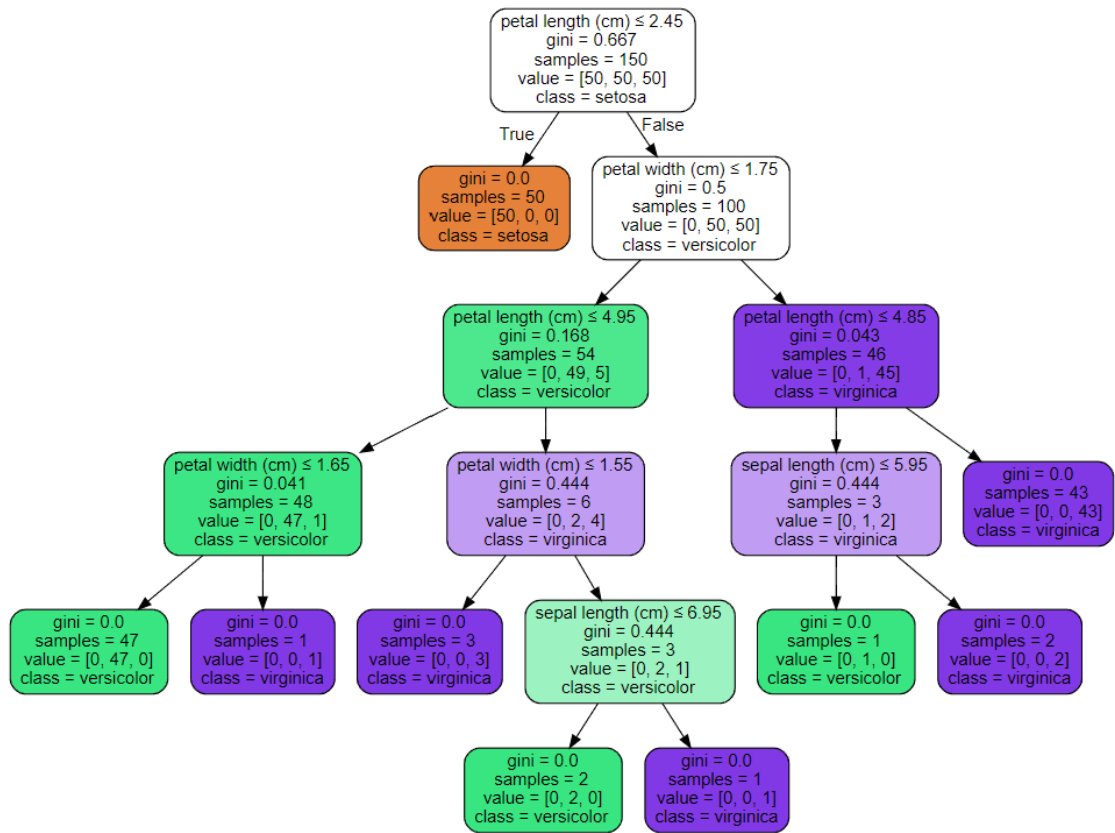


Figure 2.4: Plotting a decision tree can show what feature has been selected and the criteria that lead a sample to leaves. .

leaves. This makes decision trees useful in domains where model interpretability is important.

2.3.2 Local Model-Agnostic Methods

The local model agnostic method typically aims to explain individual samples and their predictions. To track prediction changes, the model modifies the input values. Counterfactual explanations, LIME and SHAP are three well-known examples of such methods.

- **Local interpretable model-agnostic explanations (LIME)**

LIME is an example of a local model-agnostic method [76]. Given a specific sample, LIME constructs a new dataset by generating multiple points in its local neighbourhood, which are then used to train a model. Changes in predictions are then analysed using a kernel function that assigns larger weights to nearby points,

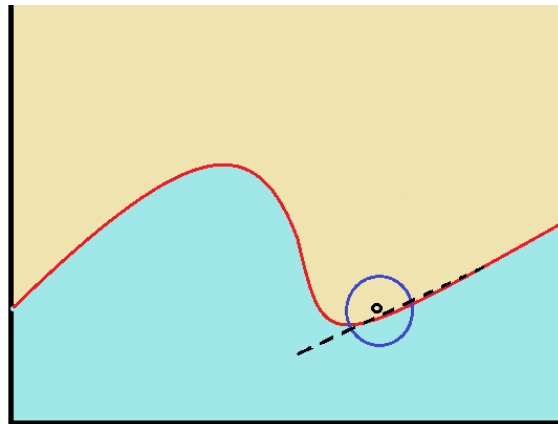


Figure 2.5: The red line indicates the boundary between two classes in a binary classification problem for a two-dimensional space. An example is depicted with a black circle enclosed by a larger blue circle. The blue area represents how far new data points can extend. The size of the area explained by a different linear model can vary.

in order to examine the behaviour of the model in that local space. The kernel width parameter is used to control the size of the meaningful radius of the weights that surrounds a given sample. The choice of the kernel function and its bandwidth can affect the explanation of the results, particularly when dealing with highly non-linear problems. This behaviour is depicted in figure 2.5.

- **SHAP and Shapley values**

The Shapley value was introduced in the context of game theory in the 1950s by the economist Lloyd Shapley [77]. A Shapley value is a mechanism for allocating the overall payoff of a cooperative game to its players in a fair manner, based on their marginal contributions to all potential coalitions.

In machine learning, Shapley values are used to explain the output of a model by attributing the contribution of each input feature to the final prediction. Specifically, the Shapley value of a feature is defined as the average change in the model output when the feature is included, over all possible subsets of features that also include the feature [78].

The Shapley values of a feature are defined as the difference between the expected model output and the conditional model output given the feature value, averaged over all possible orders of features. To compute Shapley values in practice, one needs to evaluate the model output on many subsets of features, which can be

computationally expensive or even infeasible for large datasets or complex models [79]. A common method for estimating Shapley values involves using a kernel function to sample feature subsets based on their similarity to the original input, with each subset being assigned a weight [80].

The SHAP technique uses the Shapley values to provide explanations for a result. In practice, it is necessary to approximate the predicted and conditional model outputs for each feature value. This can be effectively done using tree-based models or linear models. Depending on the extent of aggregation or granularity of the input characteristics, the SHAP results can provide a global or a local explanation of the behaviour of a model [81].

- **Counterfactual explanations** alter a given prediction. A very well-known example of its use is when the customer wants to know how to influence the bank's decision of rejecting a loan application. To accomplish the stated objective, many methods (such as trial and error and creating a loss function) may be used. Wachter et al. [82] proposed a simple loss function to change the model's decision where the distance between two produced values and samples is added up to the weighted (λ) difference between sample labels and predicted label for the new value.

$$\arg \min_{x'} \max_{\lambda} \lambda(\hat{f}(x') - y')^2 + d(x, x') \quad (2.12)$$

Dandl et al. developed a new loss function (equation 4.2) that includes a counter for the modified features [83]. This was done in consideration that the loss function might affect all of the input features. They also consider the gap between the output value and the nearest sample in the training dataset.

$$L(x, x', y', x^{obs}) = o_1(\hat{f}(x'), y'), o_2(x, x'), \\ o_3(x, x'), o_4(x', X^{obs}) \quad (2.13)$$

Measurement of a distance in each loss function can differ. In chapter 4, we will discuss some of the differences occurring when using distance functions. One problem with the possibilities presented is that modifying some of the features would be ineffective. Returning to the earlier example, if the updated values indicate that the consumer should lower their age, this would not be appropriate in this instance. A potential solution to this problem will be discussed in chapter 4.

Chapter 3

Efficient Fully Connected Layers in ANN

3.1 Preliminaries

Matrix multiplication is at the heart of various crucial algorithms and is used by a large variety of applications. It supports many applications and scientific fields like Physics (e.g., Lattice QCD), Machine Learning, and Data Science in general, where calculating correlation between variables can be important; and also, information retrieval algorithms indirectly used by many people through search engines. One aspect with matrix multiplication is that the basic algorithm's complexity does not scale linearly, which becomes problematic when processing large datasets, hence requiring expensive computational resources. Indeed, while square matrices require $O(n^2)$ storage space, $O(n^3)$ computational complexity are executed by the basic algorithm. Hence, the storage space and computational complexity of a non-square matrix would be $O(m.n)$ and $O(m.n.p)$ for two matrices, where m , n , and p denote the dimensions of the matrices.

In this chapter, a new randomised algorithm (non-deterministic) called the Mediterranean Matrix Multiplication is introduced. It is a straightforward and useful algorithm, and its performance is also discussed. Additionally, the chapter examines how this algorithm can be applied to compress matrix multiplication. The results indicate that compressing the fully connected (FC) layers can lead to a significant reduction in the size of these models. In the VGG16 network, the size was primarily determined by the FC

layers. However, employing the Mediterranean diet approach compresses the FC layers relatively negligibly in size, resulting in a state-of-the-art compression.

To facilitate understanding of the following sections, we enumerate some of the symbols used hereafter:

- A, B, C, Y, W, X : Matrices and vectors (X and Y) such that $C \simeq AB$ and $Y = WX$ (W being a weight matrix, X a layer input and Y its output).
- A_i, W_i : Rows i of respective Matrices A and W .
- B_j : Column j of Matrix B .
- k : Number of trials/hyperplanes used (and the main trade-off factor between precision and complexity, with the error decreasing at a rate of $O(k^{-0.5})$).
- m : Number of rows of A, C, W and Y .
- p : Number of columns of B and C .
- n : Number of columns of A and W , and number of rows of B and X . When discussing square matrices, we will assume that all dimensions are of size n .
- ϵ : Error made on the approximation.
- ϵ' : A constant arbitrarily close to zero used for the complexity notation.
- $1 - \gamma$: The confidence level for the error that occurs.
- ω : A complexity constant for square matrix multiplication algorithms such that their complexity is expressed as $O(n^\omega)$.
- α : A complexity constant for rectangular matrix multiplication such that the product of an $n \times n$ matrix by an $n \times n^\alpha$ matrix is known to be achievable in $O(n^{2+\epsilon'})$ [84].

3.2 Related Work

Matrix multiplication has undergone significant improvements in performance and efficiency in recent years due to major advancements in algorithm and hardware architecture

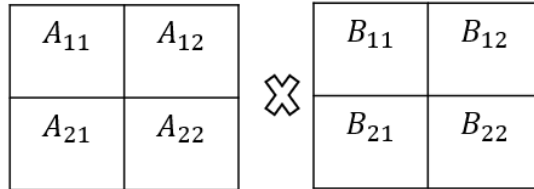


Figure 3.1: Sub-matrices in strassen's algorithm

design. New techniques, such as block matrix multiplication [85] and low-rank matrix approximation, have been developed to reduce the computational complexity of matrix multiplication.

The progress made in algorithms and hardware architecture designs have enabled significant improvements in the performance and efficiency of matrix multiplication, opening new possibilities and discoveries in various fields [86].

In addition, specialized hardware, such as tensor processing units (TPUs) [87] and graphics processing units (GPUs) [88], have demonstrated superior performance over CPUs for matrix multiplication tasks that can be divided into parallel tasks. This parallel processing is especially useful in deep learning applications, where large matrices are frequently multiplied during the training of neural networks.

3.2.1 Advances in Matrix Multiplication

The conventional technique for matrix multiplication entails calculating each element of the product matrix by summing the products of the corresponding rows and columns of two input matrices. Several research studies have shown that the standard approach for multiplying matrices can be improved. Strassen's algorithm [89], for example, divides and conquers the original matrices' block partitions through a sequence of operations to create seven temporary matrices. The algorithm works recursively dividing each matrix into sub-matrices of approximately equal dimensions, such as $A_{1,2}$, which refers to the second block in the first row of matrix A as figure 3.1. This algorithm is used for the seven products of these sub-matrices as shown below:

$$\begin{aligned}
 M_1 &= (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2}) \\
 M_2 &= (A_{2,1} + A_{2,2})B_{1,1} \\
 M_3 &= A_{1,1}(B_{1,2} - B_{2,2}) \\
 M_4 &= A_{2,2}(B_{2,1} - B_{1,1}) \\
 M_5 &= (A_{1,1} + A_{1,2})B_{2,2} \\
 M_6 &= (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2}) \\
 M_7 &= (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})
 \end{aligned} \tag{3.1}$$

After calculation M s, the final results C is obtained with the equation 3.2. Instead of performing 8 multiplications to calculate the product of two matrices, as in the standard matrix multiplication algorithm, Strassen's algorithm only requires 7 multiplications and lowers matrix multiplication's time complexity to $O(n^{\log_2 7})$.

$$\begin{aligned}
 C_{1,1} &= M_1 + M_4 - M_5 + M_7 \\
 C_{1,2} &= M_3 + M_5 \\
 C_{2,1} &= M_2 + M_4 \\
 C_{2,2} &= M_1 - M_2 + M_3 + M_6
 \end{aligned} \tag{3.2}$$

Similar to the previous method, Don Coppersmith, and Shmuel Winograd [90] introduced another approach in which the input matrices are divided into four smaller matrices. However, this approach transforms the matrix multiplication problem into a polynomial multiplication problem. The product of the smaller matrices is expressed as polynomials, which are evaluated at specific points. The resulting values are then used to interpolate a new polynomial that corresponds to the product of the original matrices.

In addition, a Fast Fourier Transform (FFT)-based algorithm can be used to accelerate matrix multiplication when the matrices are transformed into the frequency domain. First, a Discrete Fourier Transform (DFT) is applied, and then element-wise multiplication is performed. The final result is obtained by performing an inverse DFT [91].

GPUs and TPUs can benefit massively from parallel algorithms. Many approaches have been proposed to improve the performance of matrix multiplication on GPUs [92].

The key is to distribute the computation between multiple threads where it can be done concurrently. The computation can be divided based on multiple memory hierarchies, where sub-tasks use different levels of memory, or multiple computation resources such as threads or machines [93, 94, 95].

The discovery of new algorithms for matrix multiplication that are efficient for hardware and have reduced complexity has been explored using machine learning methods. Fawzi et al. [86] applied a reinforcement learning algorithm to improve upon the best-known matrix multiplication algorithms. They formulated the discovery process as a single-player game called Tensor-Game. In each step of Tensor-Game, the player selects how to combine different entries of binary matrices (each cell representing a multiplication of $a \times a$, where a is the size of the matrices) to perform the multiplication. A score is assigned based on the number of selected operations required to reach the correct multiplication result. This approach is also capable of finding efficient multiplication algorithms for specific hardware. Dominguez et al. [96] demonstrated that the search for matrix multiplication algorithms can be formulated as an Ising-Hamiltonian optimization problem.

3.2.2 Computational Aspects

Matrix multiplications are usually computed in a deterministic way but can also be calculated in non-deterministic fashion. Given the identical input matrices, deterministic matrix multiplication yields the same result each time, making it a predictable and dependable technique. However, both randomised numerical linear algebra and probabilistic algorithms can be advantageous for multiplying matrices, which while it may seem surprising at first, can generate various outcomes for the same input matrices. The precise requirements and limits of the current application can determine whether deterministic or non-deterministic matrix multiplication should be performed.

In the field of deterministic matrix multiplication, researchers have long been interested in finding algorithms that can multiply two matrices with high efficiency. Two types of matrix multiplication algorithms that have been extensively studied are rectangular matrix multiplication and square matrix multiplication.

Table 3.1 provides a summary of the upper bounds on the complexity of exact matrix multiplication algorithms, expressed in terms of the exponent ω (their complexity is expressed as $O(n^\omega)$). This table includes the various works that have improved upon the original upper bound of $\omega < 2.808$ established by Strassen in 1969.

3. Efficient Fully Connected Layers in ANN

Algorithm	Upper Bound (ω)
Strassen [89]	$\omega < 2.808$
Pan [97]	$\omega < 2.796$
Bini [98]	$\omega < 2.78$
Schönhage [99]	$\omega < 2.53$
Romani [100]	$\omega < 2.52$
Coppersmith-Winograd [90]	$\omega < 2.5$
Strassen [101]	$\omega < 2.4785$
Coppersmith-Winograd [102]	$\omega < 2.375$
Stothers [103]	$\omega < 2.374$
Williams [104]	$\omega < 2.373$
Le Gall [105]	$\omega < 2.3728639$

Table 3.1: Reduced upper bounds for exact matrix multiplication algorithms.

Rectangular matrix multiplication, on the other hand, involves multiplying a matrix with non-square dimensions by a matrix with square dimensions. This type of matrix multiplication has important applications in areas such as image processing and data analysis. Table 3.2 summarises the upper bounds on the complexity of rectangular matrix multiplication for different matrix dimensions, assuming certain conditions are met. The table includes the upper bounds on the complexity of multiplying a $\mathbb{R}^{n \times n}$ matrix by a $\mathbb{R}^{n \times \log n}$ matrix, as well as the complexity of multiplying an $n \times n$ matrix by an $n \times n^\alpha$ matrix and an $n \times n^\alpha$ matrix by an $n^\alpha \times n$ matrix.

Type of Multiplication	Matrix Dimensions	Complexity Bound
$\mathbb{R}^{n \times n} \times \mathbb{R}^{n \times \log n}$	$n \times n, n \times \log n$	$n^2 + O(n^2)$ [106]
$n \times n$ by $n \times n^\alpha$	$\alpha < 0.172$	$O(n^{2+\epsilon})$ [107]
$n \times n^\alpha$ by $n^\alpha \times n$	$\alpha < 0.30298$	$O(n^{2+\epsilon})$ [108]

Table 3.2: Complexity bounds for rectangular matrix multiplication algorithms.

Non-deterministic algorithms have also been studied extensively. Indeed, many techniques where randomised algorithms have been shown to improve on the algorithmic complexity when compared to deterministic ones, although only approaching the ideal solution to a given precision. Freivalds' paper [109] originally demonstrated the usefulness of randomised algorithms to linear algebra by showing that the likelihood of $C = AB$ can be asserted to a given, arbitrarily high probability. The running time of Freivalds' algorithm is $O(n^2)$, contrasting with the known limits of deterministic algorithms for matrix

multiplication at the time of publication. Further research on randomised algorithms has been carried out to approximate specific results in linear algebra, including Singular Value Decomposition (SVD) and low rank approximations. One can cite some notable achievements by [110, 111, 112] and [113]. More recently, [114] has shown how to find two correlated d dimensional vectors belonging to a set of n vectors with a complexity less than $O(n^{1.62} + nd)$. In general, if the sole purpose is to identify highly correlated vectors in a k dimensional space ([113]), a nearest-neighbour search can suffice ([115]). Indeed, these are problems that can easily be solved by performing matrix multiplications at some stage, but this does not necessarily need to be the case.

If one wants to compute an actual product of two matrices to a given precision, there are several iterative solutions exhibiting an $O(n^2)$ complexity per iteration for calculating the estimation. This obviously implies a trade-off between the complexity constant and the final precision wanted, with the precision factor until now being included in the complexity result as a representation for the extra number of iterations needed. While not all applications using matrix multiplications are able to deal with some error margins, there are important areas where such an error could be acceptable or needed. Often, this is due to the sheer amount of data and dimensions to be processed, giving no other option than using approximation algorithms. For instance, Cohen and Lewis [116] have demonstrated an approximation of a series of matrix multiplications in the context of information retrieval and pattern recognition where the number of features/dimensions is often in the order of 106 or even higher. Their algorithm works by creating a layered graph where each node of a layer represents a row of a matrix. This graph is then processed using a random walk that samples the final result in a Monte Carlo fashion. The main focus in [116] being the finding of highly correlated data, a good strategy for that purpose is often to quickly identify the largest dot products before performing higher precision calculations on them by for instance computing the relevant dot products fully. It is easy to see that one can approach the row/column dot product value defined as equation 3.3:

$$\sum_{k=1}^n a_{ik}b_{kj} \tag{3.3}$$

By considering a random subset of all the products $a_{ik}b_{kj}$ as an approximation. This selective sampling approach is sensitive (i.e., exhibiting unbounded variance) however to the input values of the two matrices ([117], [118]), usually when high frequencies are present in the data. To circumvent this problem, it was proposed in [119, 120] to select

a limited number of dimensions using an importance sampling scheme in accordance with the lengths of rows of A and columns of B , which guarantees a bounded variance for the estimation.

While not improving theoretical bounds, research restricted to Boolean matrix multiplication has nevertheless led to specific complexity results and produced more practical algorithms. The famous Four-Russians algorithm ([121]) led to a reduced complexity of $O(n^3/\log^2 n)$ for multiplying Boolean matrices. This result was superseded by a complexity of $O(n^3(\log \log n)^2/\log^{9/4} n)$ in [122]; $O(n^3(\log \log n)^3/\log^3 n)$ in [123]; and finally $\hat{O}(n^3/\log^4 n)$ in [124] – currently the best-known bound of this type.¹

From a purely theoretical point of view, other ways to reduce the strict number of arithmetic operations used for performing matrix multiplication exist. $O(n^2 \log n)$ was demonstrated by [125] while a better bound of $O(n^2)$ was achieved in [126] and [127]. However, these results are achieved by using extra-large integers or real values that usually have a binary size in the order of $O(n)$. This allows “packing” several operations into a single theoretical arithmetic operation that cannot however be executed in $O(1)$ steps on a regular computer. In general, while the lowest bound value for ω is not yet known, many have conjectured that the true value for ω is 2 for a deterministic algorithm.

A different, more streaming-oriented approach is the one proposed in [118] and based on random projections principles as described by Johnson and Lindenstrauss ([128]). At the heart of this technique is the computation of an $ASS^\top B$ product, where S is a Johnson-Lindenstrauss Transform (JLT) sign matrix. Indeed, [118] demonstrated that to achieve an error less than ϵ with a confidence level of $1 - \gamma$. For example, ensuring,

$$\Pr(\|AB - C\|_F < \epsilon \|A\|_F \|B\|_F) \geq 1 - \gamma \quad (3.4)$$

The number of steps required is as below:

$$O((mn + np + mp) \times (\log(1/\gamma) \times \epsilon^{-2} + \log(1/\gamma)^2)) \quad (3.5)$$

This bound was further improved [1] to

$$O((mn + np + mp) \times \log(1/\gamma) \times \epsilon^{-2}) \quad (3.6)$$

, which can theoretically be reduced to $O(n^2)$ steps for sufficiently large n values if fast rectangular matrix multiplications are used.

¹The \hat{O} notation is used to remove the $\text{poly}(\log \log n)$ factor in the expression of the complexity.

Since the pioneering work by Strassen [89] demonstrating a sub-cubic complexity, many deterministic and non-deterministic techniques for matrix multiplication have been proposed, as any progress made on such a fundamental concept can have a large impact. Nevertheless, there are issues plaguing the more theoretically optimal algorithms, and especially the deterministic ones. Typically, the presence of very large complexity constants coupled with algorithms which do not benefit much from the stream-oriented modern processor architectures (vs random access) make these low-complexity algorithms difficult to use in practical applications. In the last decade or so, randomised iterative algorithms that are able to get closer to an optimal complexity of $O(n^2)$ have gained attention. These algorithms also exhibit a run-time complexity constant as a non-linear function of the desired precision and usually in the order of ϵ^{-2} . Nevertheless, there is a point to using approximation algorithms as some applications that handle large datasets do not necessarily need an exact solution. For instance, finding approximate correlations quickly may be preferred to finding exact solutions in firm real-time systems (For example financial services and high-frequency trading), especially if the error variance can be easily quantified or at least estimated. Drineas and Mahoney have recently shown ([129]) that various scientific areas can benefit from randomised linear algebra algorithms.

Finally, several algorithms have been proposed for specific, typically sparse matrices that can be considered "compressible". In [130], it was shown that two sparse square matrices containing at most m elements could be multiplied in $O(m^{0.7}n^{1.2} + n^{2+O(1)})$ steps. [131] demonstrated that if a matrix AB has a sufficiently low number of not close to zero entries per column ($< n^\alpha$, where α is the best-known exponent for rectangular matrix multiplication), then a compressed sensing algorithm exists that executes in $O(n^{2+\epsilon})$ time steps. [132] demonstrated a randomised algorithm running in $O(n^2s^{\omega/2-1})$ that can compute the product of two Boolean matrices if they contain at most s non-zero entries, where ω is the best-known exponent for matrix multiplication [105]. Pagh [2013] improved on those results by demonstrating an algorithm for computing AB exactly with high probability, in time $O(N + nb)$ in the case where A and B have at most N non-zero entries and AB has at most b non-zero entries. Similarly, [118] and [133] computed Alon Matias Szegedy (AMS) sketches [134] in a way that reduced complexity. For this, hash functions are chosen such that they can be processed separately on matrices A and B . Counting sketches are then mainly done in the frequency domain to accelerate the count before decompressing the final result in the space domain. Kutzkov [135] worked on similar

principles using AMS sketches and for the first time demonstrated an algorithm running in $O(n^{2+\epsilon})$ when AB has fewer than $O(\sqrt{n})$ non-zero elements.

Overall, the Mediterranean diet has similar advantages to some of these approaches [3, 136, 137, 59] for inference as it combines predominant binary operators, low rank matrices and random projections in one framework.

3.3 Mediterranean Matrix Multiplication

This section proposes the Mediterranean Matrix Multiplication, a new, simple, and practical randomised algorithm that samples angles between the rows and columns of two matrices with sizes (m, n, p) to approximate matrix multiplication in $O(k(mn + np + mp))$ steps, where k is a constant related to the precision desired. The random sampling pattern, as discovered initially by Goemans and Williamson [138], ensures the estimation of each entry of the resulting matrix in constant time for a given precision. While the theoretical complexity can be shown to be equivalent to the one proposed in [1] - i.e., $n^2 + O(n^2)$ when using fast multiplication algorithms of square matrices, the number of instructions carried out is mainly bounded by bitwise operators, amenable to a simplified processing architecture and compressed matrix weights. Results show that the method is superior in size and number of operations to the standard approximation with signed matrices.

Our new algorithm will therefore improve upon past research on randomised algorithms for matrix multiplications by firstly showing better convergence results for correlated rows and column, and secondly allowing various packed bit-wise operations that can be carried out at a faster pace than equivalent fused-multiply-and-add operations. Our GPU implementation will for instance take advantage of fast population count units available on modern architectures.

3.3.1 Monte Carlo Sampling of angles between vectors

We introduce a complexity analysis for the M^3 algorithm proposed in this chapter. At the heart of these algorithms is a relatively simple principle of randomly sampling the angle between two vectors. This principle was initially introduced in [138] to provide a good approximation to the solution of the Max-Cut problem, but curiously never found its way into a matrix multiplication algorithm. In this chapter, it was noticed that the probability of having a random plane separating two vectors (i.e., the probability of getting opposite

dot-product signs) was proportional to the angle formed by these two vectors. Others have since used this principle successfully and applied it to produce algorithms identifying similarities (e.g., [139] with the aim of providing good hash functions).

This work finds a new scope of application for this idea by subsequently integrating it inside a simple Monte Carlo process eventually leading to a lower matrix multiplication complexity bound.

The starting point of our Monte Carlo evaluation is the expression of a dot product between two vectors as:

$$C_{i,j} = A_i B_j = \|A_i\| \cdot \|B_j\| \cdot \cos \theta_{ij}. \quad (3.7)$$

From there, one notices that the angle value θ_{ij} is the only limiting factor in establishing a minimal complexity result for matrix multiplication. Indeed, all the necessary $\|A_i\| \cdot \|B_j\|$ vector norm products can be calculated in $O(n^2)$ steps. If we can estimate the angle for every entry $C_{i,j}$ in k steps with a simple Monte Carlo sampling process, then we know that the variance in the estimation will be proportional to $1/k$. Furthermore, if we can compute a given number of k trials "for free" (i.e., in constant time per entry) as n increases, with a relationship $k = f(n)$ and f being a monotonic function that tends to infinity, the relative error made can then be factored out of the complexity expression. This allows the existence of a lower complexity bound for matrix multiplication approximation, with our main theorem expressed as follows:

Theorem 3.1 Let ϵ , ϵ' and γ be three positive real values arbitrarily close to 0. The product of two square matrices A and B can be approximated as a matrix C with an algorithmic complexity equivalent to that of a rectangular matrix multiplication (i.e., either $O(n^{2+\epsilon'})$ or $n^2 + O(n^2)$ steps), while satisfying

$$\Pr(\|C - AB\|_F < \epsilon \|A\|_F \|B\|_F) \geq 1 - \gamma \quad (3.8)$$

While a similar result has already been given in [1], we will demonstrate that it is still valid when sampling angles instead of using randomly signed matrices.

3.3.2 Sampling principles

Lemma 3.2 Let A_i and B_j be two non-collinear vectors of any length, and \mathcal{P} be the 2D plane defined from a linear combination of these two vectors and the origin as shown in Fig. 3.2. The angle between these two vectors can be approximated with a Monte

3. Efficient Fully Connected Layers in ANN

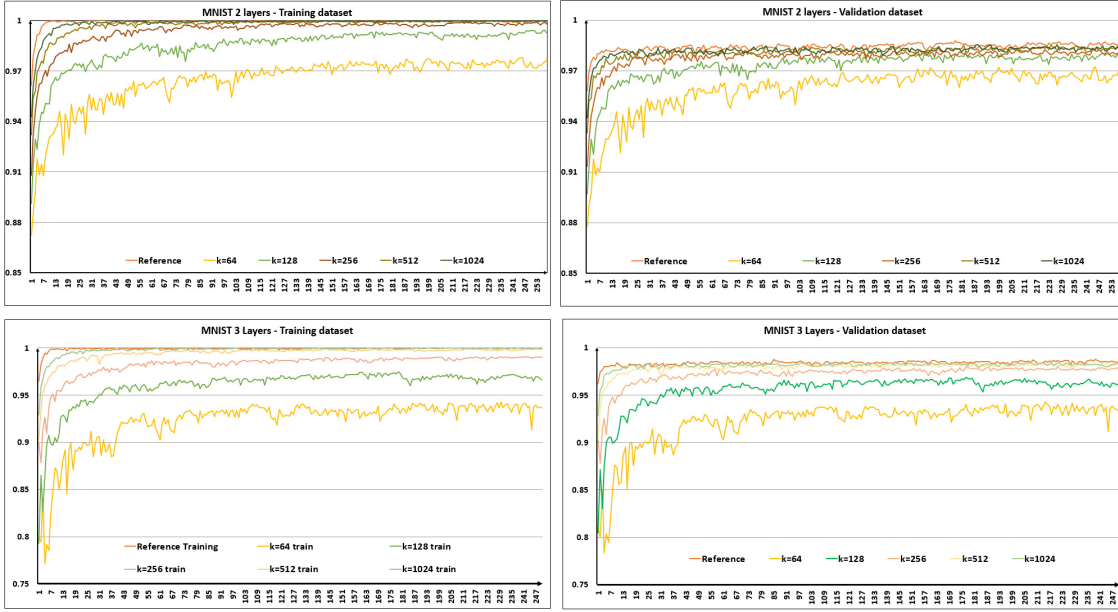


Figure 3.2: An illustration of Monte Carlo sampling of an angle between two angles which accumulates results from random tests (Algorithm 1). Left: Both points are on the same side of the hyperplane and test will return 0. Right: The hyperplane is separating the two points and the test will return +1.

Carlo simulation (Algorithm 1) that initialises the angle to 0 and then repeats $k - times1$) Choosing a random line \mathcal{L}_ϕ of \mathcal{P} crossing the origin; 2) Adding $\frac{\pi}{k}$ to the angle when \mathcal{L}_ϕ separates the two vertices A_i and B_j .

The justification for this lemma is simple and has been discussed in detail in [138]. We will provide a thorough explanation of the proof and examine some of its properties for completeness.

Proof. Let θ be the angle between two unit vectors as defined by the arc-cosine of the vectors' dot product, and within the range $[0..\pi]$ (i.e., the complement of the reflex angle). This angle can be defined as shown in Fig. 3.2 by the integral over the section related to the angle

$$\theta = \frac{1}{2} \int_0^{2\pi} Box(A_i, B_j, \mathcal{L}_\phi) d\phi. \quad (3.9)$$

Where Box is a box function that returns 1 if the line \mathcal{L}_ϕ separates the two vertices A_i and B_j , and 0 otherwise. Note that since \mathcal{L}_ϕ is equivalent to $\mathcal{L}_{\phi+\pi}$, the final result needs to be divided by a factor of two. The mathematical principles to estimate this integral using a

Algorithm 1 Iterative dot product approximation by angular sampling.

Input: Row A_i , column B_j and a number of iterations k .

Output: An approximation C_{ij} such as $C_{ij} \simeq A_i B_j$.

$$C'_{ij} = 0$$

$$NormA_i = \|A_i\|$$

$$NormB_j = \|B_j\|$$

for $s = 1$ to k **do**

$$E_s = \text{RandomNormalDistribution}(\sigma = cst, \mu = 0)$$

$$DotA_i = A_i \cdot E_s$$

$$DotB_j = E_s^T \cdot B_j$$

if $(DotA_i \cdot DotB_j < 0)$ **then**

$$C'_{i,j} = C'_{i,j} + 1$$

$$C_{i,j} = \cos\left(\frac{\pi}{k} C'_{i,j}\right) \cdot NormA_i \cdot NormB_j$$

Monte Carlo simulation are well established and θ can be evaluated as

$$\hat{\theta} = \sum_{t=1}^k \frac{\pi}{k} \text{Box}(A_i, B_j, \mathcal{L}_{\phi_t}) = \frac{\pi}{k} \sum_{t=1}^k \text{Box}(A_i, B_j, \mathcal{L}_{\phi_t}), \quad (3.10)$$

, where k is the number of random tests desired, with $k > 0$. Finally, we have been assuming that the vectors are not collinear as we would not be able to define a 2D plane otherwise. If this is the case, A_i and B_j will always be on the same side of the separating line and the approximated value for the angle will be 0, which is as expected. \square

It is useful to determine the statistical properties of such a test (Algorithm 1), which can be established from classical statistical analysis. If we assume that the \mathcal{L}_{ϕ} space is sampled uniformly, then our box test is a Bernoulli trial of well-known success probability $\frac{\theta}{\pi}$ and performing k samplings will result in a binomial distribution, which leads to the following lemma:

Lemma 3.3 There exist a sufficient number of iterations k , an error margin ϵ decreasing as k increases, and a confidence level $1 - \gamma$ such that:

$$\Pr(|C_{i,j} - A_i B_j| \leq \epsilon \cdot \|A_i\| \cdot \|B_j\|) \geq 1 - \gamma \quad (3.11)$$

Proof. The main statistical property of interest to us is the evaluation of the error made for the estimation within some level of confidence. Firstly, as we are dealing with a

well-established sum of Bernoulli trials (i.e., a binomial distribution), the variance on the estimation $\hat{\theta}$ is given by

$$\begin{aligned} \text{Var}[\hat{\theta}] &= \text{Var}\left[\sum_{t=1}^k \frac{\pi}{k} \text{Box}(A_i, B_j, \mathcal{L}_{\phi_t})\right] = \\ &k \frac{\pi^2}{k^2} \text{Var}[\text{Box}(A_i, B_j, \mathcal{L}_{\phi_t})] = \frac{\pi^2}{k} \frac{\theta}{\pi} \left(1 - \frac{\theta}{\pi}\right) \end{aligned} \quad (3.12)$$

Also, the expected value $\mathbf{E}(\hat{\theta})$ is trivially equal to θ . Note that

$$\frac{\theta}{\pi} \left(1 - \frac{\theta}{\pi}\right) \leq \frac{1}{4}, \theta \in [0.. \pi]. \quad (3.13)$$

Therefore, the maximum variance is achieved for $\theta = \pi/2$ and is bounded by

$$\text{Var}[\hat{\theta}] \leq \frac{\pi^2}{4k}. \quad (3.14)$$

Hence, the error on the real angle is expected to decrease proportionally to the square root of the number of trials k . Note that we have voluntarily remove the $\frac{\theta}{\pi}(1 - \frac{\theta}{\pi})$ factor when generalising this result to all possible angles later on. Should we have some a priori knowledge about the minimal or maximal values of the row-column angles, such a constant could be re-introduced in the subsequent estimation of the error bound.

A similar $1/k$ convergence rate is encountered in algorithms exposed in [120] and [1], with the exception that it is now weighted by a constant in the range $[0.. \pi^2/4]$. Small angles will therefore require fewer iterations while angles close to $\pi/2$ will need $2.46 \times$ more samples to reach the same error level.

We are now interested in the statistical error ϵ that results from approximating an angle with k trials, in conjunction with a confidence level of at least $1 - \gamma$. This can be expressed as

$$\Pr(|\hat{\theta} - \theta| \leq \epsilon) \geq 1 - \gamma. \quad (3.15)$$

From the variance and expected value of the Bernoulli trials, one can use Chebyshev's inequality to obtain a bound for the error defined as

$$\Pr\left(|\hat{\theta} - \theta| \leq \epsilon\right) \geq 1 - \frac{\theta(\pi - \theta)}{k\epsilon^2}. \quad (3.16)$$

The ϵ value is therefore linked to the chosen number of iterations k and the confidence level γ as follows:

$$\epsilon_{k,\gamma} = \sqrt{\frac{\theta(\pi - \theta)}{k\gamma}}, \gamma_{\epsilon,k} = \frac{\theta(\pi - \theta)}{k\epsilon^2} \text{ and } k_{\epsilon,\gamma} = \frac{\theta(\pi - \theta)}{\gamma\epsilon^2}. \quad (3.17)$$

The actual property we are interested in is the estimation of the error made on $\cos \hat{\theta}$ as it is the value needed to compute the final dot product evaluation. The \cos function is monotonic in the range $[0..\pi]$, with $|\cos' \theta| \leq 1$. Therefore

$$\forall \theta_1, \theta_2 \in [0..\pi] : |\cos \theta_1 - \cos \theta_2| \leq |\theta_1 - \theta_2|. \quad (3.18)$$

Which simply implies

$$\Pr(|\cos \hat{\theta} - \cos \theta| \leq \epsilon) \geq 1 - \frac{\theta(\pi - \theta)}{k\epsilon^2} \geq 1 - \gamma. \quad (3.19)$$

We conclude that for any ϵ and γ in the range $[0..1]$, there exist a number of iterations $k_{\epsilon,\gamma}$ such that one can estimate the value $\cos \hat{\theta}$ with an error margin ϵ at a level of confidence $1 - \gamma$. For any given confidence level, the decrease rate in the error is proportional to $\frac{1}{\sqrt{k}}$. It follows that the error made on the evaluation of each $A_i \cdot B_j$ product is bounded with a given confidence level expressed as 3.20.

$$\Pr\left(|C_{i,j} - A_i B_j| \leq \epsilon \cdot \|A_i\| \cdot \|B_j\|\right) \geq 1 - \frac{\theta(\pi - \theta)}{k\epsilon^2} \geq 1 - \gamma. \quad (3.20)$$

□

Supposing that there is an algorithm allowing this bound for every entry of the result matrix, the global error bound needs to be expressed in relationship with the Frobenius norms of A and B to be comparable with other results in the area.

Lemma 3.4 There exists a sufficient number of iterations k that verifies:

$$\Pr\left(\|AB - C\|_F \leq \epsilon \|A\|_F \|B\|_F\right) \geq 1 - \gamma \quad (3.21)$$

Proof. We have

$$\begin{aligned} & \Pr\left(\|AB - C\|_F \leq x\right) \geq \\ & \Pr\left(\sum_{i=1,j=1}^{m,p} \left(\|A_i\| \cdot \|B_j\| \cdot |\cos(\hat{\theta}_{ij}) - \cos(\theta_{ij})|\right)^2 \leq x^2\right) \quad (3.22) \\ & \geq \Pr\left(\sum_{i=1,j=1}^{m,p} \left(\|A_i\| \cdot \|B_j\| \cdot |\hat{\theta}_{ij} - \theta_{ij}|\right)^2 \leq x^2\right) \end{aligned}$$

Where x is a positive value. We now calculate the expected value of the inner sum

$$\begin{aligned}
 & \mathbf{E} \left(\sum_{i=1, j=1}^{m, p} \left(\|A_i\| \cdot \|B_j\| \cdot |\hat{\theta}_{ij} - \theta_{ij}| \right)^2 \right) = \\
 & \sum_{i=1, j=1}^{m, p} \left(\|A_i\|^2 \|B_j\|^2 \cdot \mathbf{E}((\hat{\theta}_{ij} - \theta_{ij})^2) \right) = \\
 & \sum_{i=1, j=1}^{m, p} \left(\|A_i\|^2 \|B_j\|^2 \cdot (\mathbf{Var}(\hat{\theta}_{ij} - \theta_{ij}) + \mathbf{E}(\hat{\theta}_{ij} - \theta_{ij})^2) \right) = \\
 & \sum_{i=1, j=1}^{m, p} \left(\|A_i\|^2 \|B_j\|^2 \cdot \mathbf{Var}(\hat{\theta}_{ij}) \right)
 \end{aligned} \tag{3.23}$$

We know from (3.14) that the variance on every angle estimation can be bounded, leading to

$$\begin{aligned}
 & \mathbf{E} \left(\sum_{i=1, j=1}^{m, p} \left(\|A_i\| \cdot \|B_j\| \cdot |\hat{\theta}_{ij} - \theta_{ij}| \right)^2 \right) \leq \\
 & \frac{\pi^2}{4k} \sum_{i=1, j=1}^{m, p} \|A_i\|^2 \|B_j\|^2 \leq \frac{\pi^2}{4k} \|A\|_F^2 \|B\|_F^2
 \end{aligned} \tag{3.24}$$

Markov's inequality can now be used to finalize an upper bound. Combining (3.22), (3.24) and (3.25), we get

$$\begin{aligned}
 & \Pr \left(\|AB - C\|_F \leq x \right) \geq \\
 & 1 - x^{-2} \mathbf{E} \left(\sum_{i=1, j=1}^{m, p} \left(\|A_i\| \cdot \|B_j\| \cdot |\hat{\theta}_{ij} - \theta_{ij}| \right)^2 \right)
 \end{aligned} \tag{3.25}$$

Let x be $\epsilon \|A\|_F \|B\|_F$. We finally obtain

$$\Pr \left(\|AB - C\|_F \leq \epsilon \|A\|_F \|B\|_F \right) \geq 1 - \frac{\pi^2}{4k\epsilon^2} \geq 1 - \gamma \tag{3.26}$$

We therefore conclude that for any values ϵ and γ in the range $[0..1]$, there exists a large enough integer k verifying the hypothesis. \square

Even though this proof requires k to be in the order of $O(\epsilon^{-2}\gamma^{-1})$ as it is derived from a Markov inequality, it is well known that binomial distributions will lead to a $\epsilon^{-2} \log(1/\gamma)$ bound when k is finite, which is similar to the best known bound given in [1].

3.3.3 Basic algorithm

We now extend the randomised algorithm 1 respecting the bound described in the previous section to all entries of Matrices A and B . Let vectors A_i, B_j, E_s be three non-collinear vectors of \mathbb{R}^n , with E_s chosen randomly on the hypersphere centred on the origin and defining a unique orthogonal hyperplane \mathcal{P}_s . The intersection of \mathcal{P}_s with the 2D subspace $\mathcal{P}'_{i,j}$ — defined from a linear combination of vectors A_i and B_j — provides a random line $L_{i,j,s}$ in $\mathcal{P}'_{i,j}$ that crosses the origin. It also ensures randomness with a uniform distribution over the angle, as initially demonstrated by [138]. As such, $L_{i,j,s}$ is our random, uniformly distributed line that can be used for sampling the angle, which also follows the statistical convergence properties enunciated earlier. We can now make the whole process efficient with $O(mn + np + mp)$ steps per iteration by re-using the very same \mathcal{P}_s hyperplane for all dot products occurring in a matrix multiplication. Note that choosing a random plane ensures the uniform distribution of $L_{i,j,s}$ for all the $\mathcal{P}'_{i,j}$ planes. Hence, the basic test simply consists first of computing the signs of all dot products $A_i E_s$ and $E_s^T B_j$, which requires $O(mn + np)$ operations per random split. Obtaining opposite signs at a given s iteration, with $s \in [1..k]$, means that \mathcal{P}_s is a plane separating vectors A_i and B_j . This sign test will need to be repeated $O(mp)$ times (or $O(n^2)$ times in the context of square matrices) to cater for all possible dot product combinations (A_i, B_j) . Hence, the complexity of the basic algorithm over k iterations is $O(k(mn + np + mp))$. Finally, once all the angle cosines have been sampled, a post-processing scale of complexity $O(mn + np + mp)$ will multiply each result entry by the respective norms $\|A_i\|$ and $\|B_j\|$.

To generate a uniform and unbiased distribution on the hypersphere as required by the algorithm, it suffices to generate vector E_s entries randomly using a random number generator following a normal distribution (i.e., with a fixed parameter σ and $\mu = 0$) for each dimension independently. Importantly, the likelihood of choosing E_s orthogonal to A_i or B_j becomes infinitely small as the range of distinct floating point values tend to infinity. One limitation with this algorithm though, is that the error made on each $C_{i,j}$ entry will decrease at a rate of $k^{-1/2}$, similar to results obtained by [120] and [1].

3.3.4 Reformulated algorithm and Complexity Bounds

As mentioned by Clarkson and Woodruff [1], to improve on the trivial complexity bound obtained in (3.26), one needs only to notice 1) that the error decreases as the number of iterations k increases; and 2) that Algorithm 1 can be broken down into three rectangular

3. Efficient Fully Connected Layers in ANN

matrix multiplications, which allows us to compute k iterations for “free” (i.e., with the same algorithmic complexity of computing one iteration,) where $k \ll n$ but k also increases with n . To do so, we can write the whole algorithm as in algorithm 2.

To allow acceleration of rectangular matrix multiplications, an integer k must be chosen for instance such that $k = \lfloor n^\alpha \rfloor$ (cf., algorithms from [84] and [108]); or such that $k \leq \log n$ ([106]). Indeed, algorithm 2 in its first stage multiplies the $\mathbb{R}^{n \times n}$ matrix A with an $\mathbb{R}^{n \times k}$ matrix E made of k random vectors of \mathbb{R}^n , and similarly multiplies B with E^\top . The second stage, which counts the number of separating planes, will require $2kn$ sign extraction operations to be performed beforehand. This thresholding will process every entry of the two temporary matrices AE and $E^\top B$ such that their respective entries are set to 1 when $(AE)_{i,j} > 0$ and $(E^\top B)_{i',j'} < 0$, and -1 otherwise. Finally, the two thresholder sign matrices of respective sizes $\mathbb{R}^{n \times k}$ and $\mathbb{R}^{k \times n}$ will be multiplied together, which, if implemented as a rectangular matrix product, is also possible with a complexity of either $O(n^{2+\epsilon'})$ if $k < 0.30298$ [108] or $n^2 + O(n^2)$ for $k \leq \log n$ [106]. This also completes the proof for theorem 3.1.

Algorithm 2

Algorithm 1 rewritten as a product of three rectangular matrix multiplication resulting in a theoretical $O(n^{2+\epsilon'})$ or $n^2 + O(n^2)$ algorithmic complexity.

Input: Matrix A of rows A_i , Matrix B of columns B_j .

Output: The resulting matrix entries C_{ij} such as $C \simeq AB$.

for $i = 1$ to n **do**

$NormA_i = \|A_i\|$

$NormB_i = \|B_i\|$

$k = f(n)$

$\triangleright k = \text{floor}(n^\alpha)$ or $k = \text{floor}(\log_2(n))$

$E = \text{RandomNormalDistribution}(\sigma = cst, \mu = 0)$

$\triangleright O(n^2)$

$A' = AE$

\triangleright Rectangular Matrix multiplication

$B' = E^\top B$

\triangleright Rectangular Matrix multiplication

for $i = 1$ to n **do**

for $j = 1$ to n **do**

$A'_{i,j} = (A'_{i,j} > 0) ? 1 : -1$

$B'_{i,j} = (B'_{i,j} < 0) ? 1 : -1$

$C' = A'B'$

\triangleright Rectangular Matrix multiplication

for $i = 1$ to n **do**

for $j = 1$ to n **do**

$C_{i,j} = \cos\left(\frac{\pi(C'_{i,j} + k)}{2k}\right) \cdot NormA_i \cdot NormB_j$

3.3.5 Practical considerations

As improvements in the complexity of matrix multiplications are sometimes not practical, this section discusses the details behind an implementation of matrix multiplication approximation algorithms on modern architectures like GPUs. Algorithm 2 can be broken down into three stages. The first stage will compute WE and $E^\top X$, where E is a matrix made of k columns. It would therefore seem natural for k to be smaller than n as otherwise the computing effort would be similar to that of a basic “exact” matrix multiplication algorithm. This however is not a strict requirement for our algorithm as it performs most operations bitwise and therefore any value $k < 32n$ could be beneficial. One could also use a random number generator to create E , removing the need to store this matrix. However, we can actually process up to $k = n$ separating hyperplanes by convolution in $mn \log(n)$ (WE) and $n \log(n)$ steps ($E^\top X$) if E is chosen as a Toeplitz matrix with columns defined from a rotated random vector. Indeed, our hyperplanes must be chosen independently, and it is easy to see that the columns of a randomised Toeplitz matrix satisfy $\mathbf{E}(E_i E_j) = 0, i \neq j$. Furthermore, using a Toeplitz matrix we only need to create and store the first column E_0 of E , and possibly every subsequent column E_i such that $i \pmod n = 0$ (n being the number of columns of our weight matrix W).

In the next stage, thresholding of WE and $E^\top X$ can be performed in $O(2kn)$ operations and thus has no influence on the overall complexity of the process. The last stage is what differentiates our algorithm in practical terms from other algorithms like Clarkson and Woodruff’s approach ([1]). As thresholding is not embedded in the random matrix but comes later in the pipeline, the final operation is a pair-wise computation of hamming distances of complexity $O(kmn)$, which is simpler to carry out than a full-blown matrix multiplication. It can for instance be implemented with Boolean matrix multiplication algorithms like the Four-Russian algorithm ([121]), which would reduce the number of computations by a $\log(n)$ or even a $\log^2(n)$ factor for practical matrix sizes. However, this will require implementing look-up tables and adding various barriers in the flow of operations, therefore limiting parallelism. Even though current GPUs allow fast look-up operations, the throughput obtained for random lookups in tables more than a few kilobytes would be low when compared to the throughput of other simpler operations like FMA ones. Modern processor architectures, however, include population count instructions that can output sums of 32 bits (e.g., NVIDIA CUDA cores) or 64 bits (e.g., all modern X86 CPUs) integers at every clock cycle. Being able to process so many elements at once natively

is likely to be competitive with any algorithmic speedup we could get from Boolean matrix multiplications. Table 3.3 therefore summarises the differences between the signed matrix approach and ours that we must consider for obtaining maximum performance in a practical context. While bitwise operations have a much higher throughput in general as we process chunks of 32-bit elements at a time, our approach needs three logical instructions for every corresponding Fused Multiply-Add (FMA) instruction carried out in a regular matrix multiplication. More precisely, processing bitstream chunks can be implemented with three ALU instructions and three reads and one write as $sum+ = popcount(X \oplus Y)$. Furthermore, our method requires approximately 2.46 more samples to obtain the same error level but at the same time only requires a single hamming distance kernel to be run, contrasting with the three matrix multiplications required for computing ASS^TB . As such, using bit-wise operators to compute Hamming distances may theoretically result in an effective 13× speedup, based on the assumption that each compute unit can output one operation per clock. This potential speedup is high, especially as we have not been able to find a better implementation design that could match the performance of the population count instruction. All in all, our Mediterranean multiplication requires 1 XOR, 1 POPCOUNT and 1 ADD to process 64 planes in parallel, instead of requiring a single Fused Multiply-Add (FMA) [1] for every plane, but also requires approximately 2.46× more samples in the worst case to obtain the same variance as [1]. While we will not analyse energy efficiency, we also hypothesise that the underlying circuits to implement \oplus , integer ADDs and POPCOUNT operations are much simpler than FMA circuits and could consume less energy overall if implemented on a specialised circuit.

3.3.6 GPU Implementation

All kernels implementing the steps described above have been developed using the CUDA platform, with the kernel pipeline shown in 3.3. This environment provides us with optimized kernels for linear algebra operations like matrix multiplications and batched Fourier transforms already. In theory, most of the kernels we have implemented or reused (i.e., multiplication, binarizing, norm and regularization kernels) should run at a very fast pace as their complexity is $O(n^2)$. In practice, they require expensive $O(n^2)$ memory reads/writes, and therefore their execution time may not be negligible. The four FFT kernels execute slightly more floating-point operations as their complexity is $O(n^2 \log n)$. It must however be noted that while matrix multiplication kernels usually get close to the

peak floating-point GPU performance (assuming an $O(n^3)$ algorithm), GPU FFT kernels are usually less efficient as memory accesses and barriers create additional parallelisation challenges. Therefore, the FFT kernels will have an impact on performance for practical matrix sizes. In contrast, the Hamming distance kernel has a complexity of $O(kn^2)$ when processing the two n^2 bitstreams A' and B' . Bitstreams are encoded and manipulated as 64-bit integers in our code, which delivers better performance even though current CUDA hardware and instructions like POPCOUNT work natively on 32-bit integer only. As this kernel processes k separating planes, one can simply restrict computations to two kn submatrices belonging to respectively A' and B' . The kernel itself can otherwise be optimized similarly to GPU matrix multiplications kernels, just replacing floating-point instructions by 64-bit integer ones that perform population counts. A crucial optimization however is to limit memory transfers by loading submatrices to shared/local On-GPU memory before computing hamming distances from the submatrices.

3.4 Compressing Fully Connected Layers

This section demonstrates a first application of M^3 in the ANNs by showing that weights of Fully Connected layers can be compressed between $30\times$ and $100\times$ with little to no loss in inference accuracy. The requirements for pure floating-point operations are also down as our algorithm relies mainly on simpler bit-wise operators.

Artificial Neural Networks (ANN) are composed of layers of neurons, which are connected by weight matrices. These matrices need to be updated to optimal values using back-propagation. Matrix multiplication becomes the most time-consuming process in an ANN because the number of weights can be quite large, particularly when Fully Connected (FC) layers are used. The computation of the dot product between each input feature and each neuron in an FC layer necessitates a large number of matrix multiplications.

Therefore, it is not surprising that extensive work has been done to optimize this part of the learning process. One popular way to do so is to use lower precision calculations, with register sizes between 4 to 16 bits being implemented nowadays on specialised hardware. A popular format showing little loss in overall precision is bfloat16, which is currently preferred to the more standard IEEE 16-bit half-precision floating point format in ANNs applications [47].

While it is unclear what the best precision to use is and what are the reasons behind this, some researchers have been able to push it to a limit of 1 bit by using binary network

models. In *BinaryConnect* [2], Courbariaux and Bengio provided a training algorithm to restrict weights to $\{1,-1\}$, therefore improving storage by a factor of 32 (compared to FP32) with little impact on the error rate. As integer or FP prevision calculations were still required, XNOR-Nets were proposed in [3], allowing binarized (0 or 1) weights, operations, and input. Recently, a more flexible model using a variable number of bits (e.g., 1 to 3) has been proposed in [140].

As reducing precision reaches its limits quickly, other approaches have focused on compressing matrices as a whole. A survey of some of the approaches is provided in [46]. A typical way to compress matrices in scientific applications is to use a low-rank approximation, where the less important eigenvectors of the kernel are typically removed [141, 142, 143]. Novikov et al [144] improved on low-rank approximation by proposing tensor decomposition and demonstrated several-fold improvements in the compression of Fully Connected layers.

Random projection methods in ANNs have been popularised with kitchen sinks [136], making kernel methods more scalable. The *Fastfood transform* [137] and then *Deep Fried networks* [59] improved on kitchen sinks by reducing memory space and processing time. At the heart of these methods is a diagonal random matrix (thus reducing storage costs from $O(nd)$ to $O(n)$) combined with fast transforms that can approximate the weight matrix of a Fully Connected Network [59].

3.4.1 Using the Mediterranean Matrix Multiplication in ANNs

A natural question is whether this algorithm can still find a place in applications bounded by tensor operations, like artificial neural networks. Although using it directly for training models was not ideal due to the relatively large error propagation, we were able to successfully compress the fully connected layers of a ANN for data inference.

As an input X is progressing through the layers of an artificial neural networks it follows a sequence of matrix multiplications. For an FC layer, we can write this operation as $Y = WX$ where matrix W represents the weights an FC layer and X the input. Replacing a standard matrix multiplication with the M^3 one requires the calculation of WE and $E^T X$, where E is a random matrix of k columns, and binarizing these results similarly to algorithm 2. The pipeline for this is given in Fig. 3.3. We therefore need to calculate

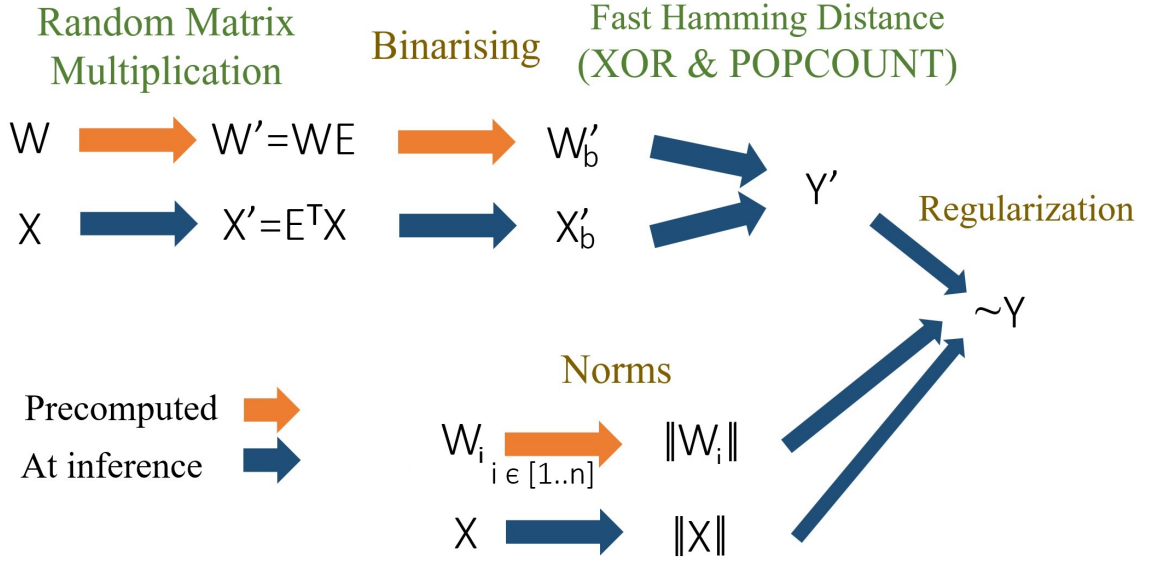


Figure 3.3: Pipeline of operations for estimating the product $Y \approx WX$. Operations needed during inference are shown in blue. Note that the entries of the constant matrix E follow a normal distribution and do not need to be stored.

first $Bin(WE)$ and $Bin(E^T X)$, with

$$Bin(X) = \{ Bin(X_i) \}, \text{ with } Bin(X_i) = \begin{cases} 1 & X_i \geq 0 \\ 0 & \text{Otherwise} \end{cases} \quad (3.27)$$

Note that the Bin operator, unlike in algorithm 2, is applied symmetrically to both matrices as we optimise code with XOR (\oplus) and POPCOUNT operators instead of using multiply and add operators. It is now trivial that $Bin(WE)$ can be pre-calculated before prediction phase as both matrices W and E are already known. Therefore, we can store the FC layer as a stream of binary data that, depending on the chosen number of hyperplanes k , may become significantly smaller than the original weight matrix W . We also need to store the vector of norms $\{\|W_i\|\}$ for the last step of the algorithm, but this space is almost negligible as this represents only a single floating-point value per row of W .

We would logically need to store a copy E as well to multiply it with the input X that is unknown at this stage. We do however have two options here given the random nature of this matrix. We can either store a random seed number and produce the same random number on the fly or store a single column vector of random numbers and rotate this vector to generate up to $n - 1$ extra columns of E , in the same spirit as the *Fastfood transform* [137] or *Deep Fried networks* [59]. It is important to notice that this gives us the opportunity to

compute the product $E^T X$ with a low number of floating-point operations by computing it in the frequency domain. As such, the complexity of the FC layer is mainly bounded by the operation combining the two binary streams obtained, which is represented in algorithm 2 by a standard matrix multiplication but can be implemented from simple XOR (\oplus) and POPCOUNT operators. These two operators require far fewer transistors than floating-point units and can be done on 32 or 64 bits at a time on modern hardware. Hence, the M^3 not only provides a way to compress FC layers during prediction, but also provides a simplified logic with far less emphasis on floating-point operations.

3.4.2 Training the Neural Network

Training needs to reflect the changes we have made to the way our FC layers work. This section describes how we ensure that a graph-based tool like TensorFlow is still able to learn patterns correctly once we are introducing our Mediterranean multiplication.

We initially train the neural network in a standard way. For some datasets (MNIST), we also augment the training dataset by adding rotated and translated samples to provide better results ([145, 146, 147]). We then consider that Convolutional Layers (CL), if present in the model, are trained and we focus on replacing the standard FC layers' multiplications with the new Mediterranean Matrix Multiplication. Indeed, FC layers are usually located at the end of a neural network. For each FC layer l we associate a unique constant random matrix E^l with entries following a normal distribution. This matrix is in our tests generated from a pseudo Random Number Generator (RNG) along with a unique seed and is created either on the fly (no storage requirement) or just from the stored columns $E_i^l \text{ (mod } n)=0$ of E^l as all other columns E_i^l can be calculated from a single-hop rotation of E_{i-1}^l . Rotating columns is preferable to just generating all the numbers from a RNG as it allows making use of the FFT algorithm to perform matrix multiplications efficiently.

From there, we can process forward, and back propagations as follows, assuming that the CLs are already trained and the output they produce will not change for the training dataset. The forward propagation is calculated by simply replacing the regular matrix multiplication by our M^3 variant in the FC layers, hence computing WE and $E^T X$ at this stage. We however keep the regular matrix multiplication algorithm when performing back-propagation because the M^3 pipeline is not differentiable, which has also the benefit of learning the extra level of error introduced in the forward pass without introducing

new errors in the back-propagation step. We then simply use the weight matrix W as the gradient for back-propagation in our TensorFlow implementation.

3.5 Results

This section presents the practical results obtained from the implementation of the M^3 method in a GPU, as well as its applications in neural networks. In order to compare the error performance of our method with [1]. We conducted experiments with different parameters and evaluated the results using standard metrics.

The implementation of the M^3 method on a GPU yielded significant improvements in computation time, enabling faster and binary operations for large matrices. Specifically, we observed a speedup factor of $10\times$ compared to standard multiplication.

Furthermore, we evaluated the performance of our method in the context of neural network training and found that it outperformed existing methods in terms of compressing. Specifically, our approach achieved an average improvement of 10% in the final test accuracy compared to the state-of-the-art method. Overall, the results demonstrate the effectiveness and practicality of the M^3 method in addressing the computational challenges of large matrix multiplication, and its potential for improving the performance of compressing neural network models.

3.5.1 Testing Environment

All tests are performed on an Intel 4770K processor running at 3.9GHz and coupled with 16 GB of RAM and an Nvidia GeForce GTX 1080TI Graphics card (11GB). TensorFlow 2.0 is used and runs on a Linux distribution.

3.5.2 Error analysis

To test the Mediterranean Matrix Multiplication, we create two matrices A and B using random number generators. Entries of these two matrices match a normal distribution with $\theta = 1$ and $\mu = 0$, except for one specific test where we analyse the effect of μ on the final error obtained. While it is difficult to allow for all possible distributions arising from specific circumstances, a normal distribution was thought to be representative of various processes. It must be noted that the content of the two matrices A and B itself should not affect performance tests but could potentially affect the accuracy of final approximation.

3. Efficient Fully Connected Layers in ANN

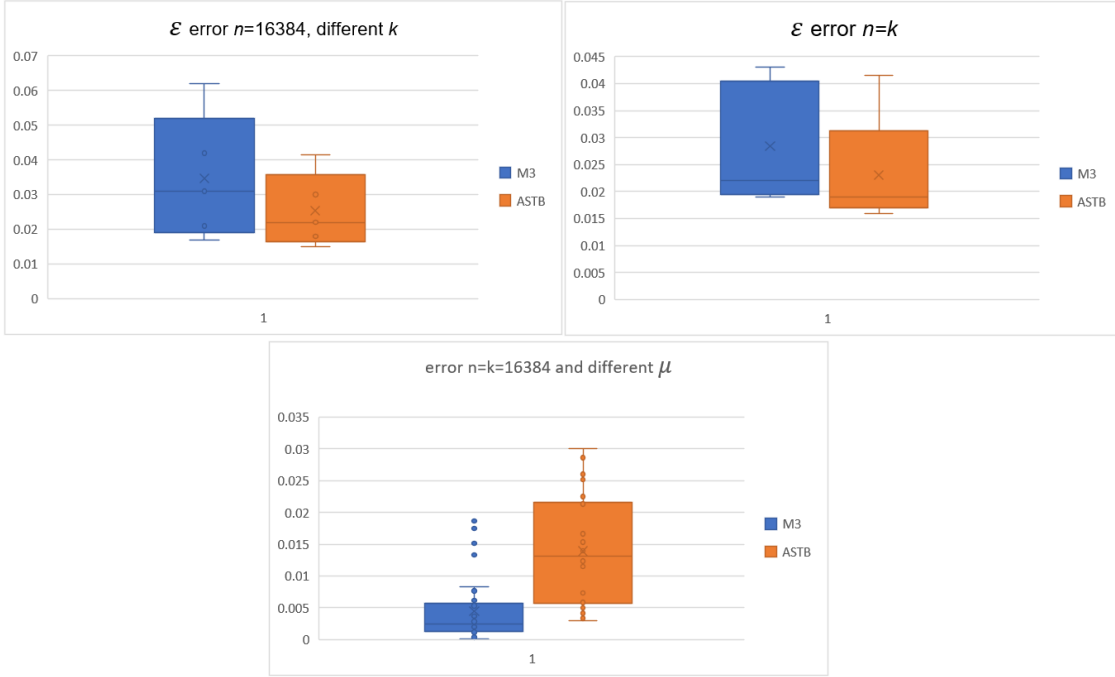


Figure 3.4: Comparison of error rates between M^3 method and [1] in approximating matrix AB , by varying the matrix sizes with $p = n$ or fixing n and changing the number of samples used. The chart also displays the final error rates of both methods after varying the μ parameter of the Gaussian random number generator, with $\sigma = 1$.

All floating-point computations are performed using the standard IEEE 32-bit precision. Errors in the approximation are measured after executing a full matrix multiplication and computing, where AB is obtained from a CUBLAS kernel call and C is the final estimation. Results are compared both to standard matrix multiplication and to $ASS^T B$, where S is a random sign matrix.

$$\|C - AB\|_F / \|A\|_F \|B\|_F \quad (3.28)$$

It can also be seen that Clarkson and Woodruff's method exhibits a lower error for the same number of iterations, with a measured error ratio close to $\pi/2$. This means that to get a similar error one needs to compute $2.46\times$ more planes with our algorithm, which still compares favourably as our pipeline is processing 32 or 64 bits per instruction vs. one. However, these variance results are obtained for the worst-case scenario where rows of A and columns of B are generated with $\mu = 0$, resulting in almost orthogonal vectors. By varying μ (Fig. 3.4e) so that these rows and columns become more correlated, the error produced by our technique is actually reduced dramatically and tends to 0, a

direct consequence of the Bernoulli trials (cf. Eq. 3.12). This is an important fact as it demonstrates the significant superiority of sampling angles to find correlated data over the original method of signed matrices.

As expected from our theoretical analysis, the error decreases linearly according to the square of the number of iterations when using normally distributed inputs (Fig. B.3). One can also observe some variance in the final error made when the μ parameter becomes large. This can be explained as follows. In general, the obtained errors are very close to the theoretical ones for $\mu = 0$ as there is usually very little variation over the error measured due to the large number of entries in the final matrices. However, signed matrices perform badly in that regard when rows of A and columns of B become similar as the entries of C tend to be highly correlated due to computations becoming very similar for all entries. Our algorithm may also more subtly inherit this problem, but as the error tends to zero, so does the error variance, which makes it more stable than the standard $ASS^T B$ approach. It must finally be noted that we used $n = k = 8192$ in this particular test as using single-float precision provides limited accuracy which may affect the error calculation. This comes to light in Fig. 3.4 where the error ratio between the two techniques starts differing noticeably from the theoretical $\pi/2$ for $n = 16384$ but in the favour of our algorithm.

3.5.3 M^3 CUDA Implementation

This section provides test results for a simple CUDA implementation of the multiplication (Fig. 3.3) of two random matrices with varying parameters. Kernels are currently optimized for power of two sizes, with a minimum size n of 256. A maximum matrix size of 16384 (1GB per matrix) has been tested, due to GPU memory limitations. Results (single floating-point precision) are compared both to standard matrix multiplication and to $ASS^T B$, where S is a sign matrix. Computing $ASS^T B$ just requires three CUBLAS calls and therefore can be considered optimal and optimized.

Table 3.3 summarises the factors affecting performance between the use of signed matrices [1] and the proposed method. While our technique requires more samples to estimate dot products at angles close to $\pi/2$ due to a higher variance, it also benefits from most operations being done by binary operators that can process 64 bits at a time (as implemented) to calculate the Hamming distance. However, a breakdown of the performance of the different components of the pipeline (Fig. 3.6) also shows

3. Efficient Fully Connected Layers in ANN

n \ k	256	512	1024	2048	4096	8192	16384
256	0.18	0.23	0.62	1.89	6.39	22.74	85.2
512		0.24	0.61	1.9	6.65	23.43	87.97
1024			0.7	2.25	6.64	23.5	88.17
2048				2.49	7.6	26.92	98.42
4096					9.11	32.73	115.46
8192						40.61	154.54
16384							238.02

(a) Timing measured in ms for M^3

n \ k	256	512	1024	2048	4096	8192	16384
256	0.29x	0.48x	0.45x	0.47x	0.50x	0.45x	0.42x
512		0.92x	0.88x	0.91x	0.85x	0.84x	0.83x
1024			1.31x	1.28x	1.79x	1.74x	1.65x
2048				2.27x	2.87x	3.01x	3.13x
4096					4.79x	4.49x	5.56x
8192						7.43x	8.68x
16384							11.45x

(c) Ratios between calculating $Ass^T B$ and M^3

n \ k	256	512	1024	2048	4096	8192	16384
Time (ms)	0.022	0.072	0.28	1.887	13.79	109.3	906.64

(e) Timings measured in ms to calculate the product of two n^2 matrices with CUBLAS.

n \ k	256	512	1024	2048	4096	8192	16384
256	0.05	0.11	0.28	0.9	3.18	10.14	35.69
512		0.22	0.54	1.72	5.63	19.77	72.95
1024			0.92	2.87	11.9	40.92	148.8
2048				5.65	21.81	80.9	308.36
4096					43.65	146.92	642.17
8192						301.85	1340.7
16384							2724.6

(b) Timing measured in ms for $Ass^T B$

n \ k	256	512	1024	2048	4096	8192	16384
256	0.12x	0.20x	0.18x	0.19x	0.20x	0.18x	0.17x
512		0.37x	0.36x	0.37x	0.34x	0.34x	0.34x
1024			0.53x	0.52x	0.73x	0.71x	0.67x
2048				0.92x	1.16x	1.22x	1.27x
4096					1.94x	1.82x	2.25x
8192						3.01x	3.52x
16384							4.64x

(d) Precision-normalised ratios between calculating $Ass^T B$ and M^3 algorithm - i.e. results obtained on the left are divided by $(\pi/2)^2$

n \ k	256	512	1024	2048	4096	8192	16384
256	0.12x	0.32x	0.45x	1.00x	2.16x	4.81x	10.64x
512		0.30x	0.46x	0.99x	2.07x	4.66x	10.31x
1024			0.40x	0.84x	2.08x	4.65x	10.28x
2048				0.76x	1.82x	4.06x	9.21x
4096					1.51x	3.34x	7.85x
8192						2.69x	5.87x
16384							3.81x

(f) Ratios between calculating the matrix product with CUBLAS and M^3

Figure 3.5: Performance comparison between our method, $Ass^T B$ [1] (b, c & d), and a direct matrix multiplication of A and B (e & f). As we mentioned, to achieve the same level of error, M^3 necessitates about 2.46 times as many samples.

that operations with a theoretically negligible complexity have a significant impact on performance, especially when the matrix size is small.

Figure 3.5 shows the performance gains obtained with our new technique and compare them to both a standard multiplication and Clarkson and Woodruff approach [1]. As expected, some significant speedup is obtained for large matrix multiplications, where our method is measured to be up to 4.64 \times as fast as the use of signed matrices [1] after correcting for the higher variance per iteration, and as implemented. When compared to a standard matrix multiplication, we can be up to 10 \times faster, depending on the maximum error tolerated.

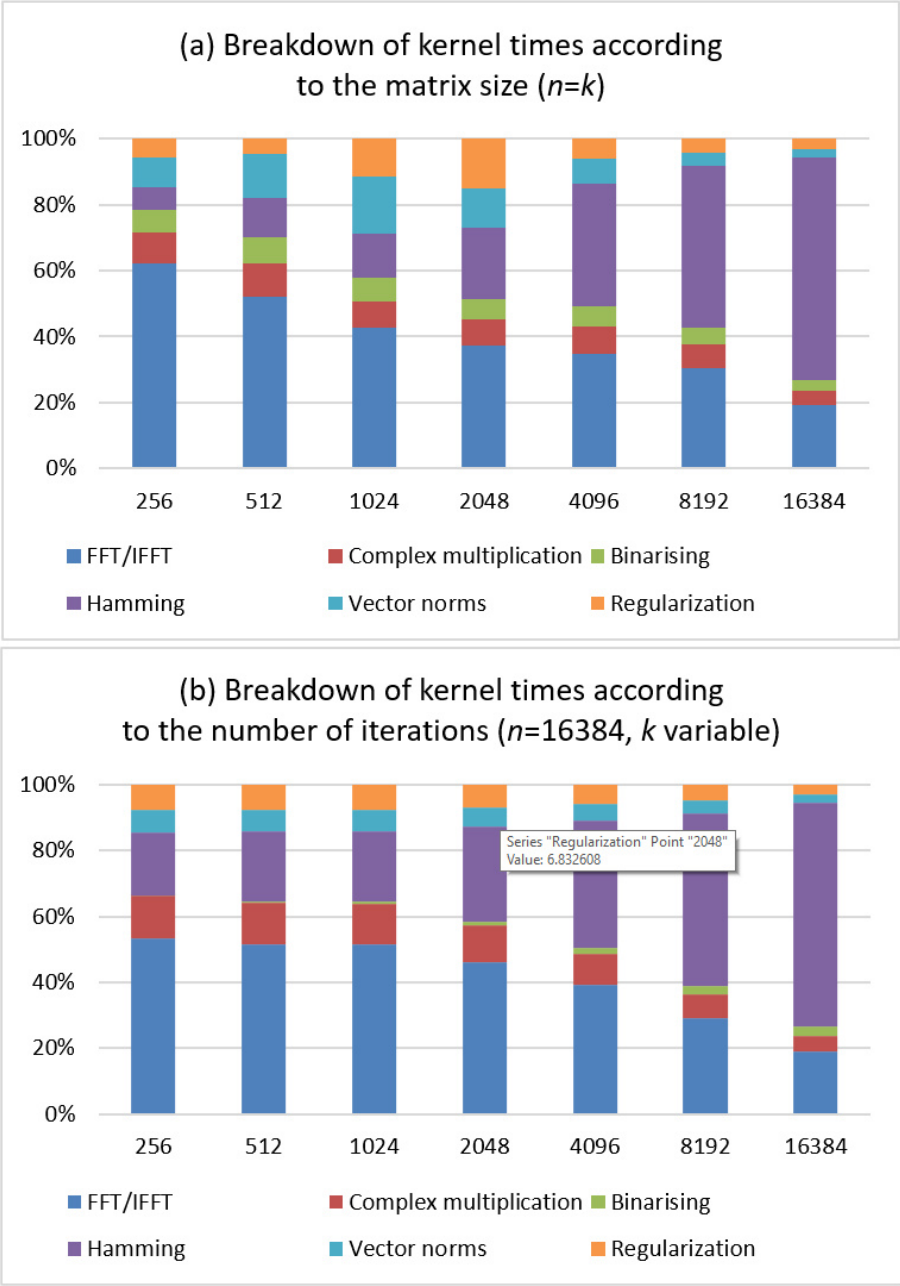


Figure 3.6: Breakdown of performance for the various kernels used in our GPU implementation. It shows the percentage of operations required by each group in separate columns.

3.5.4 Training Artificial neural network with M^3

We study in Figs. 3.7, and B.1 the effect of directly replacing the standard matrix multiplications with our M^3 version in the training of two ANN models with respectively

3. Efficient Fully Connected Layers in ANN

Table 3.3: Comparison of the different practical factors influencing the performance of the signed matrix approximation [1] and our own Hamming distance kernel. (GPU: NVIDIA 1080Ti)

Factor	Signed Matrices [1]	Our Approach
Theoretical Complexity - Square matrices	$O(n^2)$	$O(n^2)$
Base variance / Sampling constant	1 for all angles (Fig. 3.4e)	$\pi^2/4 \approx 2.46$ @ angles $\pi/2$ and $-\pi/2$ 0 @ angles 0 & π
Core arithmetic instructions per sample per entry	1 Fused Multiply and Add	1 POPCOUNT, 1 XOR, 1 ADD (32- or 64-bits instruction)
Other performance factors	3 matrix multiplications needed. 6 floating-point operations per sample per entry.	3 binary operations per sample per entry. 32 or 64 bits processed per instruction. Extra operations of lower theoretical complexity taking a non-negligible time (Fig. 3.6).
Peak throughput as measured	~ 10 TFlops	~ 82 TBits/s as implemented
Kernel implementations	CUDA libraries only (3 matrix multiplication)	Own Hamming distance kernel (Fig. 3.3)

two and three dense layers of the MNIST models (no data augmentation) being replaced. The replacements are made in either the forward pass, the backward pass, or both passes. Unless stated otherwise, we use a batch size of 1024 to be representative of a real-world application as our algorithm can only replace Matrix-Matrix multiplications efficiently (speed-wise), but not Matrix-Vector multiplications. This large batch size does not affect the convergence rate but may be impractical in some situations as this increases the memory requirements.

Fig. 3.7 demonstrates that both convergence rate and accuracy improve as we are using more planes for the approximation. The algorithm converges more or less closely to the reference model as expected. Combining both forward and back-propagation passes also logically results in a lower accuracy than only using one of them.

More surprising is the fact that just using the M^3 in back-propagation gives particularly good results, especially for $k = 1024$, but the forward pass requires a much higher precision and leads to an unsatisfactory training (Fig. 3.7c&f). Also, this may only be workable on shallow networks as the approximation error is likely be amplified with the extra layers.

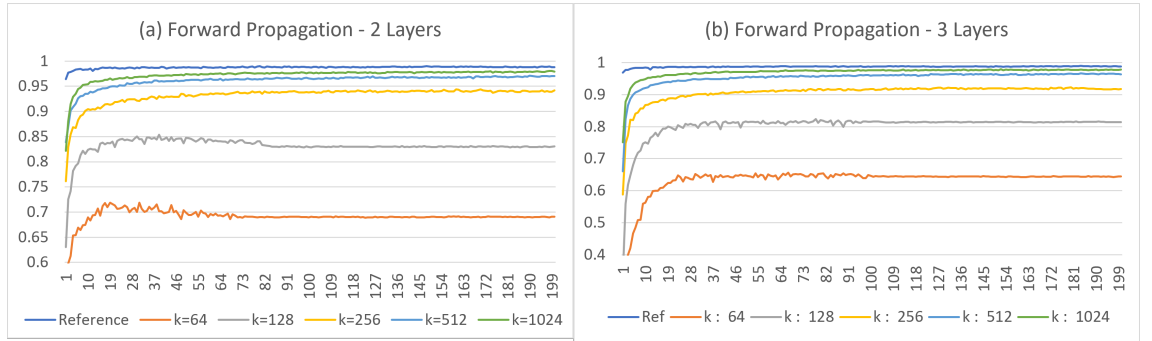


Figure 3.7: Effect of replacing the standard matrix multiplication with the M^3 version for training the MNIST datasets in all but the last Fully Connected layer. The horizontal represents the number of epochs while the vertical axis represents the accuracy obtained on the testing dataset. We study forward-only replacement cases. The number of planes k used is the same for each layer and each pass in a single experiment (batch size used is 1024). Getting good results in the forward pass with M^3 requires a much larger number of planes.

Also, in this particular example, the experiment entails multiplying two square matrices with $n = 1024$ and possibly k set to 1024 but Fig. 3.5 tells us that our current CUDA implementation of the M^3 would only perform at $0.4\times$ the speed of a standard matrix multiplication, and therefore would be impractical.

3.5.5 Compression of Fully Connected layers

We tested our Mediterranean diet on the FC layers of standard neural network models like VGG16, and mainly concerned in observing the effect of replacing the standard matrix multiplication in the pipeline of these models with our Mediterranean Matrix Multiplication. As such, the low error obtained in our tests, may be beaten by other networks not using FC layers. We use VGG16 on three datasets (CIFAR10, CIFAR100 [147], CINIC-10 [146]) and use two other standard models for MNIST [145]. Especially, VGG16 has 3 FC layers which represent 56% of the total size of the model when having 3×32^2 input images.

Details of the networks used are given in tables in the appendix B. For training, we have used a weight decay of 5×10^{-4} , batch normalisation and drop-out between convolutional layers. We have used ReLU activation functions after each layer, except for the last layer where a SoftMax function has been used. In addition, data augmentation was created by rotating images 15° for CIFAR and CINIC datasets and 8° for MNIST. Furthermore, images were shifted by 10% randomly on both x and y axes. Results with and without

3. Efficient Fully Connected Layers in ANN

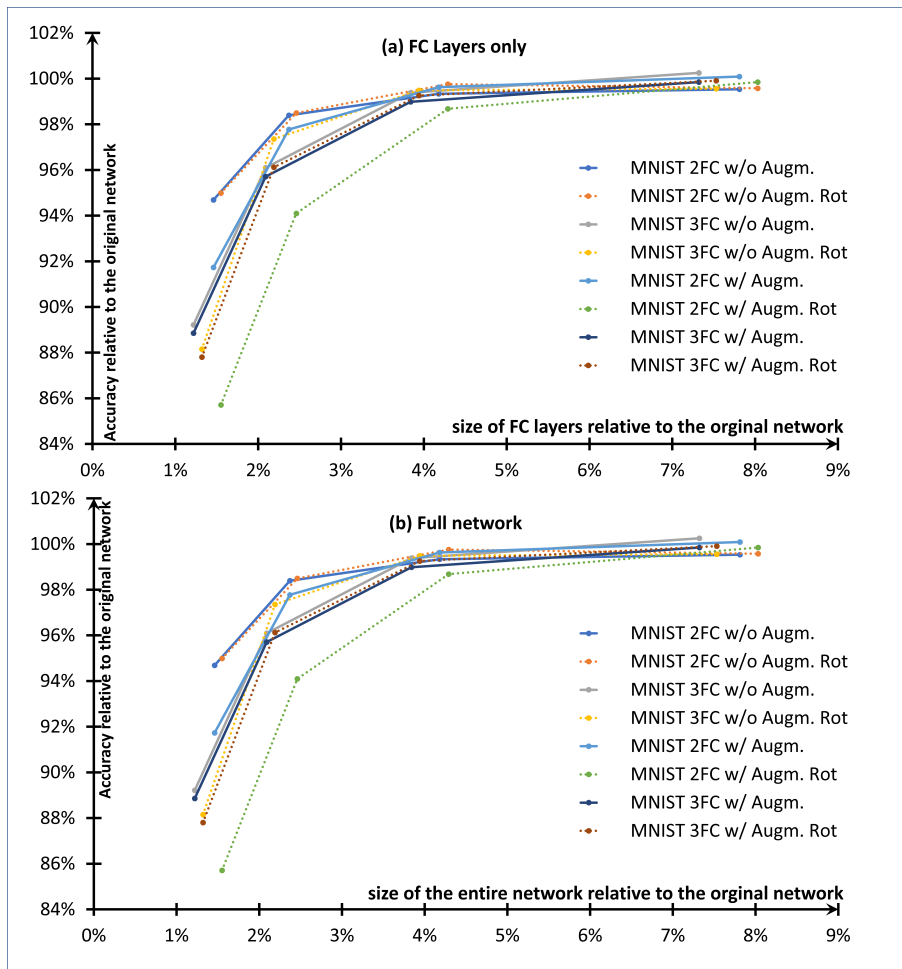


Figure 3.8: Convergence of accuracy according to the number of hyperplanes used. The horizontal axis displays the compression obtained for just the internal Fully Connected layers (a) and for the neural network as a whole (b). The vertical axis displays the obtained accuracy of the network relative to that of the original, unmodified network – with 100% meaning that the compressed network has the same accuracy as the original one. All network models have been tested with 256, 512, 1024 and 2048 hyperplanes as represented with continuous or dashed lines.

augmentation are given for MNIST. Experiments have been carried out with a varying number k of hyperplanes. The size of the compressed layers is calculated from the size of the bit representation of the layers, the norms $\|W_i^l\|$ of the weight matrix rows, and the seed value which is negligible. As our random numbers are calculated from the seed value, they do not need to be stored. We have included the storage requirements for the tests where random numbers are further obtained by rotation of columns of E (i.e., Toeplitz case) as this may have some practical implications, including removing the need

Table 3.4: Comparison of the best compression rates with Binary-Connect [2] and XNOR-Net [3]. Our technique allows changing the compression rate to favour either compression or quality.

Dataset	MNIST		
Technique	Accuracy	size	
Binary-Connect	98.78%	0.36 MB (3.26%)	
XNOR-Net	97.65%	0.38 MB (3.40%)	
$M^3, k = 2048$	99.37%	0.50 MB (7.24%)	
$M^3, k = 1024$	98.61%	0.43 MB (3.87%)	
$M^3, k = 512$	97.92%	0.24 MB (2.18)	

Dataset	CIFAR10		
Technique	Accuracy	size	FC-layers only size
Binary-Connect	90.10%	1.67 MB (3.13%)	1.13 MB (3.13%)
XNOR-Net	91.24%	1.68 MB (3.15%)	1.14 MB (3.15%)
$M^3, k = 2048$	91.81%	17.97 MB (33.59%)	0.51 MB (1.43%)
$M^3, k = 1024$	91.29%	17.72 MB (33.12%)	0.26 MB (0.73%)
$M^3, k = 512$	90.88%	17.59 MB (32.88%)	0.14 MB (0.38%)

to calculate these random numbers on the fly when inferencing. Tables in the Appendix B illustrates the results for the four datasets and Fig. 3.8, and B.2 illustrate the convergence to the original network according to the number of planes.

All experiments on VGG16 showed that the FC layers can be compressed to around 1% of their original size without any meaningful loss of accuracy. The compression rate for the MNIST models is also very significant at around $50\times$ without a significant degradation of accuracy, although lower than VGG16 results in general. We conjecture that as VGG16 FC layers are larger, they become easier to compress. It however looks like more hyperplanes are needed when using rotated random numbers. While the compression rates are still very good, we still need two to four times more space to see no difference with the original network. We do not know at this stage if this result can be improved upon, by for instance improving the random number sequence.

Obviously, compressing the FC layers translates into a significant reduction in the size of these models. For the MNIST models, as all the layers are FC layers, the model can easily be compressed up to $25\times$ without any loss of accuracy. Also, while the original VGG16 network size was mostly dictated by the FC layers, the Mediterranean diet just make them negligible in size, leading to a compression of VGG16 by more than 2. Finally, storing the random numbers does not have a big impact on the memory footprint but still

allows faster implementations and less reliance on floating-point calculations as this part of the pipeline can be now ensured by a fast convolution performed in the Fourier space.

We finally compare the compression rates we obtained (cf.3.4) with two other techniques that are Binary-Connect [2] and XNOR-Net [3]. The table shows that similar levels of compression and accuracy can be obtained by the three techniques for Fully Connected layers (The best result in 5 run). While Binary-Connect and XNOR-Net only allow a single bit of information, our approach is more flexible as it allows us to choose any random number of planes we wish and therefore allows parameterised compression levels. However, we do not propose yet a way to compress convolutional networks, and therefore the two techniques cited perform much better overall in CIFAR10. The similarities in the results make us believe that these two methods may implicitly learn the M^3 pipeline while training.

3.6 Conclusion

This chapter has firstly proposed and analysed the Mediterranean Matrix Multiplication, a new, fairly simple, unbiased algorithm for approximating matrix multiplications. While the theoretical bound obtained is optimal and similar to the currently best-known randomised methods ([1]), it does offer increased convergence when dealing with non-orthogonal rows and columns. It is also amenable to various bitwise optimizations that accelerate computations greatly as the cost of combined XOR/POPCOUNT units could be significantly lower than that of FMA units in terms of area and energy consumption.

We have also demonstrated a Mediterranean diet algorithm for Fully Connected layers, allowing a compression rate for these layers as high as $100\times$ in the case of VGG16, and so without a drop of accuracy. This result is similar or better to similar techniques published in the area, with the extended benefit that most of the operations during prediction can be performed on binary streams using a combination of XOR and POPCOUNT operators. This should allow simplifying the architecture of embedded inference processors and as such, significantly reduce their energy consumption. Our CUDA implementation indeed demonstrates that a significant speed-up can be obtained for large matrix sizes based mainly on the sole use of these operators.

All things considered, the Mediterranean diet is a technique closely related to recent research done in the area of random projections, low-rank approximations, and binary nets, and combining all the benefits of these methods into one framework. Further

investigations may include tweaking the random number generator, compressing other types of layers and networks, studying training scalability for larger networks, or even accelerating other scientific problems.

Chapter 4

Explainable AI

4.1 Introduction

In recent years, Explainable AI (ExAI) has received a lot of attention [148]. ExAI methods aim to explain how a black-box model makes decisions [149]. One of the aims of providing explainability in Sub-symbolic AI models is to ensure that the predictions made by algorithms and the input data that trigger those predictions can be understood [150]. ExAI focuses on creating prediction models that produce explanations that are understandable by humans for Sub-symbolic AI methods.

From a data science perspective, equipped with its “explanation power,” ExAI provides deeper insights from data [148]. By explaining why a model makes a certain prediction, one gains knowledge about the underlying data used to build the model. In the current development in ExAI software is fragmented, and implementations are scattered in multiple libraries written in different programming languages, predominantly intended for data science developers rather than domain experts. The lack of easy-to-use ExAI tools also hinders the further development of ExAI. Moreover, having a model and describing the decision made by this model is a common request from many end-users who are not machine learning experts. Being knowledgeable about why a given decision was made can be extremely helpful to a wide variety of professions, especially where risk assessment and analysis are critical, like in financial or medical areas. In recent years, the focus has been put on Explainable AI to meet this demand.

In Explainable AI, understanding decisions from black box models work essentially by observing the effect of small changes to a given input and their effect on the output. This

allows for identifying the features in the input space that have the strongest impact on the output. Nowadays, this approach to Explainable AI is implemented inside various open-source frameworks like [151], [152], and [153], and makes sub-symbolic AI more accessible to non-machine-learning experts. However, these approaches have some limitations. For instance, results can sometimes differ after each execution.

In this chapter, we first present ExMed, a self-contained ExAI toolkit for domain experts that performs ExAI analysis for prediction models. With its simple user interface, it supports both *global* explanations presenting patterns of the entire dataset and *instance* explanations that are local to individual predictions, for both classification and regression tasks. Although various, ExAI techniques have been proposed in recent years - for example, a good overview of these techniques is presented in [154] - we focus on feature attribution explanation techniques [77] due to the transparency of their explanations, their computational effectiveness, and general popularity. We also present two real-world case studies that demonstrate ExMed's functionalities. In case study I, a COVID-19 transmission study reveals how different COVID-19 control measures were used and impacted transmission rates. In case study II, we examine lung cancer patient life expectancy using the Simulacrum dataset.¹ Through the two case studies, we illustrate how ExMed can be used for making predictions and generating explanations.

Finally, we introduce a method that seeks to identify the decision boundary of a specific model. This approach builds upon previous research [82] by determining the closest point to the boundary using the loss function of our model. In the results section, we showcase the effectiveness of our Neighbour Migrating Generator (NMG) in both global and local contexts using three datasets: a synthetic mathematical model, the Iris dataset, and the Pima Indians Diabetes dataset.

In general, the goal of this chapter is to enhance the usability of ExAI by presenting a new framework for non-expert users and offering a novel approach to explain a model's decisions and identifying the input dimensions that have the greatest impact on those decisions.

4.2 Literature review

Individual predictions made by experts rely on the fact that humans can learn enough about which parameters have the biggest influence in the feature space, or why a model

¹<https://simulacrum.healthdatainsight.org.uk/>

inferred a particular decision. Many ways for individual predictions are explained using local interpretation tools, such as Individual Conditional Expectation (ICE) [155], Counterfactual Explanations [82] [83], Local Interpretable Model-agnostic Explanations (LIME) [76], and SHAP methods [81], which explain a model by assigning a weight to each dimension of inputs.

Due to the involvement of data sensitivity in the medical domain, there is a necessity of gaining human trust towards ML applications [156]. Therefore, we have seen a recent surge in the production of explainable results using state-of-the-art models such as Local Interpretable Model-Agnostic (LIME) [157] and SHAP [77] to supplement the outputs provided by black-box algorithms. Much work has shown the intent of expanding Explainable AI (ExAI) through new prediction model architectures [158, 159].

ICE [155] is a method for plotting the global effect of modifying each feature individually. ICE can modify one feature while retaining the same value for the rest of the feature space. It then displays the changes in predictions according to the value of the target feature. However, comprehending the differences between individual lines in the plot can be difficult at times. This can be resolved by centring the curves at a point and comparing the predictions to this point, which is known as Centered ICE (c-ICE). Nevertheless, a limitation of ICE is that it cannot allow one to see and detect any association between features [160].

LIME [76] generates a linear approximation of the local decision boundary by considering a trained model as a black box since the reasons behind a prediction for a given input are not understandable by users. It develops local surrogate models to explain the provided input and the decisions of the trained model. The explanation process consists of three steps. First, LIME alters the feature space values (creating a new training set) and, by using Gaussian kernels, associates a weight to the new sample points based on their distance to a given input whenever a new instance is passed to the LIME model. The new samples are then sent to the trained model to approximate the local decision boundary, which would be used to map the inputs and outputs together to extract explanations. LIME explains a model in a feature space locally. However, Laugel et al. have shown [161] that LIME is highly dependent on some kernel settings because it relies on the correct definition of the local Neighbourhood. This problem is not only a parameter or sampling distribution issue but has a significant impact on the relevance and quality of the local black-box decision boundary approximation and, thus, on the meaning and accuracy of the given explanation. Many studies have investigated the development of

new local training sets [162], but there are certain difficulties to their solutions, such as losing sight of a crucial notion like the tangent or inspecting small Neighbourhoods without taking linearity into account in the ML function.

SHapley Additive exPlanations (SHAP) [81], based on cooperative game theory, assigns an importance value to each feature. Cooperative game theory aims to determine fair compensation for all participants based on their contribution. SHAP computes the Shapley score by replacing a particular instance value with new values and then applying the trained model’s predictions to the new dataset. While analysing all input features can be costly, we only need to investigate a subset of them, resulting in slightly different outcomes each time. However, a limitation of SHAP is that the order of features for a given instance might affect the prediction.

Counterfactual Explanations, unlike the previous techniques mentioned, aim to identify the smallest changes needed in a sample to modify the model’s prediction. This differs from most other approaches that vary in how they define the loss function. Wachter et al. [82], for instance, minimise the loss function as shown in equation 4.1.

$$\arg \min_{x'} \max_{\lambda} \lambda (\hat{f}(x') - y')^2 + d(x, x') \quad (4.1)$$

The Manhattan distance (weighted with the inverse median absolute deviation) between the original input and the modified one is added to the square of the L^2 norm of the differences between their predictions. The weight λ indicates the influence of the input instance, with a higher value indicating a preference for infrequent alterations. However, this approach has difficulty handling categorical features. To address this, Dandl et al. proposed a new loss function that minimises a four-section loss while counting the number of altered features using the Gower metric and a counter.

$$L(x, x', y', x^{obs}) = \min(o_1(\hat{f}(x'), y'), o_2(x, x'), o_3(x, x'), o_4(x', X^{obs})) \quad (4.2)$$

The Manhattan distance between $\hat{f}(x')$ and y' is o_1 , while the Gower distance function (o_2) calculates the distance between x (original sample) and x' (close target sample on the decision boundary). o_3 calculates the number of features that have been altered. Finally, o_4 calculates the distance between the estimated sample and the nearest observed sample in the train set.

The current ML pipelines are primarily focused on identifying a specific problem, and these pipelines often need to be rebuilt when new data is available. Without a

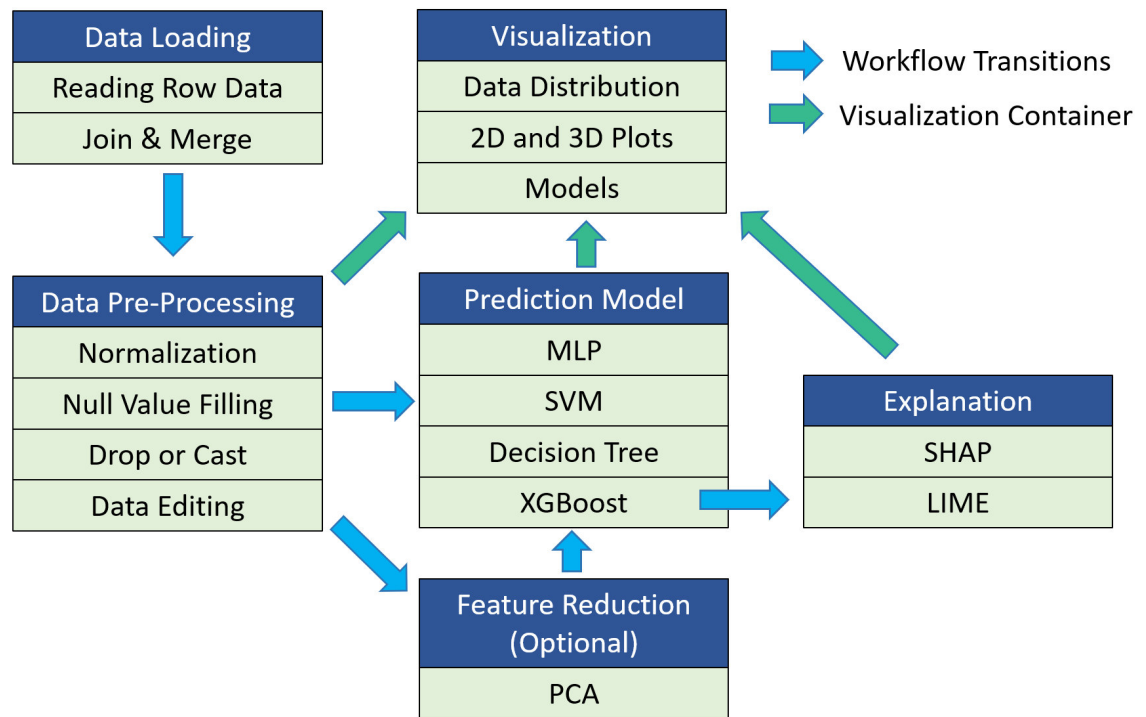


Figure 4.1: ExMed Activities. ExMed provides the user with a sequence of simple actions, including loading, merging, and editing data, and creating prediction as well as explanation models. Various visualisation techniques are supported in several stages of this pipeline.

basic understanding of ML, accessing ML applications can be challenging without the assistance of domain experts. Furthermore, there is a need for more human input and interaction in machine learning models to support explainability. To address this, research has been conducted to develop explainable architectures, and web-based interfaces have been created to support image segmentation with a user-friendly interface. One such interface is discussed in [163].

Several open-source applications have been created to simplify the implementation of sub-symbolic AI in various datasets, such as tools like Fiji, Weka, and others [164, 152, 165, 166]. In the field of biology, a significant amount of data is stored as images, and Fiji [164] is an example of an open-source tool designed specifically for biological image analysis. It allows for the quick prototyping of image processing algorithms, and scripting languages have been developed to facilitate this process. Fiji also simplifies the conversion of cutting-edge algorithms into ImageJ plugins that can be shared with users through a built-in update system, as shown in Figure 4.2.

Massive Online Analysis (MOA) is another open-source framework for classification and clustering of data streams that require quick predictions [165]. In MOA, data arrives

4. Explainable AI

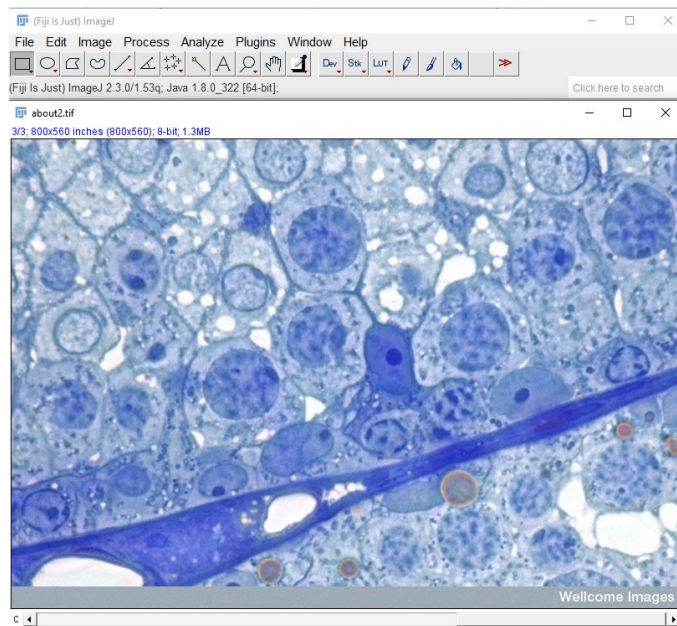


Figure 4.2: Working with biology photos is simple and quick using Fiji’s many plugins. A biological image with a filter applied to it is shown in the figure.

in a stream of instances, where each instance can be used as a test before being used as a training instance. MOA also offers features such as outlier detection and clustering of data. This framework provides many algorithms for streaming algorithms, as well as other useful implementations such as frequent pattern mining and change detection algorithms. Figure 4.3 displays one panel of MOA. The framework also provides a real-time report with various metrics for model development and data input. Both Fiji and MOA are not specifically designed for the kind of dataset that we focus on.

WEKA [152] is another workbench designed to combine different ML libraries for supporting various analyses through a graphical user interface. WEKA was developed at the University of Waikato and provides fast access to information within datasets, allowing the selection of areas of interest. This framework can perform data preparation tasks such as casting, replacing, and numeric transformations, as well as apply machine learning algorithms such as regression, classification, and clustering. However, WEKA does not support merging and concatenation of datasets, which is often required when introducing new medical data. WEKA is a well-known software in the field of machine learning and comes with ample documentation. The interface, as shown in Fig 4.4, is straightforward and provides options for data preparation and visualisation.

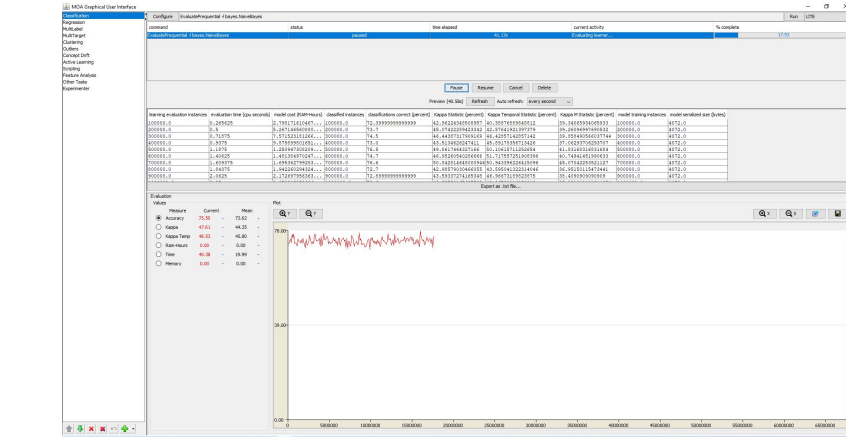


Figure 4.3: Massive Online Analysis is a framework for running machine learning algorithms on data series databases.

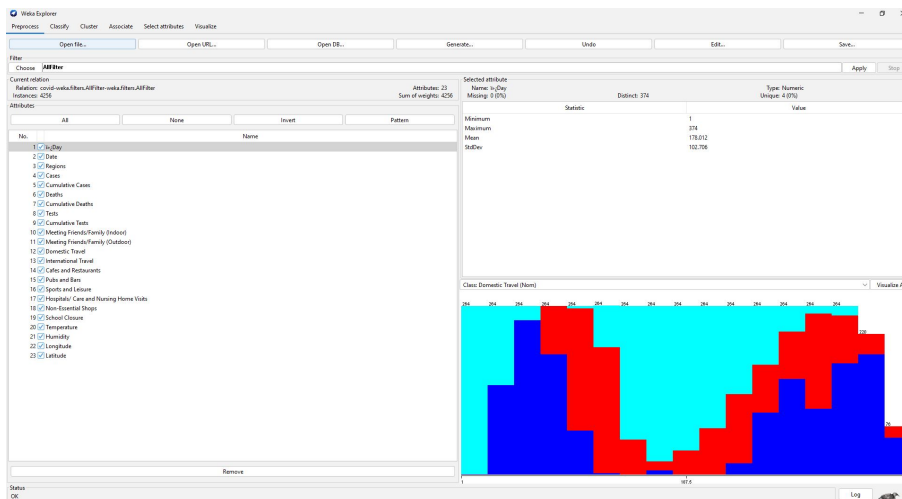


Figure 4.4: The University of Waikato in New Zealand created the Waikato Environment for Knowledge Analysis (Weka) since it has a free software license.

In ExMed, we have incorporated various features available in the other software mentioned earlier, such as visualisation. However, we have also prioritised simplicity and user-friendliness to ensure that our software does not require extensive training to use. Additionally, we have included essential data preparation steps that are commonly required for medical datasets. Moreover, our software includes a unique feature of providing explanations that is not found in other existing software.

The NMG approach is comparable to the techniques explained earlier in the sense that we consider the model as a black box without having access to its internal structure or weights. Nevertheless, our main focus is to recognise the closest Neighbouring instance in a distinct category. Our work is similar to the two approaches described above, where a separate model is trained for each case to determine the smallest modification that can change the predicted class label. However, it differs from them in how we define the loss function. Our loss function allows the model to change freely in the feature space. Moreover, we have introduced a new method of finding the most influential features for a given model. We train the model's NMG representative of the decision boundaries with all the training set samples (instead of a single sample) and analyse the modifications. Additionally, we allow the model to decide the required amount of change by self-adjusting the kernel settings.

4.3 ExMed Workflow

When sending medical data to an ML model, human errors and noise can reduce the quality of the results. Moreover, as one of the leading challenges in medical data analysis is to aggregate data from multiple data sources for performing joint analysis [167], it is crucial for medical data analytic frameworks to support the pre-processing stage. Our new application ExMed [168] addresses both challenges and makes the integration of data pre-processing tools easier in order to minimise error and increase the baseline performance of the ML model.

ExMed's main functionalities and architecture illustrations are shown in Figure 4.1 and B.6. ExMed implements a wide set of tools to load, process, predict, and explain data. Its back-end design is modular so that more tools can be easily added at a later stage. ExMed can accept the most common data files as input (e.g., Excel, CSV, SAS, and XPT files) with the possibility for easy integration of new file types. Input data can be then combined through classic database join operators, whether or not a common key

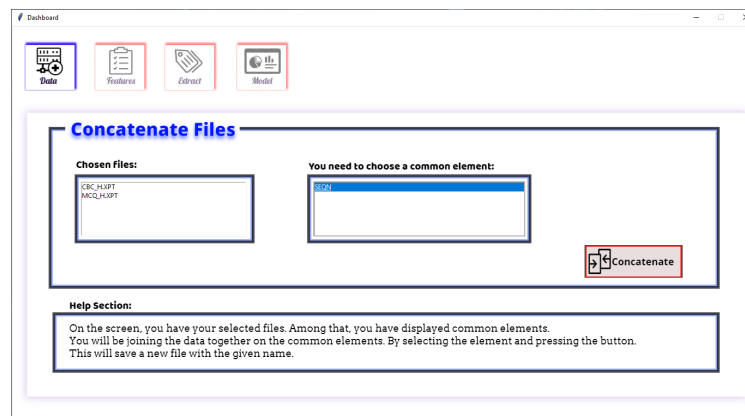


Figure 4.5: ExMed interface for some of the main activities as describe in Fig. 4.1. Data from various supported file types is loaded, with the option to combine this data with other datasets.

exists. This gives users the potential to create larger datasets from different file types – potentially collected from different sources – rapidly. Cells, rows, columns, and data types can be edited by the user directly within ExMed, allowing greater freedom for data manipulation and quality checks. Data validation is supported by various visualisation tools included with the interface. These tools can represent data trends in many ways (see Fig B.4 for a few examples) to provide fast data insight to users and can be applied to either the entire dataset or just part of it.

Creating a model can easily be done by selecting a target label (i.e., column) in the interface. Non-categorical columns selected from inference can be edited and transformed into a category (e.g., using a thresholding operator, Fig. 4.5, and B.5) prior to creating the model. Once data has been finalised and validated, and a target label has been created, a range of machine models can then be applied, including SVM, Random Forest Classifier, MLP Regression, and XGBoost. There is also an option to apply dimensionality reduction by pre-processing data with an automated Principal Component Analysis (PCA) process. Moreover, the result of the PCA can be visualised in 2D or 3D from the two or three largest eigenvectors respectively.

To interpret data, individual models have their own functions to offer specific explanations. SHAP dot plots, SHAP bar plots, SHAP dependence plots and LIME plots can be used for this purpose. This will show different ways of explaining the reasoning behind the results. We explore explanations and ExMed capabilities on two case studies in sections 4.3.1 and 4.3.2. LIME and SHAP adhere to ML local explainability requirements for patient instances; expressed as a necessity from clinicians [156], whilst also producing

global explanations. To invoke trust, we provide explanations from both LIME and SHAP as both models see a lack of ubiquity in feature priority, but may still provide valuable insight into the data as these methods still often see the same trend in feature attribution [169]. Also, feature attribution algorithms allow for a better understanding of data, as we are able to visualise bias, error, and gain insight into patient instances.

4.3.1 Case Study I: COVID-19 Control Measures

In this case study, we demonstrate how ExMed can be used in investigating relative effectiveness of COVID control measures used in the UK.

From the Public Health England website², we collect daily infection numbers reported across 9 regions in England *East Midlands, East of England, London, Northeast, Northwest, Southeast, Southwest, West Midlands, and Yorkshire and the Humber*, as well as and the other three nations in the UK: *Wales, Scotland, and Northern Ireland*. Non-pharmaceutical control measure data were collected based on UK’s COVID policies as summarised in Table 4.1. Data are collected from various sources including the Wikipedia and major news agencies such as BBC. Control Measures are coded based on the level of severity (“High”, “Moderate” or “Low”) for all control measures excluding non-essential shops and School closures, which are coded as binary choices (“Open” and “Closed”). Temperature and humidity data obtained from the weather website Raspisaniye Pogodi Ltd³ were also included. This represents a total of 4,257 data points that were collected between February 2020 and February 2021.

We study the effectiveness of control measures by observing their impacts to the virus transmission rate R_t . Specifically, from daily infection numbers, we estimate R_t using the method reported in [170, 171]. R_t is one of the most important quantities used to measure the epidemic spread. If $R_t > 1$ then the epidemic is expanding at time t , whereas if $R_t < 1$, then it is shrinking at time t . A *serial interval distribution*, which is a Gamma distribution $g(\tau)$ with mean 7 and standard deviation 4.5, is used to model the time between a person getting infected and he/she subsequently infecting another person on day τ . The number of new infections c_t on a day t is computed as:

$$c_t = R_t \sum_{\tau=0}^{t-1} c_\tau g_{t-\tau}, \quad (4.3)$$

²<https://www.gov.uk/government/organisations/public-health-england>

³https://rp5.ru/Weather_in_the_world

Table 4.1: Non-pharmaceutical COVID Control Measures.

Control Measures	Type
Meeting Friends / Family (Indoor)	Categorical
Meeting Friends / Family (Outdoor)	Categorical
Domestic Travel Control	Categorical
International Travel Control	Categorical
Cafes and Restaurants Control	Categorical
Pubs and Bars Control	Categorical
Sports and Leisure Closure	Categorical
Hospitals / Care and Nursing Home Visits	Categorical
Non-Essential Shops Closure	Binary
School Closure	Binary

where c_τ is the number of new infections on day τ ,

$$g_1 = \int_{\tau=0}^{1.5} g(\tau) d\tau,$$

and for $s = 2, 3, \dots$,

$$g_s = \int_{\tau=s-0.5}^{s+0.5} g(\tau) d\tau.$$

From Equation 4.3, we have:

$$R_t = \frac{c_t}{\sum_{\tau=0}^{t-1} c_\tau g_{t-\tau}} \quad (4.4)$$

For $x = t$ and τ , c_x is the difference between the confirmed case on day x and the confirmed case on day $x - 1$, which is available from the dataset directly.

Using this data, we pose a simple classification question:

Given the infection number and control measures implemented on a day t , can we predict $R_t \geq 1$?

As control measures take time to affect the infection rate, we expand the dataset to include the duration of control measure implementation for all control measures. For example, “Meeting Indoors (High) = 2” means that “it is the second week that meeting indoors has been banned completely”. Similarly, “International Travel (Low) = 0” means that “there is no restriction implemented on international travel”. We also drop instances before March 15, 2020 across all

4. Explainable AI

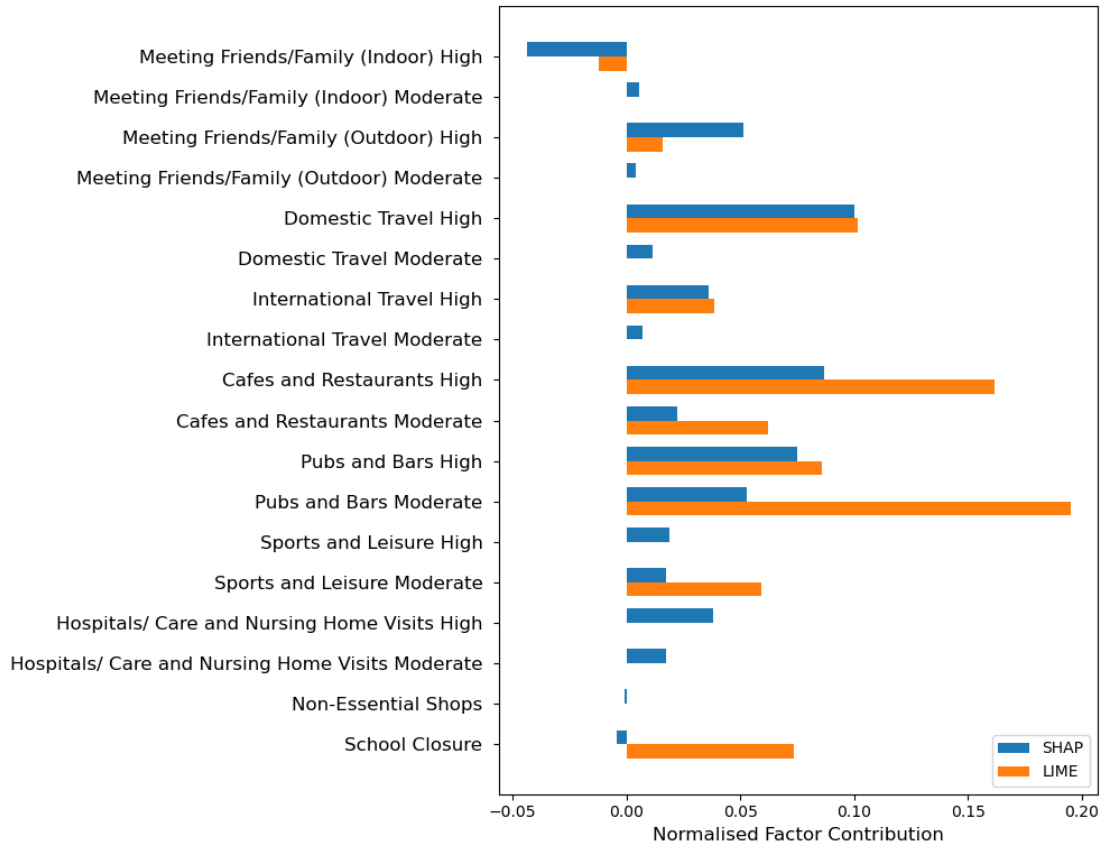


Figure 4.6: Example of an Explanation computed with SHAP and LIME. For this instance, both explainers consider top measures contributing to this prediction being *Domestic Travel*, *Cafes and Restaurants Closure* and *Pubs and Bars Closure*.

12 regions and nations in our dataset due to the low number of infections.⁴ In this way, we form a data file with 18 features and 3,937 instances with 1,550 positive ones.

Table 4.2: Prediction performance on the COVID dataset with four different classifiers with K-Fold Cross Validation ($k = 10$).

Classifier	MLP	Random Forest	SVM	XGBoost
Precision	0.82 ± 0.03	0.85 ± 0.03	0.84 ± 0.03	0.82 ± 0.03
Recall	0.81 ± 0.07	0.84 ± 0.03	0.74 ± 0.05	0.76 ± 0.03
F1-score	0.81 ± 0.02	0.87 ± 0.02	0.79 ± 0.04	0.79 ± 0.02

The classification results are summarised in Table 4.5. We can see that all four classifiers are able to achieve good performance on this dataset with a 70/30 training/testing split. As an illustration, for a prediction query instance such that:

⁴As can be seen from Equation 4.4, when c_x is small, R_t can flatten in an unrealistically large range and generate noises in the dataset.



Figure 4.7: Global explanations generated using SHAP on our COVID dataset for the prediction whether $R_t \geq 1$. We see that closing down cafes and restaurants as well as pubs and bars are the most effective control measures. When their feature values are high (red), they have strong negative impact to the prediction; whereas when their feature values are low (blue), they have strong positive impact to the prediction.

- All control measures shown in Table 4.1 except *International Travel (IT)* and *Hospital / Care and Nursing Home Visits (HCNHV)* are implemented for more than 35 days at the level *High*;
- *IT* has been implemented for more than 35 days at the level *Moderate*; and
- *HCNHV* implemented for 20-25 days at the level *High*.

Using Random Forest as our prediction model, it correctly predicts that $R_t < 1$; and SHAP and LIME explanations are shown in Figure 4.6. We see that SHAP and LIME produce similar explanations for the instance. In addition to local explanations, ExMed can also

use SHAP to compute global explanations for the entire dataset - describing the “trend” of all instances - as illustrated in figure 4.7. We observe that control measures *Cafes and Restaurants Control* and *Pubs and Bars Control* have the most influence predictions made with this dataset, this can be interpreted as:

From February 2020 to February 2021, the most effective non-pharmaceutical COVID control measures implemented in the UK are closing cafes and restaurants as well as pubs and bars.

4.3.2 Case Study II: Lung Cancer Life Expectancy

Our second case study investigates the application of ExAI to electronic patient records for cancer research instead of using public health epidemiology data in order to emphasise the transferability provided by ExMed. Especially, we use artificial data from the Simulacrum⁵, a synthetic dataset developed by Health Data Insight CiC and derived from anonymous cancer data provided by the National Cancer Registration and Analysis Service⁶, which is part of Public Health England. This dataset contains 1,322,100 cancer patient instances.

We first isolate a cohort of interest, opting for lung cancer patients as they represent a large portion of cancer-based deaths [172]. With lung cancer patients, we define the medical question as a prediction of patient survival time, and pose the following multi-class classification question:

Given a set of features for a patient, what will be the predicted survival time for the patient? Under six months, six to twelve months, or more than twelve months?

To study this, we first identify the subset of lung cancer patients in the Simulacrum with an ICD-10 code *Malignant neoplasm of bronchus and lung* and a deceased status, and includes 108,282 patients in total. We removed records from the original dataset with obvious errors and included only patients with a vital status date posterior to the diagnosis date.

A major challenge in medical data analytic, as exemplified in the Simulacrum, is missing or incomplete patient records. This results in a large number of “null” entries in the dataset. To address this, we identify a smaller cohort of patients such that each patient contains 20 features, with each patient instance only able to contain a maximum of one “null” value.

⁵<https://simulacrum.healthdatainsight.org.uk/>

⁶<http://www.ncin.org.uk/>

Table 4.3: Each patient is described with 20 features.

Feature	Value	Feature	Value
ACE	2.0	T Best	0.0
Sex	M	M Best	3.0
CNS	9.0	N Best	4.0
Age	68	Cycle Number	0.0
Grade	0.0	Ethnicity	1.0
Height	1.6	Cancer Plan	1.0
Weight	75.6	CReg Code	4.0
Morph	8041.0	Chemo Radiation	N
Laterality	901.0	Regimen Time Delay	N
Performance	1.0	Regimen Stopped Early	N

This explicit filtering isolates a cohort of 2,260 patients. This also provides a well-balanced dataset with each group containing a similar amount of patients as shown in Table 4.4.

Table 4.4: Survival Time Feature Value Count

Survival Time	Value Count
<i>Greater than 1 Year</i>	842
<i>Between 6 Months and 1 Year</i>	748
<i>Less than 6 Months</i>	670

Table 4.5: Prediction performance on the Lung Cancer dataset with four different classifiers with K-Fold Cross Validation ($k = 10$).

Classifier	MLP	Random Forest	SVM	XGBoost
Precision	0.86 \pm 0.15	0.92 \pm 0.05	0.84 \pm 0.08	0.69 \pm 0.08
Recall	0.76 \pm 0.47	0.84 \pm 0.08	0.55 \pm 0.10	0.66 \pm 0.08
F1-score	0.81 \pm 0.23	0.87 \pm 0.04	0.66 \pm 0.08	0.67 \pm 0.06

We first provide a local explanation example using both SHAP and LIME for a patient instance as shown in Table 4.3. We observe that both explainers give similar explanations as shown in Fig 4.8.

Using the entire dataset, we produce a global explanation determining feature importance towards each output class in Fig 4.3.2 (a). We then provide granularity to feature value importance towards each class with Fig 4.3.2 (b) - (d). We interpret these results as:

Cancer grades, BMI, age, patient performance and the absence of distant metastatic spread are key indicators for estimating patient survival time.

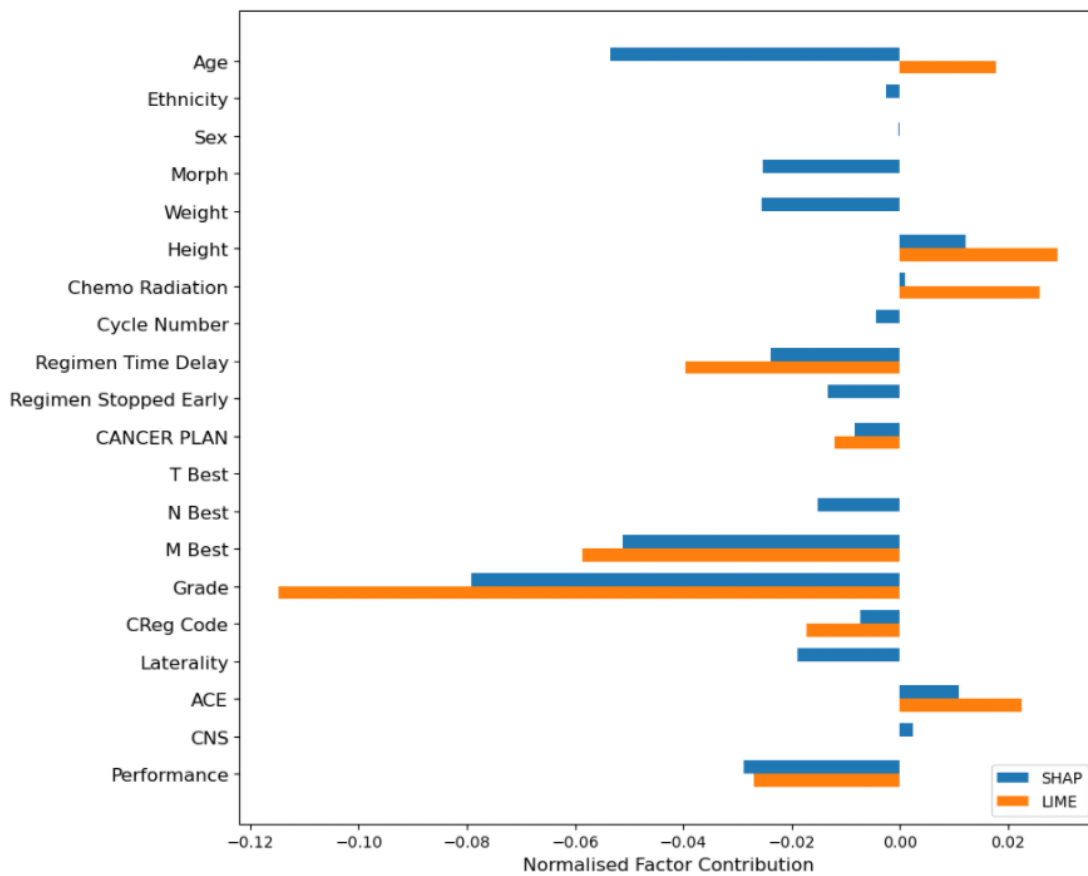


Figure 4.8: Local explanation on the Lung Cancer life expectancy data set for a patient instance. We see that the most impactful features amongst SHAP and LIME are the same: “Grade” *How the cancer cells act; the higher the grade the less normality the cell resembles, and it may act more aggressive* and “M Best” *Presence or Absence of Distant Metastatic Spread*, followed by a disagreement on age attribution.

4.4 The Neighbour Migrating Generator Model

The Neighbour Migrating Generator (NMG) is a straightforward and effective method for finding the nearest Neighbouring instance(s) with a different label for a given data point without adjusting any kernel settings. This approach enables the identification and explanation of the most significant characteristics that impact a machine learning model.

Using the NMG technique, one can transfer a particular sample to the decision boundary of the original model for its corresponding class within a small vicinity of the

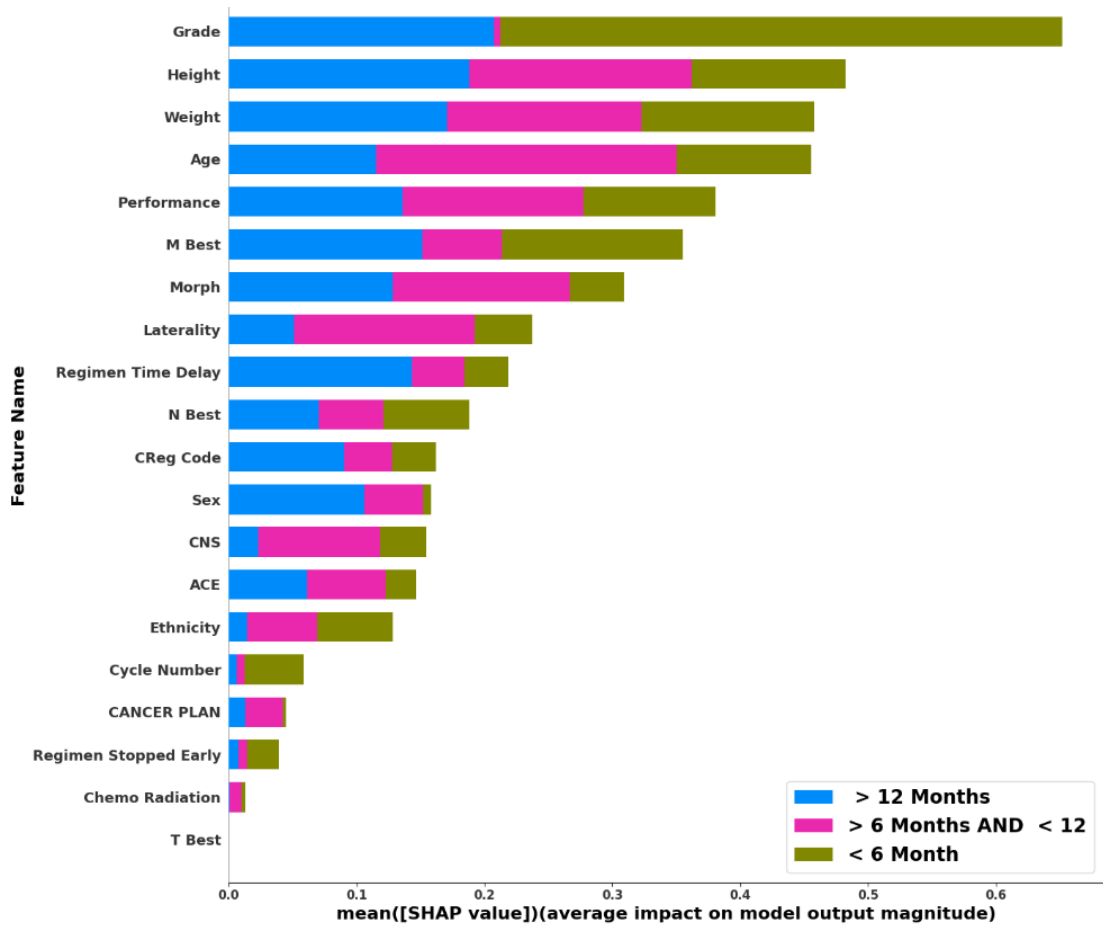


Figure 4.9: We see that the largest impact towards the survival boundaries *greater than* 1 year and *less than* 6 months is the cancer grading - having direct impact on the longest and least time survived. This, followed by an associative relationship between height, weight, and the patient age determinants of body mass index (BMI), having high attribution towards each class. This, then followed by cancer specific traits such as “M Best” and laterality of the tumour.

sample, or detect global characteristics that assist in classifying Neighbouring groups. The method employs a loss function that is divided into two parts, each of which is weighted independently by four parameters: α , β , and ω , with α being self-adjusting. Experiments have shown that this approach performs better than previous techniques in detecting even minor changes in the feature space and can also uncover issues in models such as over-fitting.

In this section, we will introduce our new model and its two variations and explain how to train them. The two variations are based on whether we want to generate local or global explanations.

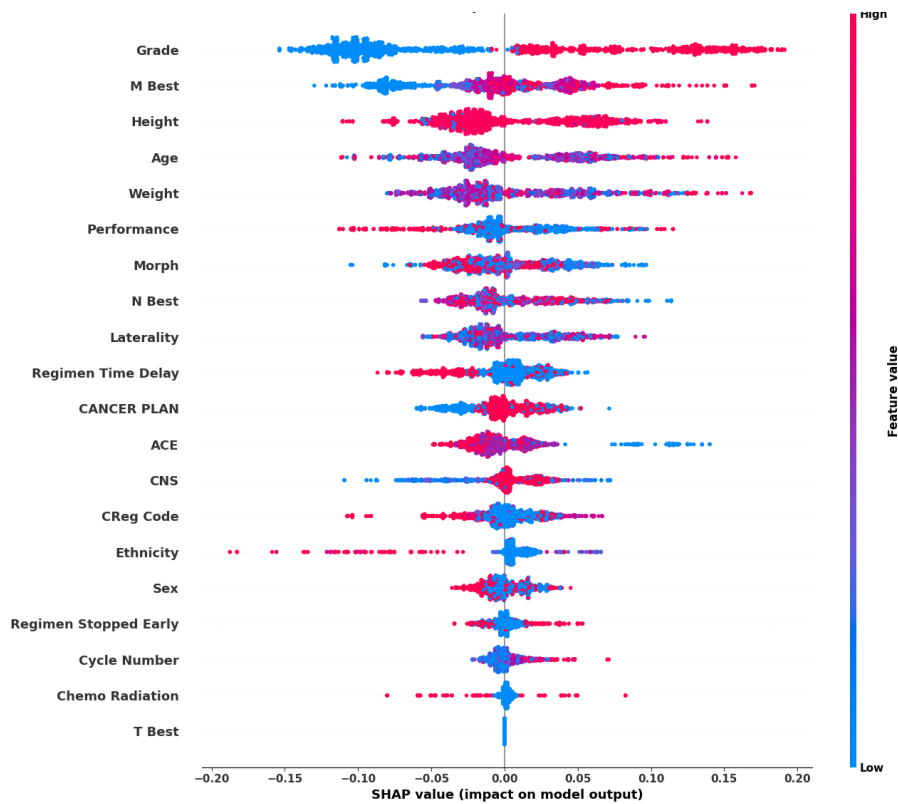


Figure 4.10: Global explanation for feature attribution measured against the class *Survival time of less than 6 months*, where we see the cancer grade of higher value - indicative of cell abnormality and more aggressive, followed by “M Best” *Presence or Absence of Distant Metastatic Spread*, with the associative BMI attributes “height”, “age” and “weight” following this.

For local explanations, the model is trained to identify the closest decision boundary to a particular data point. On the other hand, for global explanations, the model identifies the most significant features for each class. Both variations use the same architecture but are trained differently. The local variant is trained using a single data point, while the global variant is trained using all the data points.

4.4.1 Local and Global Variants

In the local explanation variant, the NM generator employs ANN to identify a Neighbour. A single input is fed into the fully connected network G , and we minimise the loss function described later to converge to a point near the original sample but with a different target label specified by the user. Therefore, the NMG aims to identify and modify only the primary features that lead to that class, indicating the most significant factors for a given sample.

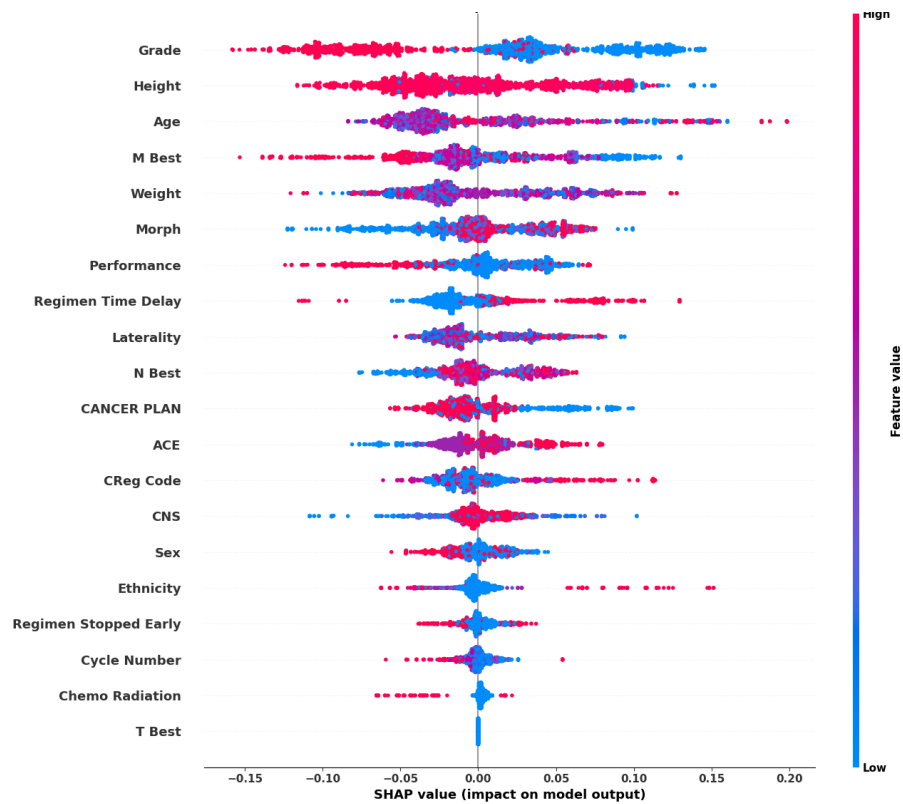


Figure 4.11: Global explanation for feature attribution measured against the class *Survival time of greater than 12 months*, we see an inverse plot of cancer grade to that shown in Fig.4.3.2 (a), such that a lower grade and what seems to be a better controlled BMI and a lower “M Best” contributing to a longer survival time.

Another potential application of our model is to observe feature migration in the global space, indicating the most influential factors for a given class. In this approach, instead of specifying a target class label to migrate towards, we use the predictions of a trained model F to generate labels. We choose the second most probable (or predicted) class label and use it as the target label. We train our Neighbour Migrating Generator G on the entire training set.

It is important to note that in the global variant, the NMG uses the same model for all samples, which aims to minimise the impact of outliers and produce more generalised results. In contrast, in the local variant, we train the model separately for each input, generating different weights each time. However, with the global variant, we may not be able to establish a minimum loss cost for outliers, but this will not affect the final results.

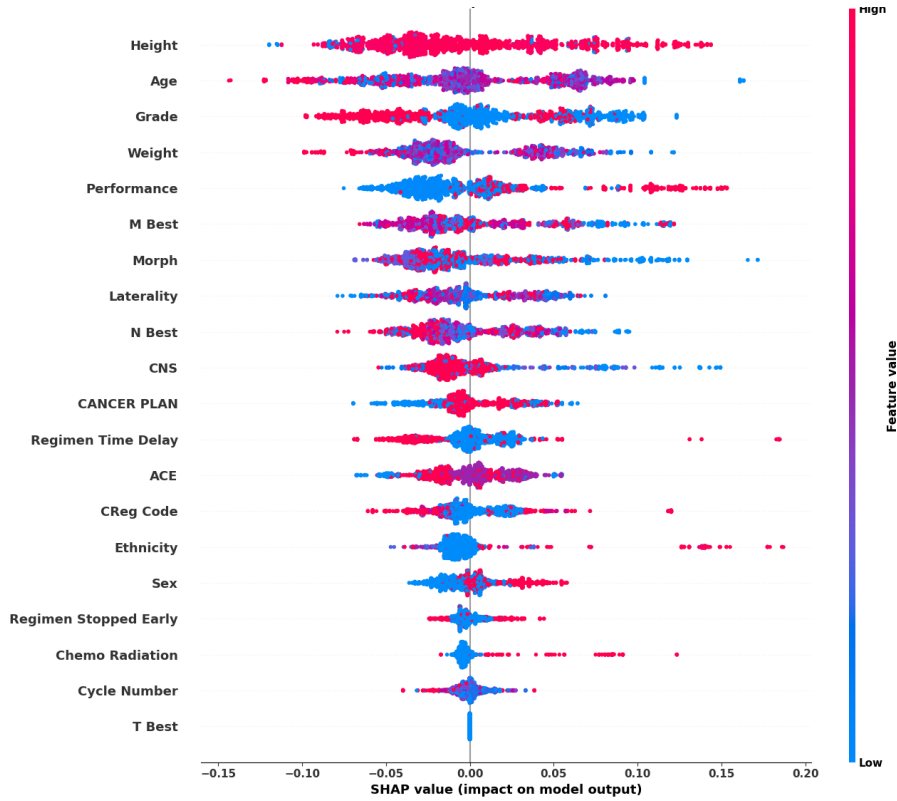


Figure 4.12: Global explanation for feature attribution measured against the class *Survival time between 6 and 12 months*, we see that a controlled BMI and lower cancer grade are attributive to this survival boundary, whilst the distributive “M Best”, performance and cancer grade containing high values in both positive and negative impacts on the model are likely the reason for the central survival boundary.

The global approach can be formulated as follows: Let d be the number of classes, n be the number of samples, and F be the original network that maps an input sample x_i to a vector of inferred probabilities for each class $F(x_i)$, with each component represented as $F_j(x_i)$. We define the best prediction of our original network $H_F^1(x_i)$ as:

$$H_F^1(x_i) = \arg \max_{j \in d} F_j(x_i) = \{j \mid F_j(x_i) \leq F_k(x_i) \forall k \in d\} \quad (4.5)$$

and the second-best class prediction $H_F^2(x_i)$ as:

$$H_F^2(x_i) = \arg \max_{j \in d} F_j(x_i) = \{j \mid F_j(x_i) \leq F_k(x_i) \forall k \in d, k \neq H_F^1(x_i)\} \quad (4.6)$$

We can now define our new NMG model G that is trained from a set of samples \mathcal{D} as follows:

$$\begin{aligned}
 G : \mathcal{D} &\rightarrow \mathcal{D} \\
 x'_i = G(x_i) & \mid H_F^2(x_i) = H_F^1(x'_i) \wedge \\
 \forall x'' \in \mathcal{D}, & H_F^2(x_i) = H_F^1(x'') \wedge \\
 & \|x_i - G(x_i)\|_l \leq \|x_i - x''\|_l
 \end{aligned} \tag{4.7}$$

with l indicating the norm used.

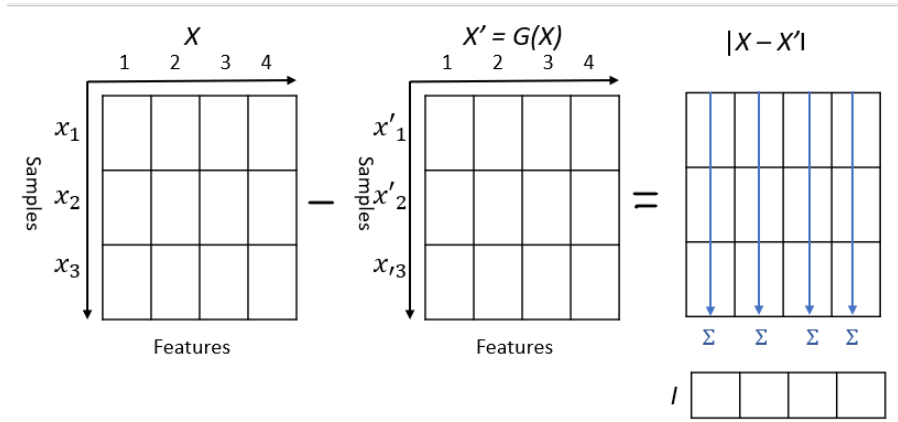


Figure 4.13: The calculation of the Feature Importance weights I provides an explanation of the global impact of features for the different classes of the original model.

After the training process, we calculate the sum of differences between each input sample x_i and the nearest corresponding sample with a different label, which is defined as $x'_i = G(x_i)$. This imaginary sample x'_i is intended to be located on the class decision boundary in the feature space and as close as possible to the initial sample x_i . For a given feature f , this sum is represented as $I_f = \sum_{i=1}^n |x_i(f) - x'_i(f)|$, as shown in Figure 4.13. Larger values of I indicate that a feature has a greater impact and is a major contributor in explaining how the model distinguishes between classes.

4.4.2 Loss function

Algorithm 3 outlines the calculation of the loss function for G during each iteration. The same loss function, as given in equation 4.8 and illustrated in figure 4.14, is used for both variants.

$$L(x, x', y, y') = CrossEntropy(y, y') + \lambda \cdot L^1(x - x') + \beta_{y, y'} \tag{4.8}$$

Algorithm 3 Algorithm for calculating the loss function at each iteration.

```

λ ← 0 /*Importance of the inner loss*/
ε ← 0.05 /*A constant for the thickness of the decision boundary*/
for each iteration do
  if (y == y') then
    λ ← clip(λ + 1.0, 100.0)
  else
    λ ← clip(λ - 1.0, 0.0)
  in_loss ← (λ/100.0) * (∑(x'_i - x_i))
  βy,y' ← |CrossEntropy(1, y == y')
    - CrossEntropy(0.5 + ε, 0.5 - ε)|
  out_loss ← CrossEntropy(y, y')
  loss = in_loss + out_loss + βy,y'

```

The equation in Algorithm 3 is divided into three parts. The first part, represented by *CrossEntropy*, calculates the difference in information between the current and the new labels, where y is the class label for x and y' is the predicted label for x' . The second part, represented by L^1 , controls how close x' should be to the original input x . Instead of using the Euclidean distance, the Manhattan distance is used to encourage changes in only a few features. This is because the L^1 norm produces an output with only a small number of features presenting a significant change, whereas using L^2 results in changes that are distributed across more features, making it harder to identify the most prominent features. Finally, the third part (*Reg*) is a regularisation term that helps to avoid overfitting during the training process. The weight λ is recalculated after each training step, and the cross-entropy function $\beta_{y,y'}$ is defined in the next paragraph. Figure 4.14 illustrates the loss function.

The last component of the loss function is the $\beta_{y,y'}$ parameter. This component is calculated as

$$\beta_{y,y'} = |CrossEntropy(1, y == y') - CrossEntropy(0.5 + \epsilon, 0.5 - \epsilon)|,$$

With $\epsilon = 0.05$.

The purpose of the λ parameter in the loss function is to balance the importance of changing the label and staying close to the original input value. By starting with a low value of λ and gradually increasing it, the model is allowed to make larger changes to the input to find a new label. As the model gets closer to the correct label, λ is increased to encourage

the model to focus more on staying close to the original input value. On the other hand, if the model predicts the wrong label, λ is decreased to allow the model to make larger changes to the input in order to find the correct label. This approach allows the model to converge to the appropriate value of x' without manual adjustment of parameters.

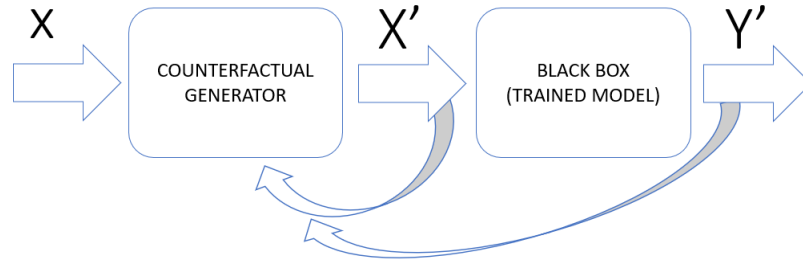


Figure 4.14: Loss function of Neighbour Migrating Generator

The vector weights

Adding weight to the loss function can limit changes in the input dimensions, which is beneficial because some input features are not easily changeable in real life. For example, if the NMG offers a modification in input dimensions to change a model decision, and the outcome shows reducing the customer age, it may not be attainable. We have demonstrated the usefulness of adding weight in section 4.4.4. By changing equation 4.8 and adding a simple weight vector (ω), we obtain equation 4.9. The weight vector ω should have the same size as the input features (with one weight per feature, calculated by element-wise product). Larger weights in ω result in a higher cost in the loss function, which means that in minimising the loss function progress, the correlated input feature changes less. Therefore, for input dimensions that we would like to change freely, the weight should be 1 or even less (less than 1 means changing that feature has a lower cost in the loss function).

$$L(x, x', y, y') = CrossEntropy(y, y') + \lambda \cdot (L^1(x - x'))\omega + \beta_{y, y'} \quad (4.9)$$

4.4.3 Experimental results

We conducted experiments using our new approach on three different datasets: a synthetic 2D heart function dataset, the Pima Indians Diabetes Database, and the IRIS dataset. The

Item	Descriptions
Dataset	Implicit Heart Function Dataset
Model F	Input(2),FC(256),RelU,FC(256),RelU,FC(2),Soft-Max
Model G	Input(2),FC(256),RelU,FC(256),RelU,FC(256),RelU,FC(2)
Dataset	PIMA
Model F	Input(8),FC(256),RelU,FC(256),RelU,FC(2),Soft-Max
Model G	Input(8),FC(256),RelU,FC(256),RelU,FC(256),RelU,FC(8)
Dataset	IRIS
Model F	Input(4),FC(128),RelU,FC(128),RelU,FC(3),Soft-Max
Model G	Input(4),FC(1024),RelU,FC(1024),RelU,FC(1024),RelU,FC(4)

Table 4.6: A description of the architecture of the models used for the three datasets. The F model is the original model that we want to study, while G is the NMG model used to migrate to decision boundaries.

tests were performed on a system with an Intel 4770K processor running at 3.9 GHz, 32 GB of RAM, and an Nvidia GeForce GTX 3070TI Graphics card. TensorFlow 2.0 was used as the AI framework, running on a Linux distribution.

To ensure that each dimension has an equal probability to change, each feature in all datasets was standardised to the range $[-1, 1]$ with a mean value of 0. We did this to account for the sensitivity of both the L^1 and L^2 norms to the units used for features.

For training, we employed an Adam optimiser with a learning rate set to 10^{-4} . Every layer in the models had a ReLU activation function, except for the last layer which was linear. The models were trained for 200 epochs, and the best model was saved. Table 4.6 provides a description of the architecture of the three original models and their associated NMG models.

Implicit Heart Function Dataset

To demonstrate the impact of the loss function on the decision of our local variant model, we created an artificial dataset from a two-dimensional slice of a heart-shaped function described in [4] (eq. 4.10). Any point inside the heart is labelled as one, and zero otherwise. We consider the original model as a black box once it is trained, and we will use it only to evaluate how far our new predictions are from the decision boundary.

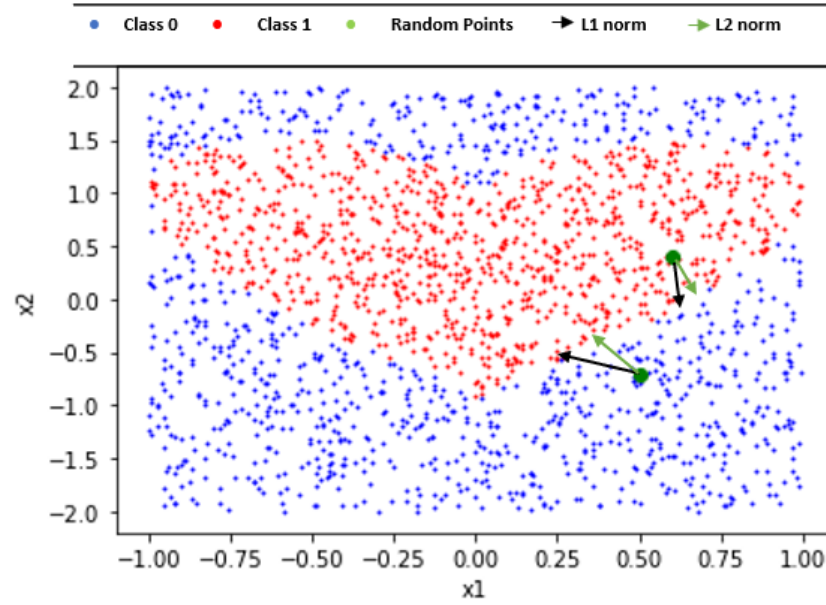


Figure 4.15: We demonstrate the effectiveness of our NMG model G by applying it to some random samples from an implicit 2D heart function dataset [4]. The dataset consists of blue dots representing class 0 samples and red dots representing class 1 samples. We also select some green dots at random from the dataset to transform using our NM Generator. The results, shown in Figure X, demonstrate that our NMG model is able to satisfactorily migrate samples to the decision boundary.

$$f(x_1, x_2) = (x_1^2 + x_2^2 - 1) - x_1^2 x_2^3$$

$$y = \begin{cases} 1 & \text{if } f(x_1, x_2) < 0 \\ 0 & \text{otherwise} \end{cases} \quad (4.10)$$

To identify the distribution of decision boundaries with our NMG, we use again a fully connected network with three layers, each with 256 neurons, and a final layer with two neurons only, as the output should have the same number of features as the input. We have assigned two random point inside and outside the heart and we have drawn the output position given by our NMG. The findings reveal that the model was able to locate the decision boundary well with both L^1 and L^2 Norms.

Pima Indians Diabetes Database

The Pima Indians Diabetes Database [173] includes seven continuous but sometimes incomplete features and a categorical binary feature that indicates whether or not a patient

4. Explainable AI

Feature Name	Description	Range
Pregnancies	Number of pregnancies	[0..17]
Glucose	Oral glucose tolerance test, measured after 2 hours.	[0..199]
Blood Pressure	Diastolic blood pressure	[0..122]
Skin Thickness	Triceps skin fold thickness	[0..99]
Insulin	Insulin level after 2 hours	[0..846]
BMI	Body Mass Index	[0..67.1]
Diabetes Pedigree Function	Diabetes history in relatives	[0.08..2.42]
Age	Age in Years	[21..81]

Table 4.7: Description of features in the Pima diabetes datasets.

Original	Migrated	Sepal L	Sepal W	Petal L	Petal W
Virginica	Versicolor	0.0662	0.0207	0.5398	0.0965
Versicolor	Virginica	0.0340	0.0081	0.3372	0.0033
Setosa	Versicolor	0.3106	0.4010	0.9289	0.7361

Table 4.8: Variation among features when changing instances. For each original class, our global NMG model returns the closest class available (left two columns). The results match what can be witnessed on Fig. 4.16. For the first two migrations, results show that the petal length is the only important feature to migrate class. For the Setosa class, all parameters have some influence, but the petals' length and width are still the most contributing factors.

has type 2 diabetes. It contains 768 individuals who are at least 21 years old, female, and of Pima Indian heritage. Table 4.7 lists the features present in the dataset.

In this example, we apply our local NMG variant to a set of 13 random samples to determine what minimal changes can be made to a sample to switch its class. Here again, our NMG migrated the samples to the decision boundary by minimising the loss function of an input sample. The very first sample we tested actually demonstrated that something was not right with the original model. We observed that the NMG prioritised a change in the blood pressure, while diabetes patients often have high blood pressure, diabetes is mainly detected from glucose or insulin levels in the blood, which can be tested two hours after ingesting 75mg of glucose (Oral Glucose Tolerance Test - OGTT). While the NMG migrated the sample to a nearby point very similar to the original one, but still clearly in the diabetic range, the original model classified it as non-diabetic. A study of 12 more samples (Tables 4.11 and B.8) indicated that this was not an exception (e.g., samples 1,3, 6), while other samples looked more logical (e.g., sample 2 migration to a diabetic

status logically increases parameters like glycemia and insulin but also BMI and blood pressure parameters, which are frequently raised for diabetic patients.).

The only explanation for small non-logical changes being able to change sample classes was that the original model was overfitting. This was confirmed afterward by noticing that a) the model had far more parameters than samples (Table 4.6), and b) the training accuracy was 100

IRIS

The IRIS dataset [174] is well-known for its incorporation of many measures into taxonomic difficulties. The original dataset includes 50 samples from each of the three Iris species (*Iris, Setosa, Iris, Virginica, and Iris, Versicolor*), for a total of 150 items. Each sample has four attributes measured in centimetres, including length and width measurements for both the sepals and petals. In particular, Fig. 4.16 shows that two out of the four features are enough to provide a good separation of the three classes. This figure also provides insight into the Neighbouring relationships between classes. The global variant of our Neighbour Migrating Generator is then applied to this dataset to see if this Neighbouring relationship can be captured by our technique. Therefore, with this global approach, the model learns how to go to the nearest border rather than specifying which class border it should learn. Figure 4.17 demonstrates all combinations of features, and as we can see, the NMG found the best two features that can separate the classes.

Our global NMG model is defined by three fully connected layers of 1024 neurons with ReLU activation. The new labels in the training set are defined by the second-best predicted label of the original model. The results in Table 4.8 show that the migration of the three different classes corresponds pretty much to what can be observed in Figure 4.16, with, for instance, the *Setosa* and *Virginica* classes migrating to the *Versicolor* class, and the latter one being migrated to the closest of the two, i.e., *Virginica*.

We can examine the most influential characteristics in financial decision-making, or in other respects, which features can distinguish classes the most from a list of features that need to be altered most of the time. These results are shown in Table 4.8.

4.4.4 Thyroid disease

The original Thyroid-disease datasets in the UCI machine learning repository contains 2800 instances for training data and 972 instances for testing, each with three classes.

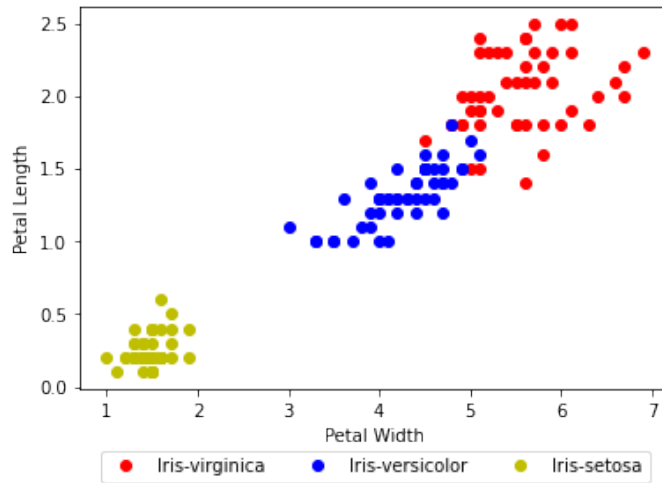


Figure 4.16: 2D plotting the IRIS dataset. The graph indicates that using only two features (Petal Length and Petal Width) out of four is enough to separate most samples of this dataset. Note that Iris Versi color appears to be in between the two other classes.

Each instance has 21 attributes, consisting of 15 categorical and six continuous numbers. This dataset includes various characteristics of patients who underwent an oral glucose tolerance test, such as insulin and glucose levels after two hours, as well as their type 2 diabetes status (0 indicating non-diabetic). For ease of displaying the experimental results, we narrowed the classes down to two and selected records based on the following criteria:

On thyroxine = Query on thyroxine &
On antithyroid medication : False &
Thyroid surgery : False &
I131 treatment : False &
Tumor : False&
Hypopituitary : False

After filtering, there were 3065 records left, which comprised of 2298 training samples and 767 test instances. Except for three columns, namely 'FTI', 'TBG', and 'Referral source', most of the categories were removed from the dataset and not utilised to train the original model F or the NM generator G . We employed the same setup as the last experiment

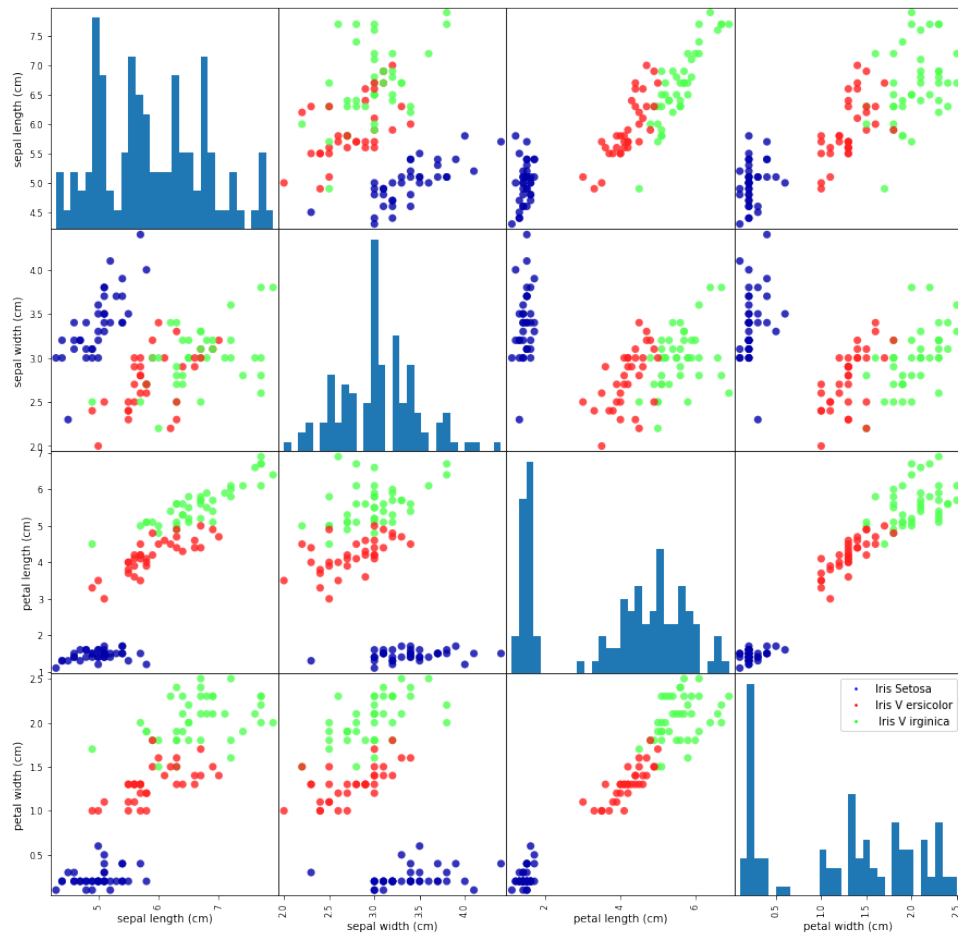


Figure 4.17: shows the relationship between each type of species, and features are displayed.

(except using L^2 norm) and normalised the dataset as before. However, the weight vector ω was altered to make some input dimensions less variable. The selected dimensions and their corresponding weights are shown in table 4.9.

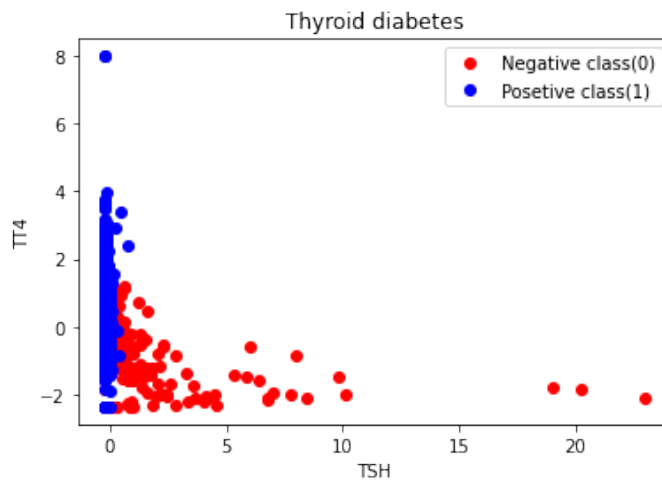
The results demonstrate that the ω vector is effective in preventing certain features from being modified, thus reducing the cost of changing them. Further analysis shows that two specific features have a significant impact on the migration of samples between classes. However, it is important to note that this observation may not be applicable to all datasets, especially when all features are changeable. The experimental results are summarised in Table 4.10.

Upon examining the differences between the original samples and their migrated counterparts, we observed that in most cases, the TSH and TT4 fields were the ones

Feature Name	Description	Weight
Age	Continuous	99
Sex	Categorical	99
TSH	Continuous	1
T3	Continuous	1
TT4	Continuous	1
T4U	Continuous	1
FTI	Continuous	99
TBG	Continuous	99
Referral source	Categorical	99

Table 4.9: Description of features in the Thyroid-disease datasets.

Figure 4.18: 2D plotting the Thyroid dataset. The graph indicates that using only two features (*TT4* and *TSH*) out of nine is enough to separate most samples of this dataset.



undergoing changes. Interestingly, these two features alone are sufficient to distinguish between the two classes, as illustrated in Figure 4.18.

In addition to what we have already discussed, we have conducted additional tests on this dataset (where the three classes are not changed) to examine the effect of the weight vector on model G. This time, we ran model G three times, either with no input dimensions frozen at all, just freezing "Sex", or two features together, namely "Sex" and "Age". The experiment layout demonstrates that the weight factor has a significant influence on how the NMG discovers the closest Neighbour with a different label while considering less change in frozen inputs. The results are shown in table B.9.

Classes	Age	Sex	TSH	T3	TT4	T4U	FTI	TBG	Ref source
Class 1	58	0	1.3	2.3	77	0	0	0	1
to 0	58	0	10.69	2.30	77.01	-0.01	0.01	0	1
Changes	-5E-04	2E-04	9.39	1E-03	0.01	-7E-03	0.01	0	9E-05
Class 0	63	1	8.20	2.10	80	1.02	78	0	1
to 1	63	1	2.37	2.10	80.22	1.02	78.01	0	1
Changes	6E-05	3E-05	-5.83	-1E-03	0.2	-5E-03	0.011	0	8E-05
Class 0	60	1	8.10	1.80	59	0.96	61	0	1
to 1	60	1	1.24	1.79	59.57	0.96	61.01	0	1
Changes	-1E-05	5E-05	-6.86	-7E-03	0.06	-3E-03	0.01	0	1E-05
Class 1	29	1	0	0	0	0	0	0	1
to 0	29	1	10.30	0.03	-1.31	0.01	0	0	1
Changes	-2E-03	9E-05	10.3	0.02	-1.31	0.01	-4E-03	0	3E-05
Class 1	59	1	0	0	0	0	0	0	1
to 0	59.01	1	10.30	0.02	-1.05	0.01	-0.01	0	1
Changes	9E-03	9E-05	10.3	0.02	-1.0	9E-03	-0.01	0	2E-07
Class 1	71	1	0	0	0	0	0	0	1
to 0	70.98	1	10.20	0.02	-1.04	0.01	0.04	0	1
Changes	-1E-02	1E-04	10.20	0.02	-1.04	9E-03	0.04	0	2E-05

Table 4.10: Description of features in the Thyroid diabetes datasets.

4.5 Conclusion

In this chapter, we introduced ExMed, a user-friendly software package that enables medical domain experts to perform Explainable AI data analysis without requiring any programming skills. With ExMed, we aimed to combine the flexibility of medical sub-domain transferability with the trustworthiness of explainability, using state-of-the-art ExAI methods. ExMed can handle various data input types and offers standard pre-processing operations, supports different prediction models and visualisation techniques, and incorporates two popular feature attribution ExAI algorithms.

To validate ExMed’s effectiveness, we conducted two real-world case studies in the domains of public health epidemiology and cancer research using electronic patient records. In the COVID case study, we analysed the effectiveness of COVID control measures in the UK from March 2020 to January 2021 and found that closing cafes, restaurants, pubs, and bars had the greatest impact in reducing the virus transmission

rate. In the cancer case study, we investigated the life expectancy of lung cancer patients using the Simulacrum dataset and observe that factors such as cancer grade, BMI, age, and M Best have a significant influence on survival.

In addition, we introduced the Neighbour Migrating Generator (NMG), a novel method for finding the closest neighbour with a different class label. The NMG searches for decision boundaries that enable changing a given input with minimal effort and identifies the most influential dimensions in the decision space that distinguish the classes. We conduct three experiments to demonstrate the effectiveness of our generator: 1) finding decision boundaries in the original model (e.g., heart function), 2) detecting overfitting in results (e.g., Pima Indians Diabetes Database), and 3) providing close Neighbouring clues between classes (Iris dataset). Notably, our method can learn decision boundaries without modifying any kernel parameters, leading to more reliable outcomes compared to previous approaches (e.g., [82]).

Samples / Class migration	Number of Pregnancies	Glucose mg/dl ≤ 140	Diastolic Blood Press., mm Hg ≤ 80	Skin Thickness mm	Insulin $\mu U/ml$ 16 – 166	BMI kg/m^2 ≤ 25	Diabetes Pedigree Function	Age year
1 : 1 \rightarrow 0	2 +3.5 (+175%)	100	66	20	90	32.9	0.867 -0.1 (-12%)	28
2 : 0 \rightarrow 1	4	129 +18.8 (15%)	86	20	270	35.1	0.231	23
3 : 1 \rightarrow 0	3	169	74	19 -0.1 (-1%)	125 -21.9 (-18%)	29.9	0.268	31
4 : 0 \rightarrow 1	2	101 +0.9 (+1%)	58	35 +0.2 (+1%)	90 -88.4 (-98%)	21.8 +2.2 (+10%)	0.155	22 +1.1 (+5%)
5 : 0 \rightarrow 1	3 +1.3 +1.3	96	56	34 +0.6 (+2%)	115	24.7 +1.3 (+5%)	0.944	39
6 : 0 \rightarrow 1	1	118	58	36	94 -23.2 (-25%)	33.3	0.261	23

Table 4.11: Variation among features when changing instances (local variant of the algorithm) for the Pima Indian Diabetes Dataset for 6 randomly selected samples. Units and standard ranges for glucose, blood pressure, insulin and Body Mass Index are given at the top, and samples outside the range are highlighted in red. The second value in cells, when present, indicates the variation generated by the migration model on some of the features. Unchanged values are not shown.

Chapter 5

Conclusions

The field of artificial neural networks has seen much growth with new exciting developments in recent years, leading to various breakthroughs in artificial intelligence, as well as in other areas of science. As artificial neural networks become more complex, it becomes crucial to reduce their complexity to improve efficiency and accuracy especially for portable devices with lower computational resources. One aspect in this thesis focused on improving fully connected layers of a neural network by proposing a different way to multiply matrices. This novel approach to matrix multiplication called the Mediterranean Matrix Multiplication (M^3) offers several advantages over traditional methods.

The M^3 is a technique that can deal with dealing with large matrices, offering a configurable and pre-calculable error rate that surpasses other algorithms in terms of performance. The algorithm works by estimating the angles between rows and columns of two matrices using randomly located planes to estimate these angles in a Monte Carlo approximation fashion. It also required mainly very simple instructions such as bitwise operations to achieve this. These operations include XOR (one), POPCOUNT (one), and a single ADD process that can operate on 64 planes in parallel instead of a single fused multiply-add (FMA) for each plane. By doing so, M^3 achieves a higher degree of energy efficiency than other matrix multiplication techniques, although it only provides an approximation.

Additionally, this study shows that the M^3 is approximately 10 times faster than the regular method for approximating large matrix multiplication. This was a significant improvement, considering that the speed of matrix multiplication has for long been a bottleneck in the learning of deep neural networks. The amount of steps required to

approximate the result using M^3 is $O(k(mn + np + mp))$, where the multiplying matrices' dimensions are (m, n, p) and k represents the number of trials required to achieve the desired accuracy. This feature allows for greater flexibility, making the M^3 a highly adaptable and customisable tool for matrix multiplication.

To empirically validate the effectiveness of the M^3 , we conducted experiments using CUDA, a well-known parallel computing platform and programming model. The M^3 shows it is compatible with modern architectures such as GPU, making it more useful in artificial neural networks.

Furthermore, artificial neural networks design frequently makes use of substantial fully connected layers. By joining a large number of neurons together, these layers are utilised to build complicated representations; the weight between the neurons is represented by matrices. Matrix multiplication, a crucial process required for training and inference, can be a bottleneck when working with big matrices, though, due to its computational complexity. In order to deal with this problem, we have developed a novel technique for compressing fully connected layers in artificial neural networks. The technique used M^3 to reduce the storage complexity of these layers based on the number of hyper-planes used. In contrast to the standard complexity for storing a matrix, which is $O(n^3)$, this technique achieves significant compression rates that can be customised to meet the specific needs of different applications. For example, we have achieved a compression rate of up to $100\times$ for fully connected layers in VGG16, with negligible or no loss of accuracy. The ability to speed up matrix multiplication for huge matrices and produce more effective computation will result in quicker prediction times.

A rapidly developing field called Explainable AI (ExAI) aims to explain ANNs and other black box models by offering comments and insights into how they make decisions [175]. This approach is based on the belief that ExAI's usability is improved by offering a framework that handles each phase of the machine learning pipeline, including data manipulation, pre-processing approaches, and explainability.

In response to the greater need for explainability, we have created a novel ExAI framework that we refer to as ExMed. With an emphasis on medical datasets, we intend to provide a framework that makes it simple for users to apply ExAI to those models and their predictions.

ExMed covers every stage of the machine learning process, from data manipulation to training various models. With its various capabilities, including data dissemination, and

explainability to the design process (through a simple and straightforward user interface), the framework may assist users in understanding every step of the design process. We have incorporated a number of algorithms, including LIME, SHAP, and plot decision tree, to further improve the utilisation of ExAI in the framework.

The Neighbour Migrating Generator (NMG) technique was another contribution to the ExAI research area. It is a new technique that explains how a model might behave (take decision) both locally and globally for new samples. The NMG is an effective method that doesn't need any hyperparameter adjustments. It enables the determination of the nearest possible neighbour(s) with a different label for a specific instance, which might assist in identifying the most crucial features that affect the model's prediction. This approach can help to check fairness and finding possible bias in a model decision.

By using NMG on models trained on various datasets, we can achieve global variance and locate the nearest neighbour with a different label. This allows explaining the features that contribute most to the model's decision-making process, providing valuable information for understanding the fairness of a model.

Moreover, NMG can be used to migrate a specific sample to the class decision boundary of the original model within a close neighbourhood of that sample or to identify global features that help localise neighbour classes. The approach minimises a loss function that is divided into two components, which are independently weighted. Results show that this approach detects the smallest changes in the feature space and can also highlight issues in models like over-fitting.

In summary, This research seeks to advance the field of artificial neural networks by making these more accessible, efficient, and trustworthy for a wider audience. One primary objective was to develop a computing pipeline for artificial neural networks that can be used on devices with limited resources, and also enhance model explainability. To accomplish this, we proposed a novel matrix multiplication algorithm that relies on Monte-Carlo principles to estimate angles between vectors, resulting in faster and more straightforward calculations. Additionally, we suggest compressing Fully Connected Layers with this algorithm. We also investigate the use of explainable models in medical domains where explainability and trust are crucial. We also introduce a novel technique for explaining a model and its input space, which can help users estimate how the model arrived at its conclusions.

Chapter 6

Future Work

The field of Artificial neural network compression offers many possibilities for advancing the usage of Machine Learning (ML) in various applications, especially in devices with lower resources such as embedded systems. Despite the significant progress in this field, many challenges remain. One such challenge is finding efficient methods to compress and accelerate ANNs while maintaining their accuracy. The recent breakthroughs in the explainability of artificial neural networks offer a new direction for developing compression techniques that are not only efficient but also preserve the accuracy of the original network. Therefore, we believe that further research and development in this area are crucial to the successful deployment of advanced deep neural networks in real-world applications.

In Chapter 3, we introduced a new matrix multiplication called Mediterranean Matrix Multiplication (M^3) and its performance in terms of inference time and storage complexity. However, as we discuss in the chapter, the effectiveness of the M^3 is limited by the nature of the columns in the first matrix, making it unsuitable for Convolutional Layers. To overcome this limitation, we might be able to propose another approach that combines M^3 with other techniques, such as XNOR-Net, which compresses the convolutional layers and utilizes M^3 for the Fully Connected Layers.

Since the M^3 was made to handle big matrix multiplications with pre-calculable error rate, it may also be interesting to improve the M^3 algorithm to handle even bigger out-of-core matrices that GPUs with less memory couldn't handle. For applications where error-free accuracy is not necessarily required, it might be a useful technique. Although the M^3 was successfully used in the fully-connected layers of a neural network, its usefulness in other fields could be investigated.

The development of ExMed can be further enhanced to add to its capabilities and simplify its usage. Incorporating a variety of technical features such as automatic data assessment and model selection, and running additional models can improve the framework's overall effectiveness. It would be even more advantageous if the framework could be modified to allow for the integration of plug-ins to add external capabilities. By tuning hyperplanes automatically and assisting end-users in model training, the usability of the framework can be further enhanced.

To assess the potency of various ExAI strategies, we consider carrying out user tests, more specifically testing ExMed in collaboration with medical experts. These investigations will offer insightful information regarding the suitability of ExMed in practical contexts. We also plan to examine capabilities like parameter adjustment, which can speed up the model selection process, as part of the framework's functionality expansion.

To further improve the accuracy and completeness of the data, we intend to implement new missing value imputation methods including Multiple Imputation by Chained Equations (MICE) and Singular Imputation by Chained Equations (SICE).

Finally, we plan to introduce additional ExAI techniques such as Anchors in ExMed, which can be particularly useful in healthcare applications. These enhancements can help advance the use of ExMed. Visualisation is a powerful tool that can greatly enhance the explainability and usefulness of machine learning models. In fact, visualisation has been shown to help end-users understand complex information and reach conclusions more easily. Therefore, in order to make ANNs more accessible and user-friendly, it is important to incorporate visualisation techniques into the design of ExAI models.

In chapter 4, we also proposed an extension to the Neighbour Migrating Generator (NMG) method, which we refer to as NMG-global. This extension enables us to identify and visualise decision boundaries and understand the influence of features on the original model's decisions. By incorporating a visualisation tool, we can provide users with a more comprehensive understanding of the model's behaviour and help them make more informed decisions based on the model's predictions. This approach has the potential to significantly enhance the performance and explainability of ANN models and could be especially useful in fields such as healthcare, finance, and marketing where accurate decision-making is critical.

Bibliography

- [1] K. L. Clarkson and D. P. Woodruff, "Numerical linear algebra in the streaming model," in *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, ser. STOC '09. New York, NY, USA: ACM, 2009, pp. 205–214. [Online]. Available: <http://doi.acm.org/10.1145/1536414.1536445>
- [2] M. Courbariaux, Y. Bengio, and J. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *CoRR*, vol. abs/1511.00363, 2015. [Online]. Available: <http://arxiv.org/abs/1511.00363>
- [3] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," *CoRR*, vol. abs/1603.05279, 2016. [Online]. Available: <http://arxiv.org/abs/1603.05279>
- [4] E. W. Weisstein, "Heart curve – from wolfram MathWorld." [Online]. Available: <https://mathworld.wolfram.com/HeartCurve.html>
- [5] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutierrez, J. E. L. Gayo, S. Kirrane, S. Neumaier, A. Polleres, R. Navigli, A. N. Ngomo, S. M. Rashid, A. Rula, L. Schmelzeisen, J. F. Sequeda, S. Staab, and A. Zimmermann, "Knowledge graphs," *CoRR*, vol. abs/2003.02320, 2020. [Online]. Available: <https://arxiv.org/abs/2003.02320>
- [6] E. N. Benderskaya and S. V. Zhukova, *Multidisciplinary Trends in Modern Artificial Intelligence: Turing's Way*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 319–343. [Online]. Available: https://doi.org/10.1007/978-3-642-29694-9_13
- [7] E. Ilkou and M. Koutraki, "Symbolic vs sub-symbolic ai methods: Friends or enemies?" 11 2020.

- [8] A. Buetti-Dinh, V. Galli, S. Bellenberg, O. Ilie, M. Herold, S. Christel, M. Boretska, I. V. Pivkin, P. Wilmes, W. Sand, M. Vera, and M. Dopson, "Deep neural networks outperform human expert's capacity in characterizing bioleaching bacterial biofilm composition," *Biotechnology Reports*, vol. 22, p. e00321, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2215017X18301954>
- [9] R. M. Haralick and L. G. Shapiro, "Image segmentation techniques," *Computer vision, graphics, and image processing*, vol. 29, no. 1, pp. 100–132, 1985.
- [10] R. V. Peel, "Social research; a study in methods of gathering data. by george a. lundberg. new york: Longmans, green, and company. 1942)," *American Political Science Review*, vol. 36, no. 5, p. 982–984, 1942.
- [11] T. Hwang, "Computational power and the social impact of artificial intelligence," Available at SSRN 3147971, 2018.
- [12] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.
- [13] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.
- [14] I. Tiddi and S. Schlobach, "Knowledge graphs as tools for explainable machine learning: A survey," *Artificial Intelligence*, vol. 302, p. 103627, 2022.
- [15] P. Biecek, "DALEX: Explainers for complex predictive models in R," *Journal of Machine Learning Research*, vol. 19, no. 84, pp. 1–5, 2018.
- [16] C. Molnar, *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*. The author, 2019. [Online]. Available: <https://christophm.github.io/interpretable-ml-book/>
- [17] F. Wu and X. Bai, "Insertgnn: Can graph neural networks outperform humans in TOEFL sentence insertion problem?" *CoRR*, vol. abs/2103.15066, 2021. [Online]. Available: <https://arxiv.org/abs/2103.15066>

-
- [18] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3642–3649.
- [19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, p. 84–90, May 2017. [Online]. Available: <https://doi.org/10.1145/3065386>
- [20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv 1409.1556*, 09 2014.
- [21] A. Rai, "Explainable ai: From black box to glass box," *Journal of the Academy of Marketing Science*, vol. 48, no. 1, pp. 137–141, 2020.
- [22] K. Gade, S. C. Geyik, K. Kenthapadi, V. Mithal, and A. Taly, "Explainable ai in industry," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 3203–3204.
- [23] J. M. Benítez, J. L. Castro, and I. Requena, "Are artificial neural networks black boxes?" *IEEE Transactions on Neural Networks*, vol. 8, no. 5, pp. 1156–1164, 1997.
- [24] D. Gunning, E. Vorm, J. Y. Wang, and M. Turek, "Darpa's explainable ai (xai) program: A retrospective," *Applied AI Letters*, vol. 2, no. 4, p. e61, 2021.
- [25] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey and benchmarking of machine learning accelerators," *CoRR*, vol. abs/1908.11348, 2019. [Online]. Available: <http://arxiv.org/abs/1908.11348>
- [26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.
- [27] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.
- [28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: <http://arxiv.org/abs/1512.03385>

- [29] J. Orbach, "Principles of Neurodynamics. Perceptrons and the Theory of Brain Mechanisms." *Archives of General Psychiatry*, vol. 7, no. 3, pp. 218–219, 09 1962. [Online]. Available: <https://doi.org/10.1001/archpsyc.1962.01720030064010>
- [30] J. Mira and F. Sandoval, *From Natural to Artificial Neural Computation: International Workshop on Artificial Neural Networks, Malaga-Torremolinos, Spain, June 7 – 9, 1995 Proceedings (Lecture Notes in Computer Science, 930)*, 1995th ed. Springer.
- [31] Y. Blumenfeld, D. Gilboa, and D. Soudry, "Beyond signal propagation: is feature diversity necessary in deep neural network initialization?" in *International Conference on Machine Learning*. PMLR, 2020, pp. 960–969.
- [32] Y. Timoshenkova, N. Safiullin, and S. Porshnev, "About influence of weight initialization algorithms on accuracy of the forecast with lstm-net for harmonic signals," in *2021 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology (USBREIT)*, 2021, pp. 0330–0333.
- [33] M. Karouia, T. Denoeux, and R. Langelle, "Influence of weight initialization on multilayer perceptron performance," in *Proc. ICANN*, vol. 1, 1995, pp. 33–38.
- [34] R. Pascanu, T. Mikolov, and Y. Bengio, "Understanding the exploding gradient problem," *CoRR*, vol. abs/1211.5063, 2012. [Online]. Available: <http://arxiv.org/abs/1211.5063>
- [35] S. Kong and M. Takatsuka, "Hexpo: A vanishing-proof activation function," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 2562–2567.
- [36] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient BackProp". Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 9–48. [Online]. Available: https://doi.org/10.1007/978-3-642-35289-8_3
- [37] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015. [Online]. Available: <http://arxiv.org/abs/1502.01852>
- [38] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

-
- [39] L. Parisi, D. Neagu, R. Ma, and F. Campean, "Qrelu and m-qrelu: Two novel quantum activation functions to aid medical diagnostics," *CoRR*, vol. abs/2010.08031, 2020. [Online]. Available: <https://arxiv.org/abs/2010.08031>
- [40] J. Bridle, *Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition*, 01 1990, pp. 227–236.
- [41] R. HECHT-NIELSEN, "Theory of the backpropagation neural network, based on "nonindent" by robert hecht-nielsen, which appeared in proceedings of the international joint conference on neural networks 1, 593–611, june 1989. © 1989 ieee." in *Neural Networks for Perception*, H. Wechsler, Ed. Academic Press, 1992, pp. 65–93. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780127412528500108>
- [42] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," *CoRR*, vol. abs/1206.5533, 2012. [Online]. Available: <http://arxiv.org/abs/1206.5533>
- [43] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.
- [44] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 07 2011.
- [45] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.
- [46] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," *CoRR*, vol. abs/1710.09282, 2017. [Online]. Available: <http://arxiv.org/abs/1710.09282>
- [47] D. D. Kalamkar, D. Mudigere, N. Mellempudi, D. Das, K. Banerjee, S. Avancha, D. T. Vooturi, N. Jammalamadaka, J. Huang, H. Yuen, J. Yang, J. Park, A. Heinecke, E. Georganas, S. Srinivasan, A. Kundu, M. Smelyanskiy, B. Kaul, and P. Dubey, "A study of BFLOAT16 for deep learning training," *CoRR*, vol. abs/1905.12322, 2019. [Online]. Available: <http://arxiv.org/abs/1905.12322>

- [48] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," in *Proceedings of the Deep Learning and Unsupervised Feature Learning NIPS 2011 Workshop*, 2011, p. 4.
- [49] D. Williamson, "Dynamically scaled fixed point arithmetic," in *[1991] IEEE Pacific Rim Conference on Communications, Computers and Signal Processing Conference Proceedings*, 1991, pp. 315–318 vol.1.
- [50] Y. H. Oh, Q. Quan, D. Kim, S. Kim, J. Heo, S. Jung, J. Jang, and J. W. Lee, "A portable, automatic data quantizer for deep neural networks," in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '18. Association for Computing Machinery, 2018. [Online]. Available: <https://doi.org/10.1145/3243176.3243180>
- [51] D. Touretzky, *Advances in Neural Information Processing Systems, 2*, mit press ed. Morgan Kaufmann Pub, 1990, vol. 14.
- [52] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," *CoRR*, vol. abs/1504.04788, 2015. [Online]. Available: <http://arxiv.org/abs/1504.04788>
- [53] A. Alqahtani, X. Xie, E. Essa, and M. W. Jones, "Neuron-based network pruning based on majority voting," in *2020 25th International Conference on Pattern Recognition (ICPR)*, 2021, pp. 3090–3097.
- [54] D. Song, P. Zhang, and F. Li, "Speeding up deep convolutional neural networks based on tucker-cp decomposition," *Proceedings of the 2020 5th International Conference on Machine Learning Technologies*, 2020.
- [55] C. Tai, T. Xiao, Y. Zhang, X. Wang, and W. E, "Convolutional neural networks with low-rank regularization," *arXiv preprint arXiv:1511.06067*, 2016.
- [56] D. B. Paul and J. M. Baker, "The design for the wall street journal-based CSR corpus," in *Speech and Natural Language: Proceedings of a Workshop Held at Harriman, New York, February 23-26, 1992*. Association for Computational Linguistics, 1992, pp. 357–362.
- [57] T. N. Sainath, B. Kingsbury, V. Sindhvani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional

- output targets,” in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 6655–6659.
- [58] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. N. Choudhary, and S. Chang, “Fast neural networks with circulant projections,” *CoRR*, vol. abs/1502.03436, 2015. [Online]. Available: <http://arxiv.org/abs/1502.03436>
- [59] Z. Yang, M. Moczulski, M. Denil, N. de Freitas, A. J. Smola, L. Song, and Z. Wang, “Deep fried convnets,” *CoRR*, vol. abs/1412.7149, 2014.
- [60] S. R. Islam, W. Eberle, S. K. Ghafoor, and M. Ahmed, “Explainable artificial intelligence approaches: A survey,” *CoRR*, vol. abs/2101.09429, 2021. [Online]. Available: <https://arxiv.org/abs/2101.09429>
- [61] Z. Hellwig, *Linear Regression and its Application to Economics*. Elsevier, 1963.
- [62] M. Fritz, “Improved output gap estimates and forecasts using a local linear regression,” *Engineering Proceedings*, vol. 5, no. 1, 2021.
- [63] X. Su, X. Yan, and C.-L. Tsai, “Linear regression,” *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 4, no. 3, pp. 259–277, 2012.
- [64] J. W. Pratt, “Dividing the indivisible: Using simple symmetry to partition variance explained,” in *Proceedings of the 2nd International Conference in Statistics*, T. Pukkila and S. Puntanen, Eds., Tampere, Finland, 1987, pp. 245–260.
- [65] J. Bring, “A geometric approach to compare variables in a regression model,” *The American Statistician*, vol. 50, no. 1, pp. 57–62, 1996.
- [66] —, “A geometric approach to compare variables in a regression model,” *The American Statistician*, vol. 50, no. 1, pp. 57–62, 1996. [Online]. Available: <http://www.jstor.org/stable/2685045>
- [67] N. Seedorff and G. Brown, “totalvis: A principal components approach to visualizing total effects in black box models,” *SN Computer Science*, vol. 2, 05 2021.
- [68] A. Brenning, “Transforming feature space to interpret machine learning models,” *CoRR*, vol. abs/2104.04295, 2021. [Online]. Available: <https://arxiv.org/abs/2104.04295>

- [69] L. Li, X. Zhang, and M. Xue, "Explaining information gain and information gain ratio in information theory," vol. 7, pp. 2385–2391, 01 2013.
- [70] R. J. Urbanowicz, M. Meeker, W. La Cava, R. S. Olson, and J. H. Moore, "Relief-based feature selection: Introduction and review," *Journal of Biomedical Informatics*, vol. 85, pp. 189–203, 2018.
- [71] E. Scornet, "Trees, forests, and impurity-based variable importance," 2021.
- [72] S. Nembrini, I. R. König, and M. N. Wright, "The revival of the Gini importance?" *Bioinformatics*, vol. 34, no. 21, pp. 3711–3718, 05 2018.
- [73] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, "Classification and regression trees. brooks," *Wadsworth and Brooks, Monterey, CA*, 1984.
- [74] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81–106, 2004.
- [75] K. Mittal, D. Khanduja, and P. C. Tewari, ""an insight into 'decision tree analysis'," *World Wide Journal of Multidisciplinary Research and Development*, vol. 3, no. 12, pp. 111–115, 2017.
- [76] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should I trust you?": Explaining the predictions of any classifier," *CoRR*, vol. abs/1602.04938, 2016. [Online]. Available: <http://arxiv.org/abs/1602.04938>
- [77] S. M. Lundberg and S. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 4765–4774.
- [78] L. S. Shapley, "A value for n-person games," *Contributions to the Theory of Games*, vol. 2, no. 28, pp. 307–317, 1953.
- [79] E. Štrumbelj and I. Kononenko, "Explaining prediction models and individual predictions with feature contributions," *Knowledge and Information Systems*, vol. 41, pp. 647–665, 12 2013.

-
- [80] S. M. Lundberg, G. G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S. Lee, “Explainable AI for trees: From local explanations to global understanding,” *CoRR*, vol. abs/1905.04610, 2019.
- [81] S. Lundberg and S.-I. Lee, “A unified approach to interpreting model predictions,” 2017.
- [82] S. Wachter, B. D. Mittelstadt, and C. Russell, “Counterfactual explanations without opening the black box: Automated decisions and the GDPR,” *CoRR*, vol. abs/1711.00399, 2017. [Online]. Available: <http://arxiv.org/abs/1711.00399>
- [83] S. Dandl, C. Molnar, M. Binder, and B. Bischl, “Multi-objective counterfactual explanations,” *Lecture Notes in Computer Science*, p. 448–469, 2020. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-58112-1_31
- [84] D. Coppersmith, “Rectangular matrix multiplication revisited,” *Journal of Complexity*, vol. 13, no. 1, pp. 42–49, 1997.
- [85] D. Coppersmith and S. Winograd, “Matrix multiplication via arithmetic progressions,” *Journal of Symbolic Computation*, vol. 9, no. 3, pp. 251–280, 1990, computational algebraic complexity editorial. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0747717108800132>
- [86] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatin, A. Novikov, F. J. R. Ruiz, J. Schrittwieser, G. Swirszcz, D. Silver, D. Hassabis, and P. Kohli, “Discovering faster matrix multiplication algorithms with reinforcement learning,” *Nature*, vol. 610, pp. 47–53, 2022.
- [87] N. P. Jouppi, C. Young, N. Patil, D. Patterson *et al.*, “In-datacenter performance analysis of a tensor processing unit,” in *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM, 2017, pp. 1–12.
- [88] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski, “Chromium: A stream-processing framework for interactive rendering on clusters,” in *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH ’02. New York, NY, USA: Association for Computing Machinery, 2002, p. 693–702. [Online]. Available: <https://doi.org/10.1145/566570.566639>

- [89] V. Strassen, "Gaussian elimination is not optimal," *Numer. Math.*, vol. 13, no. 4, pp. 354–356, Aug. 1969. [Online]. Available: <http://dx.doi.org/10.1007/BF02165411>
- [90] D. Coppersmith and S. Winograd, "On the asymptotic complexity of matrix multiplication," *SIAM Journal on Computing*, vol. 11, no. 3, pp. 472–492, 1982.
- [91] W. M. Gentleman, "Matrix multiplication and fast fourier transforms," *The Bell System Technical Journal*, vol. 47, no. 6, pp. 1099–1103, 1968.
- [92] R. C. Agarwal and F. G. Gustavson, "A parallel implementation of matrix multiplication and lu factorization on the ibm 3090," in *Proceedings of the IFIP WG*, vol. 2, 1988, pp. 217–221.
- [93] D. Ren and R. Suda, "Power efficient large matrices multiplication by load scheduling on multi-core and gpu platform with cuda," in *2009 International Conference on Computational Science and Engineering*, vol. 1, 2009, pp. 424–429.
- [94] X. Cui, Y. Chen, and H. Mei, "Improving performance of matrix multiplication and fft on gpu," in *2009 15th International Conference on Parallel and Distributed Systems*, 2009, pp. 42–48.
- [95] Z. Huang, N. Ma, S. Wang, and Y. Peng, "Gpu computing performance analysis on matrix multiplication," *The Journal of Engineering*, vol. 2019, no. 23, pp. 9043–9048, 2019. [Online]. Available: <https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/joe.2018.9178>
- [96] F. Dominguez, J. Unger, M. Traube, B. Mant, C. Ertler, and W. Lechner, "Encoding-independent optimization problem formulation for quantum computing," 2023.
- [97] V. Y. Pan, "Strassen's algorithm is not optimal trilinear technique of aggregating, uniting and canceling for constructing fast algorithms for matrix operations," in *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, ser. SFCS '78. Washington, DC, USA: IEEE Computer Society, 1978, pp. 166–176. [Online]. Available: <http://dx.doi.org/10.1109/SFCS.1978.34>
- [98] D. Bini, M. Capovani, F. Romani, and G. Lotti, " $O(n^{2.7799})$ Complexity for $n \times n$ Approximate Matrix Multiplication," *Information Processing Letters*, vol. 8, pp. 234–235, 1979.

-
- [99] A. Schönhage, "Partial and total matrix multiplication," *SIAM Journal on Computing*, vol. 10, no. 3, pp. 434–455, 1981.
- [100] F. Romani, "Some properties of disjoint sums of tensors related to matrix multiplication," *SIAM J. Comput.*, vol. 11, pp. 263–267, 1982.
- [101] V. Strassen, "Relative bilinear complexity and matrix multiplication." *Journal für die reine und angewandte Mathematik*, vol. 375, pp. 406–443, 1987.
- [102] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," *J. Symb. Comput.*, vol. 9, no. 3, pp. 251–280, Mar. 1990. [Online]. Available: [http://dx.doi.org/10.1016/S0747-7171\(08\)80013-2](http://dx.doi.org/10.1016/S0747-7171(08)80013-2)
- [103] A. J. Stothers, "On the complexity of matrix multiplication," *Contributions to the Theory of Computing*, vol. 3, no. 1, pp. 47–67, 2010.
- [104] V. V. Williams, "Multiplying matrices in $\tilde{O}(n^2.373)$ time," in *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, 2012, pp. 887–898.
- [105] F. Le Gall, "Powers of tensors and fast matrix multiplication," in *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ser. ISSAC '14. New York, NY, USA: ACM, 2014, pp. 296–303. [Online]. Available: <http://doi.acm.org/10.1145/2608628.2608664>
- [106] R. W. Brockett and D. Dobkin, "On the number of multiplications required for matrix multiplication," *SIAM Journal on Computing*, vol. 5, no. 4, pp. 624–628, 1976.
- [107] D. Coppersmith, "Rapid multiplication of rectangular matrices," *SIAM Journal on Computing*, vol. 11, no. 3, pp. 467–471, 1982.
- [108] F. Le Gall, "Faster algorithms for rectangular matrix multiplication," in *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*. IEEE, 2012, pp. 514–523.
- [109] R. Freivalds, "Probabilistic machines can use less running time." in *IFIP congress*, vol. 839, 1977, p. 842.
- [110] A. Frieze, R. Kannan, and S. Vempala, "Fast monte-carlo algorithms for finding low-rank approximations," in *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No.98CB36280)*, 1998, pp. 370–378.

- [111] D. Achlioptas and F. McSherry, "Fast computation of low-rank matrix approximations," *J. ACM*, vol. 54, p. 9, 2007.
- [112] P. Drineas, R. Kannan, and M. Mahoney, "Fast monte carlo algorithms for matrices ii: Computing a low-rank approximation to a matrix," *SIAM Journal on Computing*, vol. 36, 05 2004.
- [113] H. Y. Cheung, T. C. Kwok, and L. C. Lau, "Fast matrix rank algorithms and applications," *ArXiv*, vol. abs/1203.6705, 2013.
- [114] G. Valiant, "Finding correlations in subquadratic time, with applications to learning parities and juntas," in *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, 2012, pp. 11–20.
- [115] J. Bentley, B. Weide, and A. Yao, "Optimal expected-time algorithms for closest point problems," *ACM Trans. Math. Softw.*, vol. 6, pp. 563–580, 12 1980.
- [116] E. Cohen and D. D. Lewis, "Approximating matrix multiplication for pattern recognition tasks," in *SODA '97*, 1997.
- [117] Z. Bar-Yossef, "Sampling lower bounds via information theory," in *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*. ACM, 2003, pp. 335–344.
- [118] T. Sarlos, "Improved approximation algorithms for large matrices via random projections," in *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*. IEEE, 2006, pp. 143–152.
- [119] P. Drineas and R. Kannan, "Fast monte carlo algorithms for approximate matrix multiplication," in *In Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, 2001, pp. 452–459.
- [120] P. Drineas, R. Kannan, and M. W. Mahoney, "Fast monte carlo algorithms for matrices i: Approximating matrix multiplication," *SIAM Journal on Computing*, vol. 36, no. 1, pp. 132–157, 2006.
- [121] V. L. Arlazarov, E. A. Dinic, M. A. Kronrod, and I. A. Faradžev, "On economical construction of the transitive closure of a directed graph," *Soviet Mathematics—Doklady*, vol. 11, no. 5, pp. 1209–1210, 1970.

-
- [122] N. Bansal and R. Williams, "Regularity lemmas and combinatorial algorithms," in *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*. IEEE, 2009, pp. 745–754.
- [123] T. M. Chan, "Speeding up the four-russians algorithm by about one more logarithmic factor," in *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2015, pp. 212–217.
- [124] H. Yu, "An improved combinatorial algorithm for boolean matrix multiplication," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2015, pp. 1094–1105.
- [125] J. Wiedermann, "Fast nondeterministic matrix multiplication via derandomization of freivalds' algorithm," in *Theoretical Computer Science*. Springer, 2014, pp. 123–135.
- [126] A. Lingas and D. Sledneu, "Vector convolution in $o(n)$ steps and matrix multiplication in $o(n^2)$ steps:-)." in *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 21, 2014, p. 39.
- [127] I. Korec and J. Wiedermann, "Deterministic verification of integer matrix multiplication in quadratic time," in *SOFSEM 2014: Theory and Practice of Computer Science*. Springer, 2014, pp. 375–382.
- [128] W. B. Johnson and J. Lindenstrauss, "Extensions of lipschitz mappings into a hilbert space," *Contemporary Mathematics*, vol. 26, 1984.
- [129] P. Drineas and M. W. Mahoney, "Randnla: Randomized numerical linear algebra," *Commun. ACM*, vol. 59, no. 6, pp. 80–90, May 2016. [Online]. Available: <http://doi.acm.org/10.1145/2842602>
- [130] R. Yuster and U. Zwick, "Fast sparse matrix multiplication," vol. 1, 07 2004.
- [131] M. Iwen and C. Spencer, "A note on compressed sensing and the complexity of matrix multiplication," *Information Processing Letters*, vol. 109, no. 10, pp. 468–471, 2009. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0020019009000131>
- [132] A. Lingas, "A fast output-sensitive algorithm for boolean matrix multiplication," *Algorithmica*, vol. 61, pp. 36–50, 09 2009.

- [133] R. Pagh, “Compressed matrix multiplication,” *ACM Transactions on Computation Theory*, vol. 5, 08 2011.
- [134] N. Alon, Y. Matias, and M. Szegedy, “The space complexity of approximating the frequency moments,” *Journal of Computer and System Sciences*, vol. 58, no. 1, pp. 137–147, 1999. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0022000097915452>
- [135] K. Kutzkov, “Deterministic algorithms for skewed matrix products,” *Leibniz International Proceedings in Informatics, LIPIcs*, vol. 20, 09 2012.
- [136] A. Rahimi and B. Recht, “Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning,” in *Advances in neural information processing systems*, 2009, pp. 1313–1320.
- [137] Q. V. Le, T. Sarlós, and A. J. Smola, “Fastfood: Approximate kernel expansions in loglinear time,” *CoRR*, vol. abs/1408.3060, 2014. [Online]. Available: <http://arxiv.org/abs/1408.3060>
- [138] M. X. Goemans and D. P. Williamson, “Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming,” *J. ACM*, vol. 42, no. 6, pp. 1115–1145, Nov. 1995. [Online]. Available: <http://doi.acm.org/10.1145/227683.227684>
- [139] M. S. Charikar, “Similarity estimation techniques from rounding algorithms,” in *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, ser. STOC ’02. New York, NY, USA: ACM, 2002, pp. 380–388. [Online]. Available: <http://doi.acm.org/10.1145/509907.509965>
- [140] J. Chen, Y. Liu, H. Zhang, S. Hou, and J. Yang, “Propagating asymptotic-estimated gradients for low bit width quantized neural networks,” *IEEE Journal of Selected Topics in Signal Processing*, pp. 1–1, 2020.
- [141] R. Rigamonti, A. Sironi, V. Lepetit, and P. Fua, “Learning separable filters,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2013, pp. 2754–2761.

-
- [142] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," *CoRR*, vol. abs/1306.0543, 2013. [Online]. Available: <http://arxiv.org/abs/1306.0543>
- [143] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *arXiv preprint arXiv:1405.3866*, 2014.
- [144] A. Novikov, D. Podoprikin, A. Osokin, and D. Vetrov, "Tensorizing neural networks," in *Proceedings of the 28th Conference on Neural Information Processing Systems (NIPS)*, 2015, pp. 442–450.
- [145] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278 – 2324, 12 1998.
- [146] L. N. Darlow, E. J. Crowley, A. Antoniou, and A. J. Storkey, "CINIC-10 is not imagenet or CIFAR-10," *CoRR*, vol. abs/1810.03505, 2018. [Online]. Available: <http://arxiv.org/abs/1810.03505>
- [147] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, 05 2012.
- [148] T. Miller, "Explanation in artificial intelligence: Insights from the social sciences," *Artif. Intell.*, vol. 267, pp. 1–38, 2019.
- [149] A. Adadi and M. Berrada, "Peeking inside the black-box: A survey on explainable artificial intelligence (XAI)," *IEEE Access*, vol. 6, pp. 52 138–52 160, 2018.
- [150] D. Carvalho, E. Pereira, and J. Cardoso, "Machine learning interpretability: A survey on methods and metrics," *Electronics*, vol. 8, p. 832, 07 2019.
- [151] M. Kapcia, H. Eshkiki, J. Duell, X. Fan, S. Zhou, and B. Mora, "Exmed: An ai tool for experimenting explainable ai techniques on medical data analytics," in *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, 2021, pp. 841–845.
- [152] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

- [153] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [154] C. Molnar, *Interpretable Machine Learning*. Leanpub, 2019, <https://christophm.github.io/interpretable-ml-book/>.
- [155] A. Goldstein, A. Kapelner, J. Bleich, and E. Pitkin, "Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation," 2014.
- [156] S. Tonekaboni, S. Joshi, M. D. McCradden, and A. Goldenberg, "What clinicians want: Contextualizing explainable machine learning for clinical end use," in *Proceedings of the 4th Machine Learning for Healthcare Conference*, ser. Proceedings of Machine Learning Research, F. Doshi-Velez, J. Fackler, K. Jung, D. Kale, R. Ranganath, B. Wallace, and J. Wiens, Eds., vol. 106. Ann Arbor, Michigan: PMLR, 09–10 Aug 2019, pp. 359–380. [Online]. Available: <http://proceedings.mlr.press/v106/tonekaboni19a.html>
- [157] M. T. Ribeiro, S. Singh, and C. Guestrin, "'why should I trust you?': Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*. ACM, 2016, pp. 1135–1144.
- [158] H. F. da Cruz, B. Pfahringer, T. Martensen, F. Schneider, A. Meyer, E. Böttinger, and M.-P. Schapranow, "Using interpretability approaches to update "black-box" clinical prediction models: an external validation study in nephrology," *Artificial Intelligence in Medicine*, vol. 111, p. 101982, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0933365720312471>
- [159] P. Mróz, A. Quemy, M. Ślęzyński, K. Kluza, and P. Jemioło, "Gbex - towards graph-based explanations," in *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, 2020, pp. 112–117.
- [160] A. Hanif, X. Zhang, and S. Wood, "A survey on explainable artificial intelligence techniques and challenges," in *2021 IEEE 25th International Enterprise Distributed Object Computing Workshop (EDOCW)*, 2021, pp. 81–89.

-
- [161] T. Laugel, X. Renard, M. Lesot, C. Marsala, and M. Detyniecki, "Defining locality for surrogates in post-hoc interpretability," *CoRR*, vol. abs/1806.07498, 2018. [Online]. Available: <http://arxiv.org/abs/1806.07498>
- [162] T. Laugel, X. Renard, M.-J. Lesot, C. Marsala, and M. Detyniecki, "Defining locality for surrogates in post-hoc interpretability," 2018.
- [163] M. Pennisi, I. Kavasidis, C. Spampinato, V. Schinina, S. Palazzo, F. P. Salanitri, G. Bellitto, F. Rundo, M. Aldinucci, M. Cristofaro, P. Campioni, E. Pianura, F. Di Stefano, A. Petrone, F. Albarello, G. Ippolito, S. Cuzzocrea, and S. Conoci, "An explainable ai system for automated covid-19 assessment and lesion categorization from ct-scans," *Artificial Intelligence in Medicine*, p. 102114, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S093336572100107X>
- [164] J. Schindelin, I. Arganda-Carreras, E. Frise, V. Kaynig, M. Longair, T. Pietzsch, S. Preibisch, C. Rueden, S. Saalfeld, B. Schmid, J.-Y. Tinevez, D. J. White, V. Hartenstein, K. Eliceiri, P. Tomancak, and A. Cardona, "Fiji: an open-source platform for biological-image analysis," *Nature Methods*, vol. 9, no. 7, pp. 676–682, 2012.
- [165] M. S. Hammoodi, H. A. A. Essa, and W. A. Hanon, "The Waikato Open Source Frameworks (WEKA and MOA) for Machine Learning Techniques," *Journal of Physics: Conference Series*, vol. 1804, no. 1, p. 012133, 2021.
- [166] T. Meinl, "What's new in KNIME?" *Journal of Cheminformatics*, vol. 4, no. S1, 2012.
- [167] S. S. Dhruva, J. S. Ross, J. G. Akar, B. Caldwell, K. Childers, W. Chow, L. Ciaccio, P. Coplan, J. Dong, H. J. Dykhoff, S. Johnston, T. Kellogg, C. Long, P. A. Noseworthy, K. Roberts, A. Saha, A. Yoo, and N. D. Shah, "Aggregating multiple real-world data sources using a patient-centered health-data-sharing platform," *NPJ Digit Med*, vol. 3, p. 60, 2020.
- [168] M. Kapcia, "ExMed on Github," <https://github.com/983046/ExMed>, 2021, [Online; accessed 19-July-2021].
- [169] J. A. Duell, X. Fan, B. Burnett, G. Aarts, and S. Zhou, "A comparison of explanations given by explainable artificial intelligence methods on analysing electronic health records," in *2021 IEEE EMBS International Conference on Biomedical and Health Informatics (BHI) (IEEE BHI 2021)*, Athens, Greece, Jul. 2021.

- [170] S. Flaxman, S. Mishra, A. Gandy, H. Unwin, H. Coupland, T. Mellan, H. Zhu, T. Berah, J. Eaton, P. Perez Guzman *et al.*, “Report 13: Estimating the number of infections and the impact of non-pharmaceutical interventions on covid-19 in 11 european countries,” Imperial College London, Tech. Rep., 2020.
- [171] J. T. Wu, K. Leung, M. Bushman, N. Kishore, R. Niehus, P. M. de Salazar, B. J. Cowling, M. Lipsitch, and G. M. Leung, “Estimating clinical severity of covid-19 from the transmission dynamics in wuhan, china,” *Nature Medicine*, pp. 1–5, 2020.
- [172] H. Sung, J. Ferlay, R. L. Siegel, M. Laversanne, I. Soerjomataram, A. Jemal, and F. Bray, “Global cancer statistics 2020: Globocan estimates of incidence and mortality worldwide for 36 cancers in 185 countries,” *CA: A Cancer Journal for Clinicians*, pp. 209–249, 2021.
- [173] J. Smith, J. Everhart, W. Dickson, W. Knowler, and R. Johannes, “Using the adap learning algorithm to forcast the onset of diabetes mellitus,” *Proceedings - Annual Symposium on Computer Applications in Medical Care*, vol. 10, 11 1988.
- [174] R. A. Fisher, “The use of multiple measurements in taxonomic problems,” *Annals of Eugenics*, vol. 7, no. 2, pp. 179–188, 1936. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-1809.1936.tb02137.x>
- [175] T. Chen, E. Keravnou-Papailiou, and G. Antoniou, “Medical analytics for healthcare intelligence – recent advances and future directions,” *Artificial Intelligence in Medicine*, vol. 112, pp. 1–5, Feb. 2021.

Appendix A

Implementation of a Relevant Algorithm

Appendix B

Supplementary Data

Compression Results for MNIST, CIFAR10, CIFAR100 and CINIC-10

Compression rates of Fully Connected layers and overall network according to the number k of hyper-planes chosen and whether rotation of random numbers has been used (rot). The modified/compressed layers are in bold. FC: Fully Connected layer; BN: Batch Normalization layer; ReLU: Rectified Linear Unit layer; SFMX: SoftMax layer.

Table B.1: (a) MNIST without Data Augmentation

28×28:DropOut:FC1024:BN:ReLU:DropOut: FC1024:BN:ReLU:DropOut:FC10:SFMX			
#Hyperplanes	Accuracy	Size of	Total Network
Original Net.	99.02%	7.11 MB (100%)	7.11 MB (100%)
$k = 256$	95.51%	0.11 MB (1.48%)	0.11 MB (1.48%)
$k = 512$	98.01%	0.18 MB (2.47%)	0.18 MB (2.47%)
$k = 1024$	98.61%	0.30 MB (4.23%)	0.30 MB (4.23%)
$k = 2048$	98.90%	0.55 MB (7.75%)	0.55 MB (7.75%)
$k = 256$ (rot)	94.06%	0.11 MB (1.59%)	0.11 MB (1.59%)
$k = 512$ (rot)	97.55%	0.18 MB (2.58%)	0.18 MB (2.58%)
$k = 1024$ (rot)	98.78%	0.31 MB (4.34%)	0.31 MB (4.34%)
$k = 2048$ (rot)	98.82%	0.57 MB (7.97%)	0.57 MB (7.97%)

Table B.2: (b) MNIST without Data Augmentation

28×28:DropOut:FC1024:BN:ReLU:DropOut:FC1024:BN :ReLU:DropOut:FC1024:BN:ReLU:DropOut:FC10:SFMX			
Hyperplanes	Accuracy	Size of FC layers(%)	Total Network Size (%)
Original Net.	99.12%	11.1 MB (100%)	11.1 MB (100%)
$k = 256$	93.76%	0.14 MB (1.23%)	0.14 MB (1.23%)
$k = 512$	97.92%	0.24 MB (2.18%)	0.24 MB (2.18%)
$k = 1024$	98.61%	0.43 MB (3.87%)	0.43 MB (3.87%)
$k = 2048$	99.37%	0.80 MB (7.25%)	0.80 MB (7.25%)
$k = 256$ (rot)	94.06%	0.15 MB (1.34%)	0.15 MB (1.34%)
$k = 512$ (rot)	97.54%	0.25 MB (2.29%)	0.25 MB (2.29%)
$k = 1024$ (rot)	98.78%	0.44 MB (3.97%)	0.44 MB (3.97%)
$k = 2048$ (rot)	98.72%	0.83 MB (7.46%)	0.83 MB (7.46%)

Table B.3: (c) MNIST with Data Augmentation

28×28:DropOut:FC1024:BN:ReLU:DropOut: FC1024:BN:ReLU:DropOut:FC10:SFMX			
#Hyperplanes	Accuracy	Size of FC layers(%)	Total Network Size (%) (%)
Original Net.	99.29%	7.11 MB (100%)	7.11 MB (100%)
$k = 256$	91.08%	0.11 MB (1.48%)	0.11 MB (1.48%)
$k = 512$	97.08%	0.18 MB (2.47%)	0.18 MB (2.47%)
$k = 1024$	98.92%	0.30 MB (4.23%)	0.30 MB (4.23%)
$k = 2048$	99.38%	0.55 MB (7.75%)	0.55 MB (7.75%)
$k = 256$ (rot)	85.10%	0.11 MB (1.59%)	0.11 MB (1.59%)
$k = 512$ (rot)	93.43%	0.18 MB (2.58%)	0.18 MB (2.58%)
$k = 1024$ (rot)	97.98%	0.31 MB (4.34%)	0.31 MB (4.34%)
$k = 2048$ (rot)	99.14%	0.57 MB (7.97%)	0.57 MB (7.97%)

Table B.4: (d) MNIST with Data Augmentation

28×28:DropOut:FC1024:BN:ReLU:DropOut:FC1024:BN :ReLU:DropOut:FC1024:BN:ReLU:DropOut:FC10:SFMX			
#Hyperplanes	Accuracy	Size of FC layers(%)	Total Network Size (%)
Original Net.	99.52%	11.1 MB (100%)	11.1 MB (100%)
$k = 256$	88.43%	0.14 MB (1.23%)	0.14 MB (1.23%)
$k = 512$	95.25%	0.24 MB (2.18%)	0.24 MB (2.18%)
$k = 1024$	98.51%	0.43 MB (3.87%)	0.43 MB (3.87%)
$k = 2048$	99.37%	0.80 MB (7.25%)	0.80 MB (7.25%)
$k = 256$ (rot)	87.38%	0.15 MB (1.34%)	0.15 MB (1.34%)
$k = 512$ (rot)	95.66%	0.25 MB (2.29%)	0.25 MB (2.29%)
$k = 1024$ (rot)	98.78%	0.44 MB (3.97%)	0.44 MB (3.97%)
$k = 2048$ (rot)	99.43%	0.83 MB (7.46%)	0.83 MB (7.46%)

Table B.5: (e) CIFAR10

32×32×3:(VGG16 [20] Convolutional Part): FC1024:BN:ReLU:DropOut:FC1024:BN:ReLU:DropOut:FC10:SFMX			
#Hyperplanes	Accuracy	Size of FC layers(%)	Total Network Size (%)
Original Net.	93.03%	72.2 MB (100%)	128.42 MB (100%)
$k = 256$	92.88%	0.25 MB (0.35%)	56.39 MB (43.95%)
$k = 512$	93.01%	0.51 MB (0.70%)	56.64 MB (44.15%)
$k = 1024$	93.03%	1.01 MB (1.40%)	57.15 MB (44.54%)
$k = 2048$	93.03%	2.03 MB (2.81%)	58.30 MB (45.39%)
$k = 256$ (rot)	92.96%	0.29 MB (0.40%)	56.42 MB (43.98%)
$k = 512$ (rot)	92.96%	0.54 MB (0.75%)	56.67 MB (44.18%)
$k = 1024$ (rot)	93.03%	1.05 MB (1.45%)	57.18 MB (44.57%)
$k = 2048$ (rot)	93.03%	2.06 MB (2.86%)	58.20 MB (45.36%)

Table B.6: (f) CIFAR100

32×32×3:(VGG16 [20] Convolutional Part): FC1024:BN:ReLU:DropOut:FC1024:BN:ReLU:DropOut:FC100:SFMX			
#Hyperplanes	Accuracy	Size of FC layers(%)	Total Network Size (%)
Original Net.	71.84%	73.6 MB (100%)	129.7 MB (100%)
$k = 256$	71.15%	0.26 MB (0.35%)	56.39 MB (43.48%)
$k = 512$	71.79%	0.51 MB (0.70%)	56.64 MB (43.67%)
$k = 1024$	71.83%	1.02 MB (1.39%)	57.16 MB (44.07%)
$k = 2048$	71.84%	2.05 MB (2.78%)	58.18 MB (44.86%)
$k = 256$ (rot)	69.42%	0.29 MB (0.39%)	56.42 MB (43.50%)
$k = 512$ (rot)	71.02%	0.54 MB (0.73%)	56.67 MB (43.70%)
$k = 1024$ (rot)	71.50%	1.05 MB (1.42%)	57.18 MB (44.09%)
$k = 2048$ (rot)	71.80%	2.06 MB (2.81%)	58.20 MB (44.87%)

Table B.7: (g) CINIC10

32×32×3:(VGG16 [20] Convolutional Part): FC1024:BN:ReLU:DropOut:FC1024:BN:ReLU:DropOut:FC10:SFMX			
#Hyperplanes	Accuracy	Size of FC layers(%)	Total Network Size (%)
Original Net.	79.25%	72.2 MB (100%)	128.42 MB(100%)
$k = 256$	78.81%	0.25 MB (0.35%)	56.39 MB(43.95%)
$k = 512$	79.02%	0.51 MB (0.70%)	56.64 MB (44.15%)
$k = 1024$	79.25%	1.01 MB (1.40%)	57.15 MB(44.54%)
$k = 2048$	79.25%	2.03 MB (2.81%)	58.30 MB (45.39%)
$k = 256$ (rot)	78.92%	0.29 MB (0.40%)	56.42 MB (43.98%)
$k = 512$ (rot)	79.24%	0.54 MB (0.75%)	56.67 MB (44.18%)
$k = 1024$ (rot)	79.25%	1.05 MB (1.45%)	57.18 MB (44.57%)
$k = 2048$ (rot)	79.25%	2.06 MB (2.86%)	58.20 MB (45.36%)

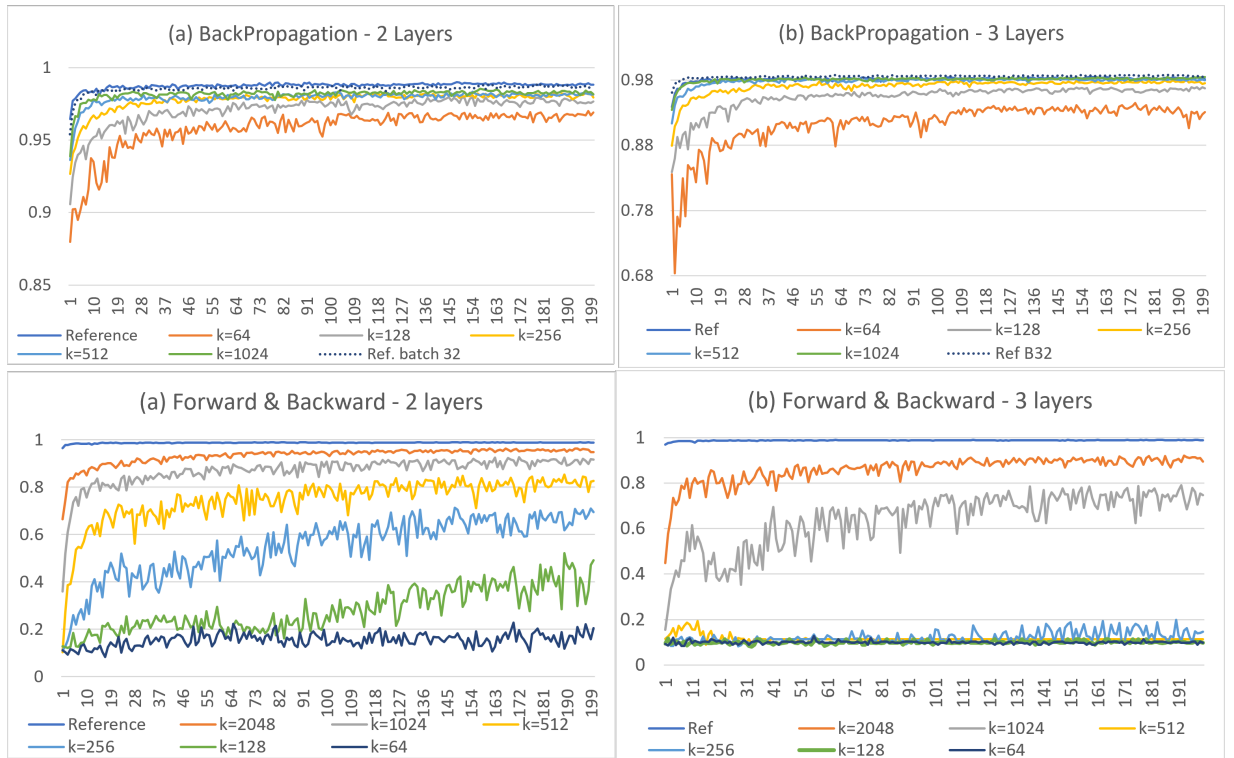


Figure B.1: Effect of replacing the standard matrix multiplication with the M^3 version for training the MNIST datasets in all but the last fully connected layer. The horizontal represents the number of epochs while the vertical axis represents the accuracy obtained on the testing dataset. We study backward-only, forward-only and backward-forward replacement cases. The number of planes k used is the same for each layer and each pass in a single experiment. The batch size used is 1024. The M^3 seems to be beneficial to back-propagation that shows a convergence similar to the use of a standard matrix multiplication with a sufficiently low number of planes.

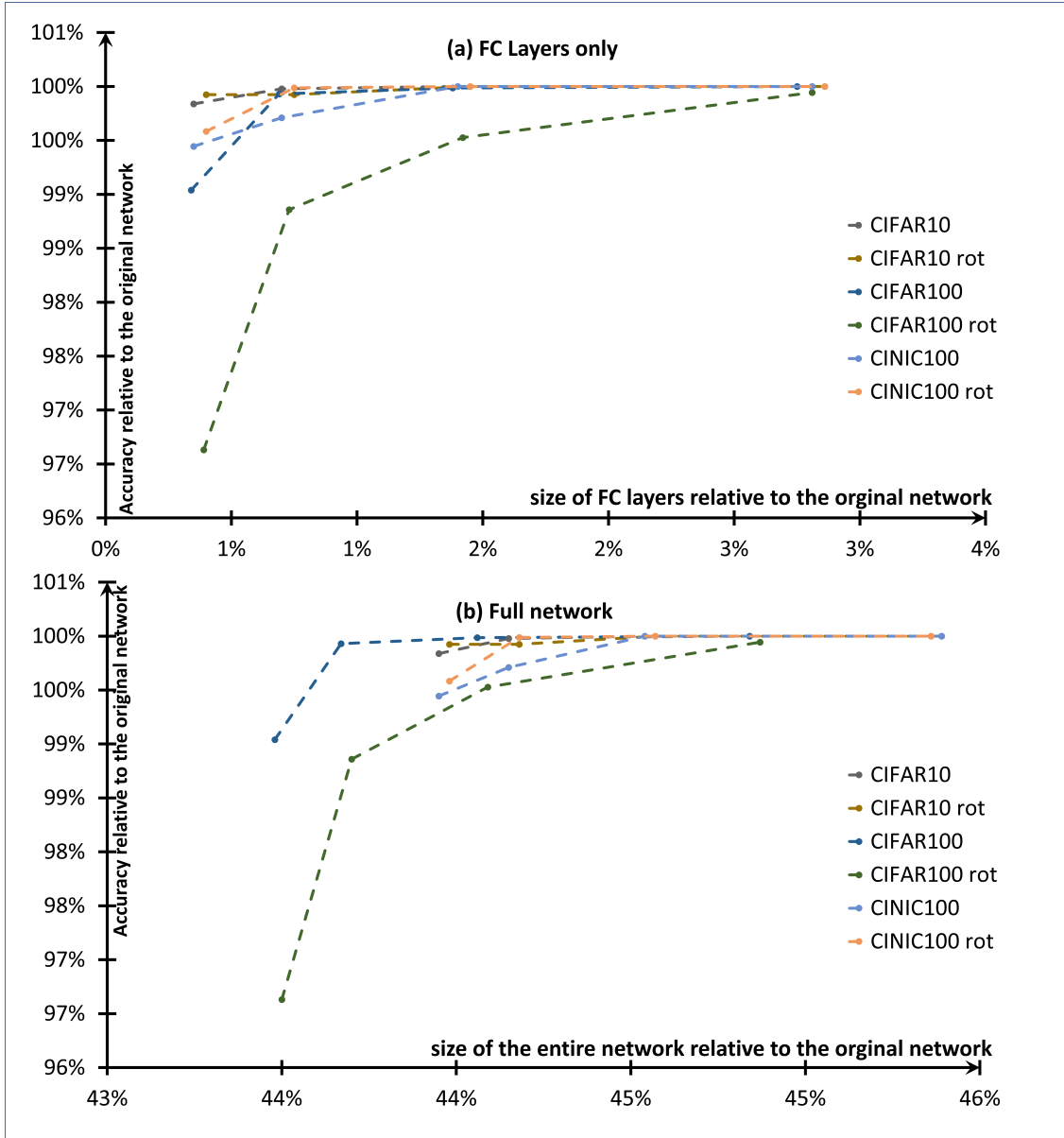


Figure B.2: Convergence of accuracy according to the number of hyperplanes used. The horizontal axis displays the compression obtained for just the internal fully-connected layers (a) and for the neural network as a whole (b). The vertical axis displays the obtained accuracy of the network relative to that of the original, unmodified network – with 100% meaning that the compressed network has the same accuracy as the original one. All network models have been tested with 256, 512, 1024 and 2048 hyperplanes as represented with continuous or dashed lines.

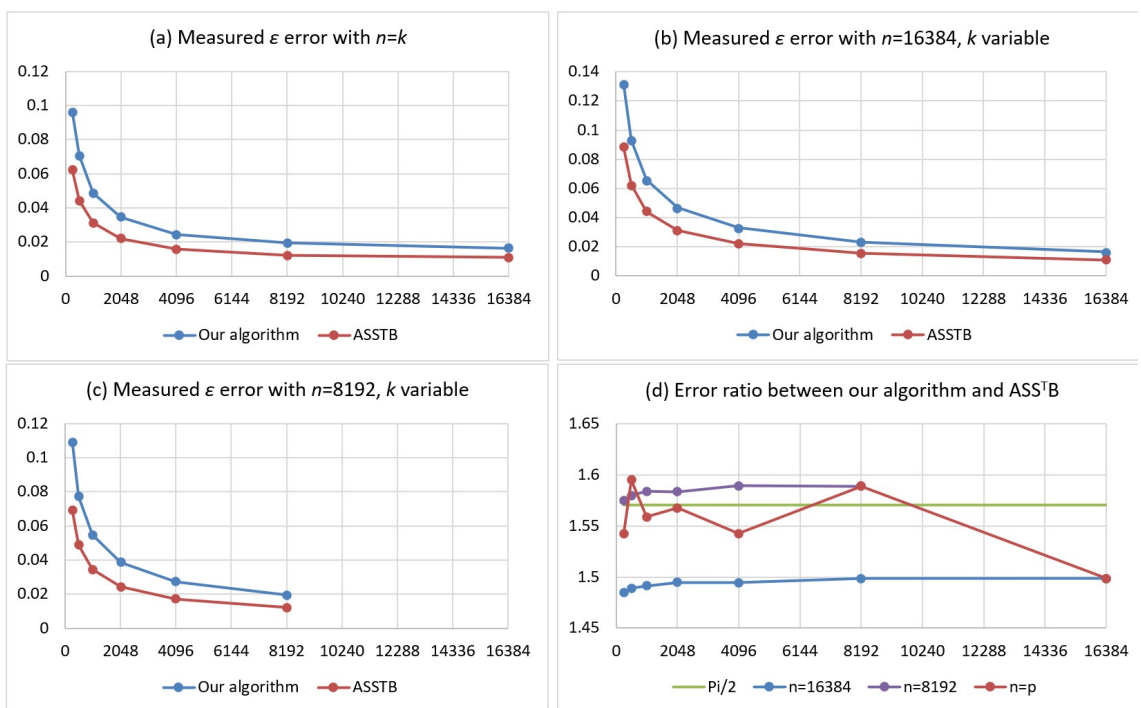
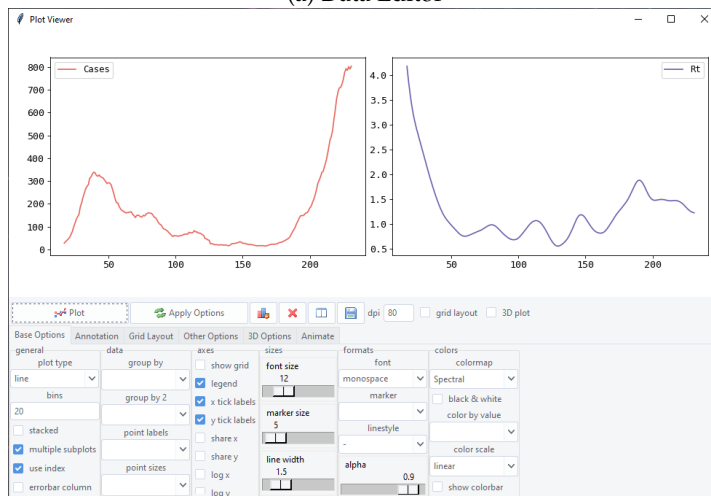


Figure B.3: Error comparison between our method and [1]. Results are given by either varying the sizes of matrix A and B with the number of samples p equal to n ($n = p$) or by fixing n and varying the number of samples used for the approximation.

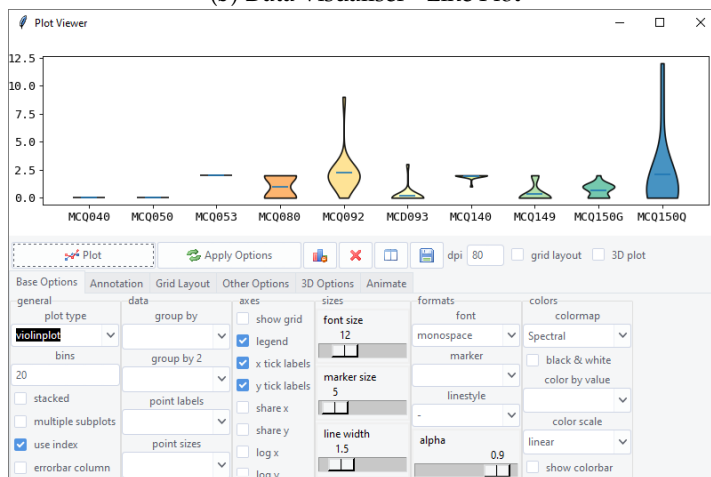
B. Supplementary Data

	SEQN	MCQ010	MCQ025	MCQ035	MCQ040	MCQ050	ASQ030	MCQ053	MCQ070	MCQ075	MC
1	73557.00	2.00	1.00	1.00	1.00	1.00	1.00	2.00	2.00	1.00	1.00
2	73558.00	1.00	8.00	1.00	1.00	2.00	2.00	2.00	2.00	1.00	2.00
3	73559.00	2.00	1.00	1.00	1.00	1.00	1.00	2.00	2.00	1.00	2.00
4	73560.00	2.00	1.00	1.00	1.00	1.00	1.00	2.00	1.00	1.00	1.00
5	73561.00	2.00	1.00	1.00	1.00	1.00	1.00	2.00	2.00	1.00	2.00
6	73562.00	2.00	1.00	1.00	1.00	1.00	1.00	2.00	2.00	1.00	1.00
7	73564.00	1.00	3.00	2.00	1.00	1.00	1.00	2.00	2.00	1.00	1.00
8	73565.00	2.00	1.00	1.00	1.00	1.00	1.00	2.00	2.00	1.00	2.00
9	73566.00	2.00	1.00	1.00	1.00	1.00	1.00	2.00	2.00	1.00	2.00
10	73567.00	2.00	1.00	1.00	1.00	1.00	1.00	2.00	2.00	1.00	2.00
11	73568.00	2.00	1.00	1.00	1.00	1.00	1.00	2.00	2.00	1.00	2.00
12	73570.00	2.00	1.00	1.00	1.00	1.00	1.00	2.00	1.00	1.00	1.00
13	73571.00	2.00	1.00	1.00	1.00	1.00	1.00	2.00	2.00	1.00	1.00
14	73572.00	1.00	7.00	1.00	2.00	2.00	2.00	2.00	1.00	1.00	1.00
15	73573.00	1.00	1.00	1.00	2.00	2.00	2.00	2.00	1.00	1.00	1.00
16	73574.00	2.00	1.00	1.00	1.00	1.00	1.00	2.00	2.00	1.00	2.00
17	73575.00	2.00	1.00	1.00	1.00	1.00	1.00	2.00	1.00	1.00	1.00
18	73576.00	2.00	1.00	1.00	1.00	1.00	1.00	2.00	2.00	1.00	2.00

(a) Data Editor

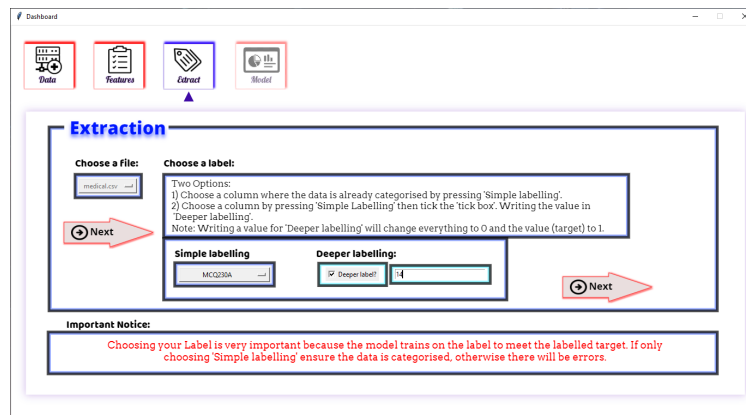


(b) Data Visualiser - Line Plot

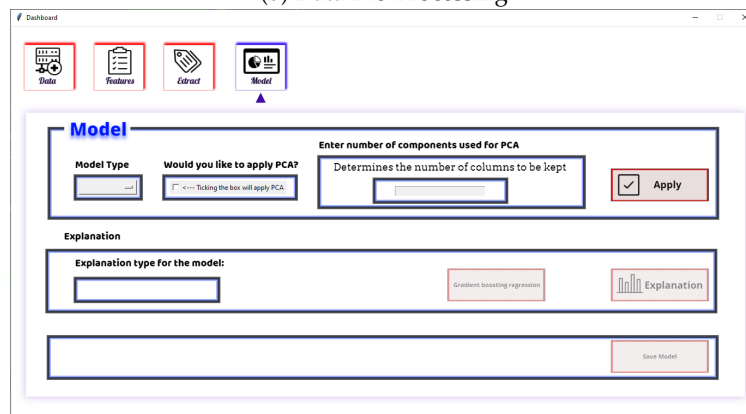


(c) Data Visualiser - Violin Plot

Figure B.4: Data exploration tools in ExMed. (a) is the Data Editor that supports standard data editing functions. (b) and (c) are the Data Visualiser that supports different plot types such as Line, Scatter, Bar, Histogram, Violin Plots and Pie chart. For each plot type, various customisation options are implemented, including changing the axes, layout, and adding texts.



(b) Data Pre-Processing



(c) Model Selection and Construction

Figure B.5: ExMed interface for some of the main activities as describe in Fig. 4.1. (b) Data is optionally pre-processed with some of the plugins available. (c) A model is created, with the option of reducing the number of features with a PCA algorithm and explanation generation with SHAP and LIME.

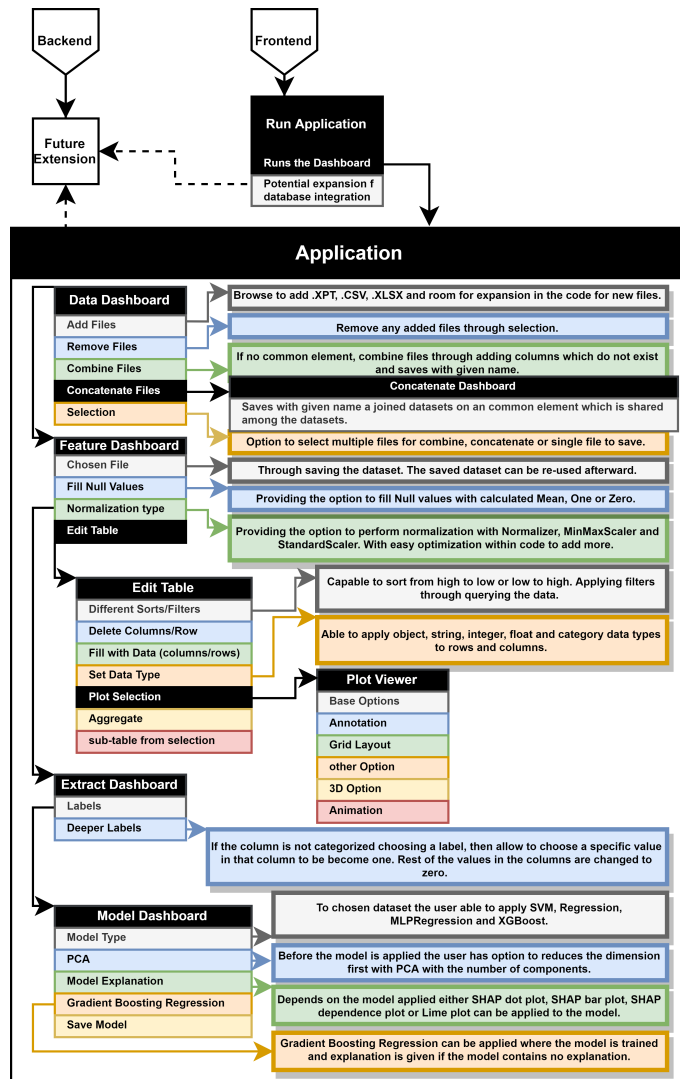


Figure B.6: ExMed Operation Overview. This figure shows the flow of ExMed operations, along with the key features available in the interface. Once the "Application" is running, the first window "Data Dashboard" is shown. Black arrows denote event-driven actions that take the user to the next window in chronological order. Colours highlight the Key features for each window, along with a short description provided for each feature. The indentation of boxes represents a dependency between windows. For instance, the 'Feature Dashboard' window can lead to the 'Edit Table' window, which subsequently can open the 'Plot Viewer' window. The dotted line represents a database extension that is to be added in the future.

Table B.8: Some additional results to table 4.11 in chapter 4

Samples / Class migration	Number of Pregnancies	Glucose mg/dl ≤ 140	Diastolic Blood Press., mm Hg ≤ 80	Skin Thickness mm	Insulin $\mu U/ml$ 16 – 166	BMI kg/m^2 ≤ 25	Diabetes Pedigree Function	Age year
7 : 0 \rightarrow 1	0	114 +46.3 (+41%)	80 -30.1 (-38%)	34 -19.9 (-59%)	285 -186 (-65%)	44.2 -22.4 (-51%)	0.167 +0.1 (+60%)	27 +37.9 (+140%)
8 : 0 \rightarrow 1	1	87	68	34	77	37.6	0.401	24 +4.4 (+18%)
9 : 0 \rightarrow 1	5 -4 (-80%)	96 +11 (+11%)	74 -6.1 (-8%)	18 +1 (+6%)	67 -54 (+81%)	33.6 -6.5 (-19%)	0.997 -0.8 (-80%)	43 -18.9 (-44%)
10 : 1 \rightarrow 0	3	107 -5 (-5%)	62	13	48	22.9	0.678	23
11 : 1 \rightarrow 0	9 -4.7 (-52%)	156 -45.4 (-29%)	86	28	155	34.3 +0.9 (+3%)	1.189 -0.5 (-42%)	42
12 : 1 \rightarrow 0	1 -0.1 (-10%)	128 -9.7 (-8%)	48 +37.9 (+79%)	45 -5.7 (-13%)	194 +25.9 (+13%)	40.5 +5 (+12%)	0.613 +0.2 (+33%)	24 +1.8 (+8%)

B. Supplementary Data

Frozen Input	Age	Sex	TSH	T3	TT4	T4U	Class
Original	74.00	0.00	1.40	0.00	123.00	1.57	1
None	60.09	0.10	-1.26	2.12	113.07	1.13	0
Age	74.02	1.02	-7.33	3.26	59.13	1.21	0
Age & Sex	74.01	0.00	0.57	0.29	142.14	0.82	0
Original	60.00	1.00	8.90	0.00	75.00	0.97	1
None	45.17	0.80	10.01	1.60	79.02	0.82	0
Age	60.00	0.82	0.77	1.85	82.53	0.93	0
Age & Sex	59.97	1.00	1.20	2.24	121.21	1.09	0
Original	46.00	0.00	0.05	2.20	97.00	0.77	2
None	57.64	0.79	1.51	1.57	85.94	0.74	0
Age	46.00	0.35	3.41	2.19	99.05	0.83	0
Age & Sex	46.00	0.00	1.60	1.83	96.84	0.74	0
Original	22.00	0.00	1.00	2.30	87.00	0.89	2
None	64.79	0.92	7.93	1.81	66.84	0.93	0
Age	22.03	0.78	0.76	2.47	77.87	1.08	0
Age & Sex	22.00	0.00	8.56	2.23	110.21	0.64	0

Table B.9: The table illustrates the weight vector impact on model G in the NMG. We run model G with three different values in the weight factor, non-frozen input ($\omega = 1$), freezing Age feature (cell correlated to age in ω equals 99), and Age and Sex are frozen. Each row shows either the input sample (top rows) or the output of model G.