

Computer Science NEA Report

An investigation into how Artificial Neural Networks work, the effects of their hyper-parameters and their applications in Image Recognition.

Max Cotton

Contents

1	Introduction	3
1.1	Project Aims	3
1.2	Overview	3
2	Analysis	4
2.1	Theory Behind Artificial Neural Networks	4
2.1.1	Structure	4
2.1.2	How Artificial Neural Networks learn	6
2.2	Theory Behind Deep Artificial Neural Networks	8
2.2.1	Network Architecture and Training	8
2.2.2	Training	10
2.2.3	Forward Propagation:	10
2.2.4	Back Propagation:	10
2.3	Theory behind training the Artificial Neural Networks	11
2.4	Datasets	11
2.4.1	XOR dataset	11
2.4.2	MNIST dataset	12
2.4.3	Cat dataset	12
2.4.4	Theory behind using Graphics Cards to train Artificial Neural Networks	12
2.5	Interview	13
2.6	Project Objectives and Requirements	14
2.6.1	Objectives	14
2.6.2	Requirements	14
3	Design	16
3.1	Introduction	16
3.2	System Architecture	16
3.3	Class Diagrams	17
3.3.1	UI Class Diagram	17
3.3.2	Model Class Diagram	17
3.4	System Flow chart	18
3.5	Algorithm implementation	18
3.6	Data Structures, Techniques and File Structures used	20

3.6.1	Data structures	20
3.6.2	Techniques	20
3.6.3	File Structure	21
3.7	Database Design	22
3.8	Queries	22
3.9	Human-Computer Interaction	23
3.10	Hardware Design	28
3.11	Workflow and source control	28
4	Technical Solution	29
4.1	Source file organisation and management	29
4.1.1	File Structure	29
4.1.2	Dependencies	32
4.1.3	Git and Github files	32
4.1.4	Organisation	37
4.2	Models package	37
4.2.1	Utils subpackage	37
4.2.2	Artificial Neural Network implementations	48
4.3	Frames package	52
4.4	Project Entrypoint - <code>__main__.py</code> module	64
5	Testing	73
5.1	Summary of tests	73
5.2	Manual Testing - Input Validation Testing	73
5.2.1	Hyper Parameter Frame	73
5.2.2	Load Model Frame	78
5.2.3	Test Frames	79
5.3	Automated Testing	81
5.3.1	Unit Tests	81
5.3.2	GitHub Automated Testing	94
5.3.3	Docker	94
6	Investigation	96
6.1	<code>test_model</code> module	96
6.2	Exploration into the effects of Hyper-Parameters	105
6.3	Conclusions	124
7	Evaluation	125
7.1	Third Party Feedback	125
7.2	Project Objectives Evaluation	125
7.2.1	Project Objectives	125
7.2.2	Project Objective Evaluations	126
7.3	Requirements Evaluation	127
7.4	Future Improvements	127

1 Introduction

Artificial Intelligence is a branch of Computer Science that attempts to mimic human cognition in order to perform tasks, understand data and predict outcomes [3]. Machine Learning is a subfield of Artificial Intelligence that uses statistical algorithms, which take as an input training datasets, to produce mathematical models that allow prediction of the outcome of unseen datasets. Deep Learning is a further subfield of Machine Learning that uses Artificial Neural Networks, a process of learning from data inspired by the human brain. Artificial Neural Networks can be trained to operate on, "learn", a vast number of problems, such as Image Recognition, and have uses across multiple fields, such as medical imaging in hospitals.

1.1 Project Aims

This project is an investigation into how Artificial Neural Networks work, the effects of changing the hyper-parameters (such as the shape of the network and the learning rate) used to tune the models, and particularly their applications in Image Recognition. To achieve this, I have derived and researched all the fundamental theory behind the project, using sources such as IBM's online research [4], and developed Neural Networks from first principles without the use of any third-party Machine Learning libraries. I have then implemented the Artificial Neural Networks in the domain of Image Recognition, by creating trained models and have allowed for experimentation in varying the hyper-parameters of each model to provide for comparison and experimentation between different model performances. A Graphical User Interface has been developed to provide a mechanism for a researcher, or interested student, to train and test models and alter key hyper-parameters to explore the effect on performance and results.

1.2 Overview

Developing an Artificial Neural Network has required understanding the fundamental theoretical maths and algorithms underpinning this important technology which has been exciting and challenging. In implementing the network I have focused on an object-orientated approach utilising design approaches such as encapsulation and suitable data structures such as doubly linked lists. I have also researched and set up a development environment utilising tools such as GitHub, Jupyter Notebook, Visual Studio Code, LaTeX and automated unit testing etc. At the start of the project, I was lucky enough to interview an expert in the field of Neural Networks and AI and his guidance was valuable in setting the scope of the project and suggesting properties of the network to test. I am particularly pleased with being able to implement the network from the first principle maths. The main learning I have taken from this project is that Artificial Neural Networks require tuning to best address a particular problem - they are not a one-size-fits-all solution - and that this is a combination of research, experimentation and experience.

2 Analysis

2.1 Theory Behind Artificial Neural Networks

From an abstract perspective, Artificial Neural Networks are inspired by the anatomy of the human brain, consisting of layers of 'neurons' all interconnected via different links, 'axons with connecting synapses', each with their own strengths, 'weights'. By adjusting these links and weights, Artificial Neural Networks can be trained to take in an input and give its best prediction as an output [4].

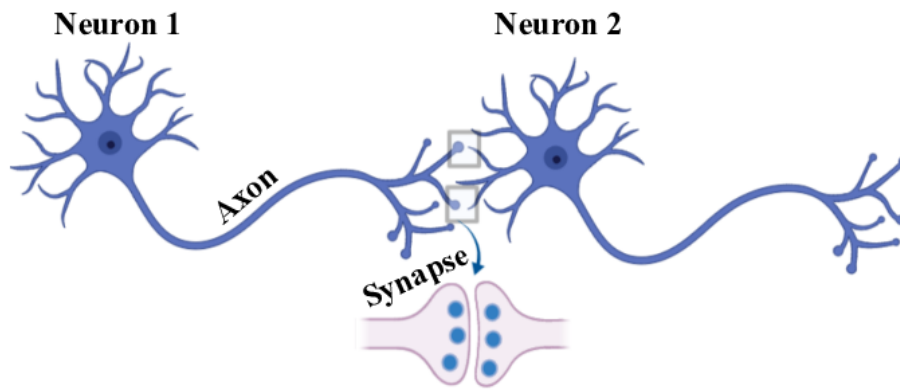


Figure 1: Two connected biological neurons sourced from <https://www.researchgate.net> [7]

2.1.1 Structure

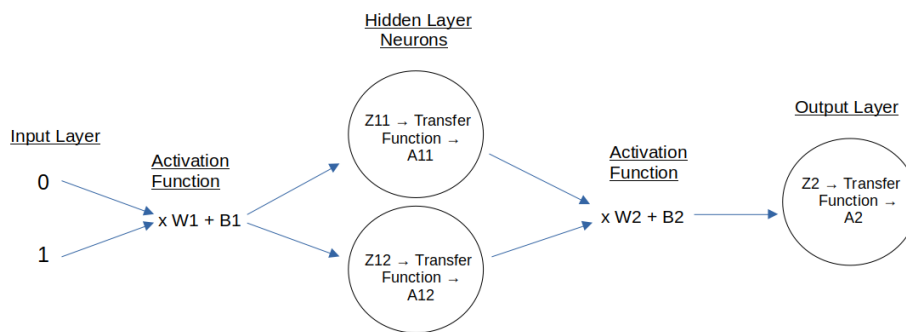


Figure 2: This shows an Artificial Neural Network with one single hidden layer and is known as a Shallow Neural Network.

I have focused on Feed-Forward Artificial Neural Networks, where values are entered to the input layer and passed forwards repetitively to the next

layer until reaching the output layer. Within this, I have investigated two types of Feed-Forward Artificial Neural Networks: Perceptron Artificial Neural Networks, that contain no hidden layers and are suitable for addressing simple linear problems, and Multi-Layer Perceptron Artificial Neural Networks, that contain at least one hidden layer. The expanded complexity of the Multi-Layer Perceptron Artificial Neural Network increases the non-linearity in the Artificial Neural Network allowing it to address more complex / non-linear problems.

Multi-Layer Perceptron Artificial Neural Networks consist of:

- An input layer of input neurons, where the input values are entered.
- Hidden layers of hidden neurons.
- An output layer of output neurons, which outputs the final prediction.

To implement an Artificial Neural Network, matrices are typically used to represent the layers, where each layer is a matrix of the layer's neuron's values. In order to use matrices for this, the following basic theory must be understood about them [6]:

- When Adding two matrices, both matrices must have the same number of rows and columns. Alternatively, a single column matrix with the same number of rows can be added by element-wise addition where each element is added to all of the elements of the corresponding rows in the associated matrix.
- In order to multiply matrices, the 'dot product' of the matrices is computed which multiplies the rows of one matrix with the columns of the other, multiplying matching members and then summing up.
- When calculating the dot product of two matrices, the number of columns of the 1st matrix must equal the number of rows of the 2nd matrix. The resulting matrix will have the same number of rows as the 1st matrix, and the same number of columns as the 2nd matrix. This is important in implementing an Artificial Neural Network, as the output of one layer must be formatted correctly to be used with the next layer.
- Alternatively, the Hadamard product of two matrices can be utilised which performs element-wise multiplication of the matrices. For this, both matrices must have the same number of rows and columns.
- Transposing a matrix is also utilised which switches all rows of the matrix into columns and all columns into rows in an output matrix.
- A matrix of values can be classified as a rank of Tensors, depending on the number of dimensions of the matrix. (Eg: A 2-dimensional matrix is a Tensor of rank 2)

I have focused on using Fully-Connected layers, that input values and apply the following functions to produce an output value for the layer:

- **Activation function** which calculates the dot product of an input matrix with a weight matrix, then sums the result with a bias matrix.

- **Transfer function** which takes the result of the activation function and calculates a suitable output value as well as adding non-linearity to the Neural Network. For example, the Sigmoid Transfer function converts the input value to an output number between zero and one - this makes it useful for logistic regression where the output value can be rounded to allow for a binary classification.

2.1.2 How Artificial Neural Networks learn

To train an Artificial Neural Network, the following processes are carried out for each of a number of training iterations called epochs:

- Forward Propagation which is the process of feeding inputs in and getting a prediction (moving forward through the network).
- Back Propagation, the process of calculating the error, known as the loss, in the prediction and then adjusting the weights and biases accordingly.

I have used Supervised Learning to train the Artificial Neural Networks, where the output prediction of the Artificial Neural Network is compared to the theoretical values it should have predicted. With this, I can calculate the loss value of the prediction. I then move back through the network and update the weights and biases via Gradient Descent which aims to reduce the Loss value of the prediction to a minimum, by subtracting the rate of change of Loss with respect to the weights / biases, multiplied with a learning rate, from the weights / biases. To calculate the rate of change of Loss with respect to the weights / biases, I used the following calculus methods:

- Partial Differentiation, which allows differentiation of multi-variable functions, by differentiating with respect to one variable and considering the rest as constants.
- The Chain Rule, where for $y = f(u)$ and $u = g(x)$, $\frac{\partial y}{\partial x}$ can be calculated as: $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} * \frac{\partial u}{\partial x}$.

This repetitive process will continue to reduce the Loss to a minimum, if the learning rate is set to an appropriate value



Figure 3: Gradient Descent
sourced from <https://www.ibm.com/topics/gradient-descent> [4]

During backpropagation, however, some issues can occur, such as the following:

- Finding a false local minimum rather than the global minimum of the function
- Having an 'Exploding Gradient' [3], where the gradient value grows exponentially to the point of overflow errors
- Having a 'Vanishing Gradient' [3], where the gradient value decreases to a very small value or zero, resulting in a lack of updating values during training.

2.2 Theory Behind Deep Artificial Neural Networks

2.2.1 Network Architecture and Training

Figure 4 below shows a simplified view of an Artificial Neural Network with multiple hidden layers, known as a Deep Neural Network, where:



Figure 4: Showing an abstracted view of an Artificial Neural Network with multiple hidden layers.

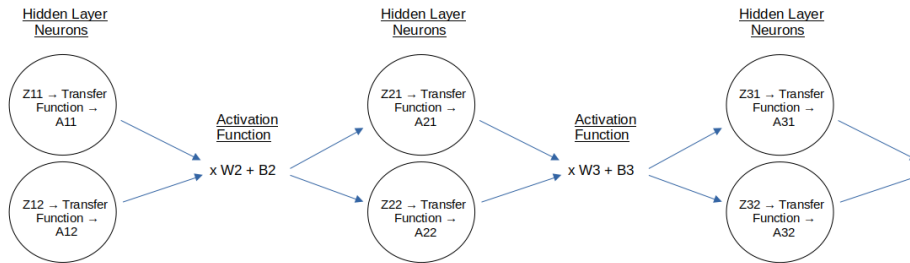


Figure 5: This shows an Artificial Neural Network with multiple hidden layers and is known as a Deep Neural Network.

Figure 5 shows the network in more detail. The layers have inputs x , weight matrix wN and biases BN .

- Each layer takes the previous layer's output as its input X
- An Activation function is applied to X to obtain Z , by taking the dot product of X with a weight matrix W , and sums the result with a bias matrix B . At first when the Neural Network is initialised pre-training the weights are initialised to random values and the biases are set to zeros.

$$Z = W * X + B$$
- A Transfer function is then applied to Z to obtain the layer's output A
 - For the output layer, the sigmoid function, explained in section 2.1, can be used for either for binary classification via logistic regression, or for multi-class classification where the output neuron, and the associated class, that has the highest value between zero and one is selected.
 - * Where $\text{sigmoid}(Z) = \frac{1}{1+e^{-Z}}$
 - However, for the input layer and the hidden layers, another transfer function known as ReLu (Rectified Linear Unit) can be better suited as it produces larger values of $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial B}$ for Gradient Descent than Sigmoid [6], so updates at a quicker rate.
 - * Where $\text{relu}(Z) = \max(0, Z)$

2.2.2 Training

Training takes place through a series of forward and backward propagations called epochs. The forward propagation generates a potential output and associated error known as the loss. Backward propagation then adjusts the weights and biases in an attempt to reduce this loss.

2.2.3 Forward Propagation:

- For each epoch the input layer is presented with a matrix of input values, which are fed through the network to obtain a final prediction A from the output layer.
- Using the process described in the previous section, backpropagation trains the network by adjusting weights and biases.

2.2.4 Back Propagation:

- The Loss value L is first calculated using the following Log-Loss function shown in the equation below, which calculates the average difference between A and the value it should have predicted Y. The average is then found by summing the result of the Loss function [1] for each value in the matrix A, then dividing by the number of predictions m, resulting in a Loss value indicating how well the network is performing. Where

$$L = -(\frac{1}{m}) * \sum(Y * \log(A) + (1 - Y) * \log(1 - A))$$

and "log()" is the natural logarithm.

- The network is then trained by moving back through the layers, adjusting the weights and biases via Gradient Descent. For each layer, the weights and biases are updated with the following formulae:

$$\begin{aligned} - W &= W - learningRate * \frac{\partial L}{\partial W} \\ - B &= B - learningRate * \frac{\partial L}{\partial B} \end{aligned}$$

- The derivation for Layer 2's $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial B}$ is shown below:

– Functions used so far:

$$1. Z = W * X + B$$

$$2. A_{relu} = \max(0, Z)$$

$$3. A_{sigmoid} = \frac{1}{1+e^{-Z}}$$

$$4. L = -(\frac{1}{m}) * \sum(Y * \log(A) + (1 - Y) * \log(1 - A))$$

$$- \frac{\partial L}{\partial A2} = \frac{\partial L}{\partial A3} * \frac{\partial A3}{\partial Z3} * \frac{\partial Z3}{\partial A2}$$

By using function 1, where A2 is X for the 3rd layer, $\frac{\partial Z3}{\partial A2} = W3$

$$\Rightarrow \frac{\partial L}{\partial A2} = \frac{\partial L}{\partial A3} * \frac{\partial A3}{\partial Z3} * W3$$

$$- \frac{\partial L}{\partial W2} = \frac{\partial L}{\partial A2} * \frac{\partial A2}{\partial Z2} * \frac{\partial Z2}{\partial W2}$$

By using function 1, where A1 is X for the 2nd layer, $\frac{\partial Z2}{\partial W2} = A1$

$$\Rightarrow \frac{\partial L}{\partial W2} = \frac{\partial L}{\partial A2} * \frac{\partial A2}{\partial Z2} * A1$$

$$\begin{aligned}
- \frac{\partial L}{\partial B2} &= \frac{\partial L}{\partial A2} * \frac{\partial A2}{\partial Z2} * \frac{\partial Z2}{\partial B2} \\
&\text{By using function 1, } \frac{\partial Z2}{\partial B2} = 1 \\
\Rightarrow \frac{\partial L}{\partial W2} &= \frac{\partial L}{\partial A2} * \frac{\partial A2}{\partial Z2} * 1
\end{aligned}$$

- As can be seen, when moving back through the network, $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial B}$ can be calculated for each layer using the rate of change of loss with respect to its output. This is calculated by the previous layer using the above formula; the derivative of the layer's transfer function, and the layers input (which in this case is A1)
 - Where by using function 2, $\frac{\partial A_{relu}}{\partial Z} = 1$ when $Z \geq 0$ otherwise $\frac{\partial A_{relu}}{\partial Z} = 0$
 - Where by using function 3, $\frac{\partial A_{sigmoid}}{\partial Z} = A * (1 - A)$
- At the start of backpropagation, the rate of change of loss with respect to the output layer's output has no previous layer's calculations, so instead it can be found with the derivative of the Log-Loss function, as shown in the following:
 - Using function 4, $\frac{\partial L}{\partial A} = (-\frac{1}{m})(\frac{Y-A}{A*(1-A)})$

2.3 Theory behind training the Artificial Neural Networks

Training an Artificial Neural Network's weights and biases to predict on a dataset creates a trained model for that dataset. This model can be used to create predictions based on future data / images inputted. However, training Artificial Neural Networks suffers problems such as Overfitting, where the trained model learns the patterns of the training dataset too well, resulting in poor predictions on new datasets. This can occur when the training dataset does not cover enough situations of inputs and the desired outputs (by being too small for example), if the model is trained for too many epochs on the poor dataset or by having too many layers in the Neural Network.

Another common problem is Underfitting, where the model has not learnt the patterns of the training dataset well enough, often when it has been trained for too few epochs, or when the Neural Network is too linear.

2.4 Datasets

I have utilised a series of open source datasets on which to train and test by Neural Networks.

2.4.1 XOR dataset

As a first step in developing and testing Artificial Neural Networks, I have utilised the XOR gate problem, where the Neural Network is fed input pairs of zeros and ones and learns to predict the output of a XOR gate used in circuits. This takes far less computation time than image datasets and was extremely useful in debugging. It is a good example of a relatively simple problem whilst not being linearly separable.

2.4.2 MNIST dataset

The MNIST dataset [9] is a well known dataset consisting of images of hand-written digits from zero to ten and is commonly used to test the performance of an Artificial Neural Network. The dataset consists of 60,000 input images representing single digits made up from 28x28 pixels with each pixel having a value from 0 to 255. To format the images into a suitable format to be inputted into the Artificial Neural Networks, each image's matrix of pixel values is commonly 'flattened' into a 1 dimensional matrix of values, where each element is divided by 255 (the maximum 8 Bit value) to produce a number between 0 and 1, to standardize the dataset. The output dataset is also loaded which represents the actual value of the number in the image. This is commonly implemented by using an array for each image where the index of the array represents the number in an image (i.e. a 1 in column 2 could represent a 2, or a 1 if zero indexed).

To create a trained Artificial Neural Network model on this dataset, the model requires 10 output neurons (one for each digit). The Sigmoid Transfer function is then utilized to output a number between one and zero to each neuron - whichever neuron received the highest value is selected as the predicted outcome. This an example of a multi-class classification, where the model must predict one of 10 classes (in this case, each class is one of the digits from zero to ten).

2.4.3 Cat dataset

I have also used a dataset of images of cats sourced from <https://github.com/marcopeix> [5], where each image is classified as either a cat or not a cat. The dataset consists of 209 input images, made up from 64x64 pixels with each pixel having an value from 0 to 255. To normalise the images into a suitable format to be input into the Artificial Neural Network, each image's matrix of values is 'flattened' into a 1 dimensional matrix of values, where each element is divided by 255 (the maximum 8 bit value) to a number between 0 and 1, to standardize the dataset.

The output dataset represents an array of binary values representing the output of each image (1 for a cat, 0 for not a cat). To create a trained Artificial Neural Network model on this dataset, the model requires only 1 output neuron - representing the chance of being a cat. By using the Sigmoid Transfer function, a number is outputted between one and zero for the neuron, if the neuron's value is closer to 1 it predicts a cat, otherwise it predicts not a cat - this is binary classification.

2.4.4 Theory behind using Graphics Cards to train Artificial Neural Networks

Graphics Cards have been designed essentially to undertake parallel matrix computations utilising many Tensor cores - which are processing units specialised for matrix operations calculating the co-ordinates of 3D graphics, however they can be used for operating on the matrices in the network at a much faster speed compared to CPUs. GPUs also include CUDA cores which act as an API to

the GPU's computing to be used for any operations (in this case training the Artificial Neural Networks).

2.5 Interview

In order to gain a better foundation for my investigation, and ensure my project would allow a user interested in Artificial Neural Networks to experiment with the fundamentals of the network and conduct experiments, I presented my prototype code and interviewed the head of Artificial Intelligence at Cambridge Consultants - Will Addison. These were their responses:

- Q: "Are there any good resources you would recommend for learning the theory behind how Artificial Neural Networks work?"
A: "There are lots of useful free resources on the internet to use. I particularly like the platform 'Medium' which offers many scientific articles as well as more obvious resources such as IBMs'."
- Q: "What do you think would be a good goal for my project?"
A: "I think it would be great to aim for applying the Neural Networks on Image Recognition for some famous datasets. For you, I would recommend the MNIST dataset as a goal."
- Q: "What features of the Artificial Neural Networks would you like to be able to experiment with?"
A: "I'd like to be able to experiment with the number of layers and the number of neurons in each layer, and then be able to see how these changes effect the performance of the model. I can see that you've utilised the Sigmoid transfer function and I would recommend having the option to test alternatives such as the ReLu transfer function, which will help stop issues such as a vanishing gradient."
- Q: "What are some practical constraints of AI?"
A: "Training AI models can require a large amount of computing power, also large datasets are needed for training models to a high accuracy which can be hard to obtain."
- Q: "What would you say increases the computing power required the most?"
A: "The number of layers and neurons in each layer will have the greatest effect on the computing power required. This is another reason why I recommend adding the ReLu transfer function as it updates the values of the weights and biases faster than the Sigmoid transfer function."
- Q: "Do you think I should explore other computer architectures for training the models?"
A: "Yes, it would be great to add support for using graphics cards for training models, as this would be a vast improvement in training time compared to using just CPU power."
- Q: "I am also creating a user interface for the program, what hyper-parameters would you like to be able to control through this?"

A: "It would be nice to control the transfer functions used, as well as the general hyper-parameters of the model. I also think you could add a progress tracker to be displayed during training for the user."

- Q: "How do you think I should measure the performance of models?"

A: "You should show the accuracy of the model's predictions, as well as example incorrect and correct prediction results for the trained model. Additionally, you could compare how the size of the training dataset effects the performance of the model after training, to see if a larger dataset would seem beneficial."

- Q: "Are there any other features you would like add?"

A: "Yes, it would be nice to be able to save a model after training and have the option to load in a trained model for testing."

2.6 Project Objectives and Requirements

Based on the interview above in section 2.5, the following high-level objectives were formulated:

2.6.1 Objectives

Objective ID	Description
1	Learn how Artificial Neural Networks work and develop them from first principles
2	Implement the Artificial Neural Networks by creating trained models based on image datasets
2.1	Allow use of Graphics Cards for faster training
2.2	Allow for the saving and loading of trained models
3	Develop a Graphical User Interface
3.1	Provide controls for hyper-parameters of models
3.2	Display and compare the results each model's predictions

2.6.2 Requirements

The following sets out the steps that must be taken to accomplish the above objectives:

ID	Description	Satisfied by	Tested by
1	Learn how Artificial Neural Networks work	Page 4	N/A
2	Develop Artificial Neural Networks from first principles		
2.1	Provide utilities for creating Artificial Neural Networks	Page 37	Page 86
2.2	Allow for the saving and loading of trained models' weights and biases	Page 41	Page 86
2.3	Allow use of Graphics Cards for faster training	Code not included in report	Page 123
3	Implement the Artificial Neural Networks on image datasets		
3.1	Allow input of unique hyper-parameters	Page 48	Page 105
3.2	Allow unique datasets and train dataset size to be loaded	Page 48	Page 112
4	Use a database to store a model's features and the location of its weights and biases	Page 64	Page 81
5	Develop a Graphical User Interface		
5.1	Provide controls for hyper-parameters of models	Page 53	Page 73
5.2	Display details of models' training	Page 53	N/A
5.3	Display the results of each model's predictions	Page 96	User Tested
5.4	Allow for the saving of trained models	Page 96	Page 79
5.5	Allow for the loading of saved trained models	Page 59	Page 78

3 Design

3.1 Introduction

The following design focuses have been made for the project:

- The program will support multiple platforms to run on, including Windows and Linux.
- The program will use python3 as its main programming language.
- An object-orientated approach is used in the design of the project.
- An option will be given to use either a Graphics Card or a CPU to train and test the Artificial Neural Networks.
- SysML will be utilised in the design of the architecture and class diagrams

3.2 System Architecture

The project is architected using object-orientated design principles, it was then implemented in Python. A SysML block diagram [2] showing the key architectural components is shown in the figure below with detailed SysML class diagrams shown in section 3.3.1.

As shown in the Figure 6 the User interacts with the software through a User Interface (UI) which is written in tkinter. The UI classes interface with the Model block which contains all the classes necessary for representing the Artificial Neural Network models. As such functional separation is maintained between the user interface elements and the Neural Network representation.

bdd [block] School Project Frame [System Architecture Diagram]

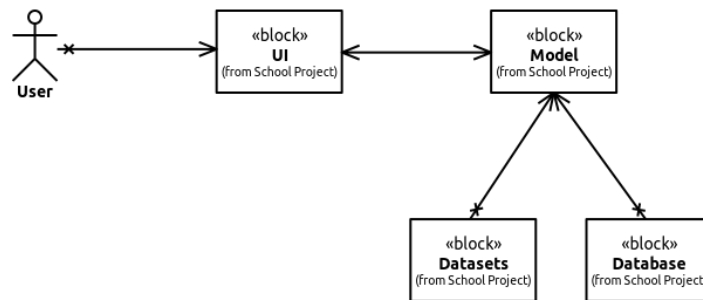


Figure 6: High level system architecture

3.3 Class Diagrams

3.3.1 UI Class Diagram

The classes utilised to implement the UI are shown in Figure 7. As can be seen the School.Project.Frame inherits the tk.Frame class - the tkinter base class which provides the basic frame functionality. The School.Project.Frame then contains a series of UI interaction specific Frames.

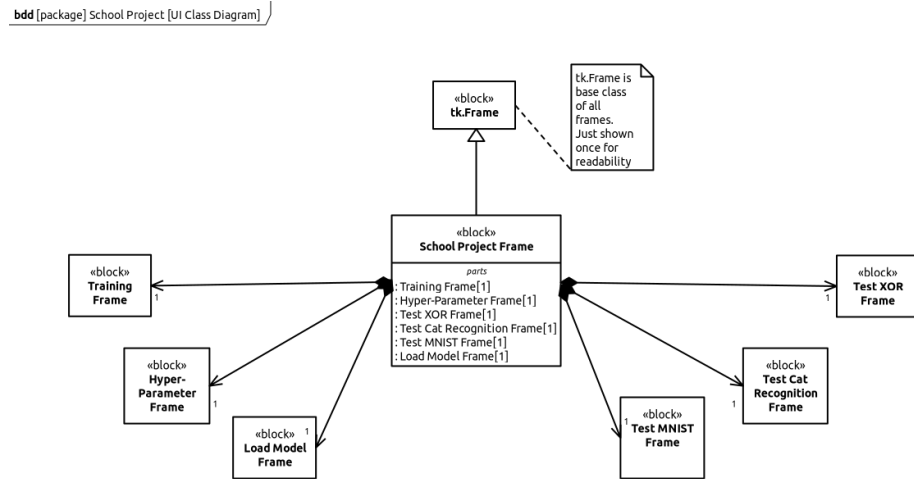


Figure 7: UI SysML Class Diagram

3.3.2 Model Class Diagram

The Model class diagram is shown below in figure 8. An AbstractModel class contains a linked list of FullyConnectedLayer classes which represent the layers within the Artificial Neural Network.

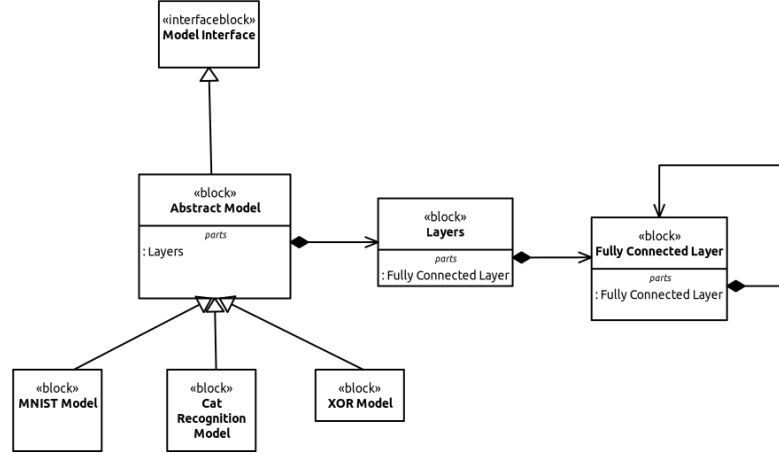


Figure 8: Artificial Neural Network and Layers SysML Class Diagram

3.4 System Flow chart

The system flow is shown in Figure 9 below. When the program is initialised the Home Frame is first displayed. The left-hand route - *Hyper-Parameter Frame* and *Training Frame* - provide the functionality to set hyper-parameters and train a new model. This model can be saved and loaded, following the right-hand route, if desired before entering the *Testing Frame* where the model can be applied to testing data and the model performance assessed. The ability to save models is important as it allows for model training on a computer with a GPU and utilisation on a non-GPU-accelerated computer. This allowed an efficient build test cycle as I could build models on a desk-based, GPU accelerated PC, and undertake testing / demonstration on a laptop.

3.5 Algorithm implementation

The detail of algorithms underpinning the Artificial Neural Network are outlined in section 2.1. To implement the algorithm within the project each layer within the Artificial Neural Network is represented as an instance of a *FullyConnectedLayer* class each of which:

- Contains a weight and bias matrix.
- Has a *transfer_function* which be selected by the user to be either Sigmoid or ReLu but can be easily extended to other functions.
- Has a *forward_propagation* function which takes an input matrix which it then multiplies by the weight matrix and sums the result with the bias matrix, this is then put through the *transfer_function*.

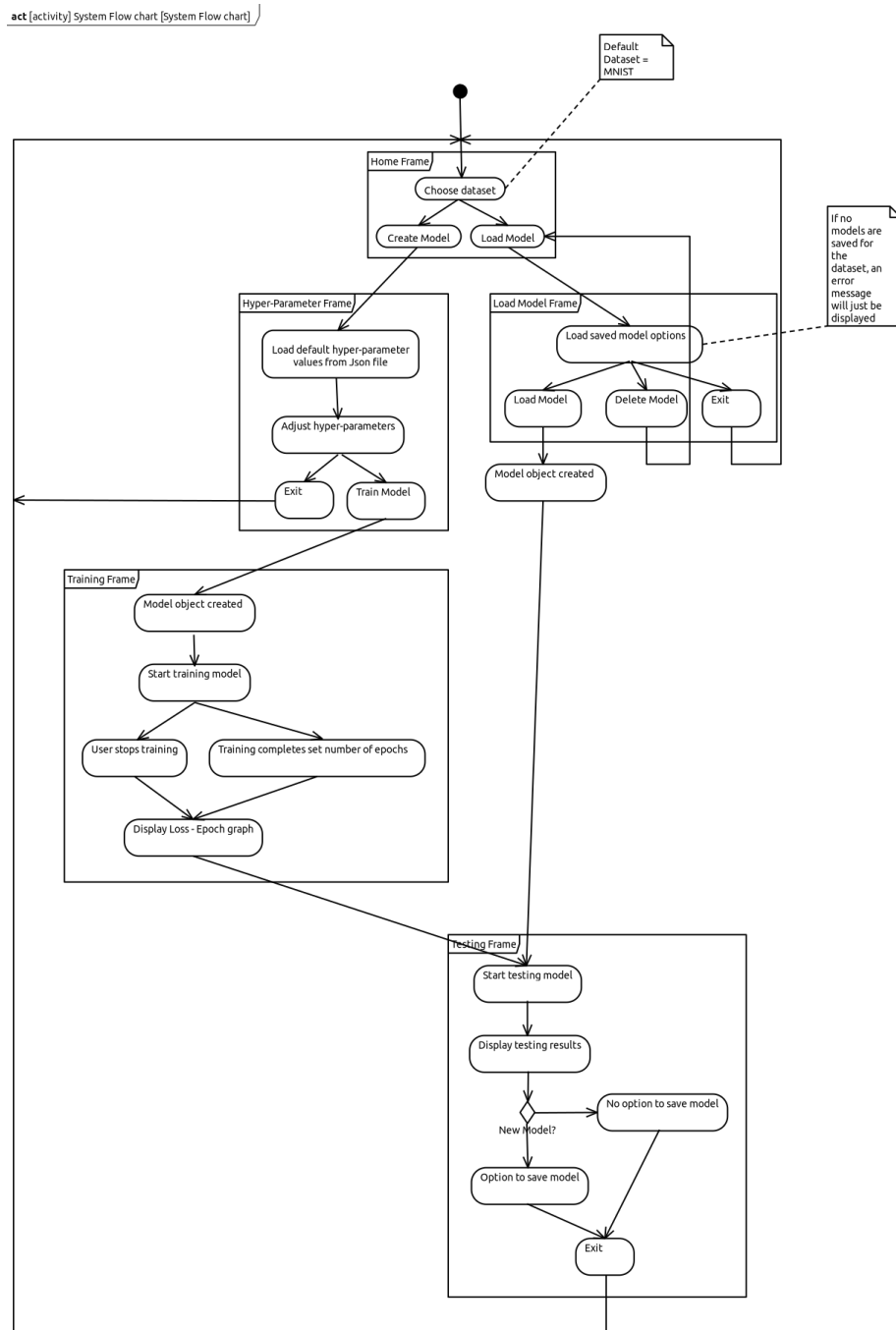


Figure 9: System flow chart

- Has a *backward_propagation* function which passes the loss with respect to each layers output allowing adjustment of the weights and biases.

Each *FullyConnectedLayer* class is held within a doubly linked list contained within the *Abstract Model Class*, which is a parent class of the data-specific classes *MNIST Model*, *Cat Recognition Model* and *XOR Model*. The use of a doubly linked list allows for a user defined number of hidden layers from 0 upwards and allows easy propagation of the list.

The algorithm to train the Artificial Neural Network iterates over a user-defined number of epochs. Each epoch starts with the presentation of data to the input layer. The doubly linked list of layers is then traversed calling the *forward_propagation* function on each. When the output layer is reached the derivative of the *Log-Loss* function is calculated alongside the absolute *Loss* for debugging and learning rate plotting.

Following the calculation of the derivative of the *Log-Loss* function, the doubly linked list is then iterated in reverse back through the layers calculating the loss with respect to the weights and biases of each layer. This process is then used to adjust the weights and biases in each layer using the specified learning rate in an attempt to reduce the loss value.

3.6 Data Structures, Techniques and File Structures used

3.6.1 Data structures

The following data structures are utilised in implementing the Artificial Neural Network:

- Standard lists for storing data, for example storing the shape of the Artificial Neural Network's layers.
- Tuples where tuple unpacking is useful, such as returning multiple values from methods.
- Dictionaries for loading the default hyper-parameter values from a JSON file.
- Matrices to represent the layers and allow for a varied number of neurons in each layer. To represent the Matrices I will use both numpy arrays and cupy arrays.
- A Doubly linked list to represent the Artificial Neural Network, where each node is a layer of the network. This allows traversing both forwards and backwards through the network, as well as storing the first and last layer to start forward and backward propagation respectively.

3.6.2 Techniques

Some techniques used include:

- Object-oriented design including inheritance, abstract classes and interfaces
- Encapsulation
- Get methods
- Abstract and Static methods

- Decorators to wrap functions and modify behaviour
- tkinter for user interface design
- SQL for database access
- Type Hinting
- Docstrings
- Generators
- Breaking down the project into subpackages and modules
- Raising and Handling of Exceptions
- Threading

3.6.3 File Structure

I have used the following file structures to store necessary data for the program:

- A JSON file [8] for storing the default hyper-parameters for creating a new model for each dataset.
- Image dataset files are stored in either a compressed archive file (such as .pkl.gz files) or of the Hierarchical Data Format (such as .h5) for storing large datasets with fast retrieval.
- Weights and biases of saved models are stored as numpy arrays in .npz files (a zipped archive file format) in a 'saved-models' directory, which allows compatibility with the standard numpy library.

3.7 Database Design

The following Relational database design is used for saving models, where the dataset, name and features of the saved model (including the location of the saved models' weights and biases and the saved models' hyper-parameters) are saved. The Model_ID field is the primary key.

Models	
Model_ID	integer
Dataset	text
File_Location	text
Hidden_Layers_Shape	text
Learning_Rate	float
Name	text
Train_Dataset_Size	integer
Use_ReLu	bool

Figure 10: Database design

The following unique constraint is also used so that each dataset can not have more than one model with the same name:

`UNIQUE (Dataset, Name)`

The following constraint is applied to ensure no attribute is left empty:

`NOT NULL`

3.8 Queries

Below are some example queries used for interacting with the database:

- Query the names of all saved models for a dataset:

```
SELECT Name FROM Models WHERE Dataset=?;
```

- Query the file location of a saved model:

```
SELECT File_Location FROM Models WHERE Dataset=? AND Name=?;
```

- Query the features of a saved model:

```
SELECT * FROM Models WHERE Dataset=? AND Name=?;
```

3.9 Human-Computer Interaction

Below are the designs of each tkinter frame in the User Interface. The flow of the frames is described in Figure 9. The final implementation of the frames is shown in the technical solution section.

- The home Frame design which acts as the entry point to the program, the implemented version is shown in Figure 22.

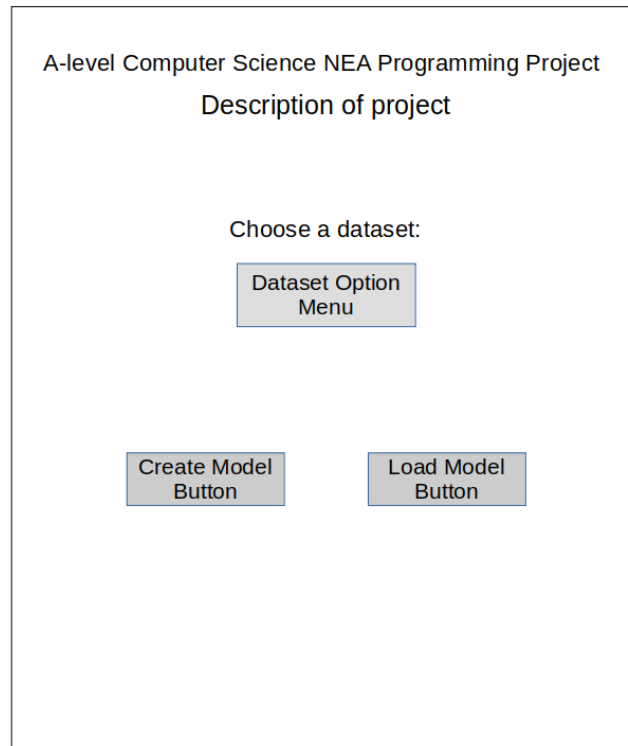


Figure 11: Home frame design - the entry point to the program

- Hyper-Parameter Frame design which allows setting of the Hyper-parameters, the implemented version is shown in Figure 17.

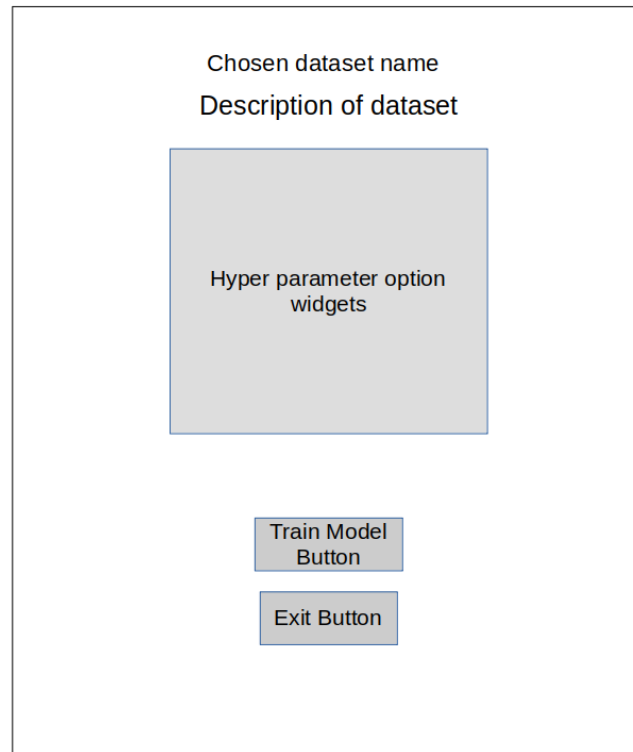


Figure 12: Hyper-Parameter Frame design which allows setting of parameters

- Training Frame design:
 - During training, the following is displayed on the Training Frame, the implemented version is shown in Figure 18

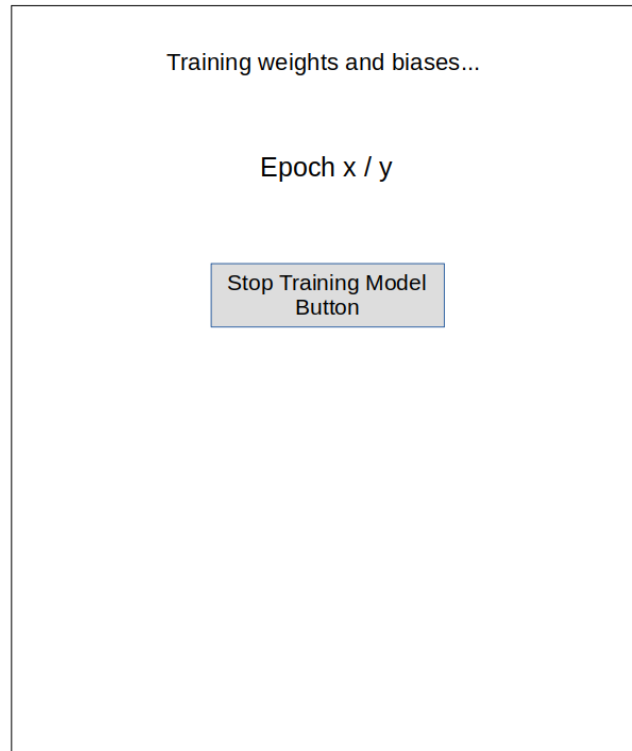


Figure 13: Training frame layout during training

- Once training has finished, the following is displayed on the Training Frame, the implemented version is shown in Figure 19.

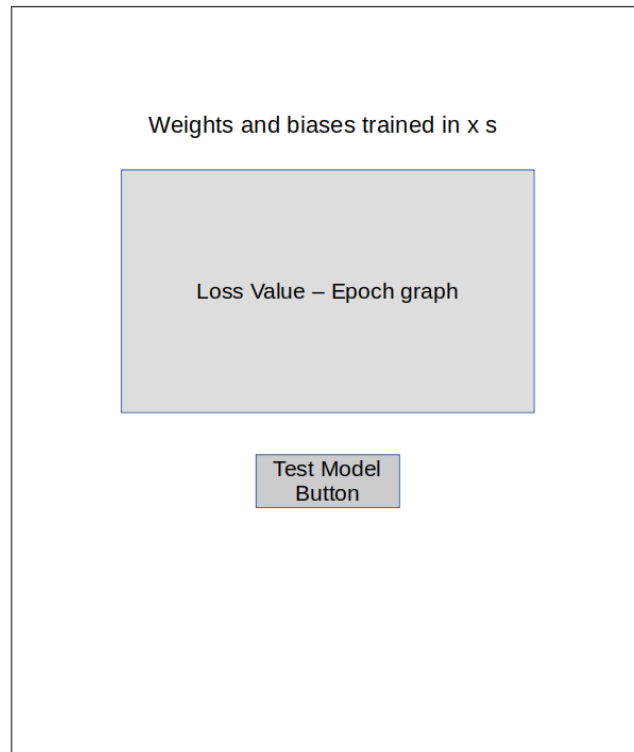


Figure 14: Training frame layout after training

- Load Model Frame design, the implemented version is shown in Figure 20.

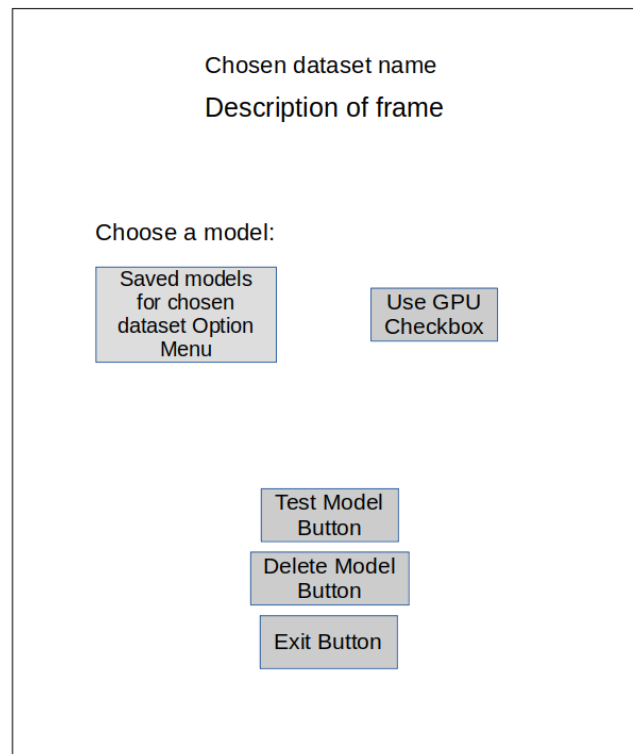


Figure 15: Load model frame design

- Test Frame design, the implemented version is shown in Figure 34.

Testing Results:
Prediction Accuracy: x %
Network Shape: y

Example Results

Enter a name for your trained model:

Text Entry for model name

Save Model Button

Exit Button

Figure 16: Test frame design

3.10 Hardware Design

To allow for faster training of an Artificial Neural Network, I gave the option to use a Graphics Card to train the Artificial Neural Network where available. I also gave the option to load pretrained weights to run on less computationally powerful hardware using just the CPU as standard. This was implemented by using the CuPy library.

3.11 Workflow and source control

Git and GitHub were used to manage the workflow and provide source control as the project was developed. In particular the following features were utilised:

- Commits and branches for adding features and fixing bugs separately.
- Using GitHub to back up the project as a repository.
- Automated testing on GitHub after each pushed commit.
- Providing the necessary instructions and information for the installation and usage of this project, as well as creating releases of the project with new patches.

4 Technical Solution

4.1 Source file organisation and management

4.1.1 File Structure

The following file structure is used to organise the code for the project, where `school_project` is the main package and is constructed of two main subpackages:

- The `models` package, which is a self-contained package for creating trained Artificial Neural Network models.
- The `frames` package, which consists of tkinter frames for the User Interface.

```

.
|-- Dockerfile
|-- .github
|   |-- workflows
|   |-- tests.yml
|-- .gitignore
|-- LICENSE
|-- notebooks
|   |-- cpu-vs-gpu-analysis.ipynb
|   |-- epoch-count-analysis.ipynb
|   |-- layer-count-analysis.ipynb
|   |-- learning-rate-analysis.ipynb
|   |-- neuron-count-analysis.ipynb
|   |-- relu-analysis.ipynb
|   |-- train-dataset-size-analysis.ipynb
|-- README.md
|-- school_project
|   |-- frames
|   |   |-- create_model.py
|   |   |-- hyper-parameter-defaults.json
|   |   |-- __init__.py
|   |   |-- load_model.py
|   |   |-- test_model.py
|   |-- __init__.py
|   |-- __main__.py
|   |-- models
|   |   |-- cpu
|   |   |   |-- cat_recognition.py
|   |   |   |-- __init__.py
|   |   |   |-- mnist.py
|   |   |   |-- utils
|   |   |   |   |-- __init__.py
|   |   |   |   |-- model.py
|   |   |   |   |-- tools.py
|   |   |   |-- xor.py
|   |   |-- datasets
|   |   |   |-- mnist.pkl.gz
|   |   |   |-- test-cat.h5
|   |   |   |-- train-cat.h5
|   |   |-- gpu
|   |   |   |-- cat_recognition.py
|   |   |   |-- __init__.py
|   |   |   |-- mnist.py
|   |   |   |-- utils
|   |   |   |   |-- __init__.py
|   |   |   |   |-- model.py
|   |   |   |   |-- tools.py
|   |   |   |-- xor.py
|   |   |-- __init__.py
|   |-- saved-models
|   |-- test
|   |   |-- __init__.py
|   |   |-- models
|   |   |   |-- cpu
|   |   |   |   |-- __init__.py
|   |   |   |   |-- utils
|   |   |   |   |   |-- __init__.py
|   |   |   |   |   |-- test_model.py
|   |   |   |   |   |-- test_tools.py
|   |   |   |-- gpu
|   |   |   |   |-- __init__.py
|   |   |   |   |-- utils

```

```
|      | |      |-- __init__.py
|      | |      |-- test_model.py
|      | |      |-- test_tools.py
|      | |-- __init__.py
|      |-- test_database.py
|-- setup.py
`-- TODO.md
```

19 directories, 50 files

Each package within the school_project package contains a `__init__.py` file, allowing the school_project package to be installed to a virtual environment so that the modules of the package can be imported from the installed package.

Show below are the contents of the frames package's `__init__.py` for example, which allows the classes of all modules in the package to be imported simultaneously:

```
1  """Package of tkinter frames for the main window."""
2
3  from .create_model import HyperParameterFrame, TrainingFrame
4  from .load_model import LoadModelFrame
5  from .test_model import TestMNISTFrame, TestCatRecognitionFrame, TestXORFrame
6
7  __all__ = ['create_model', 'load_model', 'test_model']
```

4.1.2 Dependencies

The python dependencies for the project can be installed by running the following `setup.py` file (as described in the README.md in the next section). Instructions on installing external dependencies, such as the CUDA Toolkit for using a GPU, are explained in the README.md in the following section.

- `setup.py` code:

```
1  from setuptools import setup, find_packages
2
3  setup(
4      name='school-project',
5      version='2.0.0',
6      packages=find_packages(),
7      url='https://github.com/mcttn22/school-project.git',
8      author='Max Cotton',
9      author_email='maxcotton22@gmail.com',
10     description='Year 13 Computer Science Programming Project',
11     install_requires=[
12         'cupy-cuda12x',
13         'h5py',
14         'matplotlib',
15         'numpy'
16     ],
17 )
```

4.1.3 Git and Github files

Git and Github were used extensively to manage the codebase and utilised the following files:

- A `.gitignore` file for specifying which files and directories should be ignored by Git:

```
1  # Byte compiled files
2  __pycache__
3
4  # Packaging
5  *.egg-info
```

```

6
7 # Database file
8 school_project/saved_models.db

```

- A README.md markdown file to give installation and usage instructions for the repository on GitHub:

– Markdown code:

```

1 <!-- The following lines generate badges showing the current status of
   ↳ the automated testing (Passing or Failing) and a Python3 badge
   ↳ correspondingly.) -->
2 [![tests](https://github.com/mcttn22/school-project/actions/workflows/tests.yml/badge.svg)](https://
3 [![python](https://img.shields.io/badge/Python-3-3776AB.svg?style=flat&logo=python&logoColor=white)]
4
5 # A-level Computer Science NEA Programming Project
6
7 This project is an investigation into how Artificial Neural Networks
   ↳ (ANNs) work and their applications in Image Recognition, by
   ↳ documenting all theory behind the project and developing
   ↳ applications of the theory, that allow for experimentation via a
   ↳ GUI. The ANNs are created without the use of any 3rd party Machine
   ↳ Learning Libraries and I currently have been able to achieve a
   ↳ prediction accuracy of 99.6% on the MNIST dataset. The report for
   ↳ this project is also included in this repository.
8
9 ## Installation
10
11 1. Download the Repository with:
12
13 - ```
14     git clone https://github.com/mcttn22/school-project.git
15     ```
16 - Or by downloading as a ZIP file
17
18 </br>
19
20 2. Create a virtual environment (venv) with:
21 - Windows:
22     ```
23     python -m venv {venv name}
24     ```
25 - Linux:
26     ```
27     python3 -m venv {venv name}
28     ```
29
30 3. Enter the venv with:
31 - Windows:
32     ```
33     .\{venv name}\Scripts\activate
34     ```
35 - Linux:
36     ```
37     source ./{venv name}/bin/activate
38     ```
39
40 4. Enter the project directory with:
41     ```
42     cd school-project/
43     ```
44

```

```

45 5. For normal use, install the dependencies and the project to the
    ↳ venv with:
46 - Windows:
47     ```
48     python setup.py install
49     ```
50 - Linux:
51     ```
52     python3 setup.py install
53     ```
54
55 *Note: In order to use an Nvidia GPU for training the networks, the
    ↳ latest Nvidia drivers must be installed and the CUDA Toolkit must
    ↳ be installed from
56 <a href="https://developer.nvidia.com/cuda-downloads">here</a>.*
57
58 ## Usage
59
60 Run with:
61 - Windows:
62     ```
63     python school_project
64     ```
65 - Linux:
66     ```
67     python3 school_project
68     ```
69
70 ## Development
71
72 Install the dependencies and the project to the venv in developing
    ↳ mode with:
73 - Windows:
74     ```
75     python setup.py develop
76     ```
77 - Linux:
78     ```
79     python3 setup.py develop
80     ```
81
82 Run Tests with:
83 - Windows:
84     ```
85     python -m unittest discover .\school_project\test\
86     ```
87 - Linux:
88     ```
89     python3 -m unittest discover ./school_project/test/
90     ```
91
92 Use Docker with:
93 - Build the Docker Image with:
94     ```
95     sudo docker build -t mcttn22/school-project ./
96     ```
97 - Run the Docker Image with:
98     ```
99     sudo apt-get install x11-xserver-utils
100    xhost +
101    sudo docker run -v /tmp/.X11-unix:/tmp/.X11-unix -e
    ↳ DISPLAY=unix$DISPLAY mcttn22/school-project
102    ```
103

```

```
104 Compile Project Report PDF with:
105 ~~~
106 make all
107 ~~~
108 *Note: This requires the Latexmk, pdflatex and Pygments libraries*
```

- Which will generate the following:



A-level Computer Science NEA Programming Project

This project is an investigation into how Artificial Neural Networks (ANNs) work and their applications in Image Recognition, by documenting all theory behind the project and developing applications of the theory, that allow for experimentation via a GUI. The ANNs are created without the use of any 3rd party Machine Learning Libraries and I currently have been able to achieve a prediction accuracy of 99.6% on the MNIST dataset. The report for this project is also included in this repository.

Installation

1. Download the Repository with:

- `git clone https://github.com/mcttn22/school-project.git`



- Or by downloading as a ZIP file

2. Create a virtual environment (venv) with:

- Windows:

- `python -m venv {venv name}`



- Linux:

```
python3 -m venv {venv name}
```



3. Enter the venv with:

- Windows:

```
.\{venv name}\Scripts\activate
```



- Linux:

```
source ./{venv name}/bin/activate
```



4. Enter the project directory with:

```
cd school-project/
```



5. For normal use, install the dependencies and the project to the venv with:

- Windows:

```
python setup.py install
```



- Linux:

```
python3 setup.py install
```



Note: In order to use an Nvidia GPU for training the networks, the latest Nvidia drivers must be installed and the CUDA Toolkit must be installed from [here](#).

Usage

Run with:

- Windows:

```
python school_project
```



- Linux:

```
python3 school_project
```



Development

Install the dependencies and the project to the venv in developing mode with:

- Windows:

```
python setup.py develop
```



- Linux:

```
python3 setup.py develop
```



Run Tests with:

- Windows:

```
python -m unittest discover .\school_project\test\
```

- Linux:

```
python3 -m unittest discover ./school_project/test/
```

Use Docker with:

- Build the Docker Image with:

```
sudo docker build -t mcttn22/school-project ./
```

- Run the Docker Image with:

```
sudo apt-get install x11-xserver-utils
xhost +
sudo docker run -v /tmp/.X11-unix:/tmp/.X11-unix -e DISPLAY=unix$DISPLAY mcttn22/school-prc
```

Compile Project Report PDF with:

```
make all
```

Note: This requires the Latexmk, pdflatex and Pygments libraries

- Also included was a license file that describes how others can use my code.

4.1.4 Organisation

I also utilised a TODO.md file for keeping track of what features and/or bugs need to be worked on.

4.2 Models package

This package is a self-contained package for creating the trained Artificial Neural Networks and can either be used with a CPU or a GPU, as well as containing the test and training data for all three datasets in a datasets directory. Whilst both the cpu and gpu subpackage are similar in functionality, the cpu subpackage uses NumPy for matrices whereas the gpu subpackage utilises NumPy alongside the library CuPy which requires a GPU to be utilised for operations with the matrices. I have only shown the code for the cpu subpackage - the GPU subpackage is identical apart from calling CuPy instead of NumPy.

Both the cpu and gpu subpackage contain a utils subpackage that provides the tools for creating Artificial Neural Networks, and three modules that are the implementation of Artificial Neural Networks for each dataset.

4.2.1 Utils subpackage

The utils subpackage consists of a tools.py module that provides a ModelInterface class and helper functions for the model.py module, that contains an AbstractModel class that implements every method from the ModelInterface except for the load_dataset method.

- tools.py module:

```

1  """Helper functions and ModelInterface class for model module."""
2
3  from abc import ABC, abstractmethod
4
5  import numpy as np
6
7  class ModelInterface(ABC):
8      """Interface for ANN models."""
9      @abstractmethod
10     def _setup_layers(setup_values: callable) -> None:
11         """Decorator that sets up model layers and sets up values of each
12         ↪ layer
13         with the method given.
14
15         Args:
16             setup_values (callable): the method that sets up the values of
17             ↪ each
18             layer.
19         Raises:
20             NotImplementedError: if this method is not implemented.
21
22         """
23         raise NotImplementedError
24
25     @abstractmethod
26     def create_model_values(self) -> None:
27         """Create weights and bias/biases
28
29         Raises:
30             NotImplementedError: if this method is not implemented.
31
32         """
33         raise NotImplementedError
34
35     @abstractmethod
36     def load_model_values(self, file_location: str) -> None:
37         """Load weights and bias/biases from .npz file.
38
39         Args:
40             file_location (str): the location of the file to load from.
41         Raises:
42             NotImplementedError: if this method is not implemented.
43
44         """
45         raise NotImplementedError
46
47     @abstractmethod
48     def load_datasets(self, train_dataset_size: int) -> tuple[np.ndarray,
49         ↪ np.ndarray,
50         ↪ np.ndarray]:
51         """Load input and output datasets. For the input dataset, each
52         ↪ column
53         should represent a piece of data and each row should store the
54         ↪ values
55         of the piece of data.
56
57         Args:
58             train_dataset_size (int): the number of train dataset inputs to
59             ↪ use.
60         Returns:

```

```

55         tuple of train_inputs, train_outputs,
56         test_inputs and test_outputs.
57     Raises:
58         NotImplementedError: if this method is not implemented.
59
60     """
61     raise NotImplementedError
62
63 @abstractmethod
64 def back_propagation(self, prediction: np.ndarray) -> None:
65     """Adjust the weights and bias/biases via gradient descent.
66
67     Args:
68         prediction (numpy.ndarray): the matrice of prediction values
69     Raises:
70         NotImplementedError: if this method is not implemented.
71
72     """
73     raise NotImplementedError
74
75 @abstractmethod
76 def forward_propagation(self) -> np.ndarray:
77     """Generate a prediction with the weights and bias/biases.
78
79     Returns:
80         numpy.ndarray of prediction values.
81     Raises:
82         NotImplementedError: if this method is not implemented.
83
84     """
85     raise NotImplementedError
86
87 @abstractmethod
88 def test(self) -> None:
89     """Test trained weights and bias/biases.
90
91     Raises:
92         NotImplementedError: if this method is not implemented.
93
94     """
95     raise NotImplementedError
96
97 @abstractmethod
98 def train(self, epochs: int) -> None:
99     """Train weights and bias/biases.
100
101     Args:
102         epochs (int): the number of forward and back propagations to
103             ↪ do.
104     Raises:
105         NotImplementedError: if this method is not implemented.
106
107     """
108     raise NotImplementedError
109
110 @abstractmethod
111 def save_model_values(self, file_location: str) -> None:
112     """Save the model by saving the weights then biases of each layer
113     ↪ to
114         a .npz file with a given file location.
115
116     Args:
117         file_location (str): the file location to save the model to.

```

```

117         """
118         raise NotImplementedError
119
120     def relu(z: np.ndarray | int | float) -> np.ndarray | float:
121         """Transfer function, transform input to max number between 0 and z.
122
123         Args:
124             z (numpy.ndarray | int | float):
125                 the numpy.ndarray | int | float to be transferred.
126         Returns:
127             numpy.ndarray | float,
128             with all values | the value transferred to max number between 0-z.
129         Raises:
130             TypeError: if z is not of type numpy.ndarray | int | float.
131
132         """
133         return np.maximum(0.1*z, 0) # Divide by 10 to stop overflow errors
134
135     def relu_derivative(output: np.ndarray) -> np.ndarray:
136         """Calculate derivative of ReLu Transfer function with respect to z.
137
138         Args:
139             output (numpy.ndarray):
140                 the numpy.ndarray output of the ReLu transfer function.
141         Returns:
142             numpy.ndarray,
143             derivative of the ReLu transfer function with respect to z.
144         Raises:
145             TypeError: if output is not of type numpy.ndarray.
146
147         """
148         output[output <= 0] = 0
149         output[output > 0] = 1
150
151         return output
152
153     def sigmoid(z: np.ndarray | int | float) -> np.ndarray | float:
154         """Transfer function, transform input to number between 0 and 1.
155
156         Args:
157             z (numpy.ndarray | int | float):
158                 the numpy.ndarray | int | float to be transferred.
159         Returns:
160             numpy.ndarray | float,
161             with all values | the value transferred to a number between 0-1.
162         Raises:
163             TypeError: if z is not of type numpy.ndarray | int | float.
164
165         """
166         return 1 / (1 + np.exp(-z))
167
168     def sigmoid_derivative(output: np.ndarray | int | float) -> np.ndarray |
169     ↪ float:
170         """Calculate derivative of sigmoid Transfer function with respect to z.
171
172         Args:
173             output (numpy.ndarray | int | float):
174                 the numpy.ndarray | int | float output of the sigmoid transfer
175                 ↪ function.
176         Returns:
177             numpy.ndarray | float,
178             derivative of the sigmoid transfer function with respect to z.
179         Raises:
180             TypeError: if output is not of type numpy.ndarray | int | float.

```



```

179
180     """
181     return output * (1 - output)
182
183 def calculate_loss(input_count: int,
184                   outputs: np.ndarray,
185                   prediction: np.ndarray) -> float:
186     """Calculate average loss/error of the prediction to the outputs.
187
188     Args:
189         input_count (int): the number of inputs.
190         outputs (np.ndarray):
191             the train/test outputs array to compare with the prediction.
192         prediction (np.ndarray): the array of prediction values.
193     Returns:
194         float loss.
195     Raises:
196         ValueError:
197             if outputs is not a suitable multiplier with the prediction
198             (incorrect shapes)
199
200     """
201     return np.squeeze(- (1/input_count) * np.sum(outputs *
202     ↪ np.log(prediction) + (1 - outputs) * np.log(1 - prediction)))
203
204 def calculate_prediction_accuracy(prediction: np.ndarray,
205                                  outputs: np.ndarray) -> float:
206     """Calculate the percentage accuracy of the predictions.
207
208     Args:
209         prediction (np.ndarray): the array of prediction values.
210         outputs (np.ndarray):
211             the train/test outputs array to compare with the prediction.
212     Returns:
213         float prediction accuracy
214
215     """
216     return 100 - np.mean(np.abs(prediction - outputs)) * 100

```

- model.py module:

```

1     """Provides an abstract class for Artificial Neural Network models."""
2
3     from collections.abc import Generator
4     import time
5
6     import numpy as np
7
8     from .tools import (
9         ModelInterface,
10        relu,
11        relu_derivative,
12        sigmoid,
13        sigmoid_derivative,
14        calculate_loss,
15        calculate_prediction_accuracy
16    )
17
18    class _FullyConnectedLayer():
19        """Fully connected layer for Deep ANNs,
20        represented as a node of a Doubly linked list."""
21        def __init__(self, learning_rate: float, input_neuron_count: int,
22        output_neuron_count: int, transfer_type: str) -> None:

```

```

23         """Initialise layer values.
24
25     Args:
26         learning_rate (float): the learning rate of the model.
27         input_neuron_count (int):
28         the number of input neurons into the layer.
29         output_neuron_count (int):
30         the number of output neurons into the layer.
31         transfer_type (str): the transfer function type
32         ('sigmoid' or 'relu')
33
34     """
35     # Setup layer attributes
36     self.previous_layer = None
37     self.next_layer = None
38     self.input_neuron_count = input_neuron_count
39     self.output_neuron_count = output_neuron_count
40     self.transfer_type = transfer_type
41     self.input: np.ndarray
42     self.output: np.ndarray
43
44     # Setup weights and biases
45     self.weights: np.ndarray
46     self.biases: np.ndarray
47     self.learning_rate = learning_rate
48
49     def __repr__(self) -> str:
50         """Read values of the layer.
51
52     Returns:
53         a string description of the layers's
54         weights, bias and learning rate values.
55
56     """
57     return (f"Weights: {self.weights.tolist()}\n" +
58             f"Biases: {self.biases.tolist()}\n")
59
60     def init_layer_values_random(self) -> None:
61         """Initialise weights to random values and biases to 0s"""
62         np.random.seed(1) # Sets up pseudo random values for layer weight
63         ↪ arrays
64         self.weights = np.random.rand(self.output_neuron_count,
65         ↪ self.input_neuron_count) - 0.5
66         self.biases = np.zeros(shape=(self.output_neuron_count, 1))
67
68     def init_layer_values_zeros(self) -> None:
69         """Initialise weights to 0s and biases to 0s"""
70         self.weights = np.zeros(shape=(self.output_neuron_count,
71         ↪ self.input_neuron_count))
72         self.biases = np.zeros(shape=(self.output_neuron_count, 1))
73
74     def back_propagation(self, dloss_doutput) -> np.ndarray:
75         """Adjust the weights and biases via gradient descent.
76
77     Args:
78         dloss_doutput (numpy.ndarray): the derivative of the loss of
79         ↪ the
80         layer's output, with respect to the layer's output.
81     Returns:
82         a numpy.ndarray derivative of the loss of the layer's input,
83         with respect to the layer's input.
84     Raises:
85         ValueError:
86             if dloss_doutput

```

```

83         is not a suitable multiplier with the weights
84         (incorrect shape)
85
86     """
87     match self.transfer_type:
88         case 'sigmoid':
89             dloss_dz = dloss_doutput *
90                 ↪ sigmoid_derivative(output=self.output)
91         case 'relu':
92             dloss_dz = dloss_doutput *
93                 ↪ relu_derivative(output=self.output)
94
95     dloss_dweights = np.dot(dloss_dz, self.input.T)
96     dloss_dbias = np.sum(dloss_dz)
97
98     assert dloss_dweights.shape == self.weights.shape
99
100     dloss_dinput = np.dot(self.weights.T, dloss_dz)
101
102     # Update weights and biases
103     self.weights -= self.learning_rate * dloss_dweights
104     self.biases -= self.learning_rate * dloss_dbias
105
106     return dloss_dinput
107
108 def forward_propagation(self, inputs) -> np.ndarray:
109     """Generate a layer output with the weights and biases.
110
111     Args:
112     inputs (np.ndarray): the input values to the layer.
113     Returns:
114     a numpy.ndarray of the output values.
115
116     """
117     self.input = inputs
118     z = np.dot(self.weights, self.input) + self.biases
119     if self.transfer_type == 'sigmoid':
120         self.output = sigmoid(z)
121     elif self.transfer_type == 'relu':
122         self.output = relu(z)
123     return self.output
124
125 class _Layers():
126     """Manages linked list of layers."""
127     def __init__(self) -> None:
128         """Initialise linked list."""
129         self.head = None
130         self.tail = None
131
132     def __iter__(self) -> Generator[_FullyConnectedLayer, None, None]:
133         """Iterate forward through the network."""
134         current_layer = self.head
135         while True:
136             yield current_layer
137             if current_layer.next_layer is not None:
138                 current_layer = current_layer.next_layer
139             else:
140                 break
141
142     def __reversed__(self) -> Generator[_FullyConnectedLayer, None, None]:
143         """Iterate back through the network."""
144         current_layer = self.tail
145         while True:
146             yield current_layer

```

```

145         if current_layer.previous_layer is not None:
146             current_layer = current_layer.previous_layer
147         else:
148             break
149
150     class AbstractModel(ModelInterface):
151         """ANN model with variable number of hidden layers"""
152         def __init__(self,
153                     hidden_layers_shape: list[int],
154                     train_dataset_size: int,
155                     learning_rate: float,
156                     use_relu: bool) -> None:
157             """Initialise model values.
158
159             Args:
160                 hidden_layers_shape (list[int]):
161                     list of the number of neurons in each hidden layer.
162                 train_dataset_size (int): the number of train dataset inputs to
163                     ↪ use.
164                 learning_rate (float): the learning rate of the model.
165                 use_relu (bool): True or False whether the ReLu Transfer
166                     ↪ function
167                     should be used.
168
169             """
170             # Setup model data
171             self.train_inputs, self.train_outputs, \
172             self.test_inputs, self.test_outputs = self.load_datasets(
173                 ↪ train_dataset_size=train_dataset_size
174             )
175             self.train_losses: list[float]
176             self.test_prediction: np.ndarray
177             self.test_prediction_accuracy: float
178             self.training_progress = ""
179             self.training_time: float
180
181             # Setup model attributes
182             self._running = True
183             self.input_neuron_count: int = self.train_inputs.shape[0]
184             self.input_count = self.train_inputs.shape[1]
185             self.hidden_layers_shape = hidden_layers_shape
186             self.output_neuron_count = self.train_outputs.shape[0]
187             self.layers_shape = [f'{layer}' for layer in (
188                 [self.input_neuron_count] +
189                 self.hidden_layers_shape +
190                 [self.output_neuron_count]
191             )]
192             self.use_relu = use_relu
193
194             # Setup model values
195             self.layers = _Layers()
196             self.learning_rate = learning_rate
197
198     def __repr__(self) -> str:
199         """Read current state of model.
200
201         Returns:
202             a string description of the model's shape,
203             weights, bias and learning rate values.
204
205         """
206         return (f"Layers Shape: {' '.join(self.layers_shape)}\n" +
207                 f"Learning Rate: {self.learning_rate}")

```

```

206
207 def set_running(self, value: bool) -> None:
208     """Set the running attribute to the given value.
209
210     Args:
211         value (bool): the value to set the running attribute to.
212
213     """
214     self._running = value
215
216 def _setup_layers(setup_values: callable) -> None:
217     """Decorator that sets up model layers and sets up values of each
218     ↪ layer
219     ↪ with the method given.
220
221     Args:
222         setup_values (callable): the method that sets up the values of
223         ↪ each
224         ↪ layer.
225
226     """
227     def decorator(self, *args, **kwargs) -> None:
228         # Check if setting up Deep Network
229         if len(self.hidden_layers_shape) > 0:
230             if self.use_relu:
231
232                 # Add input layer
233                 self.layers.head = _FullyConnectedLayer(
234
235                     ↪ learning_rate=self.learning_rate,
236
237                     ↪ input_neuron_count=self.input_neuron_count,
238
239                     ↪ output_neuron_count=self.hidden_layers_shape[0],
240                     transfer_type='relu'
241                 )
242                 current_layer = self.layers.head
243
244                 # Add hidden layers
245                 for layer in range(len(self.hidden_layers_shape) - 1):
246                     current_layer.next_layer = _FullyConnectedLayer(
247                         learning_rate=self.learning_rate,
248
249                         ↪ input_neuron_count=self.hidden_layers_shape[layer],
250
251                         ↪ output_neuron_count=self.hidden_layers_shape[layer
252                         ↪ + 1],
253                         transfer_type='relu'
254                     )
255                     current_layer.next_layer.previous_layer =
256                     ↪ current_layer
257                     current_layer = current_layer.next_layer
258             else:
259
260                 # Add input layer
261                 self.layers.head = _FullyConnectedLayer(
262
263                     ↪ learning_rate=self.learning_rate,
264
265                     ↪ input_neuron_count=self.input_neuron_count,
266
267                     ↪ output_neuron_count=self.hidden_layers_shape[0],
268                     transfer_type='sigmoid'
269                 )

```

```

258         current_layer = self.layers.head
259
260         # Add hidden layers
261         for layer in range(len(self.hidden_layers_shape) - 1):
262             current_layer.next_layer = _FullyConnectedLayer(
263                 learning_rate=self.learning_rate,
264
265                 ↪ input_neuron_count=self.hidden_layers_shape[layer],
266
267                 ↪ output_neuron_count=self.hidden_layers_shape[layer
268                 ↪ + 1],
269                 transfer_type='sigmoid'
270             )
271             current_layer.next_layer.previous_layer =
272             ↪ current_layer
273             current_layer = current_layer.next_layer
274
275         # Add output layer
276         current_layer.next_layer = _FullyConnectedLayer(
277             learning_rate=self.learning_rate,
278
279             ↪ input_neuron_count=self.hidden_layers_shape[-1],
280
281             ↪ output_neuron_count=self.output_neuron_count,
282             transfer_type='sigmoid'
283         )
284         current_layer.next_layer.previous_layer = current_layer
285         self.layers.tail = current_layer.next_layer
286
287         # Setup Perceptron Network
288         else:
289             self.layers.head = _FullyConnectedLayer(
290                 learning_rate=self.learning_rate,
291
292                 ↪ input_neuron_count=self.input_neuron_count,
293
294                 ↪ output_neuron_count=self.output_neuron_count,
295                 transfer_type='sigmoid'
296             )
297             self.layers.tail = self.layers.head
298
299         setup_values(self, *args, **kwargs)
300
301         return decorator
302
303     @setup_layers
304     def create_model_values(self) -> None:
305         """Create weights and bias/biases"""
306         # Check if setting up Deep Network
307         if len(self.hidden_layers_shape) > 0:
308
309             # Initialise Layer values to random values
310             for layer in self.layers:
311                 layer.init_layer_values_random()
312
313             # Setup Perceptron Network
314             else:
315
316                 # Initialise Layer values to zeros
317                 for layer in self.layers:
318                     layer.init_layer_values_zeros()
319
320     @setup_layers
321     def load_model_values(self, file_location: str) -> None:

```

```

314         """Load weights and bias/biases from .npz file.
315
316     Args:
317         file_location (str): the location of the file to load from.
318
319     """
320     data: dict[str, np.ndarray] = np.load(file=file_location)
321
322     # Initialise Layer values
323     i = 0
324     keys = list(data.keys())
325     for layer in self.layers:
326         layer.weights = data[keys[i]]
327         layer.biases = data[keys[i + 1]]
328         i += 2
329
330     def back_propagation(self, dloss_doutput) -> None:
331         """Train each layer's weights and biases.
332
333         Args:
334             dloss_doutput (np.ndarray): the derivative of the loss of the
335             output layer's output, with respect to the output layer's
336             ↪ output.
337
338         """
339         for layer in reversed(self.layers):
340             dloss_doutput =
341             ↪ layer.back_propagation(dloss_doutput=dloss_doutput)
342
343     def forward_propagation(self) -> np.ndarray:
344         """Generate a prediction with the layers.
345
346         Returns:
347             a numpy.ndarray of the prediction values.
348
349         """
350         output = self.train_inputs
351         for layer in self.layers:
352             output = layer.forward_propagation(inputs=output)
353         return output
354
355     def test(self) -> None:
356         """Test the layers' trained weights and biases."""
357         output = self.test_inputs
358         for layer in self.layers:
359             output = layer.forward_propagation(inputs=output)
360         self.test_prediction = output
361
362         # Calculate performance of model
363         self.test_prediction_accuracy = calculate_prediction_accuracy(
364             ↪ prediction=self.test_prediction,
365             outputs=self.test_outputs
366         )
367
368     def train(self, epoch_count: int) -> None:
369         """Train layers' weights and biases.
370
371         Args:
372             epoch_count (int): the number of training epochs.
373
374         """
375         self.layers_shape = [f'{layer}' for layer in (
376             [self.input_neuron_count] +

```

```

375         self.hidden_layers_shape +
376         [self.output_neuron_count]
377     ])
378     self.train_losses = []
379     training_start_time = time.time()
380     for epoch in range(epoch_count):
381         if not self.__running:
382             break
383         self.training_progress = f"Epoch {epoch} / {epoch_count}"
384         prediction = self.forward_propagation()
385         loss = calculate_loss(input_count=self.input_count,
386                             outputs=self.train_outputs,
387                             prediction=prediction)
388         self.train_losses.append(loss)
389         if not self.__running:
390             break
391         dloss_doutput = -(1/self.input_count) * ((self.train_outputs -
392             ↪ prediction)/(prediction * (1 - prediction)))
393         self.back_propagation(dloss_doutput=dloss_doutput)
394         self.training_time = round(number=time.time() -
395             ↪ training_start_time,
396                                   ndigits=2)
397
398     def save_model_values(self, file_location: str) -> None:
399         """Save the model by saving the weights then biases of each layer
400         ↪ to
401         a .npz file with a given file location.
402
403         Args:
404             file_location (str): the file location to save the model to.
405
406         """
407         saved_model: list[np.ndarray] = []
408         for layer in self.layers:
409             saved_model.append(layer.weights)
410             saved_model.append(layer.biases)
411         np.savez(file_location, *saved_model)

```

4.2.2 Artificial Neural Network implementations

The following three modules implement the AbstractModel class from the above model.py module from the utils subpackage, on the three datasets.

- cat_recognition.py module:

```

1  """Implementation of Artificial Neural Network model on Cat Recognition
2  ↪ dataset."""
3
4  import h5py
5  import numpy as np
6
7  from .utils.model import AbstractModel
8
9  class CatRecognitionModel(AbstractModel):
10     """ANN model that trains to predict if an image is a cat or not a
11     ↪ cat."""
12     def __init__(self,
13                 hidden_layers_shape: list[int],
14                 train_dataset_size: int,
15                 learning_rate: float,
16                 use_relu: bool) -> None:

```



```

15         """Initialise Model's Base class.
16
17     Args:
18         hidden_layers_shape (list[int]):
19             list of the number of neurons in each hidden layer.
20         train_dataset_size (int): the number of train dataset inputs to
21             ↪ use.
22         learning_rate (float): the learning rate of the model.
23         use_relu (bool): True or False whether the ReLu Transfer
24             ↪ function
25             should be used.
26
27     """
28     super().__init__(hidden_layers_shape=hidden_layers_shape,
29                     train_dataset_size=train_dataset_size,
30                     learning_rate=learning_rate,
31                     use_relu=use_relu)
32
33     def load_datasets(self, train_dataset_size: int) -> tuple[np.ndarray,
34         ↪ np.ndarray,
35                                     np.ndarray,
36         ↪ np.ndarray]:
37
38         """Load image input and output datasets.
39
40     Args:
41         train_dataset_size (int): the number of train dataset inputs to
42             ↪ use.
43
44     Returns:
45         tuple of image train_inputs, train_outputs,
46         test_inputs and test_outputs numpy.ndarrays.
47
48     Raises:
49         FileNotFoundError: if file does not exist.
50
51     """
52     # Load datasets from h5 files
53     # (h5 files stores large amount of data with quick access)
54     train_dataset: h5py.File = h5py.File(
55         r'school_project/models/datasets/train-cat.h5',
56         'r'
57     )
58     test_dataset: h5py.File = h5py.File(
59         r'school_project/models/datasets/test-cat.h5',
60         'r'
61     )
62
63     # Load input arrays,
64     # containing the RGB values for each pixel in each 64x64 pixel
65     ↪ image,
66     # for 209 images
67     train_inputs: np.ndarray =
68     ↪ np.array(train_dataset['train_set_x'][:])
69     test_inputs: np.ndarray = np.array(test_dataset['test_set_x'][:])
70
71     # Load output arrays of 1s for cat and 0s for not cat
72     train_outputs: np.ndarray =
73     ↪ np.array(train_dataset['train_set_y'][:])
74     test_outputs: np.ndarray = np.array(test_dataset['test_set_y'][:])
75
76     # Reshape input arrays into 1 dimension (flatten),
77     # then divide by 255 (RGB)
78     # to standardize them to a number between 0 and 1
79     train_inputs = train_inputs.reshape((train_inputs.shape[0],
80         ↪ -1)).T / 255

```

```

71     test_inputs = test_inputs.reshape((test_inputs.shape[0], -1)).T /
    ↪ 255
72
73     # Reshape output arrays into a 1 dimensional list of outputs
74     train_outputs = train_outputs.reshape((1, train_outputs.shape[0]))
75     test_outputs = test_outputs.reshape((1, test_outputs.shape[0]))
76
77     # Reduce train datasets' sizes to train_dataset_size
78     train_inputs = (train_inputs.T[:train_dataset_size]).T
79     train_outputs = (train_outputs.T[:train_dataset_size]).T
80
81     return train_inputs, train_outputs, test_inputs, test_outputs

```

- mnist.py module:

```

1  """Implementation of Artificial Neural Network model on MNIST dataset."""
2
3  import pickle
4  import gzip
5
6  import numpy as np
7
8  from .utils.model import AbstractModel
9
10 class MNISTModel(AbstractModel):
11     """ANN model that trains to predict Numbers from images."""
12     def __init__(self, hidden_layers_shape: list[int],
13                 train_dataset_size: int,
14                 learning_rate: float,
15                 use_relu: bool) -> None:
16         """Initialise Model's Base class.
17
18     Args:
19         hidden_layers_shape (list[int]):
20             list of the number of neurons in each hidden layer.
21         train_dataset_size (int): the number of train dataset inputs to
22             ↪ use.
23         learning_rate (float): the learning rate of the model.
24         use_relu (bool): True or False whether the ReLu Transfer
25             ↪ function
26             should be used.
27
28     """
29     super().__init__(hidden_layers_shape=hidden_layers_shape,
30                     train_dataset_size=train_dataset_size,
31                     learning_rate=learning_rate,
32                     use_relu=use_relu)
33
34     def load_datasets(self, train_dataset_size: int) -> tuple[np.ndarray,
35     ↪ np.ndarray,
36                                     np.ndarray,
37                                     ↪ np.ndarray]:
38
39         """Load image input and output datasets.
40
41     Args:
42         train_dataset_size (int): the number of dataset inputs to use.
43
44     Returns:
45         tuple of image train_inputs, train_outputs,
46         test_inputs and test_outputs numpy.ndarrays.
47
48     Raises:
49         FileNotFoundError: if file does not exist.
50
51     """

```

```

45     # Load datasets from pkl.gz file
46     with gzip.open(
47         'school_project/models/datasets/mnist.pkl.gz',
48         'rb'
49     ) as mnist:
50         (train_inputs, train_outputs),\
51         (test_inputs, test_outputs) = pickle.load(mnist,
52             ↪ encoding='bytes')
53
54     # Reshape input arrays into 1 dimension (flatten),
55     # then divide by 255 (RGB)
56     # to standardize them to a number between 0 and 1
57     train_inputs =
58     ↪ np.array(train_inputs.reshape((train_inputs.shape[0],
59         -1)).T / 255)
60     test_inputs = np.array(test_inputs.reshape(test_inputs.shape[0],
61         ↪ -1).T / 255)
62
63     # Represent number values
64     # with a one at the matching index of an array of zeros
65     train_outputs = np.eye(np.max(train_outputs) + 1)[train_outputs].T
66     test_outputs = np.eye(np.max(test_outputs) + 1)[test_outputs].T
67
68     # Reduce train datasets' sizes to train_dataset_size
69     train_inputs = (train_inputs.T[:train_dataset_size]).T
70     train_outputs = (train_outputs.T[:train_dataset_size]).T
71
72     return train_inputs, train_outputs, test_inputs, test_outputs

```

- xor.py module

```

1  """Implementation of Artificial Neural Network model on XOR dataset."""
2
3  import numpy as np
4
5  from .utils.model import AbstractModel
6
7  class XORModel(AbstractModel):
8      """ANN model that trains to predict the output of a XOR gate with two
9      inputs."""
10     def __init__(self,
11         hidden_layers_shape: list[int],
12         train_dataset_size: int,
13         learning_rate: float,
14         use_relu: bool) -> None:
15         """Initialise Model's Base class.
16
17     Args:
18         hidden_layers_shape (list[int]):
19             list of the number of neurons in each hidden layer.
20         train_dataset_size (int): the number of train dataset inputs to
21             ↪ use.
22         learning_rate (float): the learning rate of the model.
23         use_relu (bool): True or False whether the ReLu Transfer
24             ↪ function
25             should be used.
26
27     """
28     super().__init__(hidden_layers_shape=hidden_layers_shape,
29         train_dataset_size=train_dataset_size,
30         learning_rate=learning_rate,
31         use_relu=use_relu)

```

```

31     def load_datasets(self, train_dataset_size: int) -> tuple[np.ndarray,
↪ np.ndarray,
32                                     np.ndarray,
↪ np.ndarray]:
33         """Load XOR input and output datasets.
34
35         Args:
36             train_dataset_size (int): the number of dataset inputs to use.
37         Returns:
38             tuple of XOR train_inputs, train_outputs,
39             test_inputs and test_outputs numpy.ndarrays.
40
41         """
42         inputs: np.ndarray = np.array([[0, 0, 1, 1],
43                                       [0, 1, 0, 1]])
44         outputs: np.ndarray = np.array([[0, 1, 1, 0]])
45
46         # Reduce train datasets' sizes to train_dataset_size
47         inputs = (inputs.T[:train_dataset_size]).T
48         outputs = (outputs.T[:train_dataset_size]).T
49
50         return inputs, outputs, inputs, outputs

```

4.3 Frames package

I have used tkinter for the User Interface and the frames package which consists of tkinter frames to be loaded onto the main window when needed. The package also includes a hyper-parameter-defaults.json file, which stores optimum default values for the hyper-parameters to be set to.

- hyper-parameter-defaults.json file contents:

```

1  {
2      "MNIST": {
3          "description": "An Image model trained on recognising numbers from
↪ images.",
4          "epochCount": 150,
5          "hiddenLayersShape": [1000, 1000],
6          "minTrainDatasetSize": 1,
7          "maxTrainDatasetSize": 60000,
8          "maxLearningRate": 1
9      },
10     "Cat Recognition": {
11         "description": "An Image model trained on recognising if an image
↪ is a cat or not.",
12         "epochCount": 3500,
13         "hiddenLayersShape": [100, 100],
14         "minTrainDatasetSize": 1,
15         "maxTrainDatasetSize": 209,
16         "maxLearningRate": 0.3
17     },
18     "XOR": {
19         "description": "For experimenting with Artificial Neural Networks,
↪ a XOR gate model has been used for its lesser computation
↪ time.",
20         "epochCount": 4700,
21         "hiddenLayersShape": [100, 100],
22         "minTrainDatasetSize": 2,
23         "maxTrainDatasetSize": 4,
24         "maxLearningRate": 1

```

```

25     }
26 }

```

- create_model.py module:

```

1  """Tkinter frames for creating an Artificial Neural Network model."""
2
3  import json
4  import threading
5  import tkinter as tk
6  import tkinter.font as tkf
7
8  from matplotlib.figure import Figure
9  from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
10 import numpy as np
11
12 class HyperParameterFrame(tk.Frame):
13     """Frame for hyper-parameter page."""
14     def __init__(self, root: tk.Tk, width: int,
15                 height: int, bg: str, dataset: str) -> None:
16         """Initialise hyper-parameter frame widgets.
17
18         Args:
19             root (tk.Tk): the widget object that contains this widget.
20             width (int): the pixel width of the frame.
21             height (int): the pixel height of the frame.
22             bg (str): the hex value or name of the frame's background
23             ↪ colour.
24             dataset (str): the name of the dataset to use
25             ('MNIST', 'Cat Recognition' or 'XOR')
26         Raises:
27             TypeError: if root, width or height are not of the correct
28             ↪ type.
29
30         """
31         super().__init__(master=root, width=width, height=height, bg=bg)
32         self.root = root
33         self.WIDTH = width
34         self.HEIGHT = height
35         self.BG = bg
36
37         # Setup hyper-parameter frame variables
38         self.dataset = dataset
39         self.use_gpu: bool
40         self.default_hyper_parameters = self.load_default_hyper_parameters(
41
42                                     ↪ dataset=dataset
43                                     )
44
45         # Setup widgets
46         self.title_label = tk.Label(master=self,
47                                     bg=self.BG,
48                                     font=('Arial', 20),
49                                     text=dataset)
50
51         self.about_label = tk.Label(
52             master=self,
53             bg=self.BG,
54             font=('Arial', 14),
55
56             ↪ text=self.default_hyper_parameters['description']
57             )
58         self.learning_rate_scale = tk.Scale(
59             master=self,

```

```

55         bg=self.BG,
56         orient='horizontal',
57         label="Learning Rate",
58         length=185,
59         from_=0,
60
61         ⇨ to=self.default_hyper_parameters['maxLearningRate'],
62         resolution=0.01
63     )
64     self.learning_rate_scale.set(value=0.1)
65     self.epoch_count_scale = tk.Scale(master=self,
66                                       bg=self.BG,
67                                       orient='horizontal',
68                                       label="Epoch Count",
69                                       length=185,
70                                       from_=0,
71                                       to=10_000,
72                                       resolution=100)
73
74     ⇨ value=self.default_hyper_parameters['epochCount']
75     )
76     self.train_dataset_size_scale = tk.Scale(
77         master=self,
78         bg=self.BG,
79         orient='horizontal',
80         label="Train Dataset Size",
81         length=185,
82
83         ⇨ from_=self.default_hyper_parameters['minTrainDatasetSize'],
84         to=self.default_hyper_parameters['maxTrainDatasetSize'],
85         resolution=1
86     )
87     self.train_dataset_size_scale.set(
88         ⇨ value=self.default_hyper_parameters['maxTrainDatasetSize']
89     )
90     self.hidden_layers_shape_label = tk.Label(
91         master=self,
92         bg=self.BG,
93         font=('Arial', 12),
94         text="Enter the number of neurons in
95         ⇨ each\n" +
96         "hidden layer, separated by
97         ⇨ commas:"
98     )
99     self.hidden_layers_shape_entry = tk.Entry(master=self)
100     self.hidden_layers_shape_entry.insert(0, ",".join(
101         f"{neuron_count}" for neuron_count in
102         ⇨ self.default_hyper_parameters['hiddenLayersShape']
103     ))
104     self.use_relu_check_button_var = tk.BooleanVar(value=True)
105     self.use_relu_check_button = tk.Checkbutton(
106         master=self,
107         width=13, height=1,
108         font=tkf.Font(size=12),
109         text="Use ReLu",
110
111         ⇨ variable=self.use_relu_check_button_var
112     )
113     self.use_gpu_check_button_var = tk.BooleanVar()
114     self.use_gpu_check_button = tk.Checkbutton(
115         master=self,
116         width=13, height=1,

```

```

111         font=tkf.Font(size=12),
112         text="Use GPU",
113
114         ↪ variable=self.use_gpu_check_button_var
115     )
116     self.model_status_label = tk.Label(master=self,
117         bg=self.BG,
118         font=('Arial', 15))
119
120     # Pack widgets
121     self.title_label.grid(row=0, column=0, columnspan=3)
122     self.about_label.grid(row=1, column=0, columnspan=3)
123     self.learning_rate_scale.grid(row=2, column=0, pady=(50,0))
124     self.epoch_count_scale.grid(row=3, column=0, pady=(30,0))
125     self.train_dataset_size_scale.grid(row=4, column=0, pady=(30,0))
126     self.hidden_layers_shape_label.grid(row=2, column=1,
127         padx=30, pady=(50,0))
128     self.hidden_layers_shape_entry.grid(row=3, column=1, padx=30)
129     self.use_relu_check_button.grid(row=2, column=2, pady=(30, 0))
130     self.use_gpu_check_button.grid(row=3, column=2, pady=(30, 0))
131     self.model_status_label.grid(row=5, column=0,
132         columnspan=3, pady=50)
133
134     def load_default_hyper_parameters(self, dataset: str) -> dict[
135         str,
136         str | int | list[int] |
137         ↪ float
138     ]:
139         """Load the dataset's default hyper-parameters from the json file.
140
141         Args:
142             dataset (str): the name of the dataset to load
143             ↪ hyper-parameters
144             for. ('MNIST', 'Cat Recognition' or 'XOR')
145         Returns:
146             a dictionary of default hyper-parameter values.
147         """
148         with open('school_project/frames/hyper-parameter-defaults.json') as
149             ↪ f:
150             return json.load(f)[dataset]
151
152     def create_model(self) -> object:
153         """Create and return a Model using the hyper-parameters set.
154
155         Returns:
156             a Model object.
157         """
158         self.use_gpu = self.use_gpu_check_button_var.get()
159
160         # Validate hidden layers shape input
161         hidden_layers_shape_input = [layer for layer in
162             ↪ self.hidden_layers_shape_entry.get().replace(' ',
163             ↪ '').split(',')]]
164         for layer in hidden_layers_shape_input:
165             if not layer.isdigit():
166                 self.model_status_label.configure(
167                     text="Invalid hidden layers shape",
168                     fg='red'
169                 )
170
171                 raise ValueError
172
173         # Create Model
174         if not self.use_gpu:
175             if self.dataset == "MNIST":

```

```

169         from school_project.models.cpu.mnist import MNISTModel as
170             ↪ Model
171     elif self.dataset == "Cat Recognition":
172         from school_project.models.cpu.cat_recognition import
173             ↪ CatRecognitionModel as Model
174     elif self.dataset == "XOR":
175         from school_project.models.cpu.xor import XORModel as Model
176     model = Model(
177         hidden_layers_shape = [int(neuron_count) for neuron_count
178             ↪ in hidden_layers_shape_input],
179         train_dataset_size = self.train_dataset_size_scale.get(),
180         learning_rate = self.learning_rate_scale.get(),
181         use_relu = self.use_relu_check_button_var.get()
182     )
183     model.create_model_values()
184
185 else:
186     try:
187         if self.dataset == "MNIST":
188             from school_project.models.gpu.mnist import MNISTModel
189             ↪ as Model
190         elif self.dataset == "Cat Recognition":
191             from school_project.models.gpu.cat_recognition import
192             ↪ CatRecognitionModel as Model
193         elif self.dataset == "XOR":
194             from school_project.models.gpu.xor import XORModel as
195             ↪ Model
196     model = Model(hidden_layers_shape = [int(neuron_count) for
197         ↪ neuron_count in hidden_layers_shape_input],
198         train_dataset_size =
199         ↪ self.train_dataset_size_scale.get(),
200         learning_rate =
201         ↪ self.learning_rate_scale.get(),
202         use_relu =
203         ↪ self.use_relu_check_button_var.get())
204     model.create_model_values()
205 except ImportError as ie:
206     self.model_status_label.configure(
207         text="Failed to initialise GPU",
208         fg='red'
209     )
210     raise ImportError
211 return model
212
213 class TrainingFrame(tk.Frame):
214     """Frame for training page."""
215     def __init__(self, root: tk.Tk, width: int,
216         height: int, bg: str,
217         model: object, epoch_count: int) -> None:
218         """Initialise training frame widgets.
219
220         Args:
221             root (tk.Tk): the widget object that contains this widget.
222             width (int): the pixel width of the frame.
223             height (int): the pixel height of the frame.
224             bg (str): the hex value or name of the frame's background
225             ↪ colour.
226             model (object): the Model object to be trained.
227             epoch_count (int): the number of training epochs.
228
229         Raises:
230             TypeError: if root, width or height are not of the correct
231             ↪ type.
232         """

```



```

221     super().__init__(master=root, width=width, height=height, bg=bg)
222     self.root = root
223     self.WIDTH = width
224     self.HEIGHT = height
225     self.BG = bg
226
227     # Setup widgets
228     self.model_status_label = tk.Label(master=self,
229                                         bg=self.BG,
230                                         font=('Arial', 15))
231     self.training_progress_label = tk.Label(master=self,
232                                              bg=self.BG,
233                                              font=('Arial', 15))
234     self.loss_figure: Figure = Figure()
235     self.loss_canvas: FigureCanvasTkAgg = FigureCanvasTkAgg(
236
237         ↪ figure=self.loss_figure,
238         master=self
239     )
240
241     # Pack widgets
242     self.model_status_label.pack(pady=(30,0))
243     self.training_progress_label.pack(pady=30)
244
245     # Start training thread
246     self.model_status_label.configure(
247         text="Training weights and
248         ↪ biases...",
249         fg='red'
250     )
251     self.train_thread: threading.Thread = threading.Thread(
252
253         ↪ target=model.train,
254         ↪ args=(epoch_count,)
255     )
256     self.train_thread.start()
257
258     def plot_losses(self, model: object) -> None:
259         """Plot losses of Model training.
260
261         Args:
262             model (object): the Model object thats been trained.
263
264         """
265         self.model_status_label.configure(
266             text=f"Weights and biases trained in
267             ↪ {model.training_time}s",
268             fg='green'
269         )
270         graph: Figure.axes = self.loss_figure.add_subplot(111)
271         graph.set_title("Learning rate: " +
272                         f"{model.learning_rate}")
273         graph.set_xlabel("Epochs")
274         graph.set_ylabel("Loss Value")
275         graph.plot(np.squeeze(model.train_losses))
276         self.loss_canvas.get_tk_widget().pack()

```

This outputs the following for the hyper-parameter frame shown in figure 17:

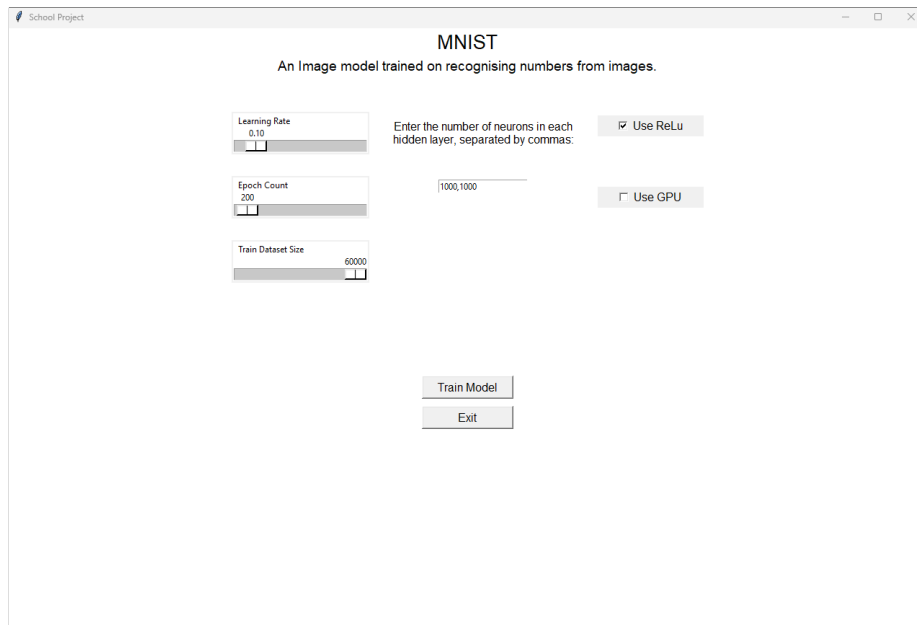


Figure 17: Hyper parameter frame - showing MNIST parameters

And outputs the following for the training frame, shown in figure 18, during training:

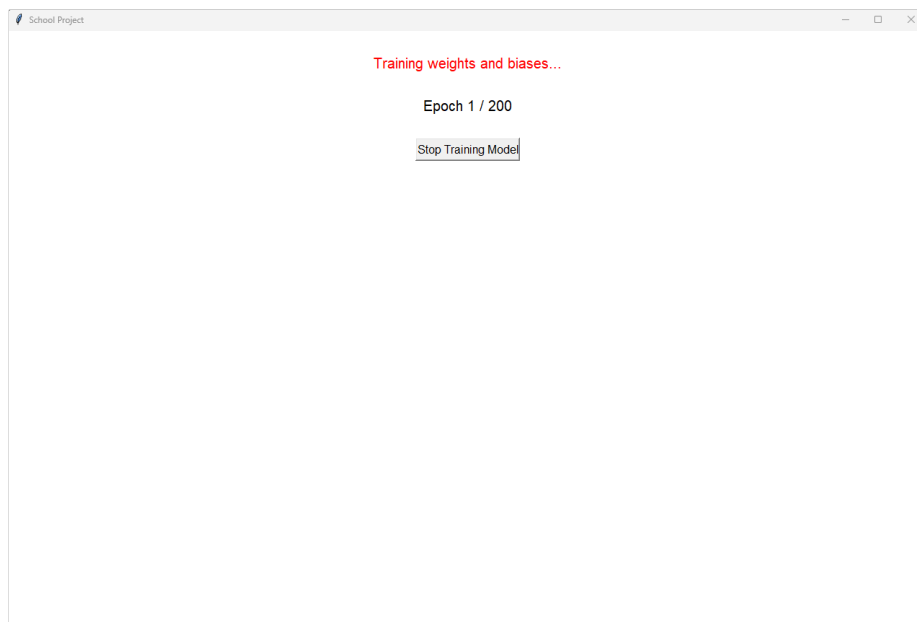


Figure 18: Training frame showing epoch count

And outputs the following for the training frame once training has completed as shown in figure 19:

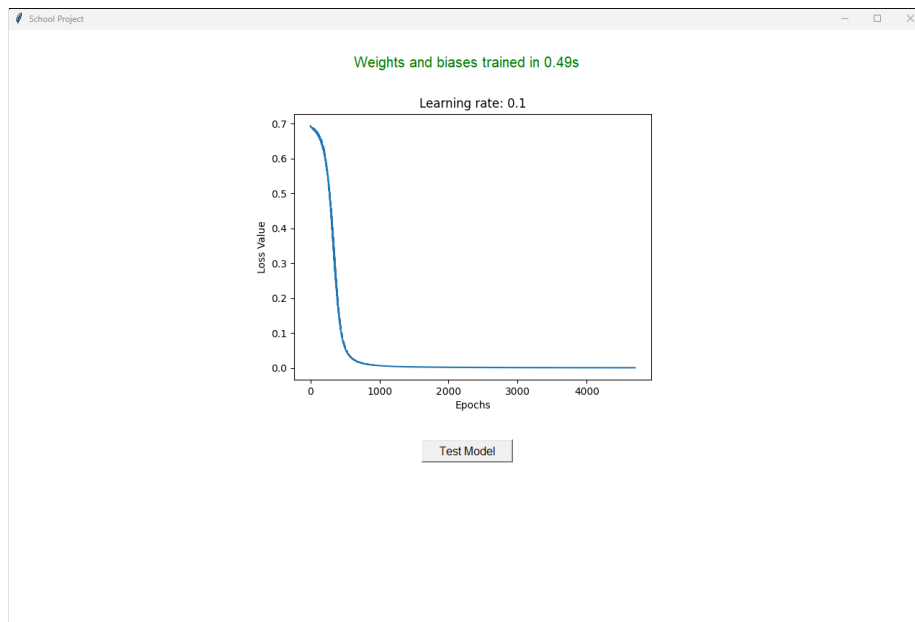


Figure 19: Training frame showing loss value against epochs

- load_model.py module:

```

1  """Tkinter frames for loading a saved Artificial Neural Network Model."""
2
3  import sqlite3
4  import tkinter as tk
5  import tkinter.font as tkf
6
7  class LoadModelFrame(tk.Frame):
8      """Frame for load model page."""
9      def __init__(self, root: tk.Tk,
10                  width: int, height: int,
11                  bg: str, connection: sqlite3.Connection,
12                  cursor: sqlite3.Cursor, dataset: str) -> None:
13          """Initialise load model frame widgets."""
14
15      Args:
16          root (tk.Tk): the widget object that contains this widget.
17          width (int): the pixel width of the frame.
18          height (int): the pixel height of the frame.
19          bg (str): the hex value or name of the frame's background
20                  ↪ colour.
21          connection (sqlite3.Connection): the database connection
22                  ↪ object.
23          cursor (sqlite3.Cursor): the database cursor object.
24          dataset (str): the name of the dataset to use
25                        ('MNIST', 'Cat Recognition' or 'XOR')
26
27      Raises:

```

```

25         TypeError: if root, width or height are not of the correct
26         ↪ type.
27
28     """
29     super().__init__(master=root, width=width, height=height, bg=bg)
30     self.root = root
31     self.WIDTH = width
32     self.HEIGHT = height
33     self.BG = bg
34
35     # Setup load model frame variables
36     self.connection = connection
37     self.cursor = cursor
38     self.dataset = dataset
39     self.use_gpu: bool
40     self.model_options = self.load_model_options()
41
42     # Setup widgets
43     self.title_label = tk.Label(master=self,
44                                bg=self.BG,
45                                font=('Arial', 20),
46                                text=dataset)
47     self.about_label = tk.Label(
48         master=self,
49         bg=self.BG,
50         font=('Arial', 14),
51         text=f"Load a pretrained model for the {dataset}"
52         ↪ dataset."
53     )
54     self.model_status_label = tk.Label(master=self,
55                                        bg=self.BG,
56                                        font=('Arial', 15))
57
58     # Don't give loaded model options if no models have been saved for
59     ↪ the
60     # dataset.
61     if len(self.model_options) > 0:
62         self.model_option_menu_label = tk.Label(
63             master=self,
64             bg=self.BG,
65             font=('Arial', 14),
66             text="Select a model to
67             ↪ load or delete:"
68         )
69         self.model_option_menu_var = tk.StringVar(
70             master=self,
71             ↪ value=self.model_options[0]
72         )
73         self.model_option_menu = tk.OptionMenu(
74             self,
75             ↪ self.model_option_menu_var,
76             *self.model_options
77         )
78         self.use_gpu_check_button_var = tk.BooleanVar()
79         self.use_gpu_check_button = tk.Checkbutton(
80             master=self,
81             width=7, height=1,
82             font=tkf.Font(size=12),
83             text="Use GPU",
84             ↪ variable=self.use_gpu_check_button_var
85         )

```

```

82     else:
83         self.model_status_label.configure(
84             text='No saved models for this
85                 ↪ dataset.',
86                 fg='red'
87             )
88
89     # Pack widgets
90     self.title_label.grid(row=0, column=0, columnspan=3)
91     self.about_label.grid(row=1, column=0, columnspan=3)
92     if len(self.model_options) > 0: # Check if options should be given
93         self.model_option_menu_label.grid(row=2, column=0, padx=(0,30),
94             ↪ pady=(30,0))
95         self.use_gpu_check_button.grid(row=2, column=2, rowspan=2,
96             ↪ pady=(30,0))
97         self.model_option_menu.grid(row=3, column=0, padx=(0,30),
98             ↪ pady=(10,0))
99     self.model_status_label.grid(row=4, column=0,
100         columnspan=3, pady=50)
101
102 def load_model_options(self) -> list[str]:
103     """Load the model options from the database.
104
105     Returns:
106         a list of the model options.
107     """
108     sql = f"""
109     SELECT Name FROM Models WHERE Dataset=?
110     """
111     parameters = (self.dataset.replace(" ", "_"),)
112     self.cursor.execute(sql, parameters)
113
114     # Save the string value contained within the tuple of each row
115     model_options = []
116     for model_option in self.cursor.fetchall():
117         model_options.append(model_option[0])
118
119     return model_options
120
121 def load_model(self) -> object:
122     """Create model using saved weights and biases.
123
124     Returns:
125         a Model object.
126     """
127     self.use_gpu = self.use_gpu_check_button_var.get()
128
129     # Query data of selected saved model from database
130     sql = """
131     SELECT * FROM Models WHERE Dataset=? AND Name=?
132     """
133     parameters = (self.dataset.replace(" ", "_"),
134         ↪ self.model_option_menu_var.get())
135     self.cursor.execute(sql, parameters)
136     data = self.cursor.fetchone()
137     hidden_layers_shape_input = [layer for layer in data[3].replace('
138         ↪ ', '').split(',') if layer != '']
139
140     # Create Model
141     if not self.use_gpu:
142         if self.dataset == "MNIST":
143             from school_project.models.cpu.mnist import MNISTModel as
144             ↪ Model

```

```

139         elif self.dataset == "Cat Recognition":
140             from school_project.models.cpu.cat_recognition import
141                 ↪ CatRecognitionModel as Model
142         elif self.dataset == "XOR":
143             from school_project.models.cpu.xor import XORModel as Model
144         model = Model(
145             hidden_layers_shape=[int(neuron_count) for neuron_count in
146                 ↪ hidden_layers_shape_input],
147             train_dataset_size=data[6],
148             learning_rate=data[4],
149             use_relu=data[7]
150         )
151         model.load_model_values(file_location=data[2])
152     else:
153         try:
154             if self.dataset == "MNIST":
155                 from school_project.models.gpu.mnist import MNISTModel
156                 ↪ as Model
157             elif self.dataset == "Cat Recognition":
158                 from school_project.models.gpu.cat_recognition import
159                 ↪ CatRecognitionModel as Model
160             elif self.dataset == "XOR":
161                 from school_project.models.gpu.xor import XORModel as
162                 ↪ Model
163             model = Model(
164                 hidden_layers_shape=[int(neuron_count) for neuron_count
165                 ↪ in hidden_layers_shape_input],
166                 train_dataset_size=data[6],
167                 learning_rate=data[4],
168                 use_relu=data[7]
169             )
170             model.load_model_values(file_location=data[2])
171         except ImportError as ie:
172             self.model_status_label.configure(
173                 text="Failed to initialise
174                 ↪ GPU",
175                 fg='red'
176             )
177             raise ImportError
178     return model

```

This outputs the following for the load model frame when models have been saved for the dataset as shown in figure 20:

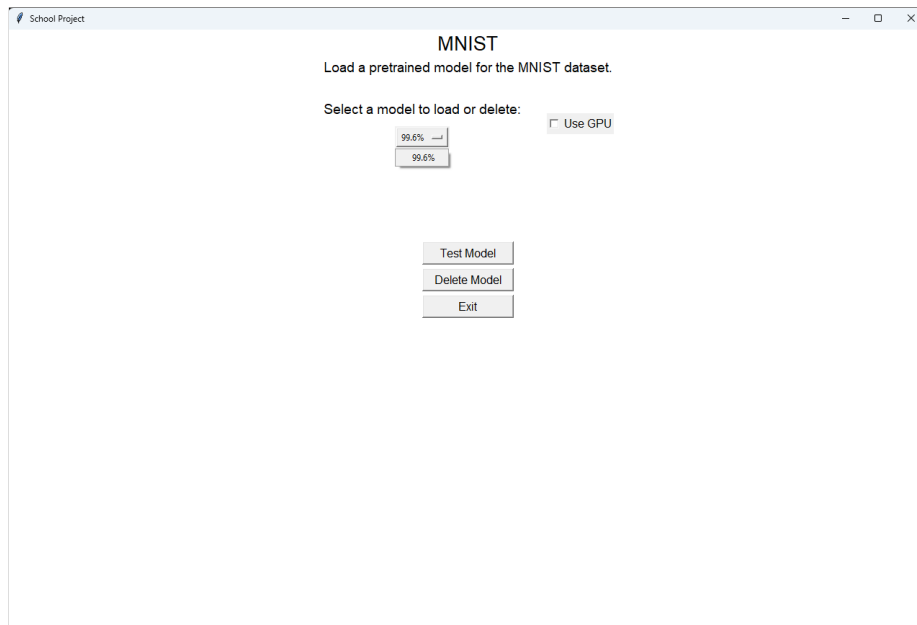


Figure 20: Load model frame

And outputs the following for the load model frame when no models have been saved for the dataset as shown in figure 21:

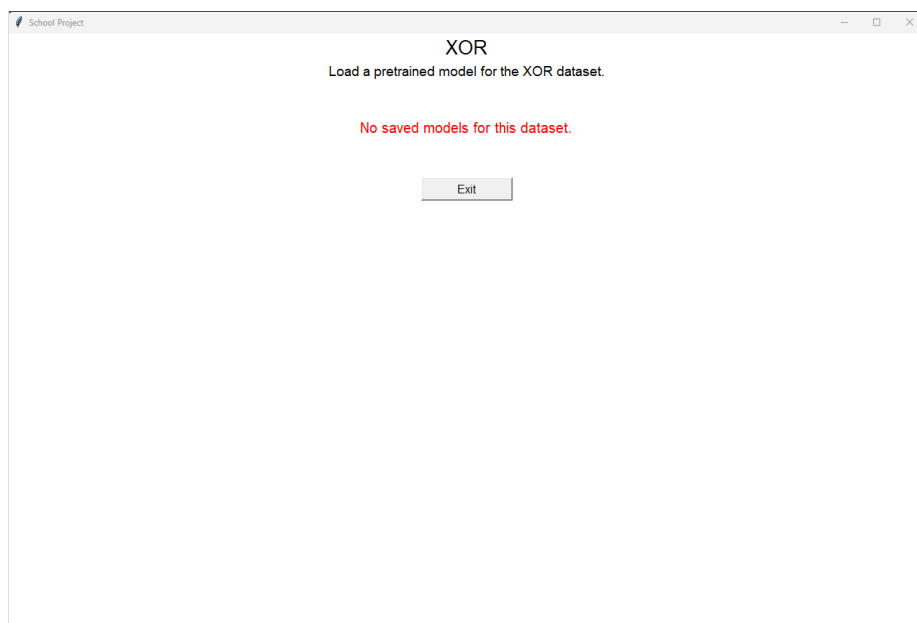


Figure 21: Load model frame showing error condition for an attempted load of a non-existent model

4.4 Project Entrypoint - __main__.py module

This module is the entrypoint to the project and loads the main window of the User Interface:

```
1  """The entrypoint of A-level Computer Science NEA Programming Project."""
2
3  import os
4  import sqlite3
5  import threading
6  import tkinter as tk
7  import tkinter.font as tkf
8  import uuid
9
10 from school_project.frames import (HyperParameterFrame, TrainingFrame,
11                                   LoadModelFrame, TestMNISTFrame,
12                                   TestCatRecognitionFrame, TestXORFrame)
13
14 class SchoolProjectFrame(tk.Frame):
15     """Main frame of school project."""
16     def __init__(self, root: tk.Tk, width: int, height: int, bg: str) -> None:
17         """Initialise school project pages.
18
19         Args:
20             root (tk.Tk): the widget object that contains this widget.
21             width (int): the pixel width of the frame.
22             height (int): the pixel height of the frame.
23             bg (str): the hex value or name of the frame's background colour.
24
25         Raises:
26             TypeError: if root, width or height are not of the correct type.
27
28         """
29         super().__init__(master=root, width=width, height=height, bg=bg)
30         self.root = root.title("School Project")
31         self.WIDTH = width
32         self.HEIGHT = height
33         self.BG = bg
34
35         # Setup school project frame variables
36         self.hyper_parameter_frame: HyperParameterFrame
37         self.training_frame: TrainingFrame
38         self.load_model_frame: LoadModelFrame
39         self.test_frame: TestMNISTFrame | TestCatRecognitionFrame | TestXORFrame
40         self.connection, self.cursor = self.setup_database()
41         self.model = None
42
43         # Record if the model should be saved after testing,
44         # as only newly created models should be given the option to be saved.
45         self.saving_model: bool
46
47         # Setup school project frame widgets
48         self.exit_hyper_parameter_frame_button = tk.Button(
49             master=self,
50             width=13,
51             height=1,
52             font=tkf.Font(size=12),
53             text="Exit",
54             command=self.exit_hyper_parameter_frame
55         )
56         self.exit_load_model_frame_button = tk.Button(
57             master=self,
58             width=13,
59             height=1,
```



```

59         font=tkf.Font(size=12),
60         text="Exit",
61         command=self.exit_load_model_frame
62     )
63     self.train_button = tk.Button(master=self,
64                                   width=13,
65                                   height=1,
66                                   font=tkf.Font(size=12),
67                                   text="Train Model",
68                                   command=self.enter_training_frame)
69     self.stop_training_button = tk.Button(
70         master=self,
71         width=15, height=1,
72         font=tkf.Font(size=12),
73         text="Stop Training Model",
74         command=lambda: self.model.set_running(
75             value=False
76         )
77     )
78     self.test_created_model_button = tk.Button(
79         master=self,
80         width=13, height=1,
81         font=tkf.Font(size=12),
82         text="Test Model",
83         command=self.test_created_model
84     )
85     self.test_loaded_model_button = tk.Button(
86         master=self,
87         width=13, height=1,
88         font=tkf.Font(size=12),
89         text="Test Model",
90         command=self.test_loaded_model
91     )
92     self.delete_loaded_model_button = tk.Button(
93         master=self,
94         width=13, height=1,
95         font=tkf.Font(size=12),
96         text="Delete Model",
97         command=self.delete_loaded_model
98     )
99     self.save_model_label = tk.Label(
100         master=self,
101         text="Enter a name for your trained model:",
102         bg=self.BG,
103         font=('Arial', 15)
104     )
105     self.save_model_name_entry = tk.Entry(master=self, width=13)
106     self.save_model_button = tk.Button(master=self,
107                                       width=13,
108                                       height=1,
109                                       font=tkf.Font(size=12),
110                                       text="Save Model",
111                                       command=self.save_model)
112     self.exit_button = tk.Button(master=self,
113                                  width=13, height=1,
114                                  font=tkf.Font(size=12),
115                                  text="Exit",
116                                  command=self.enter_home_frame)
117
118     # Setup home frame
119     self.home_frame = tk.Frame(master=self,
120                                width=self.WIDTH,
121                                height=self.HEIGHT,
122                                bg=self.BG)

```

```

123     self.title_label = tk.Label(
124         master=self.home_frame,
125         bg=self.BG,
126         font=('Arial', 20),
127         text="A-level Computer Science NEA Programming Project"
128     )
129     self.about_label = tk.Label(
130         master=self.home_frame,
131         bg=self.BG,
132         font=('Arial', 14),
133         text="An investigation into how Artificial Neural Networks work, " +
134             "the effects of their hyper-parameters and their applications " +
135             "in Image Recognition.\n\n" +
136             " - Max Cotton"
137     )
138     self.model_menu_label = tk.Label(master=self.home_frame,
139                                     bg=self.BG,
140                                     font=('Arial', 14),
141                                     text="Create a new model " +
142                                     "or load a pre-trained model "
143                                     "for one of the following datasets:")
144     self.dataset_option_menu_var = tk.StringVar(master=self.home_frame,
145                                                value="MNIST")
146     self.dataset_option_menu = tk.OptionMenu(self.home_frame,
147                                              self.dataset_option_menu_var,
148                                              "MNIST",
149                                              "Cat Recognition",
150                                              "XOR")
151     self.create_model_button = tk.Button(
152         master=self.home_frame,
153         width=13, height=1,
154         font=tkf.Font(size=12),
155         text="Create Model",
156         command=self.enter_hyper_parameter_frame
157     )
158     self.load_model_button = tk.Button(master=self.home_frame,
159                                       width=13, height=1,
160                                       font=tkf.Font(size=12),
161                                       text="Load Model",
162                                       command=self.enter_load_model_frame)
163
164     # Grid home frame widgets
165     self.title_label.grid(row=0, column=0, columnspan=4, pady=(10,0))
166     self.about_label.grid(row=1, column=0, columnspan=4, pady=(10,50))
167     self.model_menu_label.grid(row=2, column=0, columnspan=4)
168     self.dataset_option_menu.grid(row=3, column=0, columnspan=4, pady=30)
169     self.create_model_button.grid(row=4, column=1)
170     self.load_model_button.grid(row=4, column=2)
171
172     self.home_frame.pack()
173
174     # Setup frame attributes
175     self.grid_propagate(flag=False)
176     self.pack_propagate(flag=False)
177
178     @staticmethod
179     def setup_database() -> tuple[sqlite3.Connection, sqlite3.Cursor]:
180         """Create a connection to the pretrained_models database file and
181         setup base table if needed.
182
183         Returns:
184             a tuple of the database connection and the cursor for it.
185
186         """

```

```

187         connection = sqlite3.connect(
188             database='school_project/saved_models.db'
189         )
190         cursor = connection.cursor()
191         cursor.execute("""
192             CREATE TABLE IF NOT EXISTS Models
193             (Model_ID INTEGER PRIMARY KEY,
194             Dataset TEXT NOT NULL,
195             File_Location TEXT NOT NULL,
196             Hidden_Layers_Shape TEXT NOT NULL,
197             Learning_Rate FLOAT NOT NULL,
198             Name TEXT NOT NULL,
199             Train_Dataset_Size INTEGER NOT NULL,
200             Use_ReLu INTEGER NOT NULL,
201             UNIQUE (Dataset, Name))
202         """)
203         return (connection, cursor)
204
205     def enter_hyper_parameter_frame(self) -> None:
206         """Unpack home frame and pack hyper-parameter frame."""
207         self.home_frame.pack_forget()
208         self.hyper_parameter_frame = HyperParameterFrame(
209             root=self,
210             width=self.WIDTH,
211             height=self.HEIGHT,
212             bg=self.BG,
213             dataset=self.dataset_option_menu_var.get()
214         )
215         self.hyper_parameter_frame.pack()
216         self.train_button.pack()
217         self.exit_hyper_parameter_frame_button.pack(pady=(10,0))
218
219     def enter_load_model_frame(self) -> None:
220         """Unpack home frame and pack load model frame."""
221         self.home_frame.pack_forget()
222         self.load_model_frame = LoadModelFrame(
223             root=self,
224             width=self.WIDTH,
225             height=self.HEIGHT,
226             bg=self.BG,
227             connection=self.connection,
228             cursor=self.cursor,
229             dataset=self.dataset_option_menu_var.get()
230         )
231         self.load_model_frame.pack()
232
233         # Don't give option to test loaded model if no models have been saved
234         # for the dataset.
235         if len(self.load_model_frame.model_options) > 0:
236             self.test_loaded_model_button.pack()
237             self.delete_loaded_model_button.pack(pady=(5,0))
238
239         self.exit_load_model_frame_button.pack(pady=(5,0))
240
241     def exit_hyper_parameter_frame(self) -> None:
242         """Unpack hyper-parameter frame and pack home frame."""
243         self.hyper_parameter_frame.pack_forget()
244         self.train_button.pack_forget()
245         self.exit_hyper_parameter_frame_button.pack_forget()
246         self.home_frame.pack()
247
248     def exit_load_model_frame(self) -> None:
249         """Unpack load model frame and pack home frame."""
250         self.load_model_frame.pack_forget()

```

```

251     self.test_loaded_model_button.pack_forget()
252     self.delete_loaded_model_button.pack_forget()
253     self.exit_load_model_frame_button.pack_forget()
254     self.home_frame.pack()
255
256 def enter_training_frame(self) -> None:
257     """Load untrained model from hyper parameter frame,
258         unpack hyper-parameter frame, pack training frame
259         and begin managing the training thread.
260     """
261     try:
262         self.model = self.hyper_parameter_frame.create_model()
263     except (ValueError, ImportError) as e:
264         return
265     self.hyper_parameter_frame.pack_forget()
266     self.train_button.pack_forget()
267     self.exit_hyper_parameter_frame_button.pack_forget()
268     self.training_frame = TrainingFrame(
269         root=self,
270         width=self.WIDTH,
271         height=self.HEIGHT,
272         bg=self.BG,
273         model=self.model,
274         epoch_count=self.hyper_parameter_frame.epoch_count_scale.get()
275     )
276     self.training_frame.pack()
277     self.stop_training_button.pack()
278     self.manage_training(train_thread=self.training_frame.train_thread)
279
280 def manage_training(self, train_thread: threading.Thread) -> None:
281     """Wait for model training thread to finish,
282         then plot training losses on training frame.
283
284     Args:
285         train_thread (threading.Thread):
286             the thread running the model's train() method.
287     Raises:
288         TypeError: if train_thread is not of type threading.Thread.
289
290     """
291     if not train_thread.is_alive():
292         self.training_frame.training_progress_label.pack_forget()
293         self.training_frame.plot_losses(model=self.model)
294         self.stop_training_button.pack_forget()
295         self.test_created_model_button.pack(pady=(30,0))
296     else:
297         self.training_frame.training_progress_label.configure(
298             text=self.model.training_progress
299         )
300         self.after(100, self.manage_training, train_thread)
301
302 def test_created_model(self) -> None:
303     """Unpack training frame, pack test frame for the dataset
304         and begin managing the test thread."""
305     self.saving_model = True
306     self.training_frame.pack_forget()
307     self.test_created_model_button.pack_forget()
308     if self.hyper_parameter_frame.dataset == "MNIST":
309         self.test_frame = TestMNISTFrame(
310             root=self,
311             width=self.WIDTH,
312             height=self.HEIGHT,
313             bg=self.BG,
314             use_gpu=self.hyper_parameter_frame.use_gpu,

```

```

315         model=self.model
316     )
317     elif self.hyper_parameter_frame.dataset == "Cat Recognition":
318         self.test_frame = TestCatRecognitionFrame(
319             root=self,
320             width=self.WIDTH,
321             height=self.HEIGHT,
322             bg=self.BG,
323             use_gpu=self.hyper_parameter_frame.use_gpu,
324             model=self.model
325         )
326     elif self.hyper_parameter_frame.dataset == "XOR":
327         self.test_frame = TestXORFrame(root=self,
328                                         width=self.WIDTH,
329                                         height=self.HEIGHT,
330                                         bg=self.BG,
331                                         model=self.model)
332
333     self.test_frame.pack()
334     self.manage_testing(test_thread=self.test_frame.test_thread)
335
336 def test_loaded_model(self) -> None:
337     """Load saved model from load model frame, unpack load model frame,
338     pack test frame for the dataset and begin managing the test thread."""
339     self.saving_model = False
340     try:
341         self.model = self.load_model_frame.load_model()
342     except (ValueError, ImportError) as e:
343         return
344     self.load_model_frame.pack_forget()
345     self.test_loaded_model_button.pack_forget()
346     self.delete_loaded_model_button.pack_forget()
347     self.exit_load_model_frame_button.pack_forget()
348     if self.load_model_frame.dataset == "MNIST":
349         self.test_frame = TestMNISTFrame(
350             root=self,
351             width=self.WIDTH,
352             height=self.HEIGHT,
353             bg=self.BG,
354             use_gpu=self.load_model_frame.use_gpu,
355             model=self.model
356         )
357     elif self.load_model_frame.dataset == "Cat Recognition":
358         self.test_frame = TestCatRecognitionFrame(
359             root=self,
360             width=self.WIDTH,
361             height=self.HEIGHT,
362             bg=self.BG,
363             use_gpu=self.load_model_frame.use_gpu,
364             model=self.model
365         )
366     elif self.load_model_frame.dataset == "XOR":
367         self.test_frame = TestXORFrame(root=self,
368                                         width=self.WIDTH,
369                                         height=self.HEIGHT,
370                                         bg=self.BG,
371                                         model=self.model)
372
373     self.test_frame.pack()
374     self.manage_testing(test_thread=self.test_frame.test_thread)
375
376 def manage_testing(self, test_thread: threading.Thread) -> None:
377     """Wait for model test thread to finish,
378     then plot results on test frame.
379     Args:

```

```

379         test_thread (threading.Thread):
380             the thread running the model's predict() method.
381     Raises:
382         TypeError: if test_thread is not of type threading.Thread.
383
384     """
385     if not test_thread.is_alive():
386         self.test_frame.plot_results(model=self.model)
387         if self.saving_model:
388             self.save_model_label.pack(pady=(30,0))
389             self.save_model_name_entry.pack(pady=10)
390             self.save_model_button.pack()
391             self.exit_button.pack(pady=(20,0))
392         else:
393             self.after(1_000, self.manage_testing, test_thread)
394
395     def save_model(self) -> None:
396         """Save the model, save the model information to the database, then
397         enter the home frame."""
398         model_name = self.save_model_name_entry.get().strip()
399
400         # Check if model name is empty
401         if len(model_name) == 0:
402             self.test_frame.model_status_label.configure(
403                 text="Model name can not be blank",
404                 fg='red'
405             )
406             return
407
408         # Check if model contains double spaces or greater
409         elif ' ' in model_name:
410             self.test_frame.model_status_label.configure(
411                 text="Only single spaces are allowed",
412                 fg='red'
413             )
414             return
415
416         # Check if model name has already been taken
417         dataset = self.dataset_option_menu_var.get().replace(" ", "_")
418         sql = """
419         SELECT Name FROM Models WHERE Dataset=?
420         """
421         parameters = (dataset,)
422         self.cursor.execute(sql, parameters)
423         for saved_model_name in self.cursor.fetchall():
424             if saved_model_name[0] == model_name:
425                 self.test_frame.model_status_label.configure(
426                     text="Model name taken",
427                     fg='red'
428                 )
429                 return
430
431         # Save model to random hex file name
432         file_location = f"school_project/saved-models/{uuid.uuid4().hex}.npz"
433         self.model.save_model_values(file_location=file_location)
434
435         # Save the model information to the database
436         sql = """
437         INSERT INTO Models
438         (Dataset, File_Location, Hidden_Layers_Shape, Learning_Rate, Name,
439         ↪ Train_Dataset_Size, Use_ReLu)
440         VALUES (?, ?, ?, ?, ?, ?, ?)
441         """

```

```

442     parameters = (
443         dataset,
444         file_location,
445         self.hyper_parameter_frame.hidden_layers_shape_entry.get(),
446         self.hyper_parameter_frame.learning_rate_scale.get(),
447         model_name,
448         self.hyper_parameter_frame.train_dataset_size_scale.get(),
449         self.hyper_parameter_frame.use_relu_check_button_var.get()
450     )
451     self.cursor.execute(sql, parameters)
452     self.connection.commit()
453
454     self.enter_home_frame()
455
456 def delete_loaded_model(self) -> None:
457     """Delete saved model file and model data from the database."""
458     dataset = self.dataset_option_menu_var.get().replace(" ", "_")
459     model_name = self.load_model_frame.model_option_menu_var.get()
460
461     # Delete saved model
462     sql = f"SELECT File_Location FROM Models WHERE Dataset=? AND Name=?"
463     parameters = (dataset, model_name)
464     self.cursor.execute(sql, parameters)
465     os.remove(self.cursor.fetchone()[0])
466
467     # Remove model data from database
468     sql = "DELETE FROM Models WHERE Dataset=? AND Name=?"
469     parameters = (dataset, model_name)
470     self.cursor.execute(sql, parameters)
471     self.connection.commit()
472
473     # Reload load model frame with new options
474     self.exit_load_model_frame()
475     self.enter_load_model_frame()
476
477 def enter_home_frame(self) -> None:
478     """Unpack test frame and pack home frame."""
479     self.model = None # Free up trained Model from memory
480     self.test_frame.pack_forget()
481     if self.saving_model:
482         self.save_model_label.pack_forget()
483         self.save_model_name_entry.delete(0, tk.END) # Clear entry's text
484         self.save_model_name_entry.pack_forget()
485         self.save_model_button.pack_forget()
486     self.exit_button.pack_forget()
487     self.home_frame.pack()
488
489 def main() -> None:
490     """Entrypoint of project."""
491     root = tk.Tk()
492     school_project_frame = SchoolProjectFrame(root=root, width=1280,
493                                             height=835, bg='white')
494     school_project_frame.pack(side='top', fill='both', expand=True)
495     root.mainloop()
496
497     # Stop model training when GUI closes
498     if school_project_frame.model is not None:
499         school_project_frame.model.set_running(value=False)
500
501 if __name__ == "__main__":
502     main()

```

Which outputs the following for the home frame:

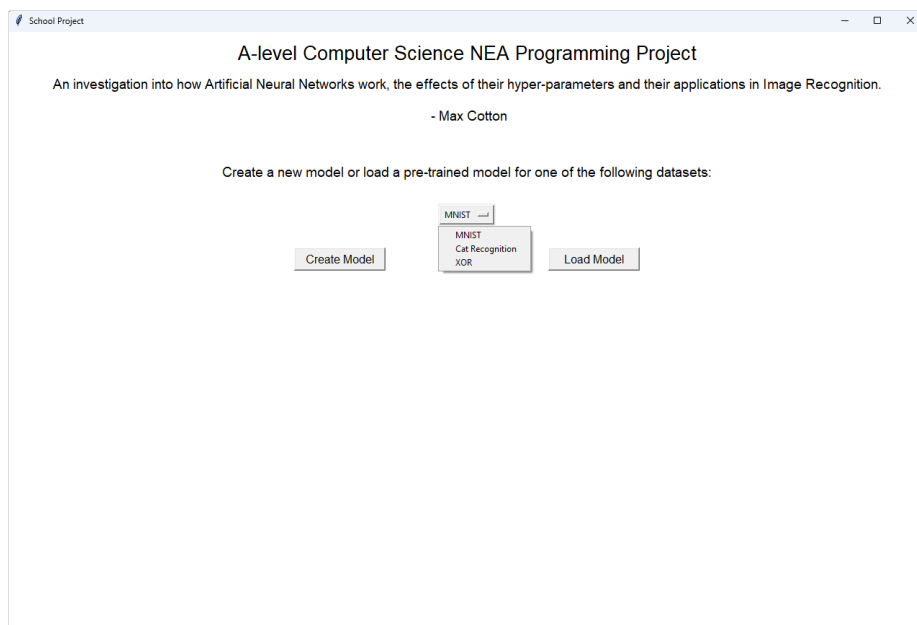


Figure 22: Home frame - the entry point to the program

5 Testing

Testing on the project source code consists of manual testing, where the inputs to frames were tested to ensure correct behaviour and error handling, and automated testing through the use of unit tests that run upon every commit to GitHub.

5.1 Summary of tests

Test Type	Description	Result
Manual	Hyper Parameter Frame - Use GPU Validation	Pass
Manual	Hyper Parameter Frame - Non-Numeric Hidden Layers Shape Validation	Pass
Manual	Hyper Parameter Frame - Negative Hidden Layers Shape Validation	Pass
Manual	Hyper Parameter Frame - Invalid Delimiter Hidden Layers Shape Validation	Pass
Manual	Load Model Frame - Use GPU Validation	Pass
Manual	Test Frames - Taken Trained Model Name Validation	Pass
Manual	Test Frames - Empty Trained Model Name Validation	Pass
Manual	Test Frames - Invalid Delimiter Trained Model Name Validation	Pass
Unit	test_database.py - test_database_structure	Pass
Unit	test_database.py - test_not_null_constraint	Pass
Unit	test_database.py - test_unique_constraint	Pass
Unit	test_database.py - test_save_load_consistency	Pass
Unit	test_model.py - test_train_dataset_size	Pass
Unit	test_model.py - test_network_shape	Pass
Unit	test_model.py - test_learning_rates	Pass
Unit	test_model.py - test_relu_model_transfer_types	Pass
Unit	test_model.py - test_sigmoid_model_transfer_types	Pass
Unit	test_model.py - test_weight_matrice_shapes	Pass
Unit	test_model.py - test_biase_matrice_shapes	Pass
Unit	test_model.py - test_layer_output_shapes	Pass
Unit	test_model.py - test_save_model	Pass
Unit	test_tools.py - test_relu	Pass
Unit	test_tools.py - test_sigmoid	Pass

5.2 Manual Testing - Input Validation Testing

The following tests check the input validation of each frames' inputs.

5.2.1 Hyper Parameter Frame

- Use GPU Validation:

Description	Select Use GPU checkbox without a GPU present.
Expected Result	The exception should be handled and a useful error message should be displayed.
Actual Result	Expected Result
Test Status	Pass

Evidence:

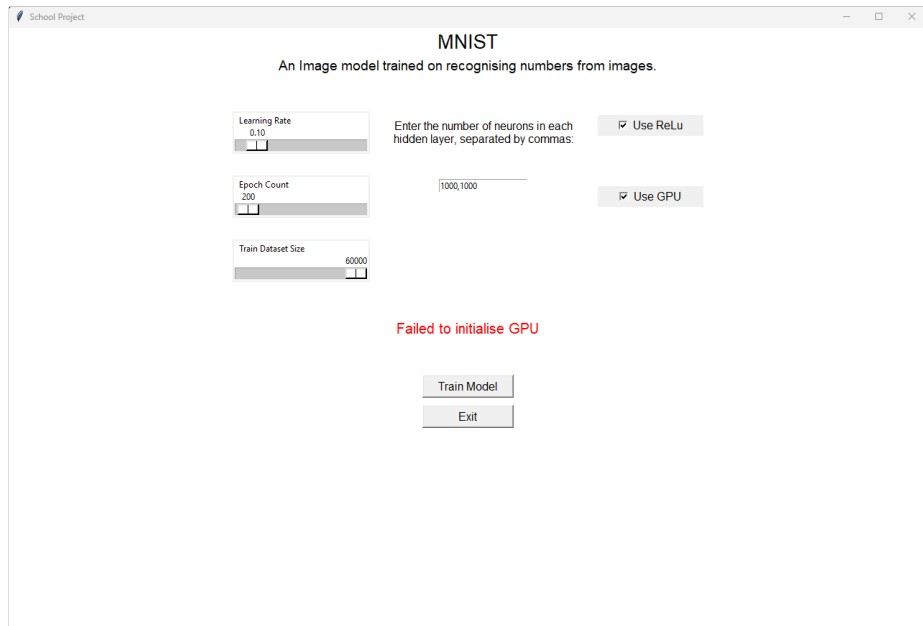


Figure 23: Use GPU Validation evidence

Link to video evidence: <https://github.com/mcttn22/school-project/blob/main/project-report/testing-videos.md/#use-gpu-validation>

• Non-Numeric Hidden Layers Shape Validation:

Description	Enter a non-numeric hidden layers shape.
Data Value	"test"
Data Type	Erroneous
Expected Result	The exception should be handled and a useful error message should be displayed.
Actual Result	Expected Result
Test Status	Pass

Evidence:

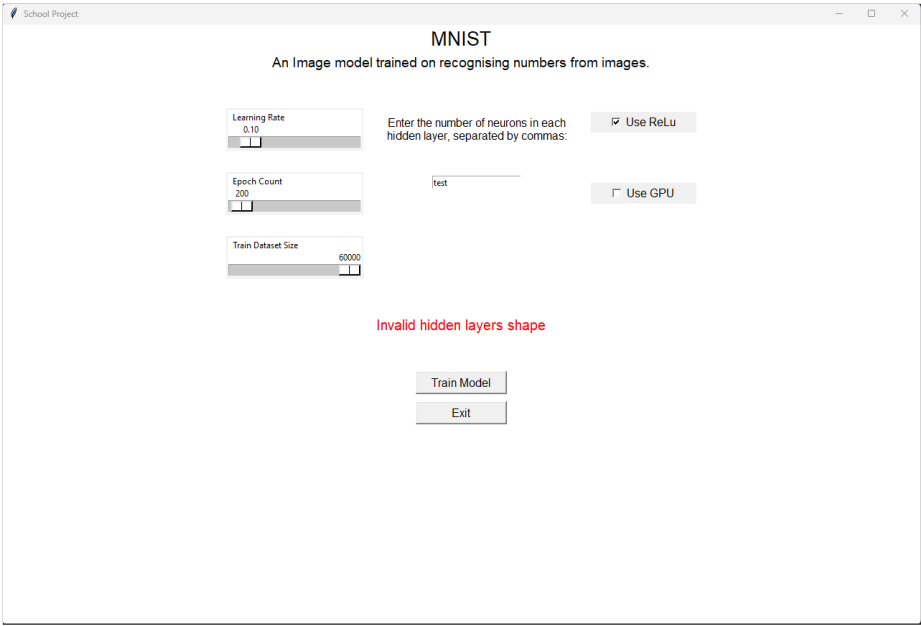


Figure 24: Non-Numeric Hidden Layers Shape Validation evidence

Link to video evidence: <https://github.com/mcttn22/school-project/blob/main/project-report/testing-videos.md/#non-numeric-hidden-layers-shape-validation>

- Negative Hidden Layers Shape Validation:

Description	Enter a negative hidden layers shape.
Data Value	"-100"
Data Type	Erroneous
Expected Result	The exception should be handled and a useful error message should be displayed.
Actual Result	Expected Result
Test Status	Pass

Evidence:

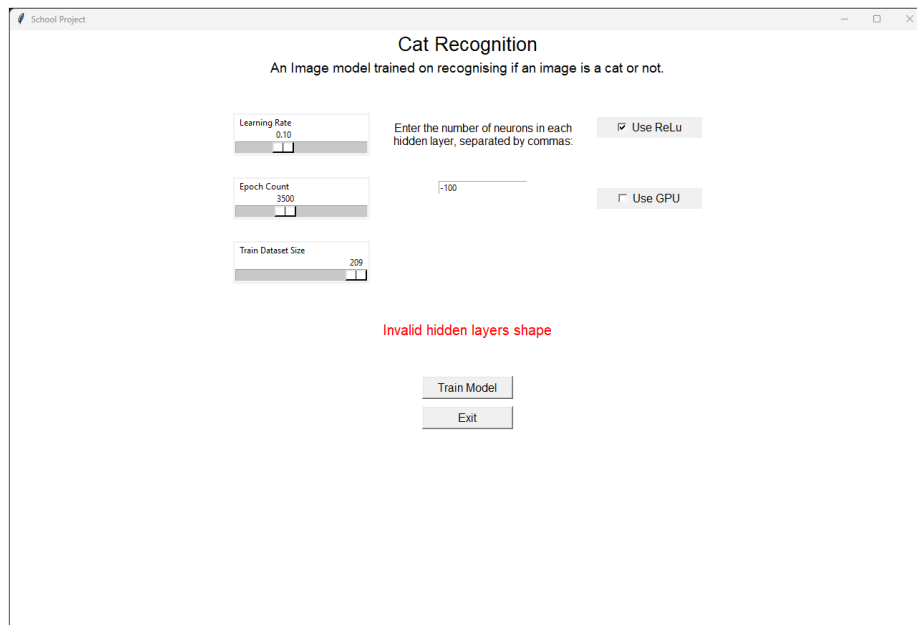


Figure 25: Negative Hidden Layers Shape Validation evidence

Link to video evidence: <https://github.com/mcttn22/school-project/blob/main/project-report/testing-videos.md/#negative-hidden-layers-shape-validation>

- Invalid Delimiter Hidden Layers Shape Validation:

Description	Enter a hidden layers shape with invalid delimiters.
Data Value	"100,,100"
Data Type	Erroneous
Expected Result	The exception should be handled and a useful error message should be displayed.
Actual Result	Expected Result
Test Status	Pass

Evidence:

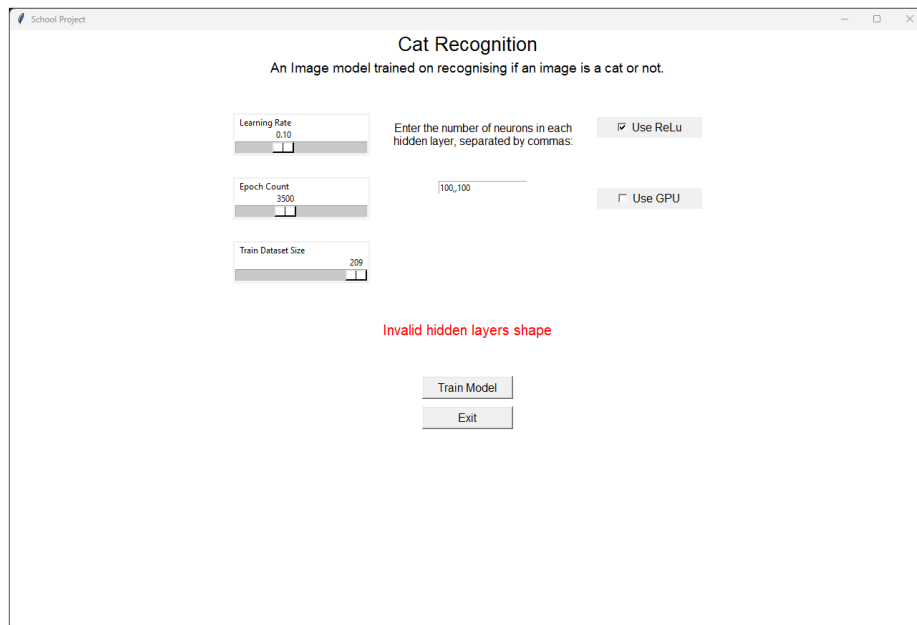


Figure 26: Invalid Delimiter Hidden Layers Shape Validation evidence

Link to video evidence: <https://github.com/mcttn22/school-project/blob/main/project-report/testing-videos.md/#invalid-delimiter-hidden-layers-shape-validation>

5.2.2 Load Model Frame

- Use GPU Validation:

Description	Select Use GPU checkbox without a GPU present.
Expected Result	The exception should be handled and a useful error message should be displayed.
Actual Result	Expected Result
Test Status	Pass

Evidence:

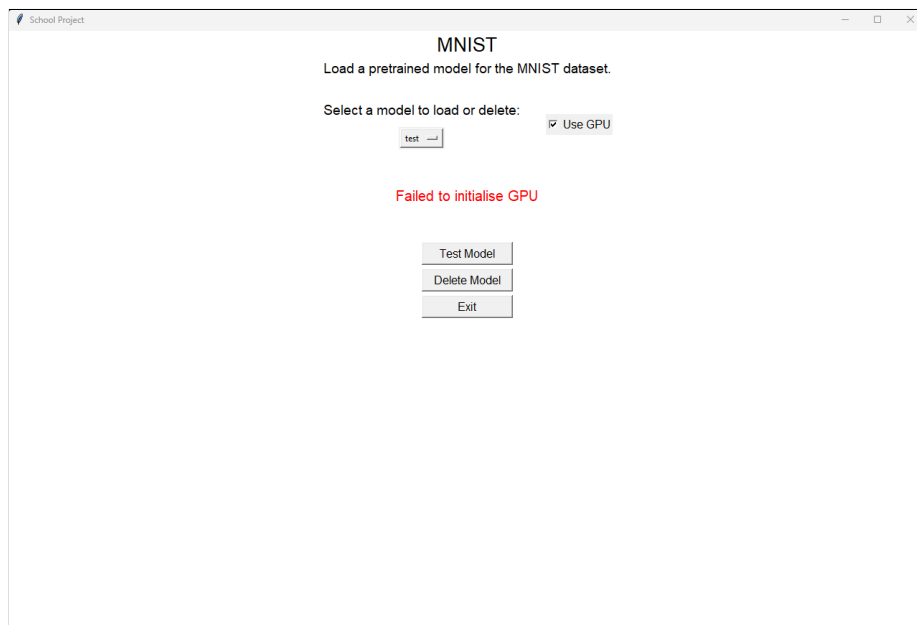


Figure 27: Use GPU Validation evidence

Link to video evidence: <https://github.com/mcttn22/school-project/blob/main/project-report/testing-videos.md/#use-gpu-validation-1>

5.2.3 Test Frames

- Taken Trained Model Name Validation:

Description	Try to save a trained model with an already taken name.
Data Value	"test"
Data Type	Erroneous
Expected Result	The exception should be handled and a useful error message should be displayed.
Actual Result	Expected Result
Test Status	Pass

Evidence:

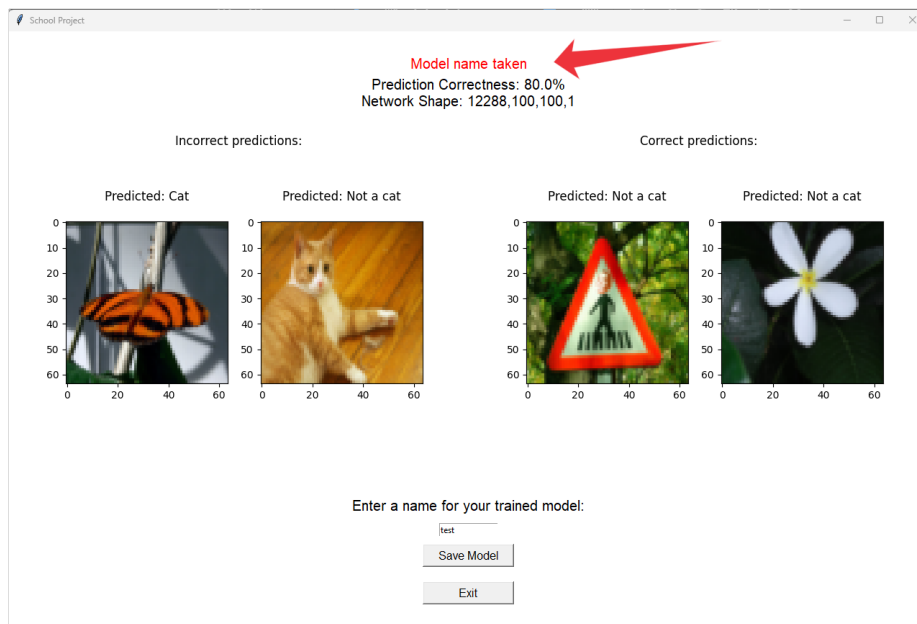


Figure 28: Taken Trained Model Name Validation evidence

Link to video evidence: <https://github.com/mcttn22/school-project/blob/main/project-report/testing-videos.md/#taken-trained-model-name-validation>

- Empty Trained Model Name Validation:

Description	Try to save a trained model with blank name.
Data Value	""
Expected Result	The exception should be handled and a useful error message should be displayed.
Actual Result	Expected Result
Test Status	Pass

Evidence:

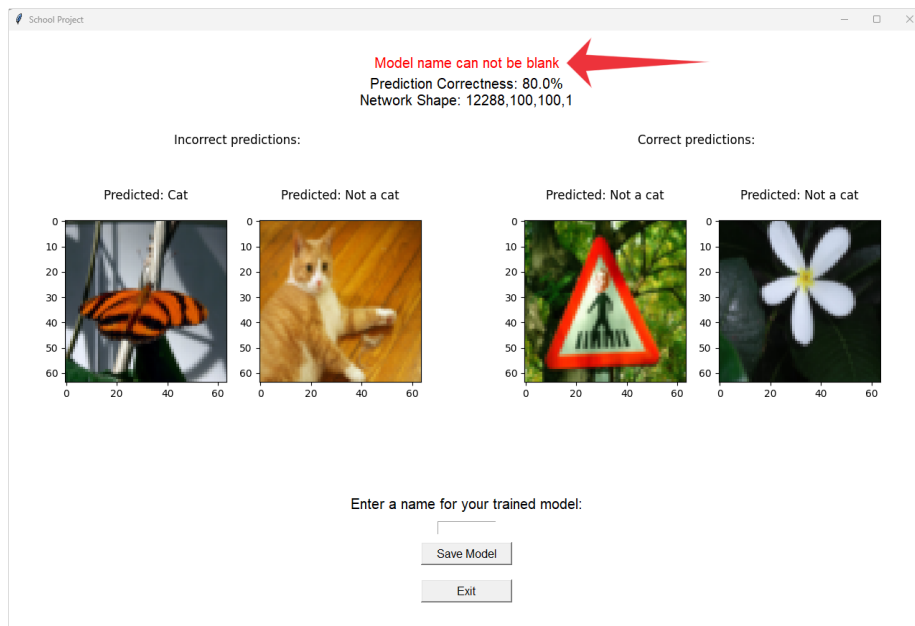


Figure 29: Empty Trained Model Name Validation evidence

Link to video evidence: <https://github.com/mcttn22/school-project/blob/main/project-report/testing-videos.md/#empty-trained-model-name-validation>

- Invalid Delimiter Trained Model Name Validation:

Description	Try to save a trained model with a name with incorrect delimiters.
Data Value	"test test"
Expected Result	The exception should be handled and a useful error message should be displayed.
Actual Result	Expected Result
Test Status	Pass

Evidence:

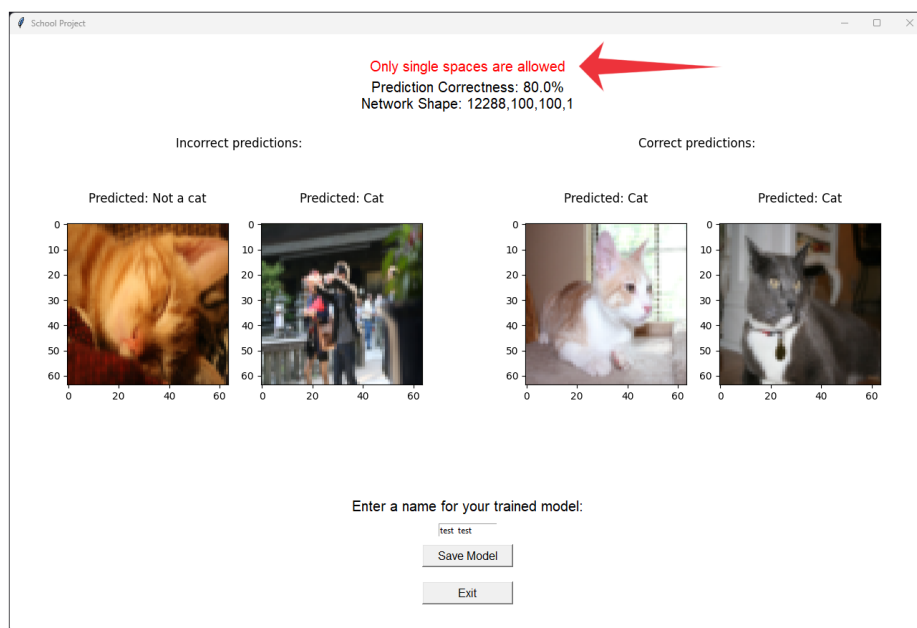


Figure 30: Invalid Delimiter Trained Model Name Validation evidence

Link to video evidence: <https://github.com/mcttn22/school-project/blob/main/project-report/testing-videos.md/#invalid-delimiter-trained-model-name-validation>

5.3 Automated Testing

5.3.1 Unit Tests

Within the test package, I have written the following unit tests:

- Unit tests for the database in a test_database.py module:
 - test_database_structure:

Description	Test that the database tables are set up correctly.
Expected Result	Check that the 'Models' table exists in the database and that the table's info matches the following format: [(0, 'Model_ID', 'INTEGER', 0, None, 1), (1, 'Dataset', 'TEXT', 1, None, 0), (2, 'File_Location', 'TEXT', 1, None, 0), (3, 'Hidden_Layers_Shape', 'TEXT', 1, None, 0), (4, 'Learning_Rate', 'FLOAT', 1, None, 0), (5, 'Name', 'TEXT', 1, None, 0), (6, 'Train_Dataset_Size', 'INTEGER', 1, None, 0), (7, 'Use_ReLu', 'INTEGER', 1, None, 0)]
Actual Result	Expected Result
Test Status	Pass

– test_not_null_constraint:

Description	Test that the NOT NULL constraint is setup.
Data Value	("Test_Dataset", f"school_project/saved- models/uuid.uuid4().hex.npz", "100, 100", 0.1, "Test_Name", 100)
Data Type	Erroneous
Expected Result	A sqlite3.IntegrityError should be raised.
Actual Result	Expected Result
Test Status	Pass

– test_unique_constraint:

Description	Test that the UNIQUE (Dataset, Name) constraint is setup.
Data Value	("Test_Dataset", f"school_project/saved- models/uuid.uuid4().hex.npz", "100, 100", 0.1, "Test_Name", 100, True)
Data Type	Erroneous
Expected Result	A sqlite3.IntegrityError should be raised.
Actual Result	Expected Result
Test Status	Pass

– test_save_load_consistency:

Description	Test that data is not changed between saving and loading.
Data Value	("Test_Dataset", f"school_project/saved- models/uuid.uuid4().hex.npz", "100, 100", 0.1, "Test_Name", 100, True)
Data Type	Normal
Expected Result	Data is not changed between saving and loading.
Actual Result	Expected Result
Test Status	Pass

– Evidence:

```

1  """Unit tests for database."""
2
3  import sqlite3
4  import unittest
5  import uuid
6
7  class TestDatabase(unittest.TestCase):
8      """Unit tests for database."""
9      def __init__(self, *args, **kwargs) -> None:
10         """Initialise unit tests."""
11         super(TestDatabase, self).__init__(*args, **kwargs)
12
13     def test_database_strucure(self) -> None:
14         """Test that the database tables are set up correctly."""
15         connection =
16         ↪ sqlite3.connect(database='school_project/saved_models.db')
17         cursor = connection.cursor()
18
19         # Check that 'Models' table is in the database
20         cursor.execute("SELECT name FROM sqlite_master WHERE
21         ↪ type='table'")
22         self.assertIn(member="Models", container=cursor.fetchall()[0])
23
24         # Check that 'Models' table has the correct attributes
25         expected_table_info = [(0, 'Model_ID', 'INTEGER', 0, None, 1),
26                                (1, 'Dataset', 'TEXT', 1, None, 0),
27                                (2, 'File_Location', 'TEXT', 1, None,
28                                ↪ 0),
29                                (3, 'Hidden_Layers_Shape', 'TEXT', 1,
30                                ↪ None, 0),
31                                (4, 'Learning_Rate', 'FLOAT', 1, None,
32                                ↪ 0),
33                                (5, 'Name', 'TEXT', 1, None, 0),
34                                (6, 'Train_Dataset_Size', 'INTEGER', 1,
35                                ↪ None, 0),
36                                (7, 'Use_ReLu', 'INTEGER', 1, None, 0)]
37         cursor.execute("PRAGMA table_info(Models)")
38         table_info = cursor.fetchall()
39         for expected_attribute, attribute in zip(expected_table_info,
40         ↪ table_info):
41             for expected_info, info in zip(expected_attribute,
42             ↪ attribute):

```

```

36         self.assertEqual(first=expected_info, second=info)
37
38     def test_not_null_constraint(self) -> None:
39         """Test that the NOT NULL constraint is setup."""
40         connection =
41             ↳ sqlite3.connect(database='school_project/saved_models.db')
42         cursor = connection.cursor()
43
44         # Try to insert record with the last attribute missing
45         test_data = ("Test_Dataset",
46
47             ↳ f"school_project/saved-models/{uuid.uuid4().hex}.npz",
48             "100, 100",
49             0.1,
50             "Test_Name",
51             100)
52
53         sql = """
54         INSERT INTO Models
55         (Dataset, File_Location, Hidden_Layers_Shape, Learning_Rate,
56         ↳ Name, Train_Dataset_Size)
57         VALUES (?, ?, ?, ?, ?, ?)
58         """
59         self.assertRaises(sqlite3.IntegrityError, cursor.execute, sql,
60             ↳ test_data)
61
62     def test_unique_constraint(self) -> None:
63         """Test that the UNIQUE (Dataset, Name) constraint is
64         ↳ setup."""
65         connection =
66             ↳ sqlite3.connect(database='school_project/saved_models.db')
67         cursor = connection.cursor()
68
69         # Save test data
70         test_data = ("Test_Dataset",
71
72             ↳ f"school_project/saved-models/{uuid.uuid4().hex}.npz",
73             "100, 100",
74             0.1,
75             "Test_Name",
76             100,
77             True)
78
79         sql = """
80         INSERT INTO Models
81         (Dataset, File_Location, Hidden_Layers_Shape, Learning_Rate,
82         ↳ Name, Train_Dataset_Size, Use_ReLu)
83         VALUES (?, ?, ?, ?, ?, ?, ?)
84         """
85         cursor.execute(sql, test_data)
86         connection.commit()
87
88         # Try to save the same data again
89         test_data = ("Test_Dataset",
90
91             ↳ f"school_project/saved-models/{uuid.uuid4().hex}.npz",
92             "100, 100",
93             0.1,
94             "Test_Name",
95             100,
96             True)
97
98         sql = """
99         INSERT INTO Models
100        (Dataset, File_Location, Hidden_Layers_Shape, Learning_Rate,
101        ↳ Name, Train_Dataset_Size, Use_ReLu)
102        VALUES (?, ?, ?, ?, ?, ?, ?)

```

```

90         """
91         self.assertRaises(sqlite3.IntegrityError, cursor.execute, sql,
92             ↳ test_data)
93
94         # Remove test data from database
95         sql = """
96         DELETE FROM Models WHERE Dataset=? AND Name=?
97         """
98         parameters = (test_data[0], test_data[4])
99         cursor.execute(sql, parameters)
100         connection.commit()
101
102     def test_save_load_consistency(self) -> None:
103         """Test that data is not changed between saving and
104         ↳ loading."""
105         connection =
106         ↳ sqlite3.connect(database='school_project/saved_models.db')
107         cursor = connection.cursor()
108
109         test_data = ("Test_Dataset",
110             ↳ f"school_project/saved-models/{uuid.uuid4().hex}.npz",
111             "100, 100",
112             0.1,
113             "Test_Name",
114             100,
115             True)
116
117         # Save test data
118         sql = """
119         INSERT INTO Models
120         (Dataset, File_Location, Hidden_Layers_Shape, Learning_Rate,
121         ↳ Name, Train_Dataset_Size, Use_ReLu)
122         VALUES (?, ?, ?, ?, ?, ?, ?)
123         """
124         cursor.execute(sql, test_data)
125         connection.commit()
126
127         # Load test data
128         sql = """
129         SELECT * FROM Models WHERE Dataset=? AND Name=?
130         """
131         cursor.execute(sql, (test_data[0], test_data[4]))
132         loaded_data = cursor.fetchall()[0]
133
134         # Delete test data from database
135         sql = """
136         DELETE FROM Models WHERE Dataset=? AND Name=?
137         """
138         parameters = (test_data[0], test_data[4])
139         cursor.execute(sql, parameters)
140         connection.commit()
141
142         # Compare test data with loaded data
143         for test_value, loaded_value in zip(test_data,
144             ↳ loaded_data[1:]):
145             self.assertEqual(first=test_value, second=loaded_value)
146
147     if __name__ == "__main__":
148         unittest.main()

```

```

bash
(venv) max@max-ThinkPad-T14-Gen-1:~/programming/projects/school-project/school-project(main)$ ls
Dockerfile LICENSE Makefile notebooks project-report README.md school_project school_project.egg-info setup.py TODO.md
(venv) max@max-ThinkPad-T14-Gen-1:~/programming/projects/school-project/school-project(main)$ python3 -m unittest school_project/test/test_database.py
.....
Ran 4 tests in 0.030s
OK
(venv) max@max-ThinkPad-T14-Gen-1:~/programming/projects/school-project/school-project(main)$

```

Figure 31: Unit tests for the database in a test_database.py module evidence

Link to video evidence: https://github.com/mcttn22/school-project/blob/main/project-report/testing-videos.md/#test_databasepy

- Unit tests for the utils subpackage of both the cpu and gpu subpackage of the models package. Similarly to the code for the cpu and gpu subpackage, it is not shown as they are identical part from the call to NumPy for the CPU and CuPy for the GPU.

– test_model.py module:

* test_train_dataset_size:

Description	Test the size of training dataset to be value chosen.
Data Value	hidden_layers_shape = [100, 100], train_dataset_size = 4, learning_rate = 0.1, use_relu = True
Data Type	Normal
Expected Result	The number of columns of the training input matrix should be equal to 4.
Actual Result	Expected Result
Test Status	Pass

* test_network_shape:

Description	Test the neuron count of each layer to match the set shape of the network.
Data Value	hidden_layers_shape = [100, 100], train_dataset_size = 4, learning_rate = 0.1, use_relu = True
Data Type	Normal
Expected Result	The input neuron count of each layer should match [2, 100, 100, 1].
Actual Result	Expected Result
Test Status	Pass

* test_learning_rates:

Description	Test learning rate of each layer to be the same.
Data Value	hidden_layers_shape = [100, 100], train_dataset_size = 4, learning_rate = 0.1, use_relu = True
Data Type	Normal
Expected Result	The learning rate of each layer should be 0.1.
Actual Result	Expected Result
Test Status	Pass

* test_relu_model_transfer_types:

Description	Test transfer type of each layer to match whats set.
Data Values	hidden_layers_shape = [100, 100], train_dataset_size = 4, learning_rate = 0.1, use_relu = True
Data Type	Normal
Expected Result	The transfer type of each layer should follow a pattern of ['relu', 'relu', 'sigmoid'].
Actual Result	Expected Result
Test Status	Pass

* test_sigmoid_model_transfer_types:

Description	Test transfer type of each layer to match whats set.
Data Values	hidden_layers_shape = [100, 100], train_dataset_size = 4, learning_rate = 0.1, use_relu = False
Data Type	Normal
Expected Result	The transfer type of each layer should follow a pattern of ['sigmoid', 'sigmoid', 'sigmoid']
Actual Result	Expected Result
Test Status	Pass

* test_weight_matrice_shapes:

Description	Test that each layer's weight matrix has the same number of columns as the layer's input matrix's number of rows, for the matrice multiplication.
Data Values	hidden_layers_shape = [100, 100], train_dataset_size = 4, learning_rate = 0.1, use_relu = True
Data Type	Normal
Expected Result	Each layer's weight matrix has the same number of columns as the layer's input matrix's number of rows.
Actual Result	Expected Result
Test Status	Pass

* test_bias_matrice_shapes:

Description	Test that each layer's bias matrix has the same number of rows as the result of the layer's weights and input multiplication, for element-wise addition of the biases.
Data Values	hidden_layers_shape = [100, 100], train_dataset_size = 4, learning_rate = 0.1, use_relu = True
Data Type	Normal
Expected Result	Each layer's bias matrix has the same number of rows as the result of the layer's weights and input multiplication.
Actual Result	Expected Result
Test Status	Pass

* test_layer_output_shapes:

Description	Test the shape of each layer's activation function's output.
Data Values	hidden_layers_shape = [100, 100], train_dataset_size = 4, learning_rate = 0.1, use_relu = True
Data Type	Normal
Expected Result	The shape of each layer's activation function's output should have the same number of rows as the layer's weight matrix and the same number of columns as the layer's input matrix.
Actual Result	Expected Result
Test Status	Pass

* test_save_model:

Description	Test that the weights and biases are saved correctly.
Data Values	hidden_layers_shape = [100, 100], train_dataset_size = 4, learning_rate = 0.1, use_relu = True, file_location = f'school_project/saved-models/uuid.uuid4().hex.npz'
Data Type	Normal
Expected Result	The weights and biases of each layer should not change between saving and loading.
Actual Result	Expected Result

* Evidence:

```

1  """Unit tests for model module."""
2
3  import os
4  import unittest
5  import uuid
6
7  import numpy as np
8
9  # Test XOR implementation of Model for its lesser computation time
10 from school_project.models.cpu.xor import XORModel
11
12 class TestModel(unittest.TestCase):
13     """Unit tests for model module."""
14     def __init__(self, *args, **kwargs) -> None:
15         """Initialise unit tests and inputs."""
16         super(TestModel, self).__init__(*args, **kwargs)
17
18     def test_train_dataset_size(self) -> None:
19         """Test the size of training dataset to be value
20         ↪ chosen."""
21         train_dataset_size = 4
22         model = XORModel(hidden_layers_shape = [100, 100],
23                           train_dataset_size = train_dataset_size,
24                           learning_rate = 0.1,
25                           use_relu = True)
26         model.create_model_values()
27         model.train(epoch_count=1)
28         self.assertEqual(first=model.layers.head.input.shape[1],
29                           second=train_dataset_size)
30
31     def test_network_shape(self) -> None:
32         """Test the neuron count of each layer to match the set
33         ↪ shape of the
34         network."""
35         layers_shape = [2, 100, 100, 1]
36         model = XORModel(hidden_layers_shape = [100, 100],
37                           train_dataset_size = 4,
38                           learning_rate = 0.1,
39                           use_relu = True)
40         model.create_model_values()
41         model.train(epoch_count=1)
42         for count, layer in enumerate(model.layers):
43             self.assertEqual(first=layer.input_neuron_count,
44                               second=layers_shape[count])

```

```

44 def test_learning_rates(self) -> None:
45     """Test learning rate of each layer to be the same."""
46     learning_rate = 0.1
47     model = XORModel(hidden_layers_shape = [100, 100],
48                       train_dataset_size = 4,
49                       learning_rate = learning_rate,
50                       use_relu = True)
51     model.create_model_values()
52     model.train(epoch_count=1)
53     for layer in model.layers:
54         self.assertEqual(first=layer.learning_rate,
55                          ↪ second=learning_rate)
56
57 def test_relu_model_transfer_types(self) -> None:
58     """Test transfer type of each layer to match whats set."""
59     transfer_types = ['relu', 'relu', 'sigmoid']
60     model = XORModel(hidden_layers_shape = [100, 100],
61                       train_dataset_size = 4,
62                       learning_rate = 0.1,
63                       use_relu = True)
64     model.create_model_values()
65     model.train(epoch_count=1)
66     for count, layer in enumerate(model.layers):
67         self.assertEqual(first=layer.transfer_type,
68                          second=transfer_types[count])
69
70 def test_sigmoid_model_transfer_types(self) -> None:
71     """Test transfer type of each layer to match whats set."""
72     transfer_types = ['sigmoid', 'sigmoid', 'sigmoid']
73     model = XORModel(hidden_layers_shape = [100, 100],
74                       train_dataset_size = 4,
75                       learning_rate = 0.1,
76                       use_relu = False)
77     model.create_model_values()
78     model.train(epoch_count=1)
79     for count, layer in enumerate(model.layers):
80         self.assertEqual(first=layer.transfer_type,
81                          second=transfer_types[count])
82
83 def test_weight_matrice_shapes(self) -> None:
84     """Test that each layer's weight matrix has the same
85     ↪ number of columns
86     as the layer's input matrix's number of rows, for the
87     ↪ matrice
88     multiplication."""
89     model = XORModel(hidden_layers_shape = [100, 100],
90                       train_dataset_size = 4,
91                       learning_rate = 0.1,
92                       use_relu = True)
93     model.create_model_values()
94     model.train(epoch_count=1)
95     for layer in model.layers:
96         self.assertEqual(first=layer.weights.shape[1],
97                          second=layer.input.shape[0])
98
99 def test_bias_matrice_shapes(self) -> None:
100     """Test that each layer's bias matrix has the same number
101     ↪ of rows
102     as the result of the layer's weights and input
103     ↪ multiplication, for
104     element-wise addition of the biases."""
105     model = XORModel(hidden_layers_shape = [100, 100],
106                       train_dataset_size = 4,
107                       learning_rate = 0.1,

```

```

103         use_relu = True)
104     model.create_model_values()
105     model.train(epoch_count=1)
106     for layer in model.layers:
107         self.assertEqual(first=layer.biases.shape[0],
108                          second=layer.weights.shape[0])
109
110     def test_layer_output_shapes(self) -> None:
111         """Test the shape of each layer's activation function's
112         ↪ output."""
113         model = XORModel(hidden_layers_shape = [100, 100],
114                          train_dataset_size = 4,
115                          learning_rate = 0.1,
116                          use_relu = True)
117         model.create_model_values()
118         model.train(epoch_count=1)
119         for layer in model.layers:
120             self.assertEqual(
121                 first=(layer.weights.shape[0],
122                        ↪ layer.input.shape[1]),
123                 second=layer.output.shape
124             )
125
126     def test_save_model(self) -> None:
127         """Test that the weights and biases are saved
128         ↪ correctly."""
129         initial_model = XORModel(hidden_layers_shape = [100,
130         ↪ 100],
131                                   train_dataset_size = 4,
132                                   learning_rate = 0.1,
133                                   use_relu = True)
134         initial_model.create_model_values()
135         initial_model.train(epoch_count=1)
136
137         # Save model values
138         file_location =
139         ↪ f"school_project/saved-models/{uuid.uuid4().hex}.npz"
140
141         ↪ initial_model.save_model_values(file_location=file_location)
142
143         # Create model from the saved values
144         loaded_model = XORModel(hidden_layers_shape = [100, 100],
145                                  train_dataset_size = 4,
146                                  learning_rate = 0.1,
147                                  use_relu = True)
148
149         ↪ loaded_model.load_model_values(file_location=file_location)
150
151         # Remove the saved model values
152         os.remove(path=file_location)
153
154         # Compare initial and loaded model values
155         for layer1, layer2 in zip(initial_model.layers,
156         ↪ loaded_model.layers):
157             self.assertTrue(np.array_equal(a1=layer1.weights,
158                                             a2=layer2.weights))
159             self.assertTrue(np.array_equal(a1=layer1.biases,
160                                             a2=layer2.biases))
161
162     if __name__ == '__main__':
163         unittest.main()

```

```

bash
(venv) max@max-ThinkPad-T14-Gen-1:~/programming/projects/school-project/school-project(main)$ ls
Dockerfile LICENSE Makefile notebooks project-report README.md school_project school_project.egg-info setup.py TODO.md
(venv) max@max-ThinkPad-T14-Gen-1:~/programming/projects/school-project/school-project(main)$ python3 school_project/test/models/cpu/utils/test_model.py
.....
Ran 9 tests in 0.011s
OK
(venv) max@max-ThinkPad-T14-Gen-1:~/programming/projects/school-project/school-project(main)$

```

Figure 32: Unit tests for model module evidence

Link to video evidence: https://github.com/mcttn22/school-project/blob/main/project-report/testing-videos.md/#test_modelpy

– test_tools.py module:

* test_relu:

Description	Test ReLu output range to be greater than or equal to zero.
Data Values	[-100, 0, 100]
Data Type	Boundary
Expected Result	The output of the ReLu transfer function should be greater than or equal to zero.
Actual Result	Expected Result
Test Status	Pass

* test_sigmoid:

Description	Test sigmoid output range to be within 0-1.
Data Values	[-100, 0, 100]
Data Type	Boundary
Expected Result	The output of Sigmoid transfer function should be between zero and one.
Actual Result	Expected Result
Test Status	Pass

* Evidence:

```

1  """Unit tests for tools module."""
2
3  import unittest
4
5  from school_project.models.cpu.utils import tools
6
7  class TestTools(unittest.TestCase):
8      """Unit tests for the tools module."""
9      def __init__(self, *args, **kwargs) -> None:
10         """Initialise unit tests."""
11         super(TestTools, self).__init__(*args, **kwargs)
12
13     def test_relu(self) -> None:
14         """Test ReLu output range to be >=0."""
15         test_inputs = [-100, 0, 100]
16         for test_input in test_inputs:
17             output = tools.relu(z=test_input)
18             self.assertGreaterEqual(a=output, b=0)

```

```
19
20     def test_sigmoid(self) -> None:
21         """Test sigmoid output range to be within 0-1."""
22         test_inputs = [-100, 0, 100]
23         for test_input in test_inputs:
24             output = tools.sigmoid(z=test_input)
25             self.assertTrue(expr=output >= 0 and output <= 1)
26
27 if __name__ == '__main__':
28     unittest.main()
```

```
bash
(venv) max@max-ThinkPad-T14-Gen-1:~/programming/projects/school-project/school-project(main)$ ls
Dockerfile LICENSE Makefile notebooks project-report README.md school_project school_project.egg-info setup.py TODO.md
(venv) max@max-ThinkPad-T14-Gen-1:~/programming/projects/school-project/school-project(main)$ python3 -m unittest school_project/test/models/cpu/utls/test_tools.py
..
-----
Ran 2 tests in 0.000s
OK
(venv) max@max-ThinkPad-T14-Gen-1:~/programming/projects/school-project/school-project(main)$
```

Figure 33: Unit tests for tools module evidence

Link to video evidence: https://github.com/mcttn22/school-project/blob/main/project-report/testing-videos.md/#test_toolspy

5.3.2 GitHub Automated Testing

With the following configuration entered in the .github/workflows/tests.yml file, the unit tests are run automatically on GitHub servers after each commit that is pushed to GitHub, and the status of the tests (either passing or failing) can be viewed on the repository's page. This automatic testing allows for a faster workflow and allows me to identify which changes (commits) cause issues within the code, allowing for easier maintenance of the project.

```
1 name: Tests
2
3 on:
4   push:
5     branches: [ "main" ]
6   pull_request:
7     branches: [ "main" ]
8
9 permissions:
10    contents: read
11
12 jobs:
13   build:
14
15     runs-on: ubuntu-latest
16
17     steps:
18     - uses: actions/checkout@v3
19     - name: Set up Python 3.10
20       uses: actions/setup-python@v3
21       with:
22         python-version: "3.10"
23     - name: Install dependencies
24       run: |
25         python -m pip install --upgrade pip
26         pip install numpy
27     - name: Test
28       run: |
29         python -m unittest discover ./school_project/test/models/cpu
```

5.3.3 Docker

Basic Dockerfile instructions are included for its use in the README.md file, this allows the project to be quickly run and tested in Docker containers. Below shows the contents of the basic Dockerfile:

```
1 FROM python:3.11
2
3 # Set a directory for the app
4 WORKDIR /usr/src/app
5
6 # Copy all the files to the container
7 COPY . .
8
9 # Install dependencies
10 RUN python setup.py install
11
12 # Run the project
13 CMD ["python", "./school_project"]
```

6 Investigation

This section outlines the code utilised to test the performance of a trained network. It then utilises this functionality to explore the effects of Hyper-Parameters on the Artificial Neural Network performance.

6.1 test_model module

The test_model module is contained within the frames package, and contains tkinter frames for investigating the behaviour of trained Artificial Neural Network models for each dataset. For each training dataset that an Artificial Neural Network is trained on, there is a corresponding test dataset with completely new images to be tested on to judge the performance of the trained model. As fewer images are needed for testing than for training, the Cat dataset only has 50 test images (compared to the 209 images for training) and the MNIST [9] dataset only has 10,000 test images (compared to the 60,000 images for training). Each frame displays the results of the testing along with a random selection of incorrect and correct predictions.

```
1  """Tkinter frames for testing a saved Artificial Neural Network model."""
2
3  import random
4  import threading
5  import tkinter as tk
6
7  from matplotlib.figure import Figure
8  from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
9  import numpy as np
10
11 class TestMNISTFrame(tk.Frame):
12     """Frame for Testing MNIST page."""
13     def __init__(self, root: tk.Tk, width: int,
14                 height: int, bg: str,
15                 use_gpu: bool, model: object) -> None:
16         """Initialise test MNIST frame widgets.
17
18         Args:
19             root (tk.Tk): the widget object that contains this widget.
20             width (int): the pixel width of the frame.
21             height (int): the pixel height of the frame.
22             bg (str): the hex value or name of the frame's background colour.
23             use_gpu (bool): True or False whether the GPU should be used.
24             model (object): The Model object to be tested.
25         Raises:
26             TypeError: if root, width or height are not of the correct type.
27
28         """
29         super().__init__(master=root, width=width, height=height, bg=bg)
30         self.root = root
31         self.WIDTH = width
32         self.HEIGHT = height
33         self.BG = bg
34
35         # Setup test MNIST frame variables
36         self.use_gpu = use_gpu
37
38         # Setup widgets
39         self.model_status_label = tk.Label(master=self,
40                                             bg=self.BG,
```



```

41         font=('Arial', 15))
42     self.results_label = tk.Label(master=self,
43                                   bg=self.BG,
44                                   font=('Arial', 15))
45     self.correct_prediction_figure = Figure()
46     self.correct_prediction_canvas = FigureCanvasTkAgg(
47         figure=self.correct_prediction_figure,
48         master=self
49     )
50     self.incorrect_prediction_figure = Figure()
51     self.incorrect_prediction_canvas = FigureCanvasTkAgg(
52         figure=self.incorrect_prediction_figure,
53         master=self
54     )
55
56     # Grid widgets
57     self.model_status_label.grid(row=0, columnspan=3, pady=(30,0))
58     self.results_label.grid(row=1, columnspan=3)
59     self.incorrect_prediction_canvas.get_tk_widget().grid(row=2, column=0)
60     self.correct_prediction_canvas.get_tk_widget().grid(row=2, column=2)
61
62     # Start test thread
63     self.model_status_label.configure(text="Testing trained model",
64                                       fg='red')
65     self.test_thread = threading.Thread(target=model.test)
66     self.test_thread.start()
67
68 def plot_results(self, model: object) -> None:
69     """Plot results of Model test.
70
71     Args:
72         model (object): the Model object thats been tested.
73
74     """
75     self.model_status_label.configure(text="Testing Results:", fg='green')
76     if not self.use_gpu:
77         self.results_label.configure(
78             text="Prediction Correctness: " +
79             f"{round(number=100 - np.mean(np.abs(model.test_prediction.round() -
80             ↪ model.test_outputs)) * 100, ndigits=1)}%\n" +
81             f"Network Shape: " +
82             f"{','.join(model.layers_shape)}\n"
83         )
84
85         test_inputs = np.squeeze(model.test_inputs).T
86         test_outputs = np.squeeze(model.test_outputs).T.tolist()
87         test_prediction = np.squeeze(model.test_prediction).T.tolist()
88
89         # Randomly shuffle order of test_inputs, test_outputs and
90         ↪ test_prediction
91         # whilst maintaining order between them
92         test_data = list(zip(test_inputs,
93                               test_outputs,
94                               test_prediction))
95         random.shuffle(test_data)
96         test_inputs, test_outputs, test_prediction = zip(*test_data)
97
98     elif self.use_gpu:
99
100         import cupy as cp
101
102         self.results_label.configure(
103             text="Prediction Correctness: " +

```

```

102         f"{round(number=100 -
↪ np.mean(np.abs(cp.asnumpy(model.test_prediction).round() -
↪ cp.asnumpy(model.test_outputs))) * 100, ndigits=1)}%\n" +
103         f"Network Shape: " +
104         f"{','.join(model.layers_shape)}\n"
105     )
106
107     test_inputs = cp.asnumpy(cp.squeeze(model.test_inputs)).T
108     test_outputs = cp.asnumpy(cp.squeeze(model.test_outputs)).T.tolist()
109     test_prediction = cp.squeeze(model.test_prediction).T.tolist()
110
111     # Randomly shuffle order of test_inputs, test_outputs and
112     ↪ test_prediction
113     # whilst maintaining order between them
114     test_data = list(zip(test_inputs,
115                         test_outputs,
116                         test_prediction))
117     random.shuffle(test_data)
118     test_inputs, test_outputs, test_prediction = zip(*test_data)
119
120     # Setup incorrect prediction figure
121     self.incorrect_prediction_figure.suptitle("Incorrect predictions:")
122     image_count = 0
123     for i in range(len(test_prediction)):
124         if test_prediction[i].index(max(test_prediction[i])) !=
125         ↪ test_outputs[i].index(max(test_outputs[i])):
126             if image_count == 2:
127                 break
128             elif image_count == 0:
129                 image = self.incorrect_prediction_figure.add_subplot(121)
130             elif image_count == 1:
131                 image = self.incorrect_prediction_figure.add_subplot(122)
132             image.set_title(f"Predicted:
133             ↪ {test_prediction[i].index(max(test_prediction[i]))}\n" +
134             ↪ f"Should have predicted:
135             ↪ {test_outputs[i].index(max(test_outputs[i]))}")
136             image.imshow(test_inputs[i].reshape((28,28)))
137             image_count += 1
138
139     # Setup correct prediction figure
140     self.correct_prediction_figure.suptitle("Correct predictions:")
141     image_count = 0
142     for i in range(len(test_prediction)):
143         if test_prediction[i].index(max(test_prediction[i])) ==
144         ↪ test_outputs[i].index(max(test_outputs[i])):
145             if image_count == 2:
146                 break
147             elif image_count == 0:
148                 image = self.correct_prediction_figure.add_subplot(121)
149             elif image_count == 1:
150                 image = self.correct_prediction_figure.add_subplot(122)
151             image.set_title(f"Predicted:
152             ↪ {test_prediction[i].index(max(test_prediction[i]))}")
153             image.imshow(test_inputs[i].reshape((28,28)))
154             image_count += 1
155
156     class TestCatRecognitionFrame(tk.Frame):
157         """Frame for Testing Cat Recognition page."""
158         def __init__(self, root: tk.Tk, width: int,
159                     height: int, bg: str,
160                     use_gpu: bool, model: object) -> None:
161             """Initialise test cat recognition frame widgets.

```

```

158         root (tk.Tk): the widget object that contains this widget.
159         width (int): the pixel width of the frame.
160         height (int): the pixel height of the frame.
161         bg (str): the hex value or name of the frame's background colour.
162         use_gpu (bool): True or False whether the GPU should be used.
163         model (object): the Model object to be tested.
164     Raises:
165         TypeError: if root, width or height are not of the correct type.
166
167     """
168     super().__init__(master=root, width=width, height=height, bg=bg)
169     self.root = root
170     self.WIDTH = width
171     self.HEIGHT = height
172     self.BG = bg
173
174     # Setup image recognition frame variables
175     self.use_gpu = use_gpu
176
177     # Setup widgets
178     self.model_status_label = tk.Label(master=self,
179                                       bg=self.BG,
180                                       font=('Arial', 15))
181     self.results_label = tk.Label(master=self,
182                                  bg=self.BG,
183                                  font=('Arial', 15))
184     self.correct_prediction_figure = Figure()
185     self.correct_prediction_canvas = FigureCanvasTkAgg(
186         figure=self.correct_prediction_figure,
187         master=self
188     )
189     self.incorrect_prediction_figure = Figure()
190     self.incorrect_prediction_canvas = FigureCanvasTkAgg(
191         figure=self.incorrect_prediction_figure,
192         master=self
193     )
194
195     # Grid widgets
196     self.model_status_label.grid(row=0, columnspan=3, pady=(30,0))
197     self.results_label.grid(row=1, columnspan=3)
198     self.incorrect_prediction_canvas.get_tk_widget().grid(row=2, column=0)
199     self.correct_prediction_canvas.get_tk_widget().grid(row=2, column=2)
200
201     # Start test thread
202     self.model_status_label.configure(text="Testing trained model...",
203                                     fg='red')
204     self.test_thread = threading.Thread(target=model.test)
205     self.test_thread.start()
206
207 def plot_results(self, model: object) -> None:
208     """Plot results of Model test
209
210     Args:
211         model (object): the Model object thats been tested.
212
213     """
214     self.model_status_label.configure(text="Testing Results:", fg='green')
215     if not self.use_gpu:
216         self.results_label.configure(
217             text="Prediction Correctness: " +
218             f"{round(number=100 - np.mean(np.abs(model.test_prediction.round() -
219             ↪ model.test_outputs)) * 100, ndigits=1)}%\n" +
219             f"Network Shape: " +
220             f"{', '.join(model.layers_shape)}\n"

```

```

221         )
222
223         # Randomly shuffle order of test_inputs, test_outputs and
224         ↪ test_prediction
225         # whilst maintaining order between them
226         test_data = list(zip(model.test_inputs.T,
227                               np.squeeze(model.test_outputs).T.tolist(),
228                               ↪ np.squeeze(model.test_prediction.round()).T.tolist()))
229         random.shuffle(test_data)
230         (test_inputs,
231          test_outputs,
232          test_prediction) = map(lambda arr: np.array(arr).T,
233                                zip(*test_data))
234
235     elif self.use_gpu:
236
237         import cupy as cp
238
239         self.results_label.configure(
240             text="Prediction Correctness: " +
241             f"{round(number=100 -
242                     ↪ np.mean(np.abs(cp.asnumpy(model.test_prediction).round() -
243                     ↪ cp.asnumpy(model.test_outputs))) * 100, ndigits=1)}%\n" +
244             f"Network Shape: " +
245             f"{','.join(model.layers_shape)}\n"
246             )
247
248         # Randomly shuffle order of test_inputs, test_outputs and
249         ↪ test_prediction
250         # whilst maintaining order between them
251         test_data = list(zip(cp.asnumpy(model.test_inputs).T,
252                               ↪ cp.asnumpy(cp.squeeze(model.test_outputs)).T.tolist(),
253                               ↪ cp.asnumpy(cp.squeeze(model.test_prediction)).round().T.tolist()))
254         random.shuffle(test_data)
255         (test_inputs,
256          test_outputs,
257          test_prediction) = map(lambda arr: np.array(arr).T,
258                                zip(*test_data))
259
260     # Setup incorrect prediction figure
261     self.incorrect_prediction_figure.suptitle("Incorrect predictions:")
262     image_count = 0
263     for i in range(len(test_prediction)):
264         if test_prediction[i] != test_outputs[i]:
265             if image_count == 2:
266                 break
267             elif image_count == 0:
268                 image = self.incorrect_prediction_figure.add_subplot(121)
269             elif image_count == 1:
270                 image = self.incorrect_prediction_figure.add_subplot(122)
271             image.set_title(f"Predicted: {'Cat' if test_prediction[i] == 1
272                     ↪ else 'Not a cat'}\n")
273             image.imshow(test_inputs[:,i].reshape((64,64,3)))
274             image_count += 1
275
276     # Setup correct prediction figure
277     self.correct_prediction_figure.suptitle("Correct predictions:")
278     image_count = 0
279     for i in range(len(test_prediction)):
280         if test_prediction[i] == test_outputs[i]:
281             if image_count == 2:

```

```

277         break
278     elif image_count == 0:
279         image = self.correct_prediction_figure.add_subplot(121)
280     elif image_count == 1:
281         image = self.correct_prediction_figure.add_subplot(122)
282     image.set_title(f"Predicted: {'Cat' if test_prediction[i] == 1
↵ else 'Not a cat'}\n")
283     image.imshow(test_inputs[:,i].reshape((64,64,3)))
284     image_count += 1
285
286 class TestXORFrame(tk.Frame):
287     """Frame for Testing XOR page."""
288     def __init__(self, root: tk.Tk, width: int,
289                 height: int, bg: str, model: object) -> None:
290         """Initialise test XOR frame widgets.
291
292         Args:
293             root (tk.Tk): the widget object that contains this widget.
294             width (int): the pixel width of the frame.
295             height (int): the pixel height of the frame.
296             bg (str): the hex value or name of the frame's background colour.
297             model (object): the Model object to be tested.
298         Raises:
299             TypeError: if root, width or height are not of the correct type.
300
301         """
302         super().__init__(master=root, width=width, height=height, bg=bg)
303         self.root = root
304         self.WIDTH = width
305         self.HEIGHT = height
306         self.BG = bg
307
308         # Setup widgets
309         self.model_status_label = tk.Label(master=self,
310                                             bg=self.BG,
311                                             font=('Arial', 15))
312         self.results_label = tk.Label(master=self,
313                                       bg=self.BG,
314                                       font=('Arial', 20))
315
316         # Pack widgets
317         self.model_status_label.pack(pady=(30,0))
318
319         # Start test thread
320         self.model_status_label.configure(text="Testing trained model...",
321                                           fg='red')
322         self.test_thread = threading.Thread(target=model.test)
323         self.test_thread.start()
324
325     def plot_results(self, model: object):
326         """Plot results of Model test.
327
328         Args:
329             model (object): the Model object thats been tested.
330
331         """
332         self.model_status_label.configure(text="Testing Results:", fg='green')
333         results = (
334             f"Prediction Accuracy: " +
335             f"{round(number=model.test_prediction_accuracy, ndigits=1)}%\n" +
336             f"Network Shape: " +
337             f"{','.join(model.layers_shape)}\n"
338         )
339         for i in range(model.test_inputs.shape[1]):

```

```
340         results += f"{model.test_inputs[0][i]}, "
341         results += f"{model.test_inputs[1][i]} = "
342         if np.squeeze(model.test_prediction)[i] >= 0.5:
343             results += "1\n"
344         else:
345             results += "0\n"
346     self.results_label.configure(text=results)
347     self.results_label.pack()
```

This code outputs the following results, as shown in figures 34 to 36 depending on the dataset used:

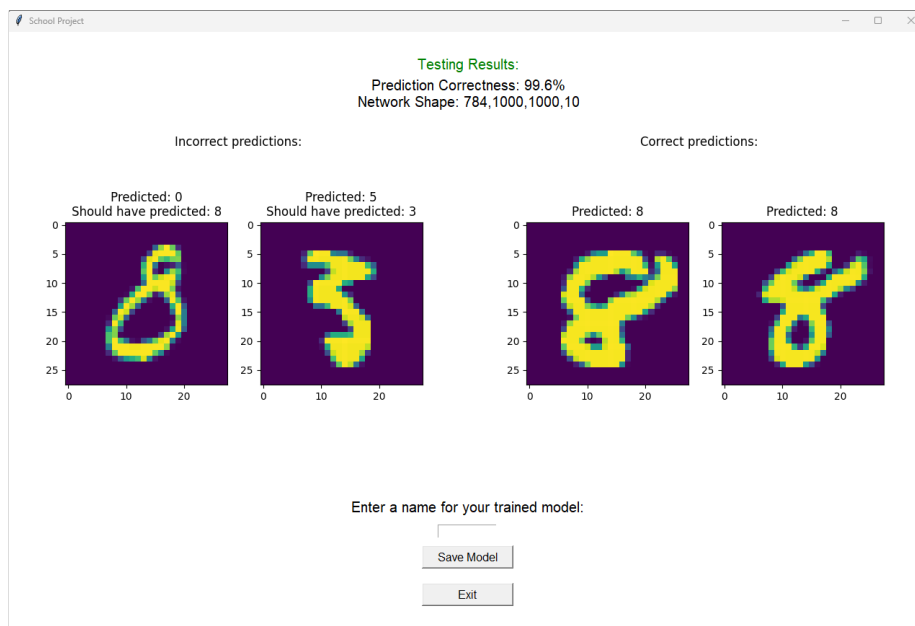


Figure 34: Model test results for MNIST database

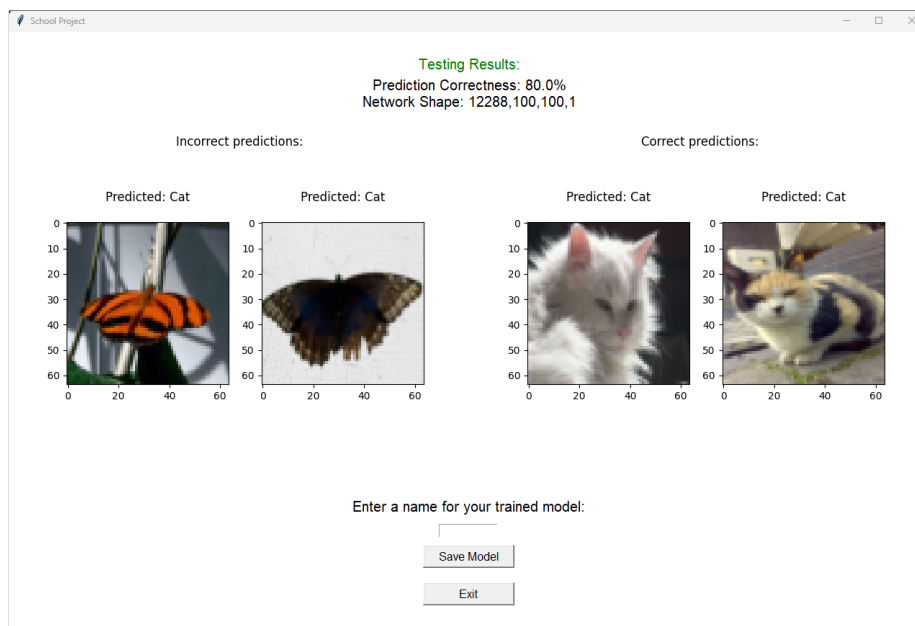


Figure 35: Model test results for Cat recognition database

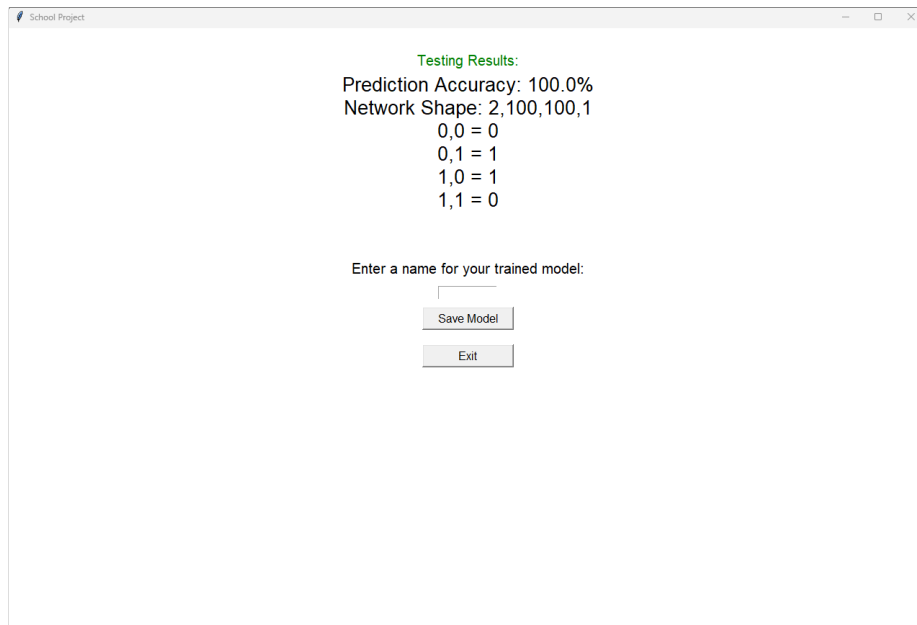


Figure 36: Model test results for XOR dataset

6.2 Exploration into the effects of Hyper-Parameters

As discussed in section 2.1 Artificial Neural Networks have a number of critical hyper-parameters which describe the shape of the network and the nature in which it learns. To explore this, I have conducted a series of experiments utilising the program to explore the fundamental impact of these hyper-parameters. In particular I explored the impact of the:

- Learning rate
- Number of Epochs undertaken during training
- Training dataset size
- Number of hidden layers
- Number of neurons in each layer
- Use of the ReLu vs Sigmoid transfer function
- Use of GPU vs CPU

For these investigations, I utilised Jupyter Notebook to run blocks of code and display the results. The output of each Jupyter Notebook is shown in the analysis below.

Analysis of the impact of the learning rate on the reduction of the loss value

The following code trains and tests models on the XOR dataset with varying learning rates, and then plots graphs of Loss Value against Epoch Count.

```
[17]: import os

import matplotlib.pyplot as plt
import numpy as np

from school_project.models.cpu.xor import XORModel as Model

# Change to root directory of project
os.chdir(os.getcwd())

# Set width and height of figure
plt.rcParams["figure.figsize"] = [5, 10]

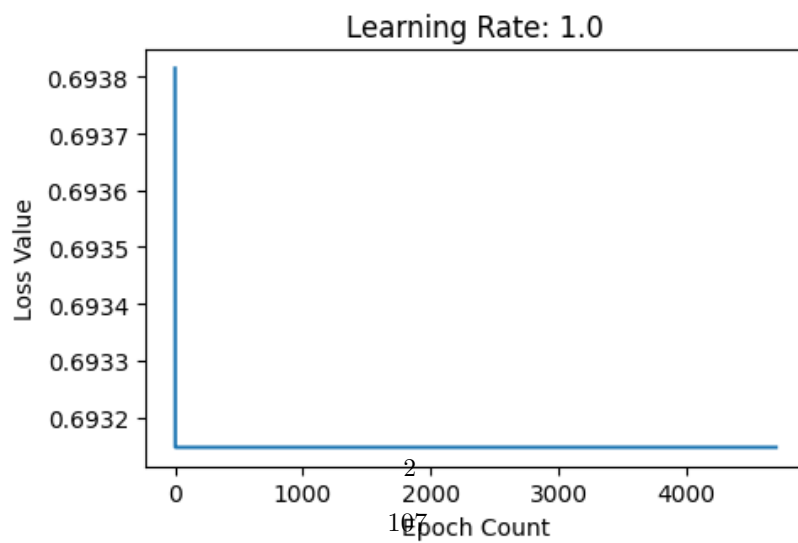
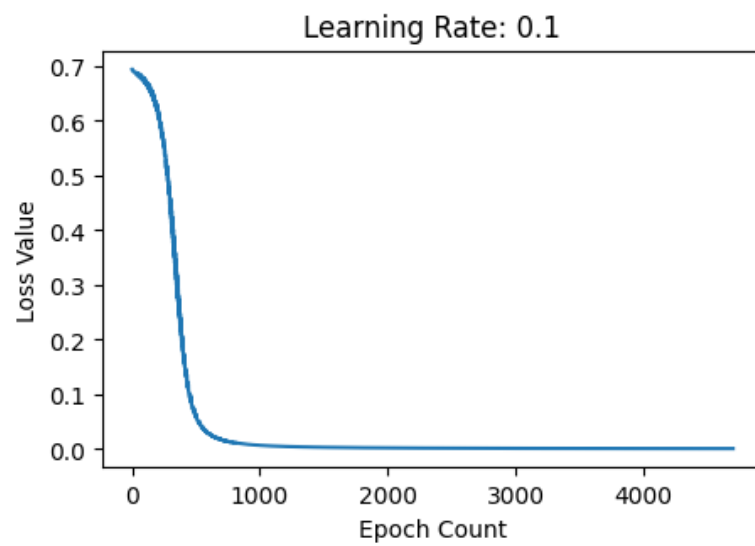
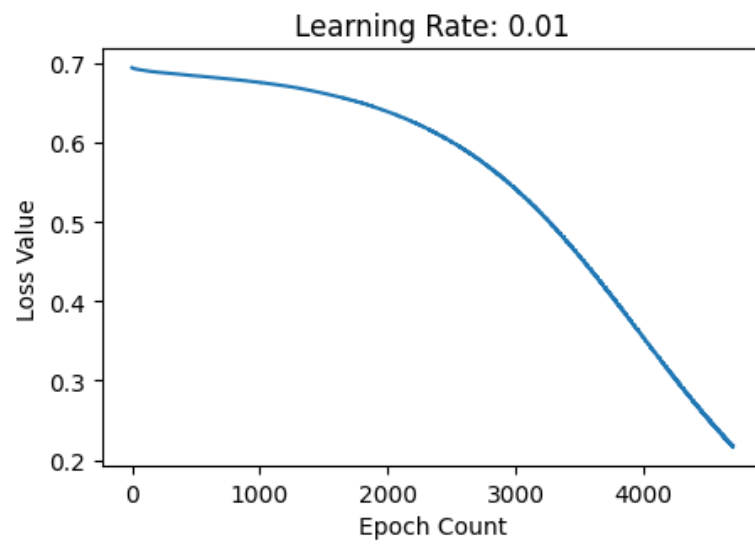
learning_rates = [0.01, 0.1, 1.0]

figure, axis = plt.subplots(nrows=len(learning_rates), ncols=1)

for count, learning_rate in enumerate(learning_rates):
    model = Model(hidden_layers_shape=[100, 100],
                  train_dataset_size=4,
                  learning_rate=learning_rate,
                  use_relu=True)
    model.create_model_values()
    model.train(epoch_count=4_700)
    model.test()

    axis[count].set_title(f"Learning Rate: {model.learning_rate}")
    axis[count].set_xlabel("Epoch Count")
    axis[count].set_ylabel("Loss Value")
    axis[count].plot(np.squeeze(model.train_losses))

plt.tight_layout()
plt.show()
```



As shown above, if the learning rate is set to too low of a value (0.01 in this case) the model will take more epochs to reduce the loss value, and may even get stuck in unwanted local minimums. If the learning rate is set to an optimal value (0.1 in this case) the model reduces the loss value efficiently and to a small enough value for predictions. On the other hand, if the learning rate is set to too high of a value (1.0 in this case) the model may learn too quickly and even ‘jump over’ minima, causing the loss value to stop reducing.

Analysis of the impact of training epoch count on network performance and training time taken

The following code trains models on the Cat Recognition dataset and tests the model at regular Epoch Count intervals, and then plots graphs of Test Prediction Accuracy against Epoch Count and Training Time against Epoch Count.

```
[6]: from IPython.display import clear_output, display
import os

import matplotlib.pyplot as plt
import numpy as np

from school_project.models.gpu.cat_recognition import CatRecognitionModel as Model

# Change to root directory of project
os.chdir(os.getcwd())

# Set width and height of figure
plt.rcParams["figure.figsize"] = [10, 5]

# Generate list of Epoch Counts from 1 to 5000, incremented by 500
epoch_count_interval = 500
epoch_counts = np.array(list(range(0, 5_000, epoch_count_interval)))

test_prediction_accuracies = np.array([])
training_times = np.array([])

# Create model object
model = Model(hidden_layers_shape=[100, 100],
               train_dataset_size=209,
               learning_rate=0.1,
               use_relu=True)
model.create_model_values()

for index, epoch_count in enumerate(epoch_counts):
    clear_output(wait=True)
```

```

display(f"Progress: {round(number=index/len(epoch_counts) * 100, ndigits=2)}%")

model.train(epoch_count=epoch_count_interval)
model.test()

test_prediction_accuracies = np.append(test_prediction_accuracies,
                                       model.test_prediction_accuracy)

# Add training times cumulatively
if len(training_times) != 0:
    training_times = np.append(training_times,
                              training_times[-1] + model.training_time)
else:
    training_times = np.append(training_times,
                              model.training_time)

clear_output(wait=True)
display("Progress: Complete")

figure, axis = plt.subplots(nrows=1, ncols=2)

axis[0].set_xlabel("Epoch Count")
axis[0].set_ylabel("Test Prediction Accuracy (%)")

# Plot regression line
axis[0].plot(epoch_counts, test_prediction_accuracies, marker='x')

# Determine gradient and y-intercept of training times regression line
m, c = np.polyfit(epoch_counts, training_times, deg=1)
print(f"Training Times Regression Line Gradient: {round(number=m, ndigits=2)}")

axis[1].set_xlabel("Epoch Count")
axis[1].set_ylabel("Training Time (s)")

# Plot scatter graph of epoch counts and training times
axis[1].scatter(epoch_counts, training_times, marker='x')

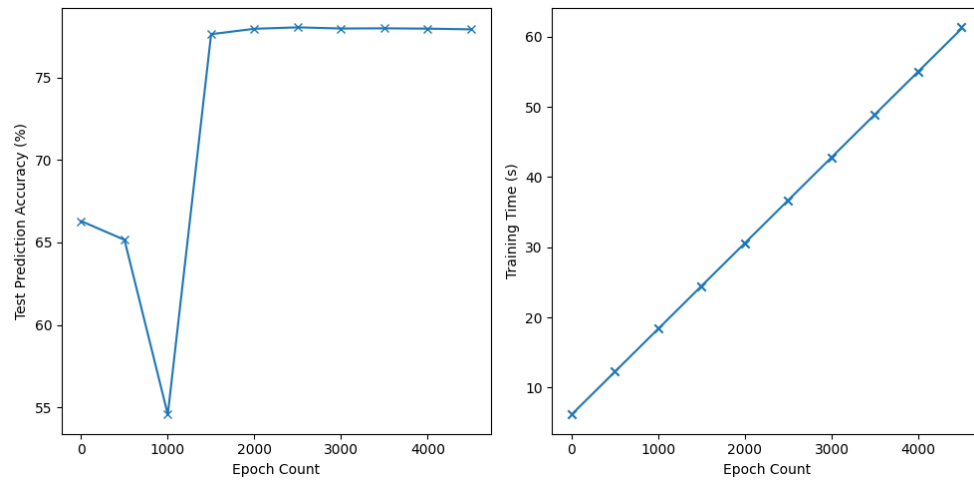
# Plot regression line
axis[1].plot(epoch_counts, m * epoch_counts + c)

plt.tight_layout()
plt.show()

```

'Progress: Complete'

Training Times Regression Line Gradient: 0.01



As shown above, as the epoch count increases so does both the test prediction accuracy and the training time taken.

Analysis of the impact of training dataset size on network performance and training time taken

The following code trains and tests models on the Cat Recognition dataset with varying Train Dataset Sizes, and then plots graphs of Test Prediction Accuracy against Train Dataset Size and Training Time against Train Dataset Size.

```
[1]: from IPython.display import clear_output, display
import os

import matplotlib.pyplot as plt
import numpy as np

from school_project.models.gpu.cat_recognition import CatRecognitionModel as Model

# Change to root directory of project
os.chdir(os.getcwd())

# Set width and height of figure
plt.rcParams["figure.figsize"] = [10, 5]

# Generate list of train dataset sizes from 1 to 210, incremented by 13
train_dataset_sizes = np.array(list(range(1, 210, 13)))

test_prediction_accuracies = np.array([])
training_times = np.array([])

for index, train_dataset_size in enumerate(train_dataset_sizes):
    clear_output(wait=True)
    display(f"Progress: {round(number=index/len(train_dataset_sizes) * 100, ndigits=2)}%")

    model = Model(hidden_layers_shape=[100, 100],
                   train_dataset_size=train_dataset_size,
                   learning_rate=0.1,
                   use_relu=True)
    model.create_model_values()
    model.train(epoch_count=2_000)
```



```

model.test()

test_prediction_accuracies = np.append(test_prediction_accuracies,
                                         model.test_prediction_accuracy)
training_times = np.append(training_times,
                             model.training_time)

clear_output(wait=True)
display("Progress: Complete")

figure, axis = plt.subplots(nrows=1, ncols=2)

# Determine gradient and y-intercept of prediction accuracies regression line
m, c = np.polyfit(train_dataset_sizes, test_prediction_accuracies, deg=1)
print(f"Test Prediction Accuracies Regression Line Gradient: {round(number=m, ndigits=2)}")

axis[0].set_xlabel("Train Dataset Size")
axis[0].set_ylabel("Test Prediction Accuracy (%)")

# Plot scatter graph of train dataset sizes and prediction accuracies
axis[0].scatter(train_dataset_sizes, test_prediction_accuracies, marker='x')

axis[0].plot(train_dataset_sizes, m * train_dataset_sizes + c)

# Determine gradient and y-intercept of training times regression line
m, c = np.polyfit(train_dataset_sizes, training_times, deg=1)
print(f"Training Times Regression Line Gradient: {round(number=m, ndigits=2)}")

axis[1].set_xlabel("Train Dataset Size")
axis[1].set_ylabel("Training Time (s)")

# Plot scatter graph of train dataset sizes and training times
axis[1].scatter(train_dataset_sizes, training_times, marker='x')

# Plot regression line
axis[1].plot(train_dataset_sizes, m * train_dataset_sizes + c)

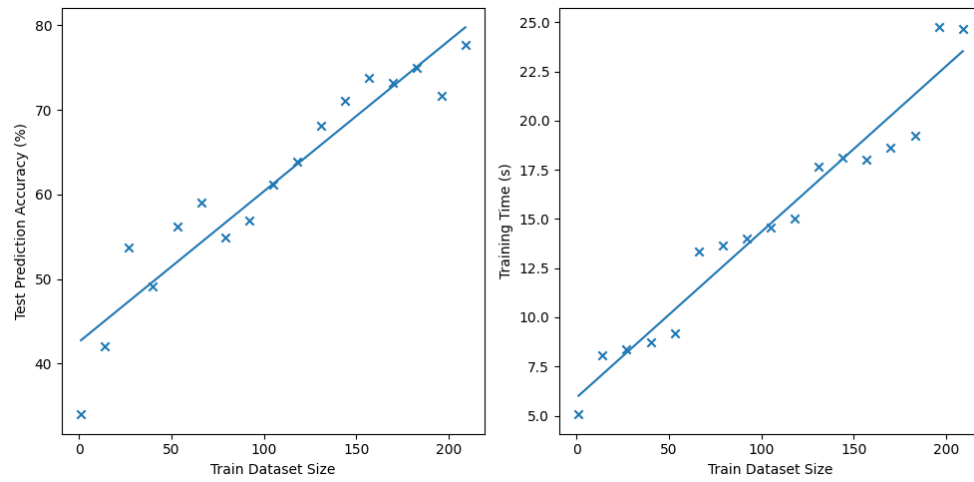
plt.tight_layout()
plt.show()

```

'Progress: Complete'

Test Prediction Accuracies Regression Line Gradient: 0.18

Training Times Regression Line Gradient: 0.08



As shown above, as the train dataset size increases so does both the prediction accuracy and the training time taken. Therefore, I can predict that if I increase the size of the Cat Recognition dataset, I could improve the accuracy of the model trained on the dataset.

Analysis of the impact of layer count on network performance and training time taken

The following code trains and tests models on the Cat Recognition dataset with a varying number of layers, and then plots graphs of Test Prediction Accuracy against Layer Count and Training Time against Layer Count.

```
[1]: from IPython.display import clear_output, display
import os

import matplotlib.pyplot as plt
import numpy as np

from school_project.models.gpu.cat_recognition import CatRecognitionModel as Model

# Change to root directory of project
os.chdir(os.getcwd())

# Set width and height of figure
plt.rcParams["figure.figsize"] = [10, 5]

layer_counts = np.array(list(range(1, 5)))
neuron_count = 100
test_prediction_accuracies = np.array([])
training_times = np.array([])

for index, layer_count in enumerate(layer_counts):
    clear_output(wait=True)
    display(f"Progress: {round(number=index/len(layer_counts) * 100, ndigits=2)}%")

    model = Model(
        hidden_layers_shape=[neuron_count for layer in range(layer_count)],
        train_dataset_size=209,
        learning_rate=0.1,
        use_relu=True
    )
    model.create_model_values()
```

```

model.train(epoch_count=3_500)
model.test()

test_prediction_accuracies = np.append(test_prediction_accuracies,
                                         model.test_prediction_accuracy)
training_times = np.append(training_times,
                             model.training_time)

clear_output(wait=True)
display("Progress: Complete")

figure, axis = plt.subplots(nrows=1, ncols=2)

axis[0].set_xlabel("Layer Count")
axis[0].set_ylabel("Test Prediction Accuracy (%)")

axis[0].plot(layer_counts, test_prediction_accuracies, marker='x')

# Determine gradient and y-intercept of training times regression line
m, c = np.polyfit(layer_counts, training_times, deg=1)
print(f"Training Times Regression Line Gradient: {round(number=m, ndigits=2)}")

axis[1].set_xlabel("Layer Count")
axis[1].set_ylabel("Training Time (s)")

# Plot scatter graph of layer Counts and training times
axis[1].scatter(layer_counts, training_times, marker='x')

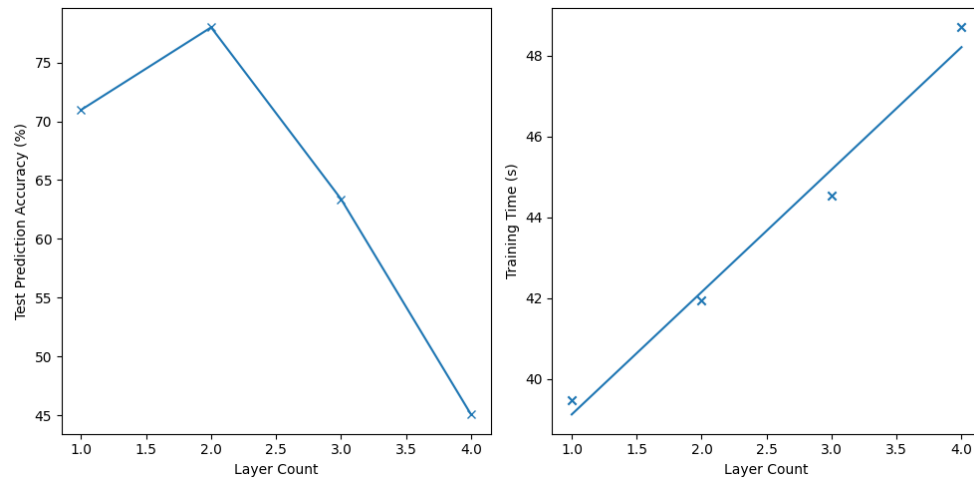
# Plot regression line
axis[1].plot(layer_counts, m * layer_counts + c)

plt.tight_layout()
plt.show()

```

'Progress: Complete'

Training Times Regression Line Gradient: 3.03



As shown above, as the layer count increases so does the training time taken and the test prediction accuracy at first. However, as the layer count continued to increase the prediction accuracy began to drop greatly (after 2 layers in this case). This is most likely due to the model overfitting and learning the training dataset too closely, causing it to fail on the new inputs of the test dataset.

Analysis of the impact of neuron count on network performance and training time taken

The following code trains and tests models on the Cat Recognition dataset with a varying number of neurons in each layer, and then plots graphs of Test Prediction Accuracy against Neuron Count and Training Time against Neuron Count.

```
[1]: from IPython.display import clear_output, display
import os

import matplotlib.pyplot as plt
import numpy as np

from school_project.models.gpu.cat_recognition import CatRecognitionModel as Model

# Change to root directory of project
os.chdir(os.getcwd())

# Set width and height of figure
plt.rcParams["figure.figsize"] = [10, 5]

# Generate list of neuron counts from 1 to 501, incremented by 100
neuron_counts = np.array(list(range(1, 501, 100)))

layer_count = 2
test_prediction_accuracies = np.array([])
training_times = np.array([])

for index, neuron_count in enumerate(neuron_counts):
    clear_output(wait=True)
    display(f"Progress: {round(number=index/len(neuron_counts) * 100, ndigits=2)}%")

    model = Model(
        hidden_layers_shape=[neuron_count for layer in range(layer_count)],
        train_dataset_size=209,
        learning_rate=0.1,
        use_relu=True
```

```

    )
    model.create_model_values()
    model.train(epoch_count=3_500)
    model.test()

    test_prediction_accuracies = np.append(test_prediction_accuracies,
                                           model.test_prediction_accuracy)
    training_times = np.append(training_times,
                               model.training_time)

clear_output(wait=True)
display("Progress: Complete")

figure, axis = plt.subplots(nrows=1, ncols=2)

axis[0].set_xlabel("Neuron Count")
axis[0].set_ylabel("Test Prediction Accuracy (%)")

axis[0].plot(neuron_counts, test_prediction_accuracies, marker='x')

# Determine gradient and y-intercept of training times regression line
m, c = np.polyfit(neuron_counts, training_times, deg=1)
print(f"Training Times Regression Line Gradient: {round(number=m, ndigits=2)}")

axis[1].set_xlabel("Neuron Count")
axis[1].set_ylabel("Training Time (s)")

# Plot scatter graph of neuron counts and training times
axis[1].scatter(neuron_counts, training_times, marker='x')

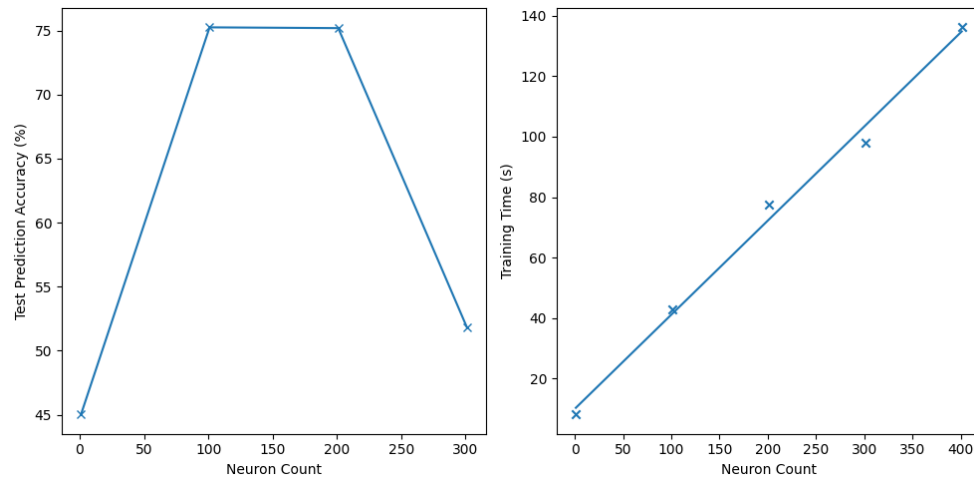
# Plot regression line
axis[1].plot(neuron_counts, m * neuron_counts + c)

plt.tight_layout()
plt.show()

```

'Progress: Complete'

Training Times Regression Line Gradient: 0.31



As shown above, as the neuron count of each layer increases so does the training time taken and the test prediction accuracy at first. However, as the neuron count continued to increase the prediction accuracy began to drop greatly (after 200 neurons in this case). This is most likely due to the model overfitting and learning the training dataset too closely, causing it to fail on the new inputs of the test dataset.

Analysis of the impact of the transfer function on the reduction of the loss value

The following code trains and tests models on the XOR dataset using ReLu and then not using ReLu, and then plots graphs of Loss Value against Epoch Count.

```
[1]: import os

import matplotlib.pyplot as plt
import numpy as np

from school_project.models.cpu.xor import XORModel as Model

# Change to root directory of project
os.chdir(os.getcwd())

# Set width and height of figure
plt.rcParams["figure.figsize"] = [10, 5]

figure, axis = plt.subplots(nrows=1, ncols=2)

model = Model(hidden_layers_shape=[100, 100],
               train_dataset_size=4,
               learning_rate=0.1,
               use_relu=True)
model.create_model_values()
model.train(epoch_count=4_700)
model.test()

axis[0].set_title("Use ReLu: True")
axis[0].set_xlabel("Epoch Count")
axis[0].set_ylabel("Loss Value")
axis[0].plot(np.squeeze(model.train_losses))

model = Model(hidden_layers_shape=[100, 100],
               train_dataset_size=4,
               learning_rate=0.1,
               use_relu=False)
model.create_model_values()
```

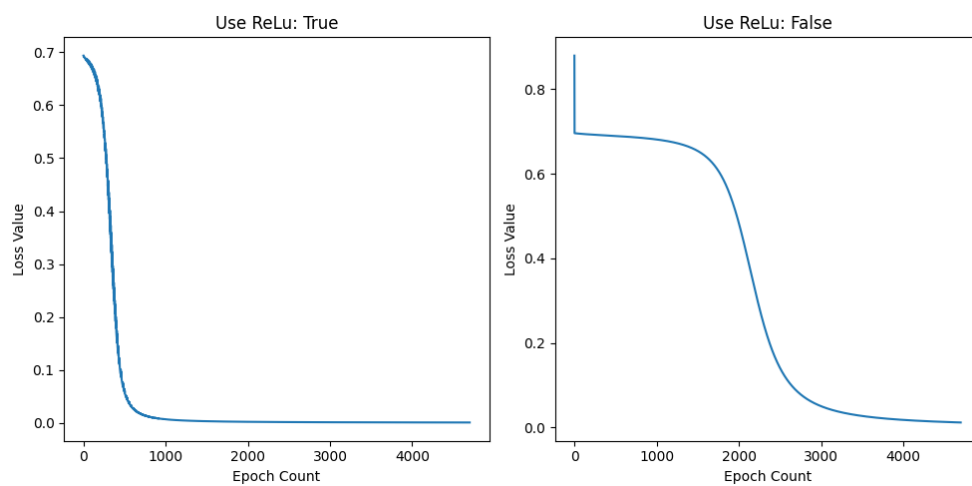
```

model.train(epoch_count=4_700)
model.test()

axis[1].set_title("Use ReLu: False")
axis[1].set_xlabel("Epoch Count")
axis[1].set_ylabel("Loss Value")
axis[1].plot(np.squeeze(model.train_losses))

plt.tight_layout()
plt.show()

```



As shown above, when using the ReLu transfer function along with the Sigmoid transfer function, the loss value decreases at a much faster rate than without. The model without the ReLu transfer function does reach the same accuracy but takes far more training epochs to do so.

Analysis of the impact of using a CPU vs GPU on training time taken

The following code trains a model on the XOR dataset using the CPU and then using the GPU to train, and then outputs the training time taken.

```
[5]: import os

from school_project.models.cpu.cat_recognition import CatRecognitionModel as CPUModel
from school_project.models.gpu.cat_recognition import CatRecognitionModel as GPUModel

# Change to root directory of project
os.chdir(os.getcwd())

model = CPUModel(hidden_layers_shape=[100, 100],
                  train_dataset_size=209,
                  learning_rate=0.1,
                  use_relu=True)
model.create_model_values()
model.train(epoch_count=3_500)

print(f"CPU Training Time: {model.training_time}s")

model = GPUModel(hidden_layers_shape=[100, 100],
                  train_dataset_size=209,
                  learning_rate=0.1,
                  use_relu=True)
model.create_model_values()
model.train(epoch_count=3_500)

print(f"GPU Training Time: {model.training_time}s")
```

CPU Training Time: 160.33s

GPU Training Time: 43.24s

As shown above, the GPU is almost four times faster at training the model than the CPU, showing how beneficial it is to utilise the parallel computations of the GPU

6.3 Conclusions

The principle conclusion from this analysis is that both the shape of the network and selection of other hyper-parameters is critical to develop an optimum Artificial Neural Network.

Firstly, when considering the shape of the network, higher numbers of neurons per layer and higher layer counts do not necessarily equate to an increase in network performance. As can be seen on page 118, increasing neuron count passes through an ideal value before prediction accuracy begins to drop. In a similar manner, as can be seen on page 115, increasing the number of layers passes through an optimum value before performance decreases. Both of these behaviours are likely due to overfitting of the network to the training dataset and was an outcome predicted during the interview I undertook at the beginning of the project. In fact, it was described to me that finding the optimum network shape is part analysis and part trial and error.

Secondly, key parameters within the forward and backward propagation methods have an impact on network performance. Most importantly, is the learning rate where selecting too small a value results in an increase in required epochs and training time - the model is also at risk of converging on a local minimum resulting in a suboptimal conclusion. Alternatively, selecting too high a learning rate can lead to an oscillation around the optimal solution. The nature of the transfer function similarly has a significant effect, with the ReLu transfer function reaching a lower loss value faster - it should be noted that the Sigmoid function did also converge but required more epochs.

With regard to the size of the training dataset, a larger dataset does appear to generate a better solution. I would expect the effect of this to reduce with a significant training dataset size but the size of the training dataset for Cat recognition did not reach this level.

Lastly the use of a GPU decreased training time by approximately a factor of 4, which is the result of the GPU architecture being optimized for matrix calculations.

7 Evaluation

7.1 Third Party Feedback

I demonstrated the final version of my program to the same third party that I interviewed in the analysis, and their response is shown below:

”In my opinion, Max has definitely met the primary and secondary goals of this project. Firstly, and most importantly, he has researched and implemented, from first principles, an Artificial Neural Network that is flexible and abstracted to the point that it can tackle a range of problems. Max started the analysis for this project from a very theoretical and mathematical point of view before implementing code which has allowed him to extend its implementation to a range of datasets from the XOR problem to image analysis.

I was particularly impressed at the level of analysis he undertook into how Artificial Neural Networks work and the impact that different kinds of design decisions can have on implementation. He took on board suggestions to explore different types of transfer functions such as the ReLu function to increase the speed of training and it was nice to see comparative studies of this and other techniques. It was also great to see the ability to save and load trained models which has allowed him to train models on a desktop PC equipped with a graphics card to be utilized on a lower power laptop.

The analysis section exploring the impact on both learning rates and epoch count was very nice to see, as well as the identification of an optimal learning rate suitable for the image dataset he was working with.

In summary, it was fantastic to see a true maths-to-code example of implementing Artificial Neural Networks that didn’t rely on the use of external AI libraries. I am certain he has learned a great deal regarding the fundamental properties and limitations of Artificial Neural Networks. I was also impressed by the usage of software engineering tools such as GitHub and Jupyter Notebook throughout the project.”

7.2 Project Objectives Evaluation

7.2.1 Project Objectives

For the reader’s convenience, I have restated the project objectives below:

Objective ID	Description
1	Learn how Artificial Neural Networks work and develop them from first principles
2	Implement the Artificial Neural Networks by creating trained models on image datasets
2.1	Allow use of Graphics Cards for faster training
2.2	Allow for the saving and loading of trained models
3	Develop a Graphical User Interface
3.1	Provide controls for hyper-parameters of models
3.2	Display and compare the results each model’s predictions

7.2.2 Project Objective Evaluations

Objective ID	Evaluation	Status	3rd Party Evaluation
1	I have learnt how Artificial Neural Networks work from online resources, reports and interviewing a subject matter expert. I have proven the key mathematical principles from first principles and implemented these structures within Python code.	Fully met	Fully met
2	I have implemented trainable Artificial Neural Networks with configurable numbers of layers, number of neurons in each layer and the nature of the Transfer Functions. The Artificial Neural Networks have been trained and tested on a variety of datasets and operates at an accuracy level comparable with the resources learnt from.	Fully met	Fully met
2.1	The Artificial Neural Networks allow the use of a graphics card where applicable.	Fully met	Fully met
2.2	The trained Artificial Neural Networks' weights and biases can be saved to a data file and the features of the corresponding Artificial Neural Networks are saved to a database. These saved Artificial Neural Networks can be loaded independently.	Fully met	Fully met
3	A Graphical User Interface allowing configuration of all hyper-parameters, loading and saving of trained models and testing has been developed.	Fully met	Fully met
3.1	The Graphical User Interface allows user configuration of all utilised model hyper-parameters.	Fully met	Fully met
3.2	The model predictions can be compared in terms of both learning rate and overall accuracy.	Fully met	Fully met

7.3 Requirements Evaluation

ID	Description	Status	3rd Party Evaluation
1	Learn how Artificial Neural Networks work	Fully met	Fully met
2	Develop Artificial Neural Networks from first principles		
2.1	Provide utilities for creating Artificial Neural Networks	Fully met	Fully met
2.2	Allow for the saving and loading of trained models' weights and biases	Fully met	Fully met
2.3	Allow use of Graphics Cards for faster training	Fully met	Fully met
3	Implement the Artificial Neural Networks on image datasets		
3.1	Allow input of unique hyper-parameters	Fully met	Fully met
3.2	Allow unique datasets and train dataset size to be loaded	Fully met	Fully met
4	Use a database to store a model's features and the location of its weights and biases	Fully met	Fully met
5	Develop a Graphical User Interface		
5.1	Provide controls for hyper-parameters of models	Fully met	Fully met
5.2	Display details of models' training	Fully met	Fully met
5.3	Display the results of each model's predictions	Fully met	Fully met
5.4	Allow for the saving of trained models	Fully met	Fully met
5.5	Allow for the loading of saved trained models	Fully met	Fully met

7.4 Future Improvements

By taking into consideration my evaluation of the objectives and feedback from the third party, I believe that the following future improvements could be made for the project:

- For the analysis of the effects of hyper-parameters, repeated tests seeded with different parts of the training dataset could provide a more accurate analysis of the average behaviour of the Artificial Neural Networks.
- Performing data augmentation to expand the size of the training datasets, such as by shifting, rotating, cropping and zooming into training images or by adding noise to training images to produce more.
- Exploring Convolutional Neural Networks.
- Utilising a standardized file format for storing trained Artificial Neural Networks, so that they can be integrated with other machine learning libraries.

References

- [1] Datacamp. Datacamp loss functions in machine learning explained. <https://www.datacamp.com/tutorial/loss-function-in-machine-learning>, 2023.

- [2] Lenny Delligatti. *SysML distilled: A brief guide to the systems modeling language*. Addison-Wesley, 2013.
- [3] Earl B Hunt. *Artificial intelligence*. Academic Press, 2014.
- [4] IBM. Ibm machine learning resources. <https://developer.ibm.com/technologies/machine-learning/>, 2023.
- [5] Marcopeix. Marcopeix. <https://github.com/marcopeix>, 2023.
- [6] Hala Nelson. *Essential Math for AI*. "O'Reilly Media, Inc.", 2023.
- [7] Researchgate. Researchgate. <https://www.researchgate.net>, 2023.
- [8] W3 schools. Json - introduction. https://www.w3schools.com/js/js_json_intro.asp, 2023.
- [9] Wikipedia. Mnist database. https://en.wikipedia.org/wiki/MNIST_database, 2023.