OCMP5329 Project Final Report

Ryan Tauhid (530638326) and Shaun Jayasinghe (530394271)

April 12, 2024

Python File
To access the ipynb file with the code, click here.

Contents

1 Introduction

1.1 What's the aim of the study

The aim of this study is to build a modified neural network architecture embedded with complex numbers (complex valued neural network) in the weights, inputs and activations. Using this complex valued neural network, we are aiming to test it on multimodal data sets and compare the results to those of traditional neural networks. To develop a complex valued neural network that can comprehend complex numbers, we are altering pytorch libraries and modules. At the conclusion of our project, we aim to have meaningful results which we can use to draw conclusions on whether neural networks embedded with complex numbers have any advantages over traditional neural networks and if so, what are these advantages. By doing so, we hope to be able to contribute to the wider understanding of deep learning and make meaningful improvements to neural networks.

1.2 Why is the study important

Especially when compared to traditional neural networks, complex valued neural networks embedded with complex numbers are relatively understudied. By studying complex valued neural networks, such as the one in this project, we can expand the use cases for neural networks and artificial intelligence. Certain fields/industries require the processing and comprehension of complex numbers, such as electrical engineering, wireless communications and physics. For these fields/industries, a modified neural network with improved performance which can assist in solving complex valued problems would be beneficial. Any improvements in efficiency or effectiveness made to complex valued neural networks could lead to breakthroughs in the fields/industries listed earlier. Overall, it is important to contribute to the published work on complex valued neural networks as even the slightest improvement in these neural networks could lead to innovation and breakthroughs in fields/industries which use complex numbers.

1.3 The general introduction of your used method in the assignment and your motivation for such a solution

To implement and define our complex valued neural network we first created a class which defined our complex activation functions and their derivatives. The activation functions defined for potential use in our model are the complex ReLU (Equation 1) and the modified ReLU (Equation 2). A complex softmax function is also defined for use in the model.

$$cReLU(z) = ReLU(Re(z)) + j * ReLU(Im(z)) \tag{1}$$

$$ModReLU(z) = ReLU(|z| + b)e^{j*\angle z} \tag{2}$$

The next class we defied is a fully connected layer for the neural network, which can handle complex numbers. This was manually created since the default layers do not have the capability to process complex numbers. We have decided to define several loss classes which can handle complex numbers. In

our code we have decided to define a base loss function, two variations of the mean squared error loss function and an absolute distance loss function. The choice to define multiple loss functions was made to determine which one produced superior results with complex valued data. Our final structural piece of code defined a complex neural network with multiple linear layers then creates, trains, tests and evaluates the neural network model. The complex neural network model was built this way through a series of tests and abolition studies which helped up narrow down on the optimal activation functions, layer sizes and hyperparameters. As the main goal of the study was to find the advantages complex neural networks have over traditional networks, the focus was not on completely optimizing the complex neural network to achieve minimal loss, instead it was to develop two models for each category of tests, regression and classification, for which a comparison could be made between the complex model and its counterpart.

2 Related Work

2.1 Existing related methods in the literature

From our extensive review of published works on the topic of complex neural networks, the methods in "On Complex Valued Convolutional Neural Networks" (Guberman 2016) were most similar to those used in our model development. In this paper, the author describes using an altered ReLU function as their activation function, however, the ReLU function used was still very faithful to the traditional ReLU function and the complex ReLU activation function (Equation 1) used in our model. The finalised model in the paper has six layers in total and three unique layers. The neural network layers used (in order) start with a convolution layer, activation function (altered ReLU) and pooling layer then repeat once. In this paper, it was determined and proven by the results that complex neural networks did show improvements over traditional neural networks in many key aspects. While the performance of the complex neural network was comparable to that of the traditional neural network, the complex network was less prone to overfitting. One of the major downsides to the complex network was that it faced significant optimisation issues during the training process. It is important to also consider the data used, in this paper they used simulated fluorescence microscopy images, which based on research we have done, we would expect complex neural networks to perform better detection, which is proven correct by the complex model in this paper detecting meaningful phase structure in the data. Existing implementations in the paper were analysed and built within our pytorch modules, as the paper did not provide code, it was up to us to interpret the mathematical representation of functions and implement accordingly. A shortcoming of the paper was the lack of exploring regression when it comes to complex neural networks, as such we implemented the functions mentioned in the paper and applied them to regression data, seeking to see the performance of an imaginary network capable of taking a single node's input as an entire complex number against a real network with two inputs representing the real and imaginary parts of a complex number.

3 Technique

3.1 The principle and justification of your method used in this assignment

3.1.1 Regression

The regression neural network model in this project has been altered such that it works with complex values. The data is input through the initial layer of the model. Then complex regression model has hidden layers which are made up using the ComplexLinear class and the Activation class. The ComplexLinear class provides a linear transformation while the Activation class provides the non-linear component. The final layer in this model is the output layer, which performs one final linear transformation on the hidden layers' output. Since there was little literature regarding complex regression, this section required careful consideration of the appropriate loss function. The traditional mean squared error (MSE) loss function is defined in Equation 33:

$$MSE = \frac{1}{n}\sum(x_{pred} - x_{true})^2 \tag{3}$$

Equation 3 is not directly applicable to complex-valued regression, as it does not account for the complex nature of the inputs and outputs.

The appropriate loss function for complex-valued regression should consider both the real and imaginary components of the complex numbers. A suitable loss function is the complex-valued MSE (CMSE), which is defined in Equation 4.

$$MSE = \frac{1}{n}\sum(x_{pred_{real}} - x_{true_{real}})^2 + (x_{pred_{imag}} - x_{true_{imag}})^2 \tag{4}$$

where $x_{pred_{real}}$ and $x_{pred_{imag}}$ are the real and imaginary parts of the predicted output, and $x_{true_{real}}$ and $x_{true_{imag}}$ are the real and imaginary parts of the true output.

This CMSE loss function ensures that the model's predictions match both the real and imaginary components of the complex-valued targets, leading to better convergence and more accurate complex-valued regression. The optimisation algorithm we decided to use for this neural network to update the model parameters during training was stochastic gradient descent. We will vary the learning rate in testing the model to determine the best value for the parameter.

3.1.2 Classification

The classification neural network model presented in this project is modified to handle complex valued inputs and weights while following a feedforward architecture. As images are more suited towards a convolutional layer, and non-image classifications do not require as such, we have created the ComplexLinear class and ComplexConv2d class. We preformed two types of classification, one on the MNIST hand written digit classification, the other on the coding assignment data.

The neural network trained on the coding assignment data has an initial layer which consists of the complex linear layer. The complex linear layer is comprised

of 128 input feature and 10 output features and uses the ComplexLinear class we have coded. This layer is the entry point for our complex valued data. The activation function is now applied in the neural network to add a non-linear component. In the Activation class we have defined two different variations of the ReLU function to accommodate complex values, they are the complex ReLU function (Equation 1) and the modified ReLU function (Equation 2). Either one of these activation functions can be used, depending on how the model is coded. As the outputs need to be real-valued, the magnitude of the output layer is taken and fed through softmax.

The data passes through hidden layers in our neural network. These hidden layers use the same ComplexLinear class as the initial layer which we have coded. These layer structures are varied through our testing, however a baseline is used with 64 hidden neurons in a single layer.

Our model initialises the weights of our complex linear layers (initial, hidden and output layers) at random by selecting values from a normally distributed set. One of the modifications we had to make to the weight initialisation process was generating the real and imaginary components of the weight separately then combining them to form the complex value.

We have defined several loss functions which can be swapped by varying the model initialisation code, however as the outputs were real valued in the classification case, the traditional CrossEntropySoftmax can be used appropriately. Similar to the regression model, the optimisation algorithm we decided to use for this neural network to update the model parameters during training was stochastic gradient descent. We will vary the learning rate in testing of the model to determine the best value for the parameter.

3.1.3 Classification With Convolution

Many parts of the traditional classification were re-used in this section, the main differences being the additional ComplexConv2d and ComplexMaxPool2d layers used. The implementation of these followed a similar thought structure as the prior, namely making these functions be compatible with imaginary inputs. ComplexMaxPool2d and ComplexConv2d were built upon the findings and discussions in the paper, (Guberman 2016). These helped us understand the underlying theory, yet the implementations had to be iterated upon and built according to our own learnings. The complex max pool was done by max magnitude of the $n \times m$ grid whilst still returning the complex-valued input and not the magnitude itself. As for the complex convolution, the paper showed that instead of storing the values of the weights and biases as the complex data type, an equivalent two sets of tensors per layer could be used according to a specific formula, a proof for which we later devised in the paper

3.2 Any advantage or novelty of the proposed method

The model we decided to create has several advantages and novelties compared to other models viewed online. For one, our model uses a relatively low number of hidden layers. While this was predominantly done due to the time restrictions/deadlines of this project, there are many purposeful side effects of this such as shorter training time and the code being more easily understandable. However, a low number of hidden layers in our model means that our model is

inherently simpler than those online, meaning our model should be less prone to overfitting. Another component of our code which was unique compared to what is seen in published papers is that we used synthetic data to run preliminary tests. While synthetic data cannot be used to evaluate real world performance reliably, it does allow us to quickly verify our model is functional and performing as expected. This was particularly useful since this saved us time on a project which had a deadline.

4 Experiment and Results

4.1 Results

4.1.1 Regression

Initial experimentations were done with complex-valued regression. The data used for this task were complex combinations of mathematical functions, where the inputs and outputs were complex numbers. The real-valued neural network used for this task had two input nodes for the real and imaginary parts of the complex number, and similarly, two output nodes for the real and imaginary parts of the predicted output.

The complex-valued neural network, on the other hand, directly worked with a single input node per complex number, and a single output node per complex number, without the need to separate the real and imaginary parts.

The use of the CMSE loss function, along with the appropriate complex-valued neural network architecture, allowed the model to converge and accurately learn the complex-valued regression task.

Let X be a complex tensor of shape ($n_{samples}$, n_{inputs}), where $n_{samples}$ is the number of samples and n_{inputs} is the number of complex inputs. Each element of X is a complex number denoted as x_{ij}, where i represents the sample index and j represents the input index. The transformation applied to each sample is Equation 5.

$$Y = \sum_{i=1}^{n} (x_i)^3 \tag{5}$$

The models were created with two hidden layers of 128 and 64 layers each, with ReLU and Complex ReLU being used accordingly between each layer except the last. As losses were exceedingly large towards the beginning of training, the following graph shows the loss over the epochs after the 200th epoch, over a range of 10000 epochs.

The complex model (Figure 1) shows the loss function being minimized, confirming that the gradient descent algorithm is correctly working over the complex weights and biases. The increased test loss suggests overfitting with an inability to reach losses below 1 due to the limited architecture.

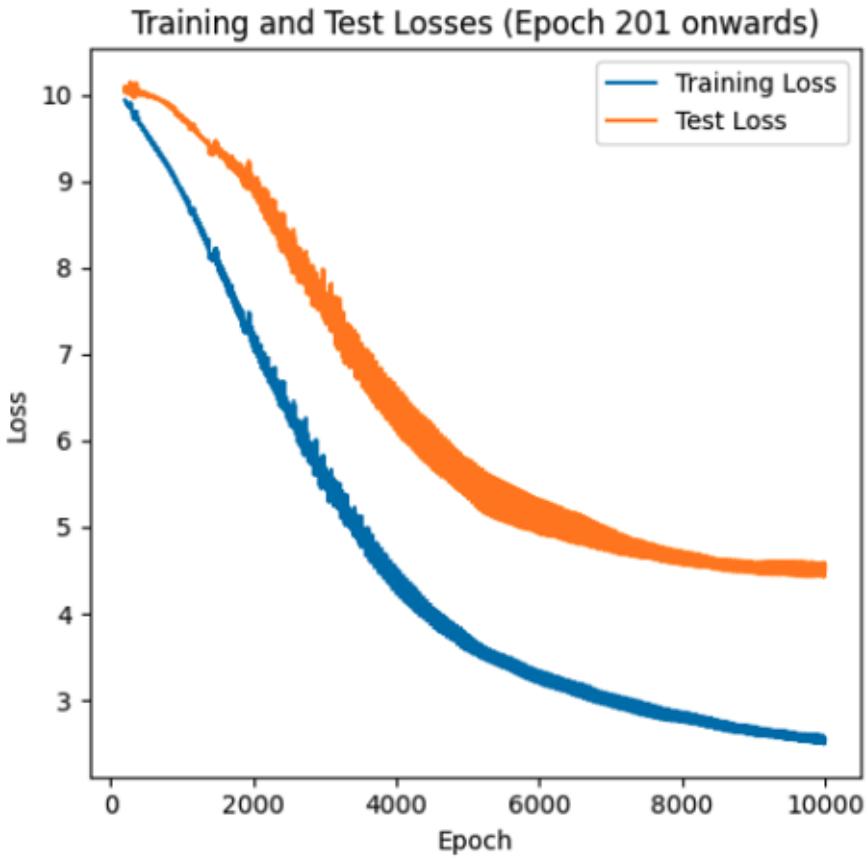

Figure 1: Initial Complex Model Training and Testing

When compared against the real model (Figure 2), a large difference in the training and test loss can be observed. Whereas the training loss of the real model marginally outperformed that of the complex model, a significantly increased test loss clearly indicated issues of over fitting. Solutions to this could have involved using batch regularization or dropout, however, this experiment shows possible advantages of a direct one to one comparison between complex and real neural networks. A disadvantage of the complex neural network is the training time.

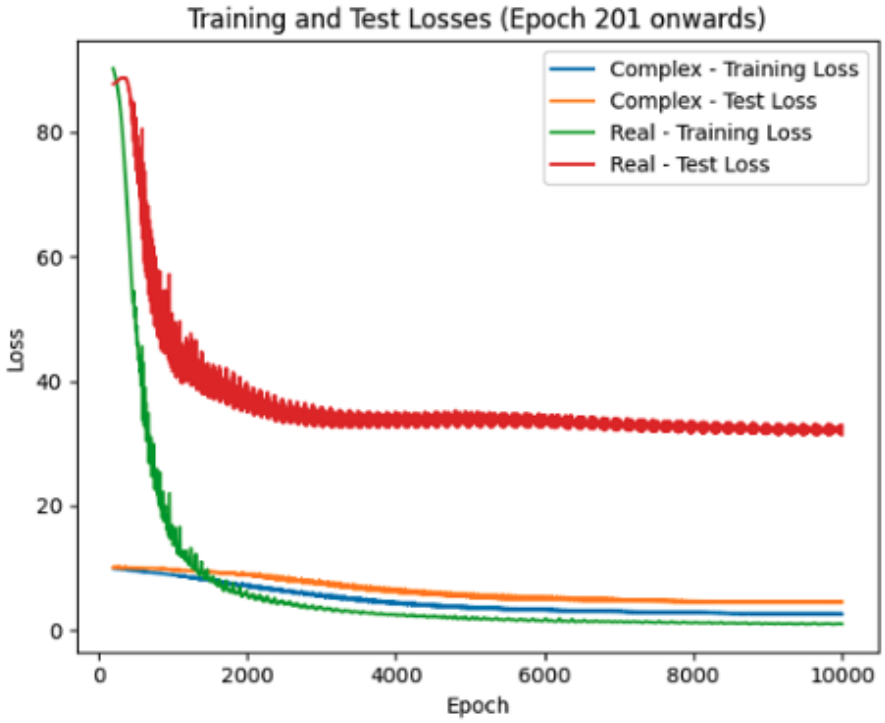

Figure 2: Initial Real vs Complex Model Training and Testing

Despite being able to outperform the real model, training times were vastly outmatched, with the real model training in 30% the time of the complex model as seen in Figure 3. A difference in training time is to be expected, with twice the parameters, each gradient in the model will require double the amount of addition computations to perform during the gradient update step, and as well as the additional compute required for the backward pass.

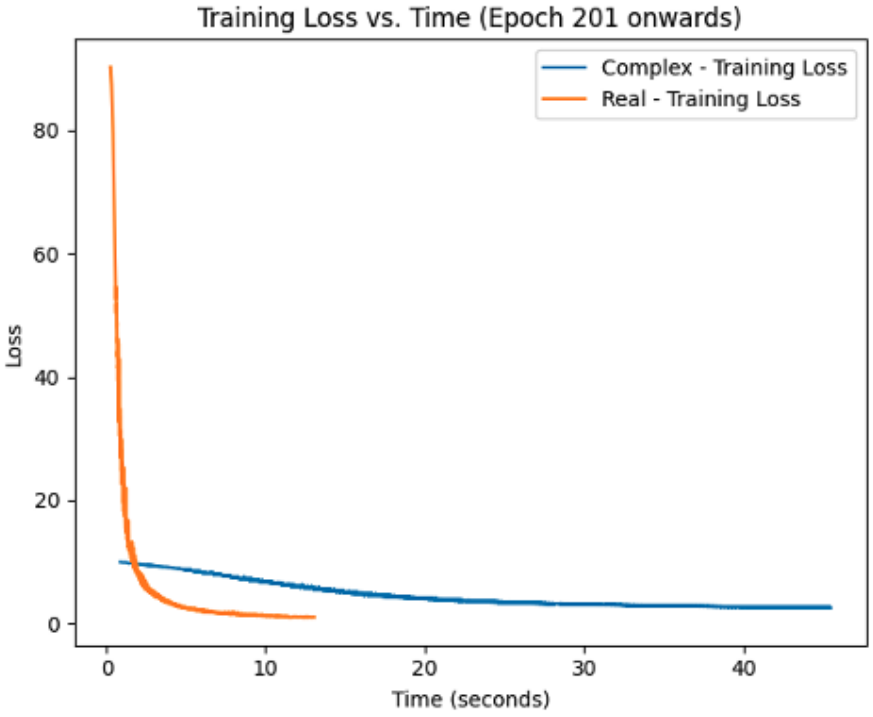

Figure 3: Real vs Complex Training Times

4.1.2 Convolutional Neural Network

Throughout the implementation of convolutional layers, a different approach towards implementation was brought to our attention. Previously, PyTorch layers were modified to handle complex data types by redefining the bias and weight attributes to accept complex data types. Alternatively, as the paper mentions, by extracting the real and imaginary coefficients of the inputs, two normal Layer instances can be used to process the real and imaginary parts separately. The outputs of these two layers can then be combined to obtain the final complex output. This approach is mathematically equivalent to using a single layer with complex weights and biases.

To show why these two approaches are equivalent, let's consider a complex-valued linear transformation:

The complex-valued linear transformation can be expressed as:

$y = Wx$

Expanding this equation using the complex components:

$(c + id) = (A + iB)(a + ib)$

Multiplying the terms on the right-hand side:

$(c + id) = (Aa - Bb) + i(Ab + Ba)$

Equating the real and imaginary parts:

$c = Aa - Bb$

$d = Ab + Ba$

The real-valued linear transformations can be expressed as:

$$y_1 = W_1 a - W_2 b$$
$$y_2 = W_1 b + W_2 a$$

If we set $W_1 = A$ and $W_2 = B$, then:

$$y_1 = Aa - Bb$$
$$y_2 = Ab + Ba$$

We can see that y_1 corresponds to the real part c and y_2 corresponds to the imaginary part d of the complex output y.

As such, we use this alternative implementation of complex layer outputs for the convolutional layer.

Running for 10 epochs each with a learning rate of 0.01 and a batch size of 64, Table 1 and Figure 4 shows the results.

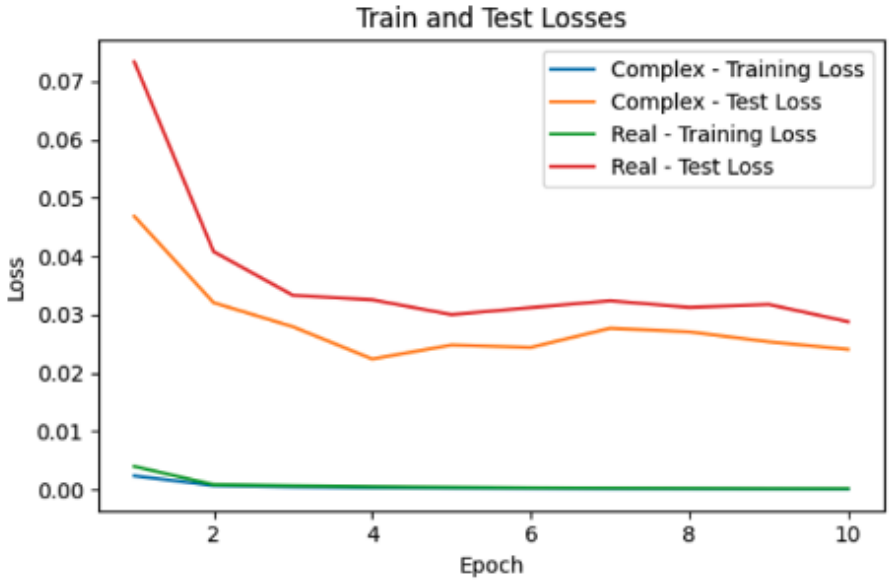

Figure 4: Convolutional Neural Network Results (Graphical)

	Train Loss	Test Loss	Accuracy
Real	0.008955	0.0288	0.992
Complex	0.009698	0.0240	0.993

Table 1: Convolutional Neural Network Results (Tabular)

As found in the paper (Guberman 2016), we validated the claims of the complex model returning test loss lower than the real model, while training loss remained relatively equal. This suggests that the complex model was able to better generalize to the test data, without overfitting the training data. The real model, on the other hand, appears to have been more prone to overfitting, resulting in higher test loss despite similar training loss. The paper failed to analyze the training time for the complex model, which can be seen to be its major downfall (as seen in Figure 5).

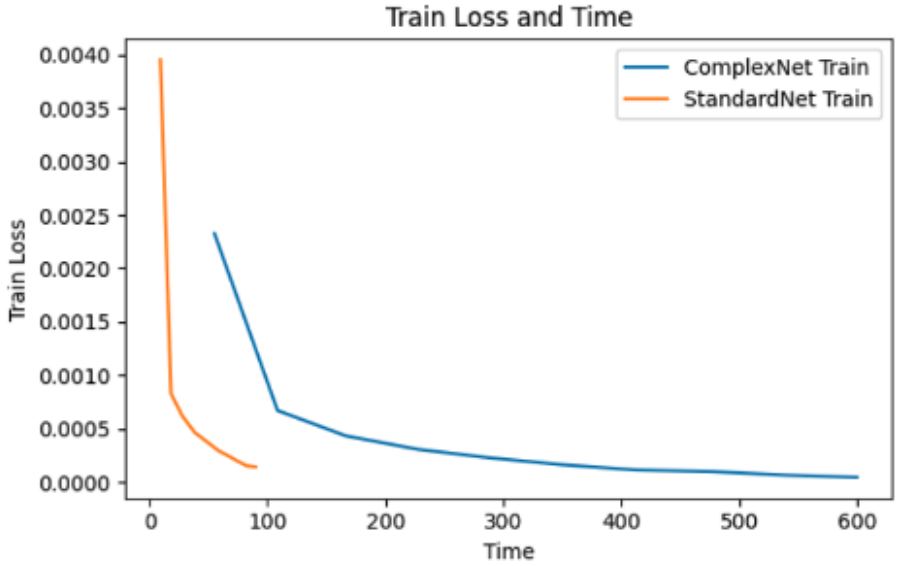

Figure 5: Standard vs Complex Training Time

4.2 Extensive Analysis

In the process of optimizing the complex model, for the following section, we will be comparing it against a baseline real model consisting of a single hidden layer of 64 nodes, on the data used for the coding assignment.

For this type of classification, the complex model showed fast convergence and ultimately a lower training loss with 300 epochs (as seen in Figure 6). This shows the complex models ability to find better local minimum's without overfitting as demonstrated in the comparison of test accuracies per epoch (as seen in Figure 7).

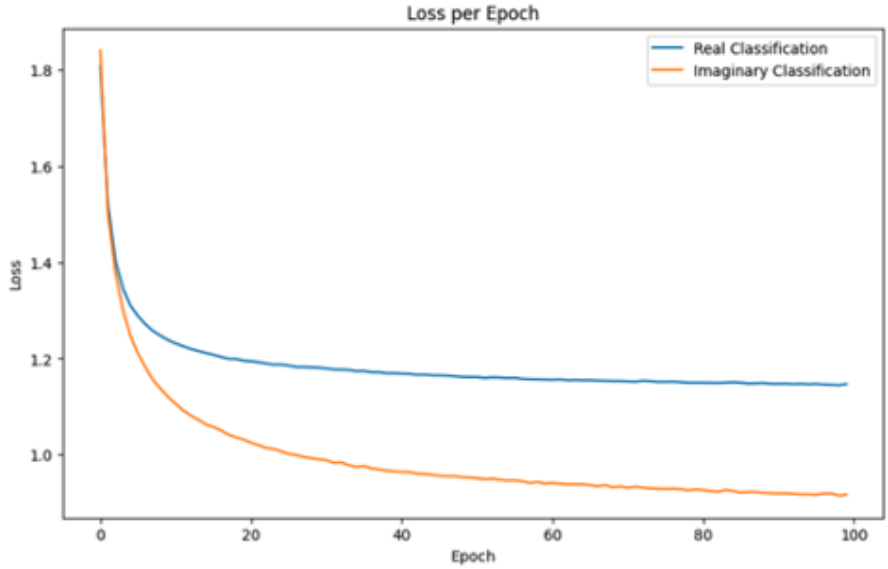

Figure 6: Complex Model Training Loss per Epoch

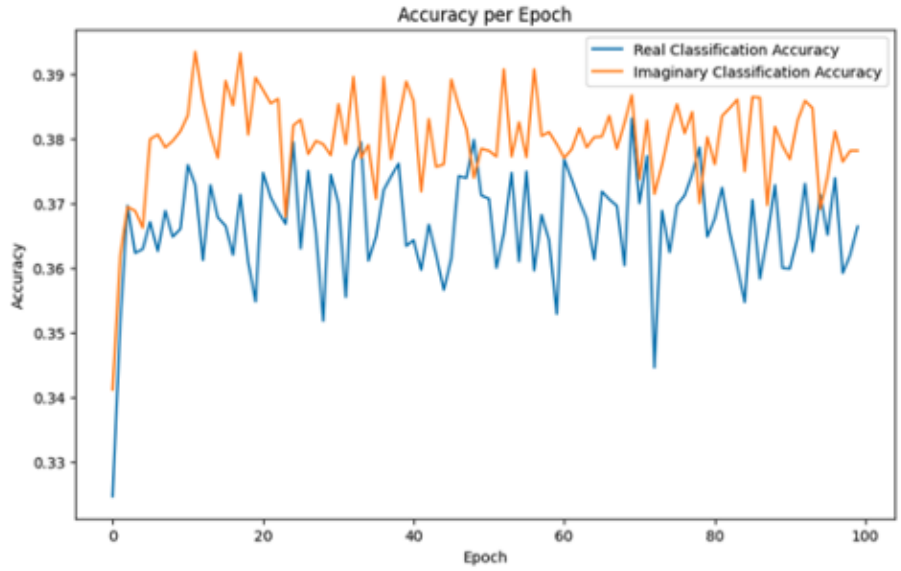

Figure 7: Complex Model Accuracy per Epoch

With the test accuracy of the complex model having a mean of 0.3799 and the real having 0.36662. These tests were then repeated multiple times to find the statistical significance of these results.

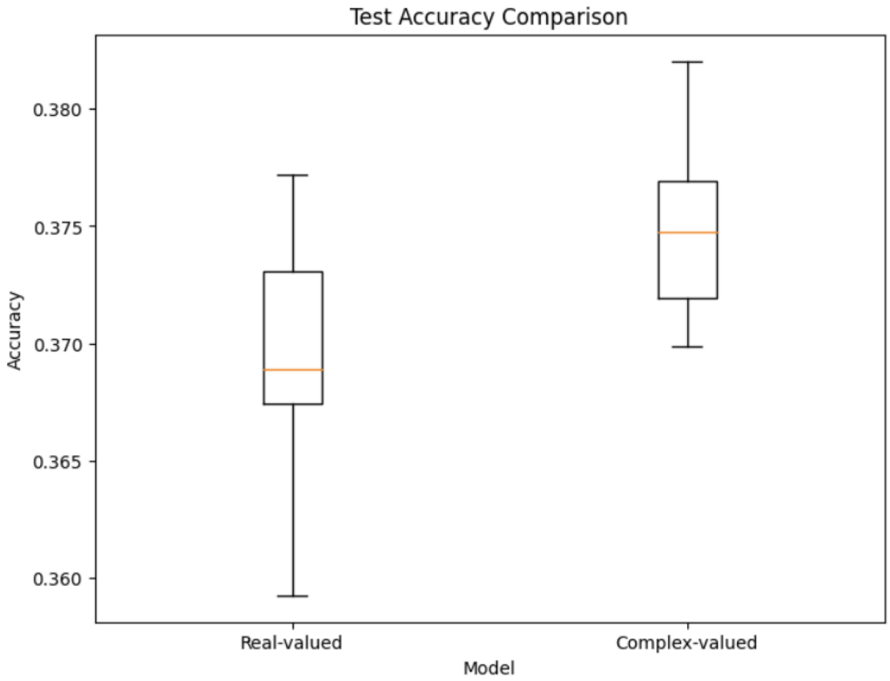

Figure 8: Real vs Complex Test Accuracy

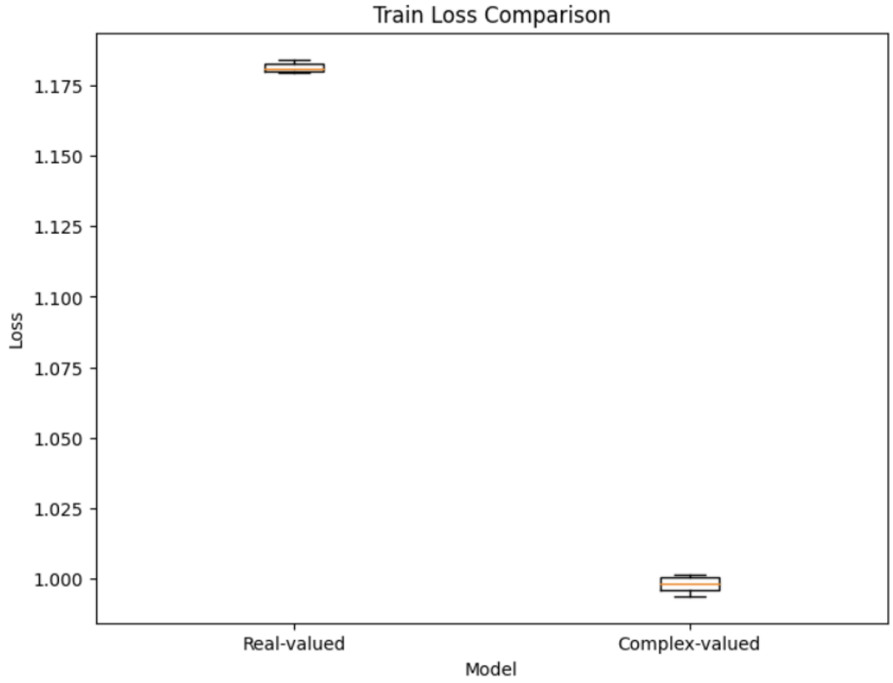

Figure 9: Real vs Complex Training Loss

By rerunning the training 10 times the box plots were returned (Figure 8 and Figure 9), it was evident that the complex model was able to consistently find better optimal which would then in turn produce better test results. The training losses had no variance, providing consistent results repeatedly. This would then translate into better test results as shown with the test accuracy, although with a slightly higher variance.

Null Hypothesis	P value	t-statistic
H0: $Y_{\text{complex pred}}$ (mean) <= $Y_{\text{real pred}}$(mean)	0.0056	2.8264

Table 2: Complex Valued Test Accuracy Significance

The complex-valued test accuracy is statistically significant higher than the real-valued test accuracy as seen in Table 2.

Following on, these studies were performed on the effects of initialization methods. The effects of this were made abundantly clear during our NumPy implementations of neural networks, during which initialization methods were the difference between gradient explosion/ vanishing, and a model which converges. Random initialization is required to break symmetry, a problem in which all neurons in a layer evolve symmetrically during training, leading to the same output and preventing the network from learning complex patterns.

By default, a random initialization is used, where the coefficients of the com-

plex and real parts of weights/biases are between 0 and 1. This approach showed extreme initial losses, however convergence was faster than Xavier Initialisation, offering some benefit. Comparision results are shown in Figure 10 and Figure 11.

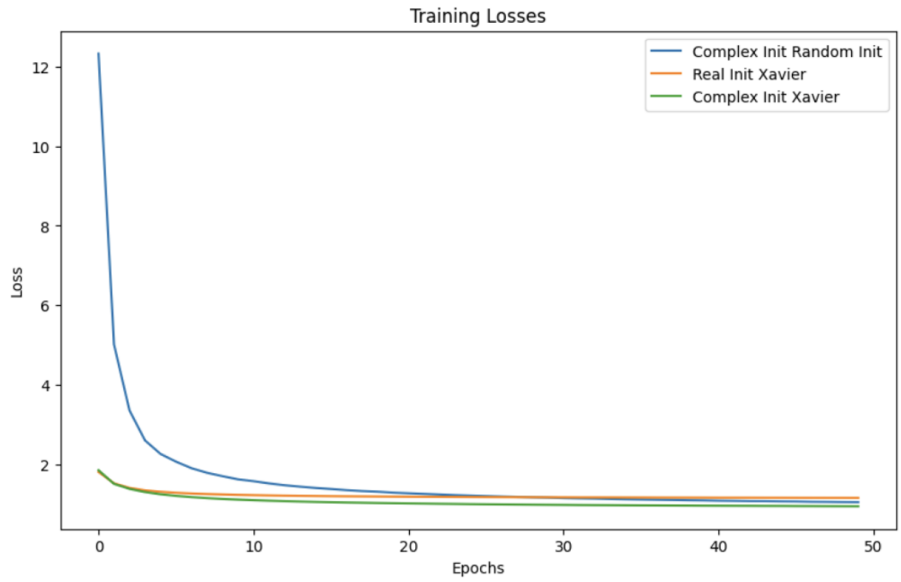

Figure 10: Training Loss Initialisation Comparison

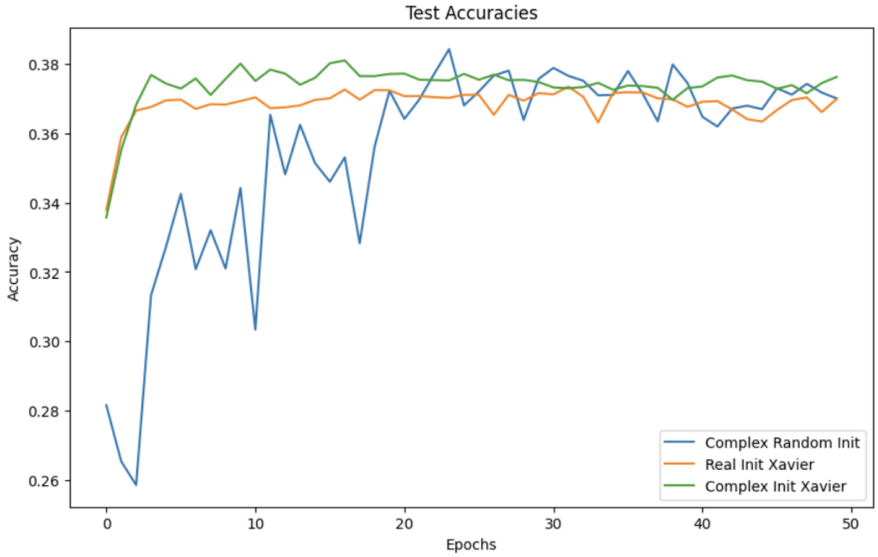

Figure 11: Test Accuracies Initialisation Comparison

4.2.1 Hidden Layers

The effect of hidden layers of test accuracy showed no improvement, although training loss was substantially affected. The models showed correlation between network complexity and training loss decreases, dropping to 0.4 with a model with three hidden layers. This in turn shows that even with over fitting on the training set, the model was able to generalize just as well as the less complicated models. Results are shown in Figure 12 and Figure 13.

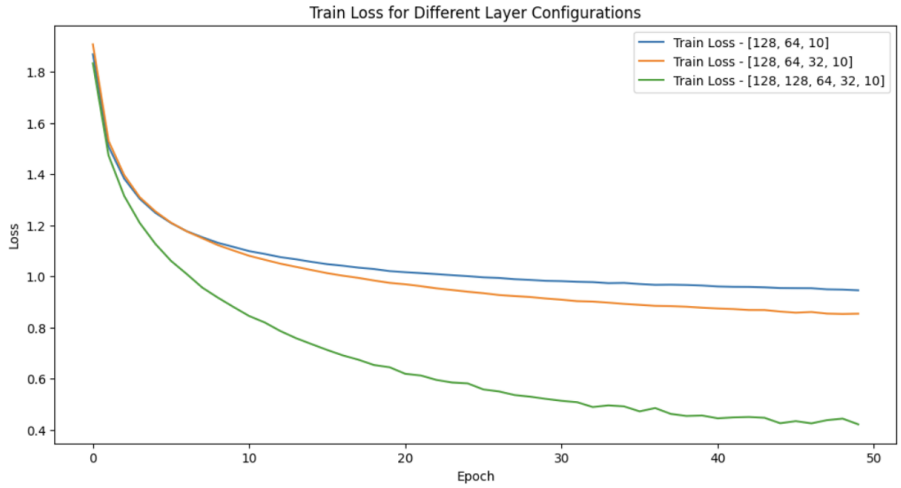

Figure 12: Training Loss for Various Layer Configurations

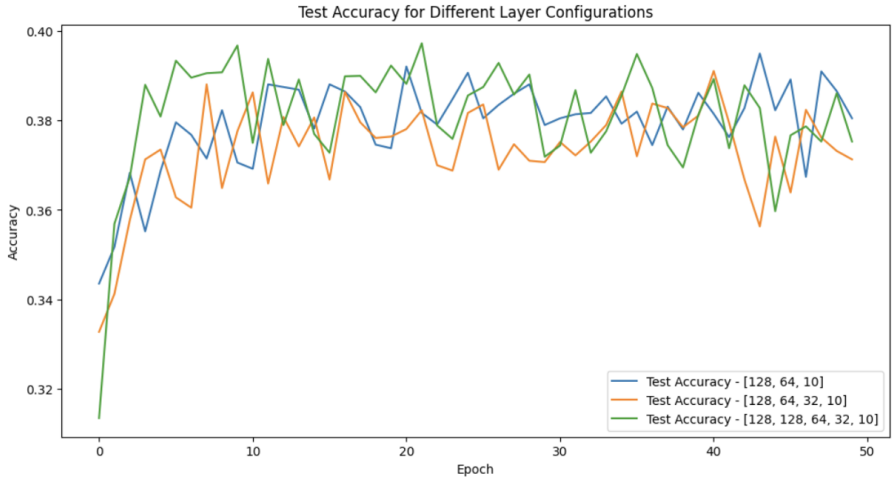

Figure 13: Test Accuracies for Various Configurations

4.2.2 Learning Rate and Loss Function

The combination of varying momentum and learning rate are summarised below in Figure 14, the best performing model had momentum set to 0.5 and a learning rate of 0.1, however the range of accuracy in this test was too low state that these hyperparameters had any statistically significance.

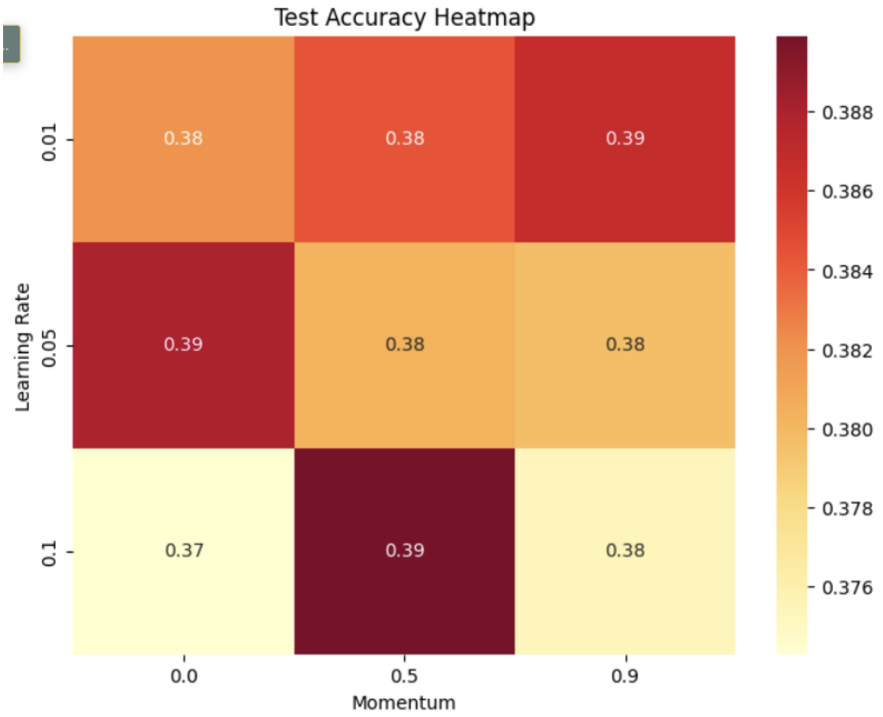

Figure 14: Loss vs Momentum Heat Map

5 Conclusion and Discussion

5.1 Discussion

One major limitation of our project was the use of CPU processing only to run our code. This was the case due to certain packages not being installed on our python client, when we realised this was the case, it was too late in the project to make the adjustment as all the results had been gathered. This will certainly be something which we would resolve if we were to do this project again. Had we had the GPU processing enabled, we would have been able to speed up the time taken to get results and given ourselves more time to improve our results or test our model on new datasets.

5.2 Conclusion

Overall, the project was a success, we managed to not only develop a complex neural network but also produce results that showed complex neural networks performing better than traditional neural networks. The significant difference in training time between the complex and real models can be a significant drawback of the complex model. Longer training times translate to higher computational and resource requirements, which may be prohibitive in many real-world applications where efficient and rapid model training is crucial. However, in instances where maximum generalisation and performance is most important, the complex model can be a suitable alternative.

6 Appendix - How to Run the Code

- Open shared Google Drive by clicking link on the front of this report.
- Open the python workbook in Google Colab.
- Run the first cell of code to install the correct Pytorch version.
- Restart the Google Colab runtime to use the updated Pytorch version.
- Comment out the first cell and run all the code cells from the second cell onwards in the Google Colab workbook.
- Run code cells in Google Colab workbook.
- Running this code will produce all the results, figures and tables presented in this report.

References

Guberman, Nitzan (2016). "On complex valued convolutional neural networks". In: *arXiv preprint arXiv:1602.09046*.