Tampere University

Joona Jokivuori

CROSS-PLATFORM PORTING OF DEEP NEURAL NETWORKS

Bachelor of Science Thesis
Faculty of Engineering
and Natural Sciences
Examiner: Pasi Pertilä
April 2020

ABSTRACT

Joona Jokivuori: Cross-platform porting of deep neural networks
Bachelor of Science Thesis
Tampere University
Science and Engineering
April 2020

Deep neural networks have become the leading choice by many for machine learning applications. With the increasing amount of storage space and increasing computational capacity in modern computers, deep neural networks are the go-to choice. However, not all devices have robust hardware, and are thus not effective to train deep neural networks.

This thesis studies how deep neural networks can be trained and then ported to another programming language, so that devices with less computing power may utilize the predictions of a pre-trained network. A network trained in Python can be ported to C++ and MATLAB.

Python, C++, and MATLAB are popular programming languages, where Python is driven by Machine Learning research, MATLAB is for signal processing, visualization, and algorithm development, and C++ for efficient running of programs.

The results were calculated by comparing the accuracy and computing speed of the original and ported networks. The results suggest that all the ported networks have identical accuracy and identical forward passes. The speed of the ported C++ network confirms that porting the network is a feasible method to allow computationally weaker devices to run forward passes on the network.

Keywords: deep neural network, porting, wrapping, C++, MATLAB

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

PREFACE

I would like to thank my employer Huawei for suggesting this thesis topic. I would also like to thank my supervisor, Pasi Pertilä, for his support and guidance throughout the writing of the thesis. Lastly, I would like to thank my family and friends for providing the support and motivation for completing this thesis.

Tampere, 12 April 2020

Joona Jokivuori

CONTENTS

1. INTRODUCTION 1
2. BACKGROUND 3
    2.1 Deep Neural Networks 3
    2.2 Python Keras 3
    2.3 C++ 4
    2.4 MATLAB 4
3. THEORY 5
    3.1 Deep Neural Networks 5
    3.2 Keras 7
    3.3 Porting 8
4. METHODS 10
    4.1 Dataset and Network Architecture 10
    4.2 Python to C++ 11
    4.3 C++ to MATLAB 12
5. RESULTS 14
    5.1 Accuracy 14
    5.2 Speed 15
6. DISCUSSION 17
7. CONCLUSION 19
REFERENCES 20

LIST OF SYMBOLS AND ABBREVIATIONS

DNN	Deep Neural Network
GPU	Graphical Processing Unit
CPU	Central Processing Unit

1. INTRODUCTION

Machine learning is a part of modern everyday life; it is covertly making life easier. For example, machine learning decides the videos and songs people will see and hear. No longer do people have to people have to spend hours searching for the perfect song to add to their playlist, just to get bored and repeat the process. Countless manhours are saved everyday just by computers running machine learning algorithms calculating the optimal movie or music suggestions for users. This is a simple example taken from the immensely larger spectrum of applications using machine learning technology.

Deep neural networks are a subclass of machine learning, which are a favorable method today. Deep learning is popular due to its performance with larger datasets, and ability to utilize high end infrastructure to train [1]. Applications utilizing this feature are found everywhere, even the smaller and weaker devices, such as smartwatches, are starting to adopt this growing trend. This is where the problem of computational power arises. A deep neural network requires a considerable amount of computational effort to train. A simple smartwatch does not have the computational power to train complex networks in a realistic timeframe. Training the network requires complex calculations like backpropagation, which is required to generate the gradient and the calculations to generate the weights. However, performing a forward pass on the network is computationally efficient, especially using a middle level programming language such as C++.

This bachelor’s thesis aims to provide a straightforward solution to the aforementioned computational problem of training a network and then using the trained network on a target platform. The answer to this is simple; the network should be first trained in a higher-level language on a system with computational power, then, it can be ported to another language to be used in a target system. The focus of the thesis will be on porting a deep neural network trained using the Python library, Keras, and porting it, such that it can run evaluations on C++ and MATLAB, utilizing the improved speed and visualization methods of C++ and MATLAB.

The rest of this thesis is organized as follows. The second section will provide background information on the reasons behind selecting the above mentioned deep neural networks and languages. The third section, the theory, will explain relevant concepts that

will support the section about conducting the experiments. The following section will explain the process of porting the network from Python to C++ and MATLAB. The last three sections will consist of the results from the experiment, analysis about the results, and finally the conclusion.

2. BACKGROUND

This section will establish the reasoning behind choosing the specific network type and languages that are used for training and testing. The section is divided into subsections describing the reasons for choosing deep neural networks, Python Keras, C++ and MATLAB.

2.1 Deep Neural Networks

Deep neural networks (DNN) have much higher performance and accuracy when dealing with large datasets compared to traditional machine learning [2]. The origin of this thesis started as a task related to porting a Python Keras trained DNN which was fed a large audio dataset. Traditional machine learning algorithms tend to suffer with performance when dealing alongside bulkier data sizes. DNN training can also require powerful computers to be able to train the large datasets in a reasonable time. DNN also very effectively utilise the graphical processing unit (GPU) of a computer [3]. This is perfect for the application where the idea is to train on a high-end computer and port the pre-trained network to another possibly weaker system.

2.2 Python Keras

The reason Python was chosen as the training language was mainly due to the simplicity and understandability that it provides, and in addition, Python has many open-source DNN libraries, such as Keras, MXNet, PyTorch, and Tensorflow. Python, since it is a high-level language is easy to write and read. Python is also the second most used programming language of all time [4], which shows the popularity of the language. With its large userbase, there are enough frameworks and open-source tools to choose from.

There are many Python frameworks that support creating neural networks. The Keras framework was chosen, because it is consistent and since it provides abstract level operations that are simple for a human to understand. The idea of Keras is to be model-level framework, allowing fast and simple creations of deep learning models [5]. Keras allows for complex use cases since it uses lower level languages, such as TensorFlow, to build the networks [5]. This allows for a time-efficient and reliable platform for training the DNN.

Keras has two different architectures for creating networks, the sequential and functional API architecture. The sequential method allows for creating models layer-by-layer and is

easy to use. The functional method is a more flexible architecture, allowing layers to be connected in a pairwise fashion and allows for more complex architectures. For this thesis, the sequential architecture will be chosen for its simplistic modular approach to building a network.

2.3 C++

The C language is an efficient medium-level language to port to. However, there does not exist any tools that support porting Keras models to pure C. All the available tools were built for C++, which supports object-oriented programming. To develop such a tool from scratch was out of the scope of this thesis, so C++ was the next best alternative.

C++ is also significantly faster in terms of performance compared to Python [6]. This makes it an excellent candidate as a language to port the DNN model to.

2.4 MATLAB

Another language that the network can be ported to is MATLAB, which is used for simulation purposes. MATLAB is also a high-level programming language, which makes it a useful platform for testing the ported networks accuracy. The idea is that before testing the network in a separate system with C++, MATLAB can be used to verify that the DNN accuracy is within an acceptable difference compared to the original Keras trained network accuracy.

MATLAB also allows for compiling and linking C++ files inside itself [7]. This will be useful when porting the network to be able to be run from MATLAB.

3. THEORY

Before exploring the realm of porting neural networks, it is important to review the relevant theory. This section will provide the necessary theory to thoroughly understand the experimental section.

3.1 Deep Neural Networks

A deep neural network can be argued to be inspired from the human brain, more specifically the neocortex, the part the brain responsible for high-order functions [8]. The human brain consists of 100 billion neurons which are connected by synapses [8]. Deep neural networks similarly contain many collections of artificial neurons that are connected in a way that allows for the individual groups to understand patterns from the data.

To understand an artificial neural network, the neuron needs to be explained. A neuron is a computational unit which takes a set of inputs and provides an output. A neuron is also given a bias and the output is multiplied by an activation function to better fit the data. The equation *f(x)* for a single neuron is,

$$y = f(x) = a\left(\sum_{i=1}^{n} \boldsymbol{w}_i \boldsymbol{x}_i + b\right)$$

Where *a()* is the activation function, *n* is the number of inputs, *b* is the bias, and ***w*** is the weight vector and x_i is the *i*th element of the input vector ***x*** [9].

A layer consists of multiple neurons hooked together. The output of neurons are the inputs of other neurons, this essentially creates a network. [10]

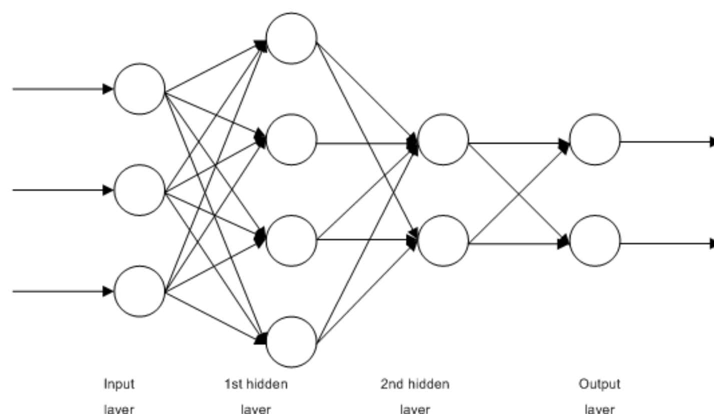

Figure 1. *An example network graph from John Salatas, licensed under CC BY-SA 3.0* [11]

There are different types of layers: the input layer, hidden layers, and the output layer. In a simple feedforward network, the input layer is the leftmost layer of the network, and this layer receives the data and forwards it to the hidden layers. The data goes through the hidden layers without being observed, and ends up at the rightmost layer, the output layer, see Figure 1. The output layer is the final layer and provides the result from the network. There are more complex networks, where there can be multiple input or output layers and more complication connections between layers. [12] An example of a more complex network is the recursive neural network, which uses a backpropagation variation to apply the same set of weights recursively to a structured input [13].

In the case of convolutional deep neural networks, each hidden layer learns higher levels of abstract features from the data. For example, when fed image data, the first layer learns first order features like colour and edges. The second layer learns features such as corners. The third layer recognizes texture. Each layer learns more advanced features gradually. Eventually the high-order features end up at output layers that can predict a classification or a regression. [8]

There are numerous amounts of different hidden layer types, such as convolution, normalization, dropout, recursive and so on. For the purpose of this thesis, there is no need to go in-depth into the mathematics of these different layer types, it is enough to know that the different types of layers exist.

The term deep means that the network contains multiple hidden groups of neurons. This is especially useful for enabling the network to automatically learn features from different abstraction levels. This means that the network does not depend on manually calculated features. The performance of a deep neural network excels especially with large unprocessed data. This allows for dumping a large amount of data into the network, and letting the network handle the feature extraction. However, the network does not always predict accurately, and the accuracy is strongly correlated to the quality of the data.

Training the network requires an error function and an optimization function. The error function, also known as a loss function, is usually the mean squared error when dealing with regression or an entropy function when dealing with classification.

The mean squared error for a prediction $\hat{\boldsymbol{y}}$ with the target $\boldsymbol{y}$ is [9],

$$MSE(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{\mathrm{N}} \sum_{i=1}^{\mathrm{N}} \|\hat{\boldsymbol{y}} - \boldsymbol{y}\|^2$$

For binary classification problems, a binary cross-entropy function can be used, which is [14],

$$CE(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\frac{1}{\mathrm{N}}\sum_{i=1}^{\mathrm{N}} \hat{\boldsymbol{y}}_i log\left(p(\hat{\boldsymbol{y}}_i)\right) + (1-\hat{\boldsymbol{y}}_i) log\left(1-p(\hat{\boldsymbol{y}}_i)\right) ,$$

Where N is the number of observations, $\hat{y}_i$ is the binary label, and $p(\hat{y}_i)$ is the predicted probability for the point being value of the correct class.

When training the network, the weights need to be updated so that the output becomes closer to the target output with each update. This is known as minimizing the error for the entire network. In the case of backpropagation, this is done by calculating the partial derivative of the error function $E(\hat{\boldsymbol{y}}, \boldsymbol{y})$ with respect to the weights, $\frac{\partial E}{\partial W_{ij}}$ [9]

A batch defines the number of samples, which are single rows of data, to go through before updating the parameters of the model. If the batch size is equal to the size of the training dataset, the learning algorithm is called a batch gradient descent. In the case if the batch size equalling one, the learning algorithm is the stochastic gradient descent. And finally, if the batch size is between one and the size of the training dataset, the algorithm is the mini-batch gradient descent. [15]

During the training process, there needs to be multiple epochs, which are cycles of the whole dataset through the network. A forward pass is the process of calculating the output from the input data. Due to the number of epochs required, training the network is more computationally intensive than calculating a forward pass.

3.2 Keras

Keras is a library built for the programming language, Python. Keras allows to create deep neural networks in a simple modular way. A network is expressed by a sequence of configurable autonomous modules that are connected to each other. Every layer is an individual module and combining these modules can create complex models. [16]

The simplest sequential network in Keras can be created in only four steps, preparing the input and output data, creating the first layer module, adding middle layer modules, and wrapping up with an output layer module [16].

The first step requires the user to configure the data so that it can be fitted to the input layer. This usually involves converting the data into arrays of numbers. In the case of audio signals, one way to prepare the data can be to convert the audio signal into a spectrogram. The spectrogram can be separated into magnitude and phase features, which can be fed as inputs [17]. Often, only the magnitude is used, and the phase is ignored.

Subsequently, the intermediate layers need to be configured. The number of intermediate layers can be chosen by the user, but for the network to be a proper deep network, there needs to be multiple intermediate layers. These middle layers can be chosen from the layer types to fit the purpose of the network.

Finally, the output layer is meant to manage the output data. This is the last layer of the network and will provide the output.

Keras has many pre-made layers that the user can choose from. Additionally, Keras has support for custom user made layers. The user can create them with two different approaches, lambda layers and custom layer classes.

For simple custom operations, using the lambda approach is recommended. The layer works similarly to ready-made layers, as it is a pre-built module that accepts a lambda function as a parameter. The lambda function is provided by the user. For example, to create a layer that does,

$$\mathrm{y} = x^2$$

The custom layer may look like [5],

```
model.add(Lambda(lambda x: x**2)).
```

Alternatively, the user can create a new class for their layer. This approach is recommended when the layer is more complex. The implementation of the custom class is out of the scope of this paper.

3.3 Porting

In relation to computers, porting, is the process of translating from one programming language or protocol to another[1]. Essentially this means translating the programming language.

Porting a neural network, is the act of translating the language, so that the network can be run using another language. There are two main ways of porting a language. The first being the more official and accepted way, which requires the user to manually re-write the framework in the target language. The process does not always need to be done line-by-line as there exists a plethora of open-source tools that help do this. The ported network should have the same output as the original network.

---

[1] https://www.techopedia.com/definition/8925/porting

Re-writing the network involves saving the architecture, bias, and weights for each layer. The network can be reconstructed in the target language by loading the architecture, bias, and weights. This requires the target language to support the original network's functions.

Another way to port a network, which theoretically is not true porting, is to wrap the precedent language by the newer language. A wrapper is an entity that encloses and hides the complexity of another entity using interfaces[2]. Essentially, in terms of porting, this means the newer language wraps the older language inside it, making it seem like the function was completed by the newer language.

[2] https://www.techopedia.com/definition/4389/wrapper-software-engineering

4. METHODS

This section will go through the process of porting the deep neural network from a Python Keras model to C++ and MATLAB. Firstly, the dataset and the network architecture are explained. The second subsection will explain the process for converting from Python to C++, and the third subsection will go through wrapping the C++ files in MATLAB.

4.1 Dataset and Network Architecture

The dataset used in this thesis was the Pima diabetes dataset by National Institute of Diabetes and Digestive and Kidney Disease [18]. The data consists of females with Pima Indian heritage. The data classifies whether the female has or does not have diabetes. There are eight inputs: amount of pregnancies, glucose concentration, blood pressure, skin thickness, insulin level, body mass index diabetes pedigree function, and age. The dataset also includes the outcome label, a 0 or 1 depending on whether the patient has diabetes.

The dataset was split into training and testing data, 80% and 20% respectively. This can help prevent the occurrence of over- or underfitting during the testing phase. The data was not pre-processed in anyway and was left as is in the dataset.

The test network used, is a very simple sequential network consisting of a few layers. The network consists of 3 Dense layers. With the activation layers: rectified linear unit and the sigmoid function. The summary of the model can be seen in the figure below.

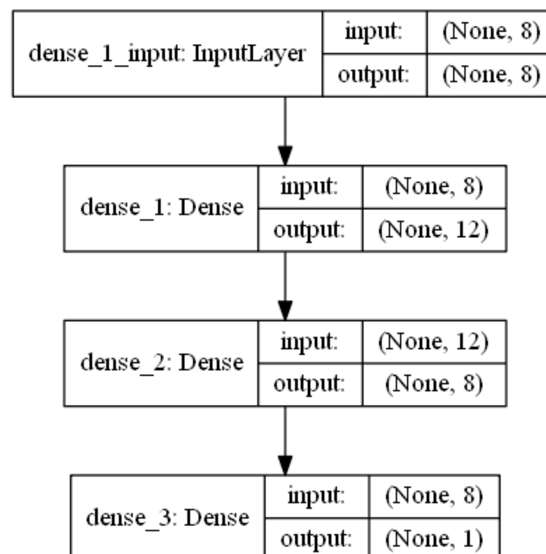

Figure 2. *The summary of the model using the plot_model() function in Keras. The left most box-is the name and type of layer. The right-most box describes the input and output dimensions. "None" means that the batch size is not specified.*

The network was chosen to be simple, due to there being no requirement for a complex network with hyperparameters. The purpose of the experiment is to simply compare the original network's and ported network's accuracy and speed. Adding more layers to complicate the network does not conform to the original purpose.

The model was compiled with the binary cross entropy loss function and the stochastic gradient descent algorithm. Binary cross entropy was chosen because it works well with binary classification problems. The stochastic gradient descent function is simple function, so it was chosen.

The training parameters for the model were chosen to be simple to save time, the number of epochs used for training was 15, and the batch size was set to 10.

The network was tested in Python using the built-in prediction function in Keras, which returns the outputs for each input list supplied. The outputs where then rounded using the basic mathematical rounding function,

$$\mathrm{round}(x) = \left\lceil \frac{\lfloor 2x \rfloor}{2} \right\rceil.$$

The rounded outputs are then compared to the given ground truth values provided by the dataset. The accuracy was then calculated by the number of correct predictions.

4.2 Python to C++

The network is first ported from Python to C++. This section will go through the step-by-step process utilizing a pre-existing tool that was modified.

The process is based on a modified version of the GitHub project, Kerasify [19]. The original project is designed to run trained Keras models from a C++ application [19]. Firstly, the Keras model is saved using Keras' own save function in Python. Then the saved model is converted into another format, using a conversion script provided by the original project, which is then opened using C++, and using the converted model, the network is reconstructed. The original project was modified to automate the porting process. The modified project follows the same concept and contains two main Python scripts.

The first Python script `kerasify.py` was provided by the original GitHub project[3], and goes through the saved Keras network model layer by layer, and re-writes them in a format that is supported by the premade C++ files. Each supported layer type is given its own numerical value. If the layer is not an activation layer, the corresponding weights and biases are saved into a struct with the layers numerical value. The C++ files unpack these structs and construct the network architecture to match the Keras network's architecture.

The second Python script `keraport.py` was made to automate the process of writing the main C++ file used to run predictions for the ported network. The script saves a template C++ file and fills in the previously converted model file name. The output for this script is a C++ file with MATLAB Mex function calls, that will be used in the later subsection. Alternatively, if the target language is C++ and not MATLAB, the output C++ file can be chosen to be pure C++ and contain no MATLAB Mex function calls. In both cases, the C++ files need to import premade C++ header files. Using this second script is optional for the porting process and exists only to reduce the requirement of manual programming. An example of a pure C++ file is shown in APPENDIX A.

The premade header C++ files are from the original GitHub project. These files reconstruct the neural network from the converted model file. The network can then be fed a tensor of input data and output a prediction which matches the prediction by the Keras model. The dimensions of the tensor need to match the Keras model dimensions. However, there is a built-in option in the modified project to override the dimensionality check.

4.3 C++ to MATLAB

The network can also be ported to be run from MATLAB. The network is not ported to pure MATLAB code, and instead utilizes wrapping C++ from MATLAB using Mex functions. This is a simpler solution than creating a new conversion tool to convert the Python model to a MATLAB model, since there are no pre-existing tools that do this.

MATLAB has an in-built system for running C++ files, the process requires the C++ to be modified to use MATLAB provided functions. Compiling the modified C++ will output a Mex file which MATLAB can run.

[3] https://github.com/moof2k/kerasify

Firstly, if the previously created C++ files do not include the MATLAB provided Mex functions, the files need to be updated to utilize the functions.

The C++ file includes a MexFunction class which inherits from the existing MATLAB provided class [7]. The class contains an array factory, a shared pointer to the MATLAB engine, and a string stream. The class needs to contain a public operator class which takes in the input and output variables. To be able to show output and errors in the MATLAB command line, functions utilizing the string stream are used.

Once the C++ file is updated to contain Mex functions, the file can be compiled into a Mex file that can be run from MATLAB. The compilation links the MATLAB provided libraries and the C++ file.

Finally, to be able to utilize the Mex file from MATLAB, the location of the file needs to be added to the MATLAB path. The predictions can be run by calling the Mex file name with the input tensor as the first parameter. Optionally, an extra Boolean parameter can be used if the dimension checking needs to be bypassed. [7]

5. RESULTS

Accuracy and speed are important factors to consider when comparing the original and ported networks. The accuracy of the ported networks should be equal to the original network's accuracy. The values are not expected to be identical due to differences in variable types and storing variables with different allocations of bytes.

Keras has inbuilt functions for calculating accuracy and making mass predictions. However, the ported C++ model only takes an input tensor and returns the output tensor. Hence, the diabetes dataset was an excellent choice due to the simplicity of the data. With a more complex dataset, the C++ code would have had to have been updated to allow for multi-dimensional tensors, as the ported C++ network only accepts one-dimensional tensors as an input. The small number of inputs in the dataset allowed for a simple algorithm to calculate accuracy and speed.

The results from the following chapters were calculated using Python 3.6.5 and Keras 2.2.4.

5.1 Accuracy

The accuracy of the network was calculated using the testing data. 154 testing samples were fed into the network, and the output was rounded get a classification of having diabetes or not.

To provide a fair comparison, the accuracy for The Python implementation was calculated using a similar method used by the C++ and MATLAB ports.

The first method to calculate accuracy was to feed the samples one-by-one and save the classified result to a list, which was then compared to the given truth values from the dataset. However, this method results in all 3 networks having near identical accuracies. The only difference was caused by the different languages' way of storing the accuracy in memory, resulting in the accuracy changing around the sixth decimal point.

The second method to compare the accuracy of each network's predictions, was to take a single sample and compare the unrounded forward pass result. This way, it is possible to compare the results for each network. Similarly, to the previous method, the problem with this method is that the results only differ due to the different storing methods for

each language. However, this is the closest method of comparing the theoretical accuracy of each ported network, as this method gives an idea of the similarities of the results from the networks.

The following table shows the results from testing. Columns indicate the results for each language. In the case of method one, the values are the accuracy as a percentage. In the second method, the values are the raw output for a single sample.

Table 1: *Accuracy of programming languages.*

Accuracy / Lang.	Python	C++	MATLAB
Method 1 (%)	67.5324675325	67.5324707031	67.5324675325
Method 2 (raw result)	0.4166727066	0.4166726768	0.4166726768

The table below, shows the difference of the networks' results compared to the result obtained in Python. The columns contain the difference between prediction results of the language and Python for each method. The values are calculated using $\Delta \text{Lang} = Target - Python$.

Table 2: *Change in accuracy of programming language compared to the result obtained in Python.*

ΔAccuracy / Lang.	C++	MATLAB
Δ Method 1	+3.1706E-06	0
Δ Method 2	-2.9800E-08	-2.9800E-08

5.2 Speed

The speed of the ported networks was calculated using the testing data. The testing data was fed into the network, and the processing time taken was measured when evaluating the data 1000 times. This way the average time taken is more accurate than calculating the time for a single case, as processing time depends on multiple factors that change constantly. The Python GPU was not used for comparison as the C++ and MATLAB implementations do not support it.

The following table shows the time results for each network.

Table 3: *Speed of programming languages.*

Time / Lang.	Python GPU	Python CPU	C++	MATLAB
Time * 1000 epochs (s)	1.711428	2.189304	1.265145	16.423488
Avg. Time / epoch (s)	0.001714	0.002189	0.001265	0.016428

The table below shows the difference of the networks' speed compared to the speed for Python CPU. The columns contain the difference between the language and Python for each method. The values are calculated using $\Delta \mathrm{Lang} = Target - Python\ CPU$. A negative value signifies that the language was faster than Python, and a positive value shows the language was slower. The values for the Python GPU are not compared to the results for C++ and MATLAB, to keep the comparison fair.

Table 4: *Change in speed of programming language compared to the result obtained in Python CPU.*

ΔTime / Lang.	Python GPU	C++	MATLAB
Δ Method 1 (s)	-0.4779	-0.9242	+14.2342
Δ Method 2 (s)	-4.779E-04	-9.242E-04	+0.0142

6. DISCUSSION

In line with the hypothesis, the results from Table 1 show that the ported networks have near-identical accuracy, and the speed improves with the lower-level C++ language. The accuracy using the first method for each network was identical to the fifth significant digit. This was expected, as the amount of correct classifications for each network, was the same, and the change was only due to each accuracy result being calculated inside the respective language. It was also expected that the C++ calculation would differ so much compared to the MATLAB and Python calculations, because the C++ calculation used a float variable, which uses less bytes in memory.

The second method to test accuracy was a better way, as it provided an idea on the actual differences with each network. Comparing the unrounded output for a single sample allowed to compare how similar each network truly was. With the previous method, rounding the output may have disguised a large difference. The second method verified that there did not exist a major difference between the output of the networks.

On the other hand, the MATLAB network's speed was a surprising result. The MATLAB result uses the wrapped C++ code, which would have thought to have been faster than Python, as it wraps the C++ code. The true resulting speed is most likely a result from the way MATLAB uses the Mex functionality to utilize C++ files. The best guess is that initializing the Mex file takes most of the CPU processing time.

These results should be taken into account when considering which language should be chosen to be ported to from Python. The results support the claim that C++ is an efficient language, especially when considering speed. C++ also had better speed compared to the Python GPU result. MATLAB is still a reasonable choice to port to when speed is not a decisive factor. MATLAB is an excellent choice when dealing with testing that requires fast prototyping, as MATLAB is simple to write, leading to the test programs being written faster than they would in the C++ language.

The methodological choices were constrained by the small dataset, and simple network architecture. Using a larger and more complex dataset could have provided more accurate results comparable to real life applications, as in reality, the state-of-the-art deep neural networks are not often shallow and narrow feedforward networks. However, due to the limited scope of this thesis, the simple architecture and small dataset were used, since the underlying mechanism to train and predict is the same.

It is beyond the scope of this study to provide results based on more complex network architectures, as the porting tool has limited support for Keras layers. The chosen GitHub project only supports eight different layer types and seven function *a()* types (called layers in Keras).

The reliability of using MATLAB as a destination language is impacted by the method used to port to MATLAB. The MATLAB porting method is built based on C++, and thus depends on MATLAB being able to compile and run C++ within MATLAB using the Mex library. It is possible that the outcome would vary if porting to MATLAB was done using a similar method to port to C++. The current outcomes may be impacted by MATLAB relying entirely on the C++ implementation.

The results from the experimentation are nonetheless valid, because they provide valuable insight on the validity of porting deep neural networks from Python Keras to C++ or MATLAB. The results show that using the method of porting networks is a justifiable option, as accuracy within an acceptable frame, and the network speed improves over Python when running in C++.

7. CONCLUSION

In this thesis, the idea of porting deep neural networks from Python to C++ and MATLAB was studied. This was done by training the neural network in Python first, and then saving it using the Keras save function. The saved network was converted to a custom model file, which can be used to load the network in C++. Then, a C++ file was created, which loaded the converted network file, and could be be fed an input tensor to receive the output tensor. After this, the C++ file was converted into a MATLAB Mex file, which allows the file to be run using MATLAB.

The results of the experimental phase led to the conclusion that porting neural networks is a valid method to utilize when there exists the requirement of running networks outside of Python. However, the CPU time taken in MATLAB was surprisingly high, making MATLAB a less desirable choice when thinking about the network's speed. Nevertheless, MATLAB is still a valid choice if speed is not an important factor to consider.

For future work, more layer types can be supported, this can be done by updating the current code or utilizing a GitHub project that is more up to date. Measuring the results for a more complex network and larger dataset can also be done to verify that the results are the same as for the simple network. Additionally, comprehensive debugging and monitoring the memory usage, would provide more insight to this study.

REFERENCES

[1] M. Najafabadi, F. Villanustre, T. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, “Deep learning applications and challenges in big data analytics,” *Journal of Big Data*, vol. 2, no. 1, pp. 1–21, 2015.

[2] J. S. Finizola, J. M. Targino, F. G. S. Teodoro, and C. A. de Moraes Lima, “A comparative study between deep learning and traditional machine learning techniques for facial biometric recognition,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018, vol. 11238, pp. 217–228.

[3] Ching T *et al.*, “Opportunities and obstacles for deep learning in biology and medicine,” *Journal of the Royal Society Interface*, vol. 15, no. 141, 2018, doi: https://doi.org/10.1098/rsif.2017.0387.

[4] Hasan M, “Top 20 Most Popular Programming Languages To Learn For Your Open-source Project,” *Web*, 2018. https://www.ubuntupit.com/top-20-most-popular-programming-languages-to-learn-for-your-open-source-project/ (accessed May 03, 2020).

[5] Chollet F and others, “Keras,” *Web*, 2015. https://keras.io (accessed Mar. 05, 2020).

[6] L. Prechelt, “Are Scripting Languages Any Good? A Validation of Perl, Python, Rexx, and Tcl against C, C++, and Java,” in *Advances In Computers*, vol. 57, Elsevier Science & Technology, 2003, pp. 205–270.

[7] MathWorks, “C/C++, Fortran, Java, and Python API Reference,” *Guide*. 2020, [Online]. Available: https://www.mathworks.com/help/pdf_doc/matlab/matlab_apiref.pdf.

[8] W. Di, *Deep learning essentials : your hands-on guide to the fundamentals of deep learning and neural network modeling* . Packt Publishing, 2018.

[9] S. A. Zhang A, Lipton ZC, Li M, *Dive into Deep Learning*, 0.7.1. 2020.

[10] Boehmke B, “UC Business Analytics R Programming Guide,” *University of Cincinnati*. http://uc-r.github.io/feedforward_DNN (accessed Apr. 10, 2020).

[11] John Salatas, “Implementation of Elman Recurrent Neural Network in WEKA,” *Web*, 2011. https://jsalatas.ictpro.gr/implementation-of-elman-recurrent-neural-network-in-weka/ (accessed Apr. 22, 2020).

[12] S. C. Ng A, Ngiam J, Foo CY, Mai Y, “UFLDL tutorial,” *Web*, 2010. http://ufldl.stanford.edu/tutorial/StarterCode/ (accessed Mar. 11, 2020).

[13] J. Patterson, *Deep learning : a practitioner's approach*, First edition. O'Reilly Media, 2017.

[14] Godoy D., "Understanding binary cross-entrpy," *Web*, 2018. https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a (accessed Apr. 12, 2020).

[15] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016, [Online]. Available: http://arxiv.org/abs/1609.04747.

[16] G. Ciaburro, *Keras 2. x projects : 9 projects demonstrating faster experimentation of neural network and deep learning applications using Keras* . Packt Publishing, 2018.

[17] P. I. Chilton E, "Combination of magnitude and phase statistical features for audio classification," *Acoustics Research Letters Online*, vol. 5, 2004, [Online]. Available: https://asa.scitation.org/doi/pdf/10.1121/1.1755731.

[18] J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler, and R. S. Johannes, "Using the ADAP learning algorithm to forecast the onset of diabetes mellitus." IEEE Computer Society Press, pp. 261--265, 1988.

[19] Rose R, "kerasify." 2017, [Online]. Available: https://github.com/moof2k/kerasify.

APPENDIX A: USING TEXT STYLES IN MS WORD

```cpp
// Auto-generated by keraport.py, do not modify
#include "keras_model.hpp"
#include <iostream>
#include <fstream>
#include <vector>
#include <math.h>
#include <iomanip>
#include <cmath>

int main() {
    // Load the model using converted model file
    KerasModel model;
    model.LoadModel("B:/python2cpp/tmp/converted.temp");
    // Load the input data and initialize to in Tensor
    std::vector<std::vector<float>> vector = {{11.00000, ..., 23.00000}};
    std::vector<float> predictions;
    Tensor in(8);
    in.data_ = vector[0];
    // Initialize out Tensor and apply to the model to get output.
    Tensor out;
    model.Apply(&in, &out);
    // Print the output data with high precision
    std::cout << std::setprecision(10) << out.data_[0] << std::endl;

    // Calculate accuracy of the predictions using real values vector
    int c = 0;
    std::vector<float> realValues = {1.00000, ..., 0.00000};
    for (int i2=0; i2 < predictions.size(); i2++) {
        if (round(predictions[i2]) == realValues[i2]) {
            c++;
        }
    }
    // Print the accuracy of the network
    std::cout << "c: " << c << "size: " << predictions.size() << std::endl;
    float acc = float(c)/float(predictions.size()) * float(100);
    std::cout << std::setprecision(10) << "accuracy: " << acc << std::endl;
    return 0;
};
```

Figure 3. *C++ example source file for a ported network.*