

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	Analysis	2
1.1	About	2
1.2	Interview	3
1.3	Project Objectives	4
1.4	Requirements	4
1.5	Theory behind Artificial Neural Networks	5
1.5.1	Structure	5
1.5.2	How Artificial Neural Networks learn	7
1.6	Theory Behind Deep Artificial Neural Networks	8
1.6.1	Setup	8
1.6.2	Forward Propagation:	9
1.6.3	Back Propagation:	9
1.7	Theory behind training the Artificial Neural Networks	11
1.7.1	Datasets	11
1.7.2	Theory behind using Graphics Cards to train Artificial Neural Networks	12
2	Design	13
2.1	Introduction	13
2.2	System Architecture	13
2.3	Class Diagrams	14
2.3.1	UI Class Diagram	14
2.3.2	Model Class Diagram	14
2.4	System Flow chart	15
2.5	Algorithms	15
2.6	Data Structures	16
2.7	File Structure	16
2.8	Database Design	17
2.9	Queries	17
2.10	Human-Computer Interaction	18
2.11	Hardware Design	23
2.12	Workflow and source control	23

3	Technical Solution	24
3.1	Setup	24
3.1.1	File Structure	24
3.1.2	Dependencies	27
3.1.3	Git and Github files	27
3.1.4	Organisation	32
3.2	models package	32
3.2.1	utils subpackage	32
3.2.2	Artificial Neural Network implementations	43
3.3	frames package	47
3.4	__main__.py module	59
4	Testing	68
4.1	Investigation	68
4.1.1	test_model module	68
4.1.2	Effects of Hyper-Parameters	77
4.2	Manual Testing	96
4.2.1	Input Validation Testing	96
4.3	Automated Testing	100
4.3.1	Unit Tests	100
4.3.2	GitHub Automated Testing	104
4.3.3	Docker	105
5	Evaluation TODO	105
5.1	Project Objectives Evaluation	105
5.1.1	Project Objectives	105
5.1.2	Project Objective Evaluations	106
5.2	Third Party Feedback TODO	106
5.3	Future Improvements TODO	106

1 Analysis

1.1 About

Artificial Intelligence mimics human cognition in order to perform tasks and learn from them, Machine Learning is a subfield of Artificial Intelligence that uses algorithms trained on data to produce models (trained programs) and Deep Learning is a 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 learn a vast number of problems, such as Image Recognition, and have uses across multiple fields, such as medical imaging in hospitals. This project is an investigation into how Artificial Neural Networks work, the effects of changing their hyper-parameters and their applications in Image Recognition. To achieve this, I will derive and research all theory behind the project, using sources such as IBM's online research, and develop Neural Networks from first principles without the use of any third-party Machine Learning libraries. I then will implement the Artificial Neural Networks in Image Recognition, by creating trained models and will allow for experimentation of the hyper-parameters of each model to allow for comparisons between each model's performances, via a Graphical User Interface.

1.2 Interview

In order to gain a better foundation for my investigation, I presented my prototype code and interviewed the head of Artificial Intelligence at Cambridge Consultants for input on what they would like to see in my project, 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.”

1.3 Project 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 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

1.4 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 5	N/A
2	Develop Artificial Neural Networks from first principles		
2.1	Provide utilities for creating Artificial Neural Networks	Page 32	Page 100
2.2	Allow for the saving and loading of trained models' weights and biases	Page 36	Page 100
2.3	Allow use of Graphics Cards for faster training	Code not included in report	Page 95
3	Implement the Artificial Neural Networks on image datasets		
3.1	Allow input of unique hyper-parameters	Page 43	Page 77
3.2	Allow unique datasets and train dataset size to be loaded	Page 43	Page 84
4	Use a database to store a model's features and the location of its weights and biases	Page 59	TODO Unit Test
5	Develop a Graphical User Interface		
5.1	Provide controls for hyper-parameters of models	Page 48	Page 96
5.2	Display details of models' training	Page 48	N/A
5.3	Display the results of each model's predictions	Page 68	User Tested
5.4	Allow for the saving of trained models	Page 68	Page 99
5.5	Allow for the loading of saved trained models	Page 55	Page 98

1.5 Theory behind Artificial Neural Networks

From an abstract perspective, Artificial Neural Networks are inspired by how the human mind works, by consisting of layers of 'neurons' all interconnected via different links, each with their own strength. By adjusting these links, Artificial Neural Networks can be trained to take in an input and give its best prediction as an output.

1.5.1 Structure

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 learnt two types of Feed-Forward Artificial Neural Networks: Perceptron Artificial Neural Networks, that contain no hidden layers and are best at learning more linear patterns and Multi-Layer Perceptron Artificial Neural Networks, that contain at least one hidden layer, as a result increasing the non-linearity in the Artificial Neural Network and allowing it to learn 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.

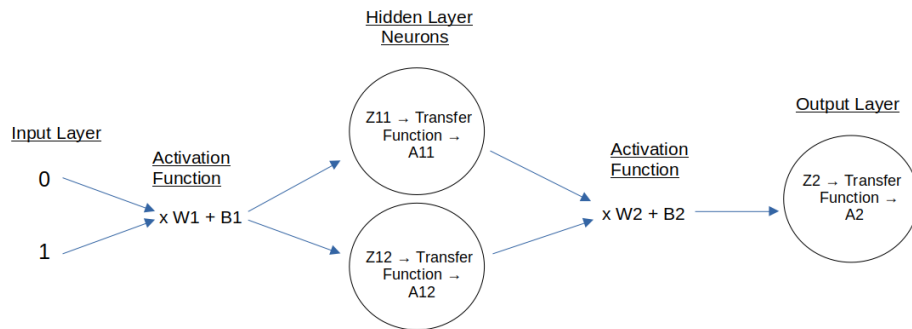


Figure 1: This shows an Artificial Neural Network with one single hidden layer and is known as a Shallow Neural Network.

To implement an Artificial Neural Network, matrices are 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 known about them:

- When Adding two matrices, both matrices must have the same number of rows and columns. Or one of the matrices can have the same number of rows but only one column, then be added by element-wise addition where each element is added to all of the elements of the other matrix in the same row.
- In order to multiply matrices, I take the 'dot product' of the matrices, which multiplies the row of one matrix with the column of the other, by multiplying matching members and then summing up.
- When taking the dot product of two matrices, the number of columns of the 1st matrix must equal the number of rows of the 2nd matrix. And the result will have the same number of rows as the 1st matrix, and the same number of columns as the 2nd matrix. This is important, as the output of one layer must be formatted correctly to be used with the next layer.
- Alternatively, at times I take the Hadamard product of two matrices which performs element-wise multiplication of the matrices. For this, both matrices must have the same number of rows and columns.
- Transposing a matrix will turn all rows of the matrix into columns and all columns into rows.
- 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 just using Fully-Connected layers, that will take in input values and apply the following calculations to produce an output of the layer:

- An Activation function

- This calculates the dot product of the input matrix with a weight matrix, then sums the result with a bias matrix
- A Transfer function
 - This takes the result of the Activation function and transfers it to a suitable output value as well as adding more non-linearity to the Neural Network.
 - For example, the Sigmoid Transfer function converts the input to a number between zero and one, making it useful for logistic regression where the output value can be considered as closer to zero or one allowing for a binary classification of predicting zero or one.

1.5.2 How Artificial Neural Networks learn

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

- Forward Propagation:
 - The process of feeding inputs in and getting a prediction (moving forward through the network)
- Back Propagation:
 - The process of calculating 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 values it should have predicted. With this, I can calculate the Loss value of the prediction (how wrong the prediction is from the actual value).
 - I then move back through the network and update the weights and biases via Gradient Descent:
 - * Gradient Descent 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, you must use the following calculus methods:
 - Partial Differentiation, in order to differentiate the multi-variable functions, by taking respect to one variable and treating the rest as constants.
 - The Chain Rule, where for $y = f(u)$ and $u = g(x)$, $\frac{\partial y}{\partial u} * \frac{\partial u}{\partial x} =$
 - For a matrix of $f(x)$ values, the matrix of $\frac{\partial f(x)}{\partial x}$ values is known as the Jacobian matrix

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

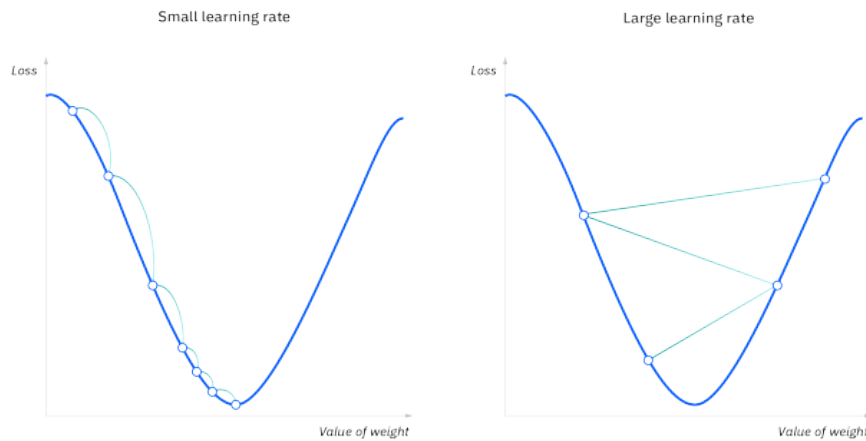


Figure 2: Gradient Descent
sourced from <https://www.ibm.com/topics/gradient-descent>

- * However, during backpropagation 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', where the gradient value grows exponentially to the point of overflow errors
 - Having a 'Vanishing Gradient', where the gradient value decreases to a very small value or zero, resulting in a lack of updating values during training.

1.6 Theory Behind Deep Artificial Neural Networks

1.6.1 Setup



Figure 3: This shows an abstracted view of an Artificial Neural Network with multiple hidden layers and is known as a Deep Neural Network.

- Where a layer takes the previous layer's output as its input X
- Then it applies an Activation function to X to obtain Z , by taking the dot product of X with a weight matrix W , then sums the result with a bias matrix B . At first the weights are initialised to random values and the biases are set to zeros.

$$- Z = W * X + B$$

- Then it applies a Transfer function to Z to obtain the layer's output
 - For the output layer, the sigmoid function (explained previously) must be used for either for binary classification via logistic regression, or for multi- class classification where it predicts the output neuron, and the associated class, that has the highest value between zero and one.

$$* \text{ Where } \text{sigmoid}(Z) = \frac{1}{1+e^{-Z}}$$

- 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, so updates at a quicker rate.

$$* \text{ Where } \text{relu}(Z) = \max(0, Z)$$

1.6.2 Forward Propagation:

- For each epoch the input layer is given a matrix of input values, which are fed through the network to obtain a final prediction A from the output layer.

1.6.3 Back Propagation:

- First the Loss value L is calculated using the following Log-Loss function, which calculates the average difference between A and the value it should have predicted Y . Then the average is found by summing the result of the Loss function for each value in the matrix A , then dividing by the number of predictions m , resulting in a Loss value to show 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
- I then move back through the network, adjusting the weights and biases via Gradient Descent. For each layer, the weights and biases are updated with the following formulae:
 - $W = W - learningRate * \frac{\partial L}{\partial W}$
 - $B = B - learningRate * \frac{\partial L}{\partial B}$
- The derivation for Layer 2's $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial B}$ can be seen 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, $\frac{\partial Z2}{\partial W2} = A1$
 $\Rightarrow \frac{\partial L}{\partial W2} = \frac{\partial L}{\partial A2} * \frac{\partial A2}{\partial Z2} * A1$
 - $\frac{\partial L}{\partial B2} = \frac{\partial L}{\partial A2} * \frac{\partial A2}{\partial Z2} * \frac{\partial Z2}{\partial B2}$
 By using function 1, $\frac{\partial Z2}{\partial B2} = 1$
 $\Rightarrow \frac{\partial L}{\partial W2} = \frac{\partial L}{\partial A2} * \frac{\partial A2}{\partial Z2} * 1$
- As you can see, when moving back through the network, the $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial B}$ of the layer can be calculated with the rate of change of loss with respect to its output, which 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, $\frac{\partial L}{\partial A} = (-\frac{1}{m}) * (\frac{Y-A}{A*(1-A)})$

1.7 Theory behind training the Artificial Neural Networks

Training an Artificial Neural Network's weights and biases to predict on a dataset, will create a trained model for that dataset, so that it can predict on future images inputted. However, training Artificial Neural Networks can involve some problems such as Overfitting, where the trained model learns the patterns of the training dataset too well, causing worse prediction on a different test dataset. 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 and having too many layers in the Neural Network. Another 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 simple (too linear).

1.7.1 Datasets

- MNIST dataset
 - The MNIST dataset is a famous dataset of images of handwritten 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, made up from 28x28 pixels and each pixel has an RGB 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 RGB values are 'flattened' into a 1 dimensional matrix of values, where each element is also divided by 255 (the max RGB value) to a number between 0 and 1, to standardize the dataset.
 - The output dataset is also loaded, where each output for each image is an array, where the index represents the number of the image, by having a 1 in the index that matches the number represented and zeros for all other indexes.
 - To create a trained Artificial Neural Network model on this dataset, the model will require 10 output neurons (one for each digit), then by using the Sigmoid Transfer function to output a number between one and zero to each neuron, whichever neuron has the highest value is predicted. This is 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).
- Cat dataset
 - I will also use a dataset of images sourced from <https://github.com/marcopeix>, where each image is either a cat or not a cat.
 - The dataset consists of 209 input images, made up from 64x64 pixels and each pixel has an RGB 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 RGB values are 'flattened' into a 1 dimensional array of values, where each element is also divided by 255 (the max RGB value) to a number between 0 and 1, to standardize the dataset.
 - The output dataset is also loaded, and is reshaped into a 1 dimensional array of 1s and 0s, to store the output of each image (1 for cat, 0 for non cat)
 - To create a trained Artificial Neural Network model on this dataset, the model will require only 1 output neuron, then by using the Sigmoid Transfer function to output a number between one and zero for the neuron, if the neuron's value is closer to 1 it predicts cat, otherwise it predicts not a cat. This is binary classification, where the model must use logistic regression to predict whether it is a cat or not a cat.
- XOR dataset
 - For experimenting with Artificial Neural Networks, I solve 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 much less computation time than image datasets, so is useful for quickly comparing different hyper-parameters of a Network, whilst still not being linearly separable.

1.7.2 Theory behind using Graphics Cards to train Artificial Neural Networks

Graphics Cards consist of many Tensor cores which are processing units specialised for matrix operations for calculating the co-ordinates of 3D graphics, however they can be used here 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 Design

2.1 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.
- I will take an object-orientated approach to the project.
- I will give an option to use either a Graphics Card or a CPU to train and test the Artificial Neural Networks.

I will also be using SysML for designing the following diagrams.

2.2 System Architecture

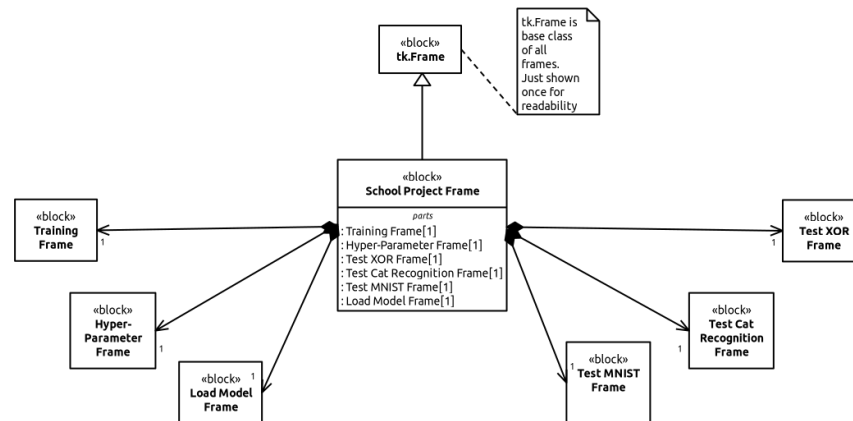
bdd [block] School Project Frame [System Architecture Diagram]



2.3 Class Diagrams

2.3.1 UI Class Diagram

bdd [package] School Project [UI Class Diagram]



2.3.2 Model Class Diagram

bdd [package] School Project [Model Class Diagram]



2.4 System Flow chart

act [activity] System Flow chart [System Flow chart]



2.5 Algorithms

Refer to Analysis for the algorithms behind the Artificial Neural Networks.

2.6 Data Structures

I will use the following data structures in the program:

- 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 will allow me to traverse both forwards and backwards through the network, as well as storing the first and last layer to start forward and backward propagation respectively.

2.7 File Structure

I will use the following file structures to store necessary data for the program:

- A JSON file for storing the default hyper-parameters for creating a new model for each dataset.
- I will store the image dataset files in a 'datasets' directory. The dataset files will either be 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.
- I will save the weights and biases of saved models as numpy arrays in .npz files (a zipped archive file format) in a 'saved-models' directory, due to their compatibility with the numpy library.

2.8 Database Design

I will use the following Relational database design 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:

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

- I will also use the following unique constraint, so that each dataset can not have more than one model with the same name:

```
UNIQUE (Dataset, Name)
```

2.9 Queries

Here are some example queries for interacting with the database:

- I can query the names of all saved models for a dataset with:

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

- I can query the file location of a saved model with:

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

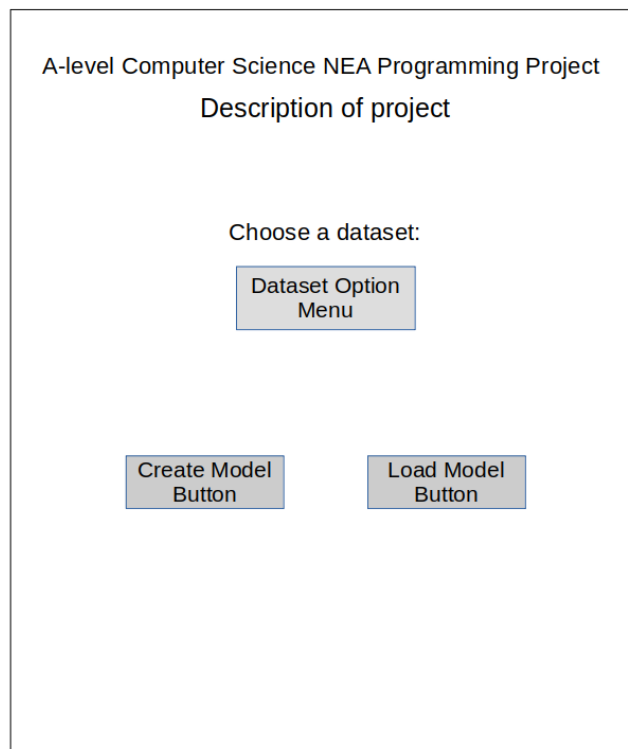
- I can query the features of a saved model with:

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

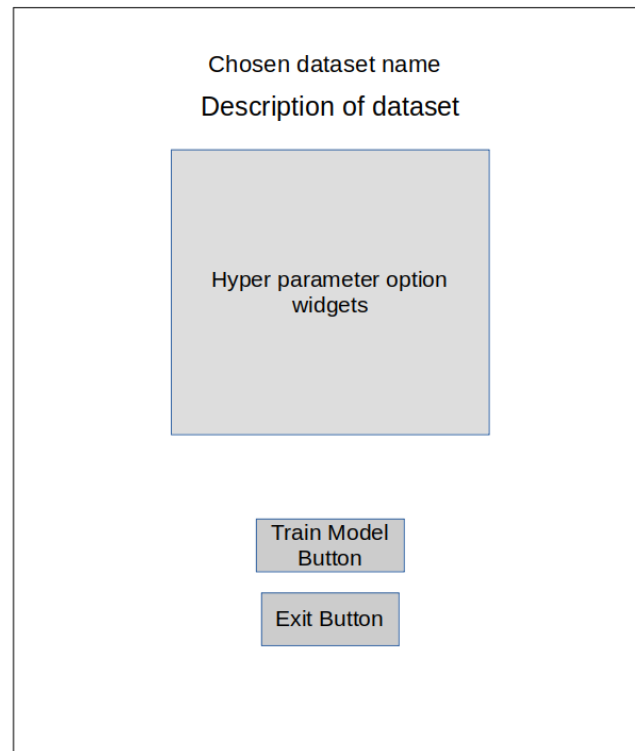
2.10 Human-Computer Interaction

Here are the designs of each tkinter frame in the User Interface:

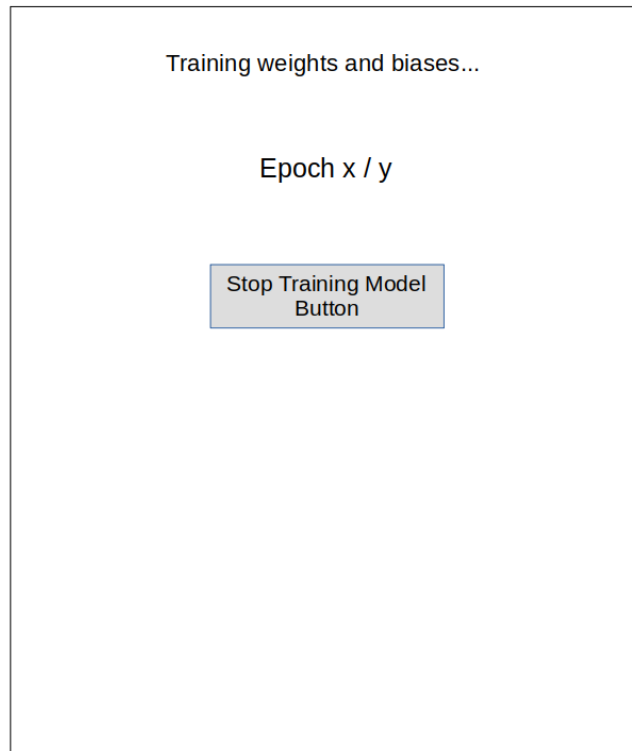
- Home Frame design:



- Hyper-Parameter Frame design:



- Training Frame design:
 - During training, the following is displayed on the Training Frame:



- Once training has finished, the following is displayed on the Training Frame:



- Load Model Frame design:



- Test Frame design:

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

2.11 Hardware Design

To allow for faster training of an Artificial Neural Network, I will give the option to use a Graphics Card to train the Artificial Neural Network if available. I will also give the option to load pretrained weights to run on less computationally powerful hardware using just the CPU as standard.

2.12 Workflow and source control

I will use Git along with GitHub to manage my workflow and source control as I develop the project, by utilising the following features:

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

3 Technical Solution

3.1 Setup

3.1.1 File Structure

I used the following file structure 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
|-- setup.py
|-- TODO.md
```

18 directories, 49 files

Each package within the school_project package contains a `__init__.py` file, which allows 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.

- Here is the contents of the frames package's `__init__.py` for example, which allows the classes of all modules in the package to be imported at once:

```
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,
6  ↪ TestXORFrame
7  __all__ = ['create_model', 'load_model', 'test_model']
```

I have omitted the source code for this report, which included a Makefile for its compilation.

3.1.2 Dependencies

The python dependencies for the project can be installed simply 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 next section also.

- `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 )
```

3.1.3 Git and Github files

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

```

```

99     sudo apt-get install x11-xserver-utils
100     xhost +
101     sudo docker run -v /tmp/.X11-unix:/tmp/.X11-unix -e
102     ↩ DISPLAY=unix$DISPLAY mcttn22/school-project
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:

Tests passing Python 3

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

- A LICENSE file that describes how others can use my code.

3.1.4 Organisation

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

3.2 models package

This package is a self-contained package for creating trained Artificial Neural Networks and can either be used for 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 utilise NumPy and another library CuPy which requires a GPU to be utilised for operations with the matrices. For that reason it is only worth showing the code for the cpu subpackage.

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.

3.2.1 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
↪ layer
12         with the method given.
13
14         Args:
15             setup_values (callable): the method that sets up the values of
↪ each
16             layer.
17         Raises:
18             NotImplementedError: if this method is not implemented.
19
20         """
21         raise NotImplementedError
22
23     @abstractmethod
24     def create_model_values(self) -> None:
25         """Create weights and bias/biases
26
27         Raises:
28             NotImplementedError: if this method is not implemented.
29
30         """
31         raise NotImplementedError
32
33     @abstractmethod
34     def load_model_values(self, file_location: str) -> None:
35         """Load weights and bias/biases from .npz file.
36
37         Args:
38             file_location (str): the location of the file to load from.
39         Raises:
40             NotImplementedError: if this method is not implemented.
41
42         """
43         raise NotImplementedError
44
45     @abstractmethod
46     def load_datasets(self, train_dataset_size: int) -> tuple[np.ndarray,
↪ np.ndarray,
47                                                                np.ndarray,
↪ np.ndarray]:
48         """Load input and output datasets. For the input dataset, each
↪ column
49         should represent a piece of data and each row should store the
↪ values
50         of the piece of data.
51
52         Args:
53             train_dataset_size (int): the number of train dataset inputs to
↪ use.

```

```

54         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
↪ do.
103         Raises:
104             NotImplementedError: if this method is not implemented.
105
106         """
107         raise NotImplementedError
108
109     @abstractmethod
110     def save_model_values(self, file_location: str) -> None:
111         """Save the model by saving the weights then biases of each layer
↪ to
112             a .npz file with a given file location.
113

```

```

114         Args:
115             file_location (str): the file location to save the model to.
116
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.

```

```

174     Returns:
175         numpy.ndarray / float,
176         derivative of the sigmoid transfer function with respect to z.
177     Raises:
178         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:

```

```

75         dloss_doutput (numpy.ndarray): the derivative of the loss of
↪ the
76         layer's output, with respect to the layer's output.
77     Returns:
78         a numpy.ndarray derivative of the loss of the layer's input,
79         with respect to the layer's input.
80     Raises:
81         ValueError:
82             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 *
↪ sigmoid_derivative(output=self.output)
90         case 'relu':
91             dloss_dz = dloss_doutput *
↪ relu_derivative(output=self.output)
92
93     dloss_dweights = np.dot(dloss_dz, self.input.T)
94     dloss_dbases = np.sum(dloss_dz)
95
96     assert dloss_dweights.shape == self.weights.shape
97
98     dloss_dinput = np.dot(self.weights.T, dloss_dz)
99
100     # Update weights and biases
101     self.weights -= self.learning_rate * dloss_dweights
102     self.biases -= self.learning_rate * dloss_dbases
103
104     return dloss_dinput
105
106 def forward_propagation(self, inputs) -> np.ndarray:
107     """Generate a layer output with the weights and biases.
108
109     Args:
110         inputs (np.ndarray): the input values to the layer.
111     Returns:
112         a numpy.ndarray of the output values.
113
114     """
115     self.input = inputs
116     z = np.dot(self.weights, self.input) + self.biases
117     if self.transfer_type == 'sigmoid':
118         self.output = sigmoid(z)
119     elif self.transfer_type == 'relu':
120         self.output = relu(z)
121     return self.output
122
123 class _Layers():
124     """Manages linked list of layers."""
125     def __init__(self) -> None:
126         """Initialise linked list."""
127         self.head = None
128         self.tail = None
129
130     def __iter__(self) -> Generator[_FullyConnectedLayer, None, None]:
131         """Iterate forward through the network."""
132         current_layer = self.head
133         while True:

```

```

134         yield current_layer
135         if current_layer.next_layer is not None:
136             current_layer = current_layer.next_layer
137         else:
138             break
139
140     def __reversed__(self) -> Generator[_FullyConnectedLayer, None, None]:
141         """Iterate back through the network."""
142         current_layer = self.tail
143         while True:
144             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
174             ↪ train_dataset_size=train_dataset_size
175             )
176
177         self.train_losses: list[float]
178         self.test_prediction: np.ndarray
179         self.test_prediction_accuracy: float
180         self.training_progress = ""
181         self.training_time: float
182
183         # Setup model attributes
184         self._running = True
185         self.input_neuron_count: int = self.train_inputs.shape[0]
186         self.input_count = self.train_inputs.shape[1]
187         self.hidden_layers_shape = hidden_layers_shape
188         self.output_neuron_count = self.train_outputs.shape[0]
189         self.layers_shape = [f'{layer}' for layer in (
190             [self.input_neuron_count] +
191             self.hidden_layers_shape +
192             [self.output_neuron_count]
193         )]
194
195         self.use_relu = use_relu
196
197         # Setup model values

```

```

193         self.layers = _Layers()
194         self.learning_rate = learning_rate
195
196     def __repr__(self) -> str:
197         """Read current state of model.
198
199         Returns:
200             a string description of the model's shape,
201             weights, bias and learning rate values.
202
203         """
204         return (f"Layers Shape: {'.'.join(self.layers_shape)}\n" +
205                 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
↪ layer
218         with the method given.
219
220         Args:
221             setup_values (callable): the method that sets up the values of
↪ each
222             layer.
223
224         """
225     def decorator(self, *args, **kwargs) -> None:
226         # Check if setting up Deep Network
227         if len(self.hidden_layers_shape) > 0:
228             if self.use_relu:
229
230                 # Add input layer
231                 self.layers.head = _FullyConnectedLayer(
232
233                     ↪ learning_rate=self.learning_rate,
234
235                     ↪ input_neuron_count=self.input_neuron_count,
236
237                     ↪ output_neuron_count=self.hidden_layers_shape[0],
238                     transfer_type='relu'
239                 )
240                 current_layer = self.layers.head
241
242                 # Add hidden layers
243                 for layer in range(len(self.hidden_layers_shape) - 1):
244                     current_layer.next_layer = _FullyConnectedLayer(
245                         learning_rate=self.learning_rate,
246
247                         ↪ input_neuron_count=self.hidden_layers_shape[layer],
248
249                         ↪ output_neuron_count=self.hidden_layers_shape[layer
250                         ↪ + 1],
251                         transfer_type='relu'
252                     )

```



```

247         current_layer.next_layer.previous_layer =
248             ↪ current_layer
249         current_layer = current_layer.next_layer
250     else:
251         # Add input layer
252         self.layers.head = _FullyConnectedLayer(
253             ↪ learning_rate=self.learning_rate,
254             ↪ input_neuron_count=self.input_neuron_count,
255             ↪ output_neuron_count=self.hidden_layers_shape[0],
256             transfer_type='sigmoid'
257         )
258         current_layer = self.layers.head
259         # Add hidden layers
260         for layer in range(len(self.hidden_layers_shape) - 1):
261             current_layer.next_layer = _FullyConnectedLayer(
262                 learning_rate=self.learning_rate,
263                 ↪ input_neuron_count=self.hidden_layers_shape[layer],
264                 ↪ output_neuron_count=self.hidden_layers_shape[layer
265                     ↪ + 1],
266                 transfer_type='sigmoid'
267             )
268             current_layer.next_layer.previous_layer =
269                 ↪ current_layer
270             current_layer = current_layer.next_layer
271         # Add output layer
272         current_layer.next_layer = _FullyConnectedLayer(
273             learning_rate=self.learning_rate,
274             ↪ input_neuron_count=self.hidden_layers_shape[-1],
275             ↪ output_neuron_count=self.output_neuron_count,
276             transfer_type='sigmoid'
277         )
278         current_layer.next_layer.previous_layer = current_layer
279         self.layers.tail = current_layer.next_layer
280     # Setup Perceptron Network
281     else:
282         self.layers.head = _FullyConnectedLayer(
283             learning_rate=self.learning_rate,
284             ↪ input_neuron_count=self.input_neuron_count,
285             ↪ output_neuron_count=self.output_neuron_count,
286             transfer_type='sigmoid'
287         )
288         self.layers.tail = self.layers.head
289     setup_values(self, *args, **kwargs)
290
291     return decorator
292
293 @setup_layers
294 def create_model_values(self) -> None:

```

```

297         """Create weights and bias/biases"""
298         # Check if setting up Deep Network
299         if len(self.hidden_layers_shape) > 0:
300
301             # Initialise Layer values to random values
302             for layer in self.layers:
303                 layer.init_layer_values_random()
304
305         # Setup Perceptron Network
306         else:
307
308             # Initialise Layer values to zeros
309             for layer in self.layers:
310                 layer.init_layer_values_zeros()
311
312     @setup_layers
313     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:

```

```

357         output = layer.forward_propagation(inputs=output)
358     self.test_prediction = output
359
360     # Calculate performance of model
361     self.test_prediction_accuracy = calculate_prediction_accuracy(
362
363                                     ↪ prediction=self.test_prediction,
364                                     ↪ outputs=self.test_outputs
365                                     )
366
367     def train(self, epoch_count: int) -> None:
368         """Train layers' weights and biases.
369
370         Args:
371         epoch_count (int): the number of training epochs.
372
373         """
374         self.layers_shape = [f'{layer}' for layer in (
375             [self.input_neuron_count] +
376             self.hidden_layers_shape +
377             [self.output_neuron_count]
378         )]
379         self.train_losses = []
380         training_start_time = time.time()
381         for epoch in range(epoch_count):
382             if not self.__running:
383                 break
384             self.training_progress = f"Epoch {epoch} / {epoch_count}"
385             prediction = self.forward_propagation()
386             loss = calculate_loss(input_count=self.input_count,
387                                 ↪ outputs=self.train_outputs,
388                                 ↪ prediction=prediction)
389             self.train_losses.append(loss)
390             if not self.__running:
391                 break
392             dloss_doutput = -(1/self.input_count) * ((self.train_outputs -
393                 ↪ prediction)/(prediction * (1 - prediction)))
394             self.back_propagation(dloss_doutput=dloss_doutput)
395             self.training_time = round(number=time.time() -
396                 ↪ training_start_time,
397                                     ↪ ndigits=2)
398
399     def save_model_values(self, file_location: str) -> None:
400         """Save the model by saving the weights then biases of each layer
401         ↪ to
402         a .npz file with a given file location.
403
404         Args:
405         file_location (str): the file location to save the model to.
406
407         """
408         saved_model: list[np.ndarray] = []
409         for layer in self.layers:
410             saved_model.append(layer.weights)
411             saved_model.append(layer.biases)
412         np.savez(file_location, *saved_model)

```

3.2.2 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
    ↪ dataset."""
2
3  import h5py
4  import numpy as np
5
6  from .utils.model import AbstractModel
7
8  class CatRecognitionModel(AbstractModel):
9      """ANN model that trains to predict if an image is a cat or not a
    ↪ cat."""
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
    ↪ use.
21             learning_rate (float): the learning rate of the model.
22             use_relu (bool): True or False whether the ReLu Transfer
    ↪ function
23                 should be used.
24
25         """
26         super().__init__(hidden_layers_shape=hidden_layers_shape,
27                          train_dataset_size=train_dataset_size,
28                          learning_rate=learning_rate,
29                          use_relu=use_relu)
30
31     def load_datasets(self, train_dataset_size: int) -> tuple[np.ndarray,
    ↪ np.ndarray,
32                                                                np.ndarray,
    ↪ np.ndarray]:
33
34         """Load image input and output datasets.
35
36         Args:
37             train_dataset_size (int): the number of train dataset inputs to
    ↪ use.
38         Returns:
39             tuple of image train_inputs, train_outputs,
40             test_inputs and test_outputs numpy.ndarrays.
41
42         Raises:
43             FileNotFoundError: if file does not exist.
44
45         """
46         # Load datasets from h5 files
47         # (h5 files stores large amount of data with quick access)
48         train_dataset: h5py.File = h5py.File(
49             r'school_project/models/datasets/train-cat.h5',
50             'r'
51         )
52         test_dataset: h5py.File = h5py.File(
53             r'school_project/models/datasets/test-cat.h5',
54             'r'

```

```

54         )
55
56     # Load input arrays,
57     # containing the RGB values for each pixel in each 64x64 pixel
58     ↪ image,
59     # for 209 images
60     train_inputs: np.ndarray =
61     ↪ np.array(train_dataset['train_set_x'][:])
62     test_inputs: np.ndarray = np.array(test_dataset['test_set_x'][:])
63
64     # Load output arrays of 1s for cat and 0s for not cat
65     train_outputs: np.ndarray =
66     ↪ np.array(train_dataset['train_set_y'][:])
67     test_outputs: np.ndarray = np.array(test_dataset['test_set_y'][:])
68
69     # Reshape input arrays into 1 dimension (flatten),
70     # then divide by 255 (RGB)
71     # to standardize them to a number between 0 and 1
72     train_inputs = train_inputs.reshape((train_inputs.shape[0],
73     -1)).T / 255
74     test_inputs = test_inputs.reshape((test_inputs.shape[0], -1)).T /
75     ↪ 255
76
77     # Reshape output arrays into a 1 dimensional list of outputs
78     train_outputs = train_outputs.reshape((1, train_outputs.shape[0]))
79     test_outputs = test_outputs.reshape((1, test_outputs.shape[0]))
80
81     # Reduce train datasets' sizes to train_dataset_size
82     train_inputs = (train_inputs.T[:train_dataset_size]).T
83     train_outputs = (train_outputs.T[:train_dataset_size]).T
84
85     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.

```

```

25
26         """
27         super().__init__(hidden_layers_shape=hidden_layers_shape,
28                           train_dataset_size=train_dataset_size,
29                           learning_rate=learning_rate,
30                           use_relu=use_relu)
31
32     def load_datasets(self, train_dataset_size: int) -> tuple[np.ndarray,
33     ↪ np.ndarray,
34                                                           np.ndarray,
35                                                           ↪ np.ndarray]:
36
37         """Load image input and output datasets.
38         Args:
39             train_dataset_size (int): the number of dataset inputs to use.
40         Returns:
41             tuple of image train_inputs, train_outputs,
42             test_inputs and test_outputs numpy.ndarrays.
43
44         Raises:
45             FileNotFoundError: if file does not exist.
46
47         """
48         # Load datasets from.pkl.gz file
49         with gzip.open(
50             'school_project/models/datasets/mnist.pkl.gz',
51             'rb'
52         ) as mnist:
53             (train_inputs, train_outputs), \
54             (test_inputs, test_outputs) = pickle.load(mnist,
55             ↪ encoding='bytes')
56
57         # Reshape input arrays into 1 dimension (flatten),
58         # then divide by 255 (RGB)
59         # to standardize them to a number between 0 and 1
60         train_inputs =
61         ↪ np.array(train_inputs.reshape((train_inputs.shape[0],
62                                         -1)).T / 255)
63
64         test_inputs = np.array(test_inputs.reshape(test_inputs.shape[0],
65         ↪ -1).T / 255)
66
67         # Represent number values
68         # with a one at the matching index of an array of zeros
69         train_outputs = np.eye(np.max(train_outputs) + 1)[train_outputs].T
70         test_outputs = np.eye(np.max(test_outputs) + 1)[test_outputs].T
71
72         # Reduce train datasets' sizes to train_dataset_size
73         train_inputs = (train_inputs.T[:train_dataset_size]).T
74         train_outputs = (train_outputs.T[:train_dataset_size]).T
75
76         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)
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 XOR input and output datasets.
39
40         Args:
41             train_dataset_size (int): the number of dataset inputs to use.
42         Returns:
43             tuple of XOR train_inputs, train_outputs,
44             test_inputs and test_outputs numpy.ndarrays.
45
46         """
47         inputs: np.ndarray = np.array([[0, 0, 1, 1],
48                                       [0, 1, 0, 1]])
49         outputs: np.ndarray = np.array([[0, 1, 1, 0]])
50
51         # Reduce train datasets' sizes to train_dataset_size
52         inputs = (inputs.T[:train_dataset_size]).T
53         outputs = (outputs.T[:train_dataset_size]).T
54
55         return inputs, outputs, inputs, outputs

```

3.3 frames package

I decided to use tkinter for the User Interface and the frames package 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
↪ colour.
23             dataset (str): the name of the dataset to use
24             ('MNIST', 'Cat Recognition' or 'XOR')
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 hyper-parameter frame variables
36     self.dataset = dataset
37     self.use_gpu: bool
38     self.default_hyper_parameters = self.load_default_hyper_parameters(
39
40                                     ↪ dataset=dataset
41                                     )
42
43     # Setup widgets
44     self.title_label = tk.Label(master=self,
45                                 bg=self.BG,
46                                 font=('Arial', 20),
47                                 text=dataset)
48     self.about_label = tk.Label(
49         master=self,
50         bg=self.BG,
51         font=('Arial', 14),
52
53         ↪ text=self.default_hyper_parameters['description']
54         )
55     self.learning_rate_scale = tk.Scale(
56         master=self,
57         bg=self.BG,
58         orient='horizontal',
59         label="Learning Rate",
60         length=185,
61         from_=0,
62
63         ↪ to=self.default_hyper_parameters['maxLearningRate'],
64         resolution=0.01
65         )
66     self.learning_rate_scale.set(value=0.1)
67     self.epoch_count_scale = tk.Scale(master=self,
68                                       bg=self.BG,
69                                       orient='horizontal',
70                                       label="Epoch Count",
71                                       length=185,
72                                       from_=0,
73                                       to=10_000,
74                                       resolution=100)
75     self.epoch_count_scale.set(
76
77         ↪ value=self.default_hyper_parameters['epochCount']
78         )
79     self.train_dataset_size_scale = tk.Scale(
80         master=self,
81         bg=self.BG,
82         orient='horizontal',
83         label="Train Dataset Size",
84         length=185,
85
86         ↪ from_=self.default_hyper_parameters['minTrainDatasetSize'],
87         to=self.default_hyper_parameters['maxTrainDatasetSize'],
88         resolution=1
89         )
90     self.train_dataset_size_scale.set(
91
92         ↪ value=self.default_hyper_parameters['maxTrainDatasetSize']
93         )

```

```

88     self.hidden_layers_shape_label = tk.Label(
89         master=self,
90         bg=self.BG,
91         font=('Arial', 12),
92         text="Enter the number of neurons in
           ↳ each\n" +
           "hidden layer, separated by
           ↳ commas:"
93     )
94
95     self.hidden_layers_shape_entry = tk.Entry(master=self)
96     self.hidden_layers_shape_entry.insert(0, ",".join(
97         f"{neuron_count}" for neuron_count in
           ↳ self.default_hyper_parameters['hiddenLayersShape']
98     ))
99     self.use_relu_check_button_var = tk.BooleanVar(value=True)
100    self.use_relu_check_button = tk.Checkbutton(
101        master=self,
102        width=13, height=1,
103        font=tkf.Font(size=12),
104        text="Use ReLu",
105        variable=self.use_relu_check_button_var
106    )
107    self.use_gpu_check_button_var = tk.BooleanVar()
108    self.use_gpu_check_button = tk.Checkbutton(
109        master=self,
110        width=13, height=1,
111        font=tkf.Font(size=12),
112        text="Use GPU",
113        variable=self.use_gpu_check_button_var
114    )
115    self.model_status_label = tk.Label(master=self,
116        bg=self.BG,
117        font=('Arial', 15))
118
119    # Pack widgets
120    self.title_label.grid(row=0, column=0, columnspan=3)
121    self.about_label.grid(row=1, column=0, columnspan=3)
122    self.learning_rate_scale.grid(row=2, column=0, pady=(50,0))
123    self.epoch_count_scale.grid(row=3, column=0, pady=(30,0))
124    self.train_dataset_size_scale.grid(row=4, column=0, pady=(30,0))
125    self.hidden_layers_shape_label.grid(row=2, column=1,
126        padx=30, pady=(50,0))
127    self.hidden_layers_shape_entry.grid(row=3, column=1, padx=30)
128    self.use_relu_check_button.grid(row=2, column=2, pady=(30, 0))
129    self.use_gpu_check_button.grid(row=3, column=2, pady=(30, 0))
130    self.model_status_label.grid(row=5, column=0,
131        columnspan=3, pady=50)
132
133    def load_default_hyper_parameters(self, dataset: str) -> dict[
134        str,
135        str | int | list[int] |
           ↳ float
136        ]:
137        """Load the dataset's default hyper-parameters from the json file.
138
139        Args:
140            dataset (str): the name of the dataset to load
141            ↳ hyper-parameters
142                for. ('MNIST', 'Cat Recognition' or 'XOR')
143        Returns:

```

```

143         a dictionary of default hyper-parameter values.
144         """
145         with open('school_project/frames/hyper-parameter-defaults.json') as
146             ↪ f:
147             return json.load(f)[dataset]
148
149 def create_model(self) -> object:
150     """Create and return a Model using the hyper-parameters set.
151
152     Returns:
153     a Model object.
154     """
155     self.use_gpu = self.use_gpu_check_button_var.get()
156
157     # Validate hidden layers shape input
158     hidden_layers_shape_input = [layer for layer in
159     ↪ self.hidden_layers_shape_entry.get().replace(' ',
160     ↪ '').split(',') if layer != '']
161     for layer in hidden_layers_shape_input:
162         if not layer.isdigit():
163             self.model_status_label.configure(
164                 text="Invalid hidden layers shape",
165                 fg='red'
166             )
167             raise ValueError
168
169     # Create Model
170     if not self.use_gpu:
171         if self.dataset == "MNIST":
172             from school_project.models.cpu.mnist import MNISTModel as
173             ↪ Model
174         elif self.dataset == "Cat Recognition":
175             from school_project.models.cpu.cat_recognition import
176             ↪ CatRecognitionModel as Model
177         elif self.dataset == "XOR":
178             from school_project.models.cpu.xor import XORModel as Model
179         model = Model(
180             hidden_layers_shape = [int(neuron_count) for neuron_count
181             ↪ in hidden_layers_shape_input],
182             train_dataset_size = self.train_dataset_size_scale.get(),
183             learning_rate = self.learning_rate_scale.get(),
184             use_relu = self.use_relu_check_button_var.get()
185         )
186         model.create_model_values()
187
188     else:
189         try:
190             if self.dataset == "MNIST":
191                 from school_project.models.gpu.mnist import MNISTModel
192                 ↪ as Model
193             elif self.dataset == "Cat Recognition":
194                 from school_project.models.gpu.cat_recognition import
195                 ↪ CatRecognitionModel as Model
196             elif self.dataset == "XOR":
197                 from school_project.models.gpu.xor import XORModel as
198                 ↪ Model
199             model = Model(hidden_layers_shape = [int(neuron_count) for
200             ↪ neuron_count in hidden_layers_shape_input],
201                 train_dataset_size =
202                 ↪ self.train_dataset_size_scale.get(),
203                 learning_rate =
204                 ↪ self.learning_rate_scale.get(),

```

```

193         use_relu =
194             ↪ self.use_relu_check_button_var.get()
195         model.create_model_values()
196     except ImportError as ie:
197         self.model_status_label.configure(
198             text="Failed to initialise GPU",
199             fg='red'
200         )
201         raise ImportError
202     return model
203
204 class TrainingFrame(tk.Frame):
205     """Frame for training page."""
206     def __init__(self, root: tk.Tk, width: int,
207                 height: int, bg: str,
208                 model: object, epoch_count: int) -> None:
209         """Initialise training frame widgets.
210
211         Args:
212             root (tk.Tk): the widget object that contains this widget.
213             width (int): the pixel width of the frame.
214             height (int): the pixel height of the frame.
215             bg (str): the hex value or name of the frame's background
216             ↪ colour.
217             model (object): the Model object to be trained.
218             epoch_count (int): the number of training epochs.
219
220         Raises:
221             TypeError: if root, width or height are not of the correct
222             ↪ type.
223
224         """
225         super().__init__(master=root, width=width, height=height, bg=bg)
226         self.root = root
227         self.WIDTH = width
228         self.HEIGHT = height
229         self.BG = bg
230
231         # Setup widgets
232         self.model_status_label = tk.Label(master=self,
233             bg=self.BG,
234             font=('Arial', 15))
235         self.training_progress_label = tk.Label(master=self,
236             bg=self.BG,
237             font=('Arial', 15))
238         self.loss_figure: Figure = Figure()
239         self.loss_canvas: FigureCanvasTkAgg = FigureCanvasTkAgg(
240             ↪ figure=self.loss_figure,
241             master=self
242         )
243
244         # Pack widgets
245         self.model_status_label.pack(pady=(30,0))
246         self.training_progress_label.pack(pady=30)
247
248         # Start training thread
249         self.model_status_label.configure(
250             text="Training weights and
251             ↪ biases...",
252             fg='red'
253         )
254         self.train_thread: threading.Thread = threading.Thread(

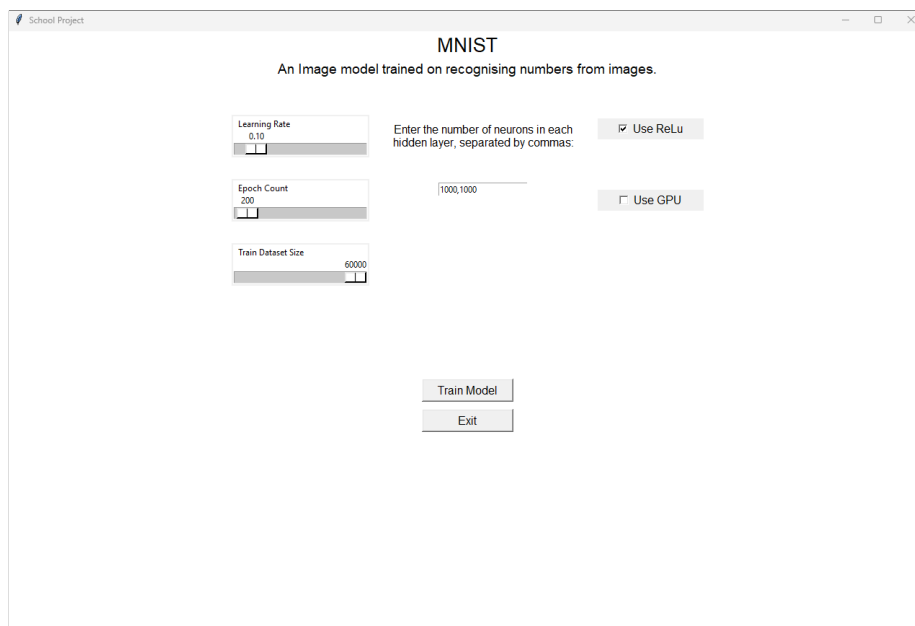
```

```

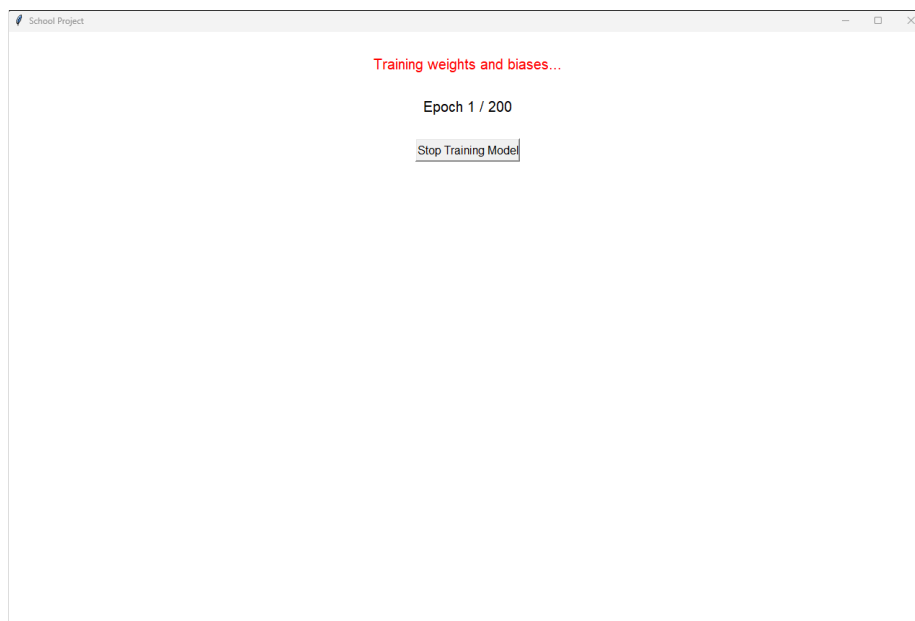
250                                     ↪ target=model.train,
251                                     ↪ args=(epoch_count,)
252                                     )
253     self.train_thread.start()
254
255     def plot_losses(self, model: object) -> None:
256         """Plot losses of Model training.
257
258         Args:
259             model (object): the Model object thats been trained.
260
261         """
262         self.model_status_label.configure(
263             text=f"Weights and biases trained in
264             ↪ {model.training_time}s",
265             fg='green'
266         )
267         graph: Figure.axes = self.loss_figure.add_subplot(111)
268         graph.set_title("Learning rate: " +
269             f"{model.learning_rate}")
270         graph.set_xlabel("Epochs")
271         graph.set_ylabel("Loss Value")
272         graph.plot(np.squeeze(model.train_losses))
273         self.loss_canvas.get_tk_widget().pack()

```

Which outputs the following for the hyper-parameter frame:



And outputs the following for the training frame during training:



And outputs the following for the training frame once training has completed:



- 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:
28              TypeError: if root, width or height are not of the correct
29          ↪ type.
30
31          """
```

```

28     super().__init__(master=root, width=width, height=height, bg=bg)
29     self.root = root
30     self.WIDTH = width
31     self.HEIGHT = height
32     self.BG = bg
33
34     # Setup load model frame variables
35     self.connection = connection
36     self.cursor = cursor
37     self.dataset = dataset
38     self.use_gpu: bool
39     self.model_options = self.load_model_options()
40
41     # Setup widgets
42     self.title_label = tk.Label(master=self,
43                                bg=self.BG,
44                                font=('Arial', 20),
45                                text=dataset)
46     self.about_label = tk.Label(
47         master=self,
48         bg=self.BG,
49         font=('Arial', 14),
50         text=f"Load a pretrained model for the {dataset}"
51         ↪ dataset."
52     )
53     self.model_status_label = tk.Label(master=self,
54                                        bg=self.BG,
55                                        font=('Arial', 15))
56
57     # Don't give loaded model options if no models have been saved for
58     ↪ the
59     # dataset.
60     if len(self.model_options) > 0:
61         self.model_option_menu_label = tk.Label(
62             master=self,
63             bg=self.BG,
64             font=('Arial', 14),
65             text="Select a model to
66             ↪ load or delete:"
67         )
68         self.model_option_menu_var = tk.StringVar(
69             master=self,
70             ↪ value=self.model_options[0]
71         )
72         self.model_option_menu = tk.OptionMenu(
73             self,
74             ↪ self.model_option_menu_var,
75             *self.model_options
76         )
77         self.use_gpu_check_button_var = tk.BooleanVar()
78         self.use_gpu_check_button = tk.Checkbutton(
79             master=self,
80             width=7, height=1,
81             font=tkf.Font(size=12),
82             text="Use GPU",
83             ↪ variable=self.use_gpu_check_button_var
84         )
85     else:
86         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

```

Which outputs the following for the load model frame when models have been saved for the dataset:



And outputs the following for the load model frame when no models have been saved for the dataset:



3.4 __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         Raises:
25             TypeError: if root, width or height are not of the correct type.
26
27         """
28         super().__init__(master=root, width=width, height=height, bg=bg)
29         self.root = root.title("School Project")
30         self.WIDTH = width
31         self.HEIGHT = height
32         self.BG = bg
33
34         # Setup school project frame variables
35         self.hyper_parameter_frame: HyperParameterFrame
36         self.training_frame: TrainingFrame
37         self.load_model_frame: LoadModelFrame
38         self.test_frame: TestMNISTFrame | TestCatRecognitionFrame | TestXORFrame
39         self.connection, self.cursor = self.setup_database()
40         self.model = None
41
42         # Record if the model should be saved after testing,
43         # as only newly created models should be given the option to be saved.
44         self.saving_model: bool
45
46         # Setup school project frame widgets
47         self.exit_hyper_parameter_frame_button = tk.Button(
48             master=self,
49             width=13,
50             height=1,
51             font=tkf.Font(size=12),
52             text="Exit",
53             command=self.exit_hyper_parameter_frame
54         )
55         self.exit_load_model_frame_button = tk.Button(
56             master=self,
57             width=13,
58             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,
195         File_Location TEXT,
196         Hidden_Layers_Shape TEXT,
197         Learning_Rate FLOAT,
198         Name TEXT,
199         Train_Dataset_Size INTEGER,
200         Use_ReLu INTEGER,
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
288         Raises:
289             TypeError: if train_thread is not of type threading.Thread.
290
291         """
292         if not train_thread.is_alive():
293             self.training_frame.training_progress_label.pack_forget()
294             self.training_frame.plot_losses(model=self.model)
295             self.stop_training_button.pack_forget()
296             self.test_created_model_button.pack(pady=(30,0))
297         else:
298             self.training_frame.training_progress_label.configure(
299                 text=self.model.training_progress
300             )
301             self.after(100, self.manage_training, train_thread)
302
303     def test_created_model(self) -> None:
304         """Unpack training frame, pack test frame for the dataset
305         and begin managing the test thread."""
306         self.saving_model = True
307         self.training_frame.pack_forget()
308         self.test_created_model_button.pack_forget()
309         if self.hyper_parameter_frame.dataset == "MNIST":
310             self.test_frame = TestMNISTFrame(
311                 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     self.test_frame.pack()
333     self.manage_testing(test_thread=self.test_frame.test_thread)
334
335 def test_loaded_model(self) -> None:
336     """Load saved model from load model frame, unpack load model frame,
337     pack test frame for the dataset and begin managing the test thread."""
338     self.saving_model = False
339     try:
340         self.model = self.load_model_frame.load_model()
341     except (ValueError, ImportError) as e:
342         return
343     self.load_model_frame.pack_forget()
344     self.test_loaded_model_button.pack_forget()
345     self.delete_loaded_model_button.pack_forget()
346     self.exit_load_model_frame_button.pack_forget()
347     if self.load_model_frame.dataset == "MNIST":
348         self.test_frame = TestMNISTFrame(
349             root=self,
350             width=self.WIDTH,
351             height=self.HEIGHT,
352             bg=self.BG,
353             use_gpu=self.load_model_frame.use_gpu,
354             model=self.model
355         )
356     elif self.load_model_frame.dataset == "Cat Recognition":
357         self.test_frame = TestCatRecognitionFrame(
358             root=self,
359             width=self.WIDTH,
360             height=self.HEIGHT,
361             bg=self.BG,
362             use_gpu=self.load_model_frame.use_gpu,
363             model=self.model
364         )
365     elif self.load_model_frame.dataset == "XOR":
366         self.test_frame = TestXORFrame(root=self,
367                                         width=self.WIDTH,
368                                         height=self.HEIGHT,
369                                         bg=self.BG,
370                                         model=self.model)
371     self.test_frame.pack()
372     self.manage_testing(test_thread=self.test_frame.test_thread)

```

```

373
374 def manage_testing(self, test_thread: threading.Thread) -> None:
375     """Wait for model test thread to finish,
376     then plot results on test frame.
377
378     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()
399
400     # Check if model name is empty
401     if model_name == '':
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 name has already been taken
409     dataset = self.dataset_option_menu_var.get().replace(" ", "_")
410     sql = """
411     SELECT Name FROM Models WHERE Dataset=?
412     """
413     parameters = (dataset,)
414     self.cursor.execute(sql, parameters)
415     for saved_model_name in self.cursor.fetchall():
416         if saved_model_name[0] == model_name:
417             self.test_frame.model_status_label.configure(
418                 text="Model name taken",
419                 fg='red'
420             )
421             return
422
423     # Save model to random hex file name
424     file_location = f"school_project/saved-models/{uuid.uuid4().hex}.npz"
425     self.model.save_model_values(file_location=file_location)
426
427     # Save the model information to the database
428     sql = """
429     INSERT INTO Models
430     (Dataset, File_Location, Hidden_Layers_Shape, Learning_Rate, Name,
↪ Train_Dataset_Size, Use_ReLu)
431     VALUES (?, ?, ?, ?, ?, ?, ?)
432     """
433     parameters = (

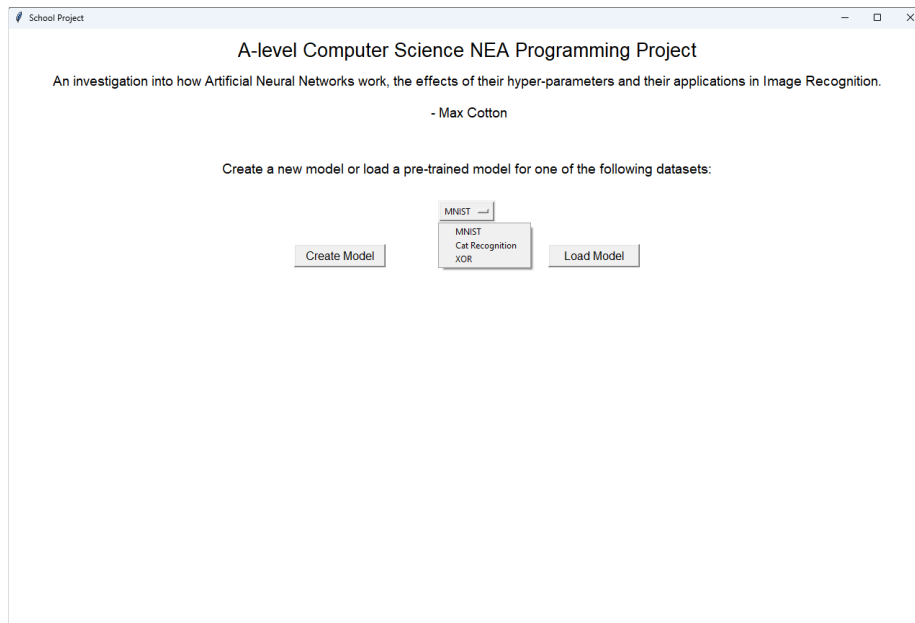
```

```

434         dataset,
435         file_location,
436         self.hyper_parameter_frame.hidden_layers_shape_entry.get(),
437         self.hyper_parameter_frame.learning_rate_scale.get(),
438         model_name,
439         self.hyper_parameter_frame.train_dataset_size_scale.get(),
440         self.hyper_parameter_frame.use_relu_check_button_var.get()
441     )
442     self.cursor.execute(sql, parameters)
443     self.connection.commit()
444
445     self.enter_home_frame()
446
447 def delete_loaded_model(self) -> None:
448     """Delete saved model file and model data from the database."""
449     dataset = self.dataset_option_menu_var.get().replace(" ", "_")
450     model_name = self.load_model_frame.model_option_menu_var.get()
451
452     # Delete saved model
453     sql = f"SELECT File_Location FROM Models WHERE Dataset=? AND Name=?"
454     parameters = (dataset, model_name)
455     self.cursor.execute(sql, parameters)
456     os.remove(self.cursor.fetchone()[0])
457
458     # Remove model data from database
459     sql = "DELETE FROM Models WHERE Dataset=? AND Name=?"
460     parameters = (dataset, model_name)
461     self.cursor.execute(sql, parameters)
462     self.connection.commit()
463
464     # Reload load model frame with new options
465     self.exit_load_model_frame()
466     self.enter_load_model_frame()
467
468 def enter_home_frame(self) -> None:
469     """Unpack test frame and pack home frame."""
470     self.model = None # Free up trained Model from memory
471     self.test_frame.pack_forget()
472     if self.saving_model:
473         self.save_model_label.pack_forget()
474         self.save_model_name_entry.delete(0, tk.END) # Clear entry's text
475         self.save_model_name_entry.pack_forget()
476         self.save_model_button.pack_forget()
477     self.exit_button.pack_forget()
478     self.home_frame.pack()
479
480 def main() -> None:
481     """Entrypoint of project."""
482     root = tk.Tk()
483     school_project_frame = SchoolProjectFrame(root=root, width=1280,
484                                              height=835, bg='white')
485     school_project_frame.pack(side='top', fill='both', expand=True)
486     root.mainloop()
487
488     # Stop model training when GUI closes
489     if school_project_frame.model is not None:
490         school_project_frame.model.set_running(value=False)
491
492 if __name__ == "__main__":
493     main()

```

Which outputs the following for the home frame:



4 Testing

4.1 Investigation

4.1.1 test_model module

The test_model module is contained within the frames package, and contains tkinter frames for testing the 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 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() -
↪ model.test_outputs)) * 100, ndigits=1)}%\n" +
80         f"Network Shape: " +
81         f"{'.'.join(model.layers_shape)}\n"
82     )
83
84     test_inputs = np.squeeze(model.test_inputs).T
85     test_outputs = np.squeeze(model.test_outputs).T.tolist()
86     test_prediction = np.squeeze(model.test_prediction).T.tolist()
87
88     # Randomly shuffle order of test_inputs, test_outputs and
89     ↪ test_prediction
90     # whilst maintaining order between them
91     test_data = list(zip(test_inputs,
92                         test_outputs,
93                         test_prediction))
94     random.shuffle(test_data)
95     test_inputs, test_outputs, test_prediction = zip(*test_data)
96
97     elif self.use_gpu:
98
99         import cupy as cp
100
101         self.results_label.configure(
102             text="Prediction Correctness: " +
103             f"{round(number=100 -
104             ↪ np.mean(np.abs(cp.asnumpy(model.test_prediction).round() -
105             ↪ cp.asnumpy(model.test_outputs))) * 100, ndigits=1)}%\n" +
106             f"Network Shape: " +
107             f"{'.'.join(model.layers_shape)}\n"
108         )
109
110         test_inputs = cp.asnumpy(cp.squeeze(model.test_inputs)).T
111         test_outputs = cp.asnumpy(cp.squeeze(model.test_outputs)).T.tolist()
112         test_prediction = cp.squeeze(model.test_prediction).T.tolist()
113
114         # Randomly shuffle order of test_inputs, test_outputs and
115         ↪ test_prediction
116         # whilst maintaining order between them
117         test_data = list(zip(test_inputs,
118                             test_outputs,
119                             test_prediction))
120         random.shuffle(test_data)
121         test_inputs, test_outputs, test_prediction = zip(*test_data)
122
123         # Setup incorrect prediction figure
124         self.incorrect_prediction_figure.suptitle("Incorrect predictions:")
125         image_count = 0
126         for i in range(len(test_prediction)):
127             if test_prediction[i].index(max(test_prediction[i])) !=
128             ↪ test_outputs[i].index(max(test_outputs[i])):
129                 if image_count == 2:
130                     break
131                 elif image_count == 0:
132                     image = self.incorrect_prediction_figure.add_subplot(121)
133                 elif image_count == 1:
134                     image = self.incorrect_prediction_figure.add_subplot(122)
135                 image.set_title(f"Predicted:
136                 ↪ {test_prediction[i].index(max(test_prediction[i]))}\n" +
137                 f"Should have predicted:
138                 ↪ {test_outputs[i].index(max(test_outputs[i]))}")

```

```

132         image.imshow(test_inputs[i].reshape((28,28)))
133         image_count += 1
134
135     # Setup correct prediction figure
136     self.correct_prediction_figure.suptitle("Correct predictions:")
137     image_count = 0
138     for i in range(len(test_prediction)):
139         if test_prediction[i].index(max(test_prediction[i])) ==
140             ↪ test_outputs[i].index(max(test_outputs[i])):
141             if image_count == 2:
142                 break
143             elif image_count == 0:
144                 image = self.correct_prediction_figure.add_subplot(121)
145             elif image_count == 1:
146                 image = self.correct_prediction_figure.add_subplot(122)
147             image.set_title(f"Predicted:
148                 ↪ {test_prediction[i].index(max(test_prediction[i]))}")
149             image.imshow(test_inputs[i].reshape((28,28)))
150             image_count += 1
151
152     class TestCatRecognitionFrame(tk.Frame):
153         """Frame for Testing Cat Recognition page."""
154         def __init__(self, root: tk.Tk, width: int,
155             height: int, bg: str,
156             use_gpu: bool, model: object) -> None:
157             """Initialise test cat recognition frame widgets.
158
159             Args:
160                 root (tk.Tk): the widget object that contains this widget.
161                 width (int): the pixel width of the frame.
162                 height (int): the pixel height of the frame.
163                 bg (str): the hex value or name of the frame's background colour.
164                 use_gpu (bool): True or False whether the GPU should be used.
165                 model (object): the Model object to be tested.
166
167             Raises:
168                 TypeError: if root, width or height are not of the correct type.
169
170             """
171             super().__init__(master=root, width=width, height=height, bg=bg)
172             self.root = root
173             self.WIDTH = width
174             self.HEIGHT = height
175             self.BG = bg
176
177             # Setup image recognition frame variables
178             self.use_gpu = use_gpu
179
180             # Setup widgets
181             self.model_status_label = tk.Label(master=self,
182                 bg=self.BG,
183                 font=('Arial', 15))
184             self.results_label = tk.Label(master=self,
185                 bg=self.BG,
186                 font=('Arial', 15))
187             self.correct_prediction_figure = Figure()
188             self.correct_prediction_canvas = FigureCanvasTkAgg(
189                 figure=self.correct_prediction_figure,
190                 master=self
191             )
192             self.incorrect_prediction_figure = Figure()
193             self.incorrect_prediction_canvas = FigureCanvasTkAgg(
194                 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" +
220             f"Network Shape: " +
221             f"{','.join(model.layers_shape)}\n"
222         )
223
224         # Randomly shuffle order of test_inputs, test_outputs and
225         ↪ test_predicition
226         # whilst maintaining order between them
227         test_data = list(zip(model.test_inputs.T,
228                             np.squeeze(model.test_outputs).T.tolist(),
229
230                             ↪ np.squeeze(model.test_prediction.round()).T.tolist()))
231
232         random.shuffle(test_data)
233         (test_inputs,
234          test_outputs,
235          test_prediction) = map(lambda arr: np.array(arr).T,
236                               zip(*test_data))
237
238     elif self.use_gpu:
239         import cupy as cp
240
241         self.results_label.configure(
242             text="Prediction Correctness: " +
243             f"{round(number=100 -
244             ↪ np.mean(np.abs(cp.asnumpy(model.test_prediction).round() -
245             ↪ cp.asnumpy(model.test_outputs))) * 100, ndigits=1)}%\n" +
246             f"Network Shape: " +
247             f"{','.join(model.layers_shape)}\n"
248         )
249
250         # Randomly shuffle order of test_inputs, test_outputs and
251         ↪ test_predicition
252         # whilst maintaining order between them
253         test_data = list(zip(cp.asnumpy(model.test_inputs).T,

```



```

248         ↪ cp.asnumpy(cp.squeeze(model.test_outputs)).T.tolist(),
249         ↪ cp.asnumpy(cp.squeeze(model.test_prediction)).round().T.tolist()))
250     random.shuffle(test_data)
251     (test_inputs,
252      test_outputs,
253      test_prediction) = map(lambda arr: np.array(arr).T,
254                             zip(*test_data))
255
256     # Setup incorrect prediction figure
257     self.incorrect_prediction_figure.suptitle("Incorrect predictions:")
258     image_count = 0
259     for i in range(len(test_prediction)):
260         if test_prediction[i] != test_outputs[i]:
261             if image_count == 2:
262                 break
263             elif image_count == 0:
264                 image = self.incorrect_prediction_figure.add_subplot(121)
265             elif image_count == 1:
266                 image = self.incorrect_prediction_figure.add_subplot(122)
267             image.set_title(f"Predicted: {'Cat' if test_prediction[i] == 1
268                             ↪ else 'Not a cat'}\n")
269             image.imshow(test_inputs[:,i].reshape((64,64,3)))
270             image_count += 1
271
272     # Setup correct prediction figure
273     self.correct_prediction_figure.suptitle("Correct predictions:")
274     image_count = 0
275     for i in range(len(test_prediction)):
276         if test_prediction[i] == test_outputs[i]:
277             if image_count == 2:
278                 break
279             elif image_count == 0:
280                 image = self.correct_prediction_figure.add_subplot(121)
281             elif image_count == 1:
282                 image = self.correct_prediction_figure.add_subplot(122)
283             image.set_title(f"Predicted: {'Cat' if test_prediction[i] == 1
284                             ↪ else 'Not a cat'}\n")
285             image.imshow(test_inputs[:,i].reshape((64,64,3)))
286             image_count += 1
287
288     class TestXORFrame(tk.Frame):
289         """Frame for Testing XOR page."""
290         def __init__(self, root: tk.Tk, width: int,
291                     height: int, bg: str, model: object) -> None:
292             """Initialise test XOR frame widgets.
293
294             Args:
295                 root (tk.Tk): the widget object that contains this widget.
296                 width (int): the pixel width of the frame.
297                 height (int): the pixel height of the frame.
298                 bg (str): the hex value or name of the frame's background colour.
299                 model (object): the Model object to be tested.
300
301             Raises:
302                 TypeError: if root, width or height are not of the correct type.
303
304             """
305         super().__init__(master=root, width=width, height=height, bg=bg)
306         self.root = root
307         self.WIDTH = width
308         self.HEIGHT = height

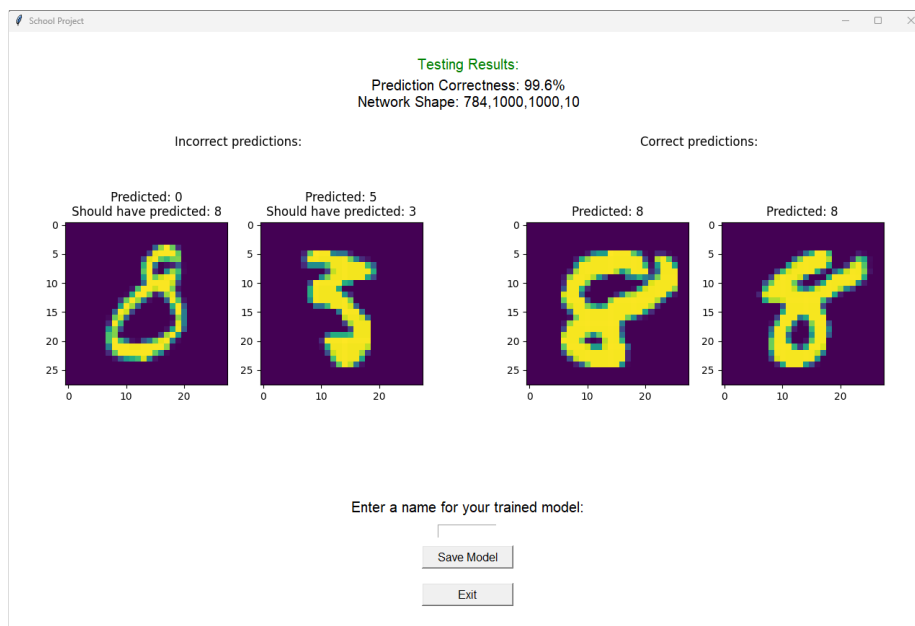
```

```

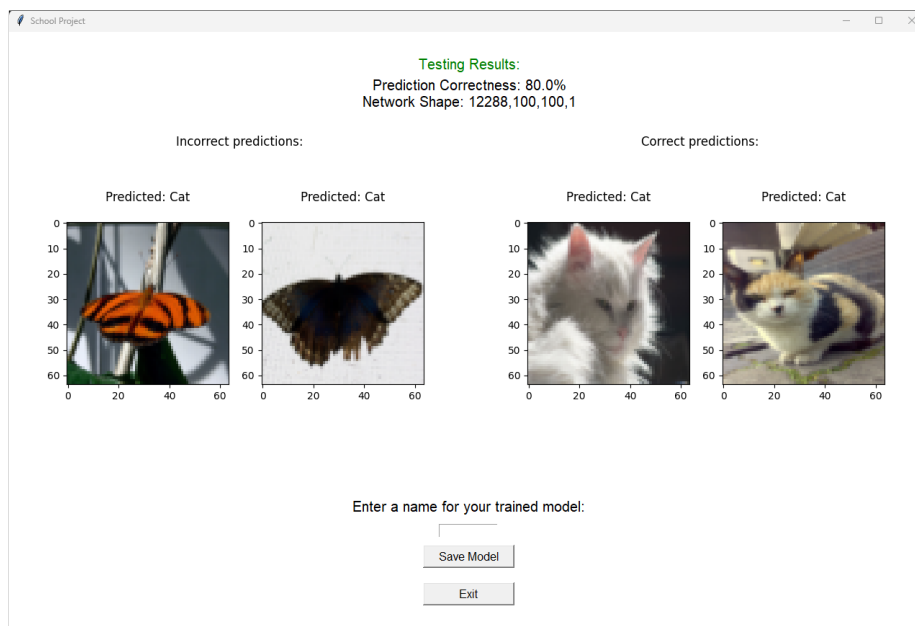
306     self.BG = bg
307
308     # Setup widgets
309     self.model_status_label = tk.Label(master=self,
310                                       bg=self.BG,
311                                       font=('Arial', 15))
312     self.results_label = tk.Label(master=self,
313                                  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()

```

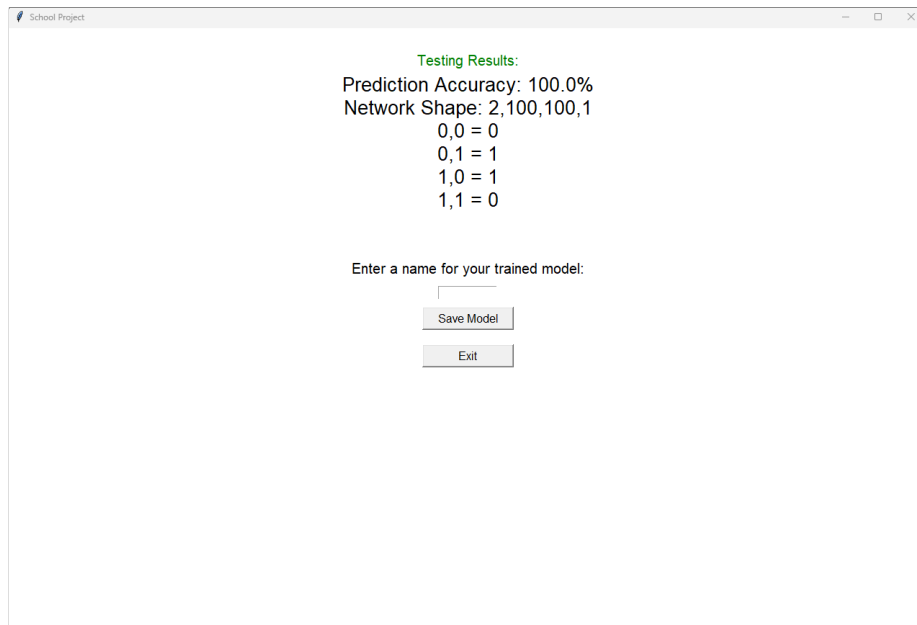
Which outputs the following for the MNIST dataset:



And outputs the following for the Cat Recognition dataset:



And outputs the following for the XOR dataset:



4.1.2 Effects of Hyper-Parameters

For the following investigations, I utilised Jupyter Notebook and have displayed the results below:

Learning Rate Analysis

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]

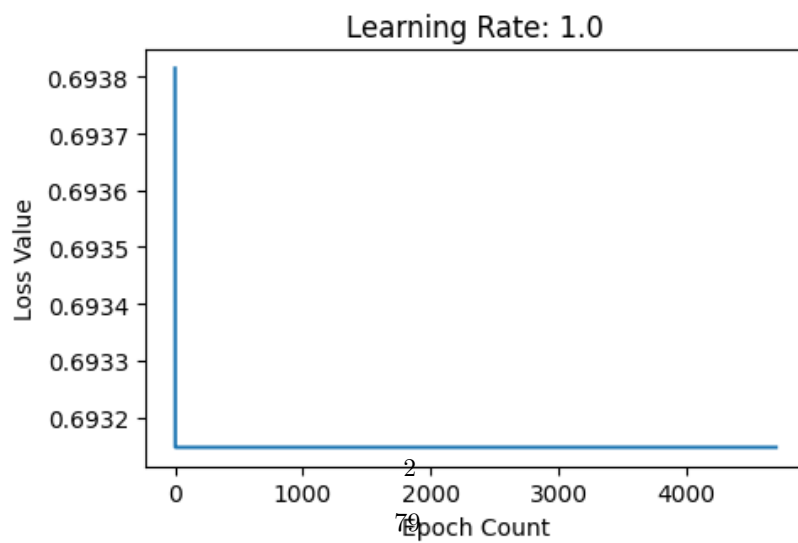
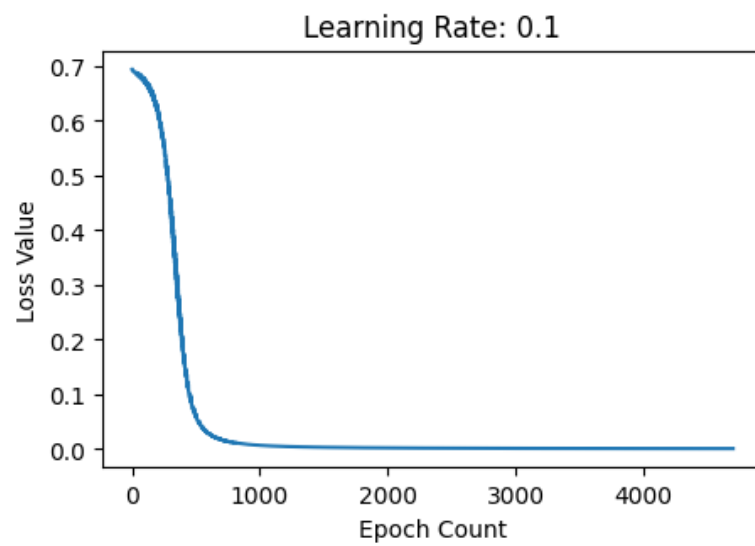
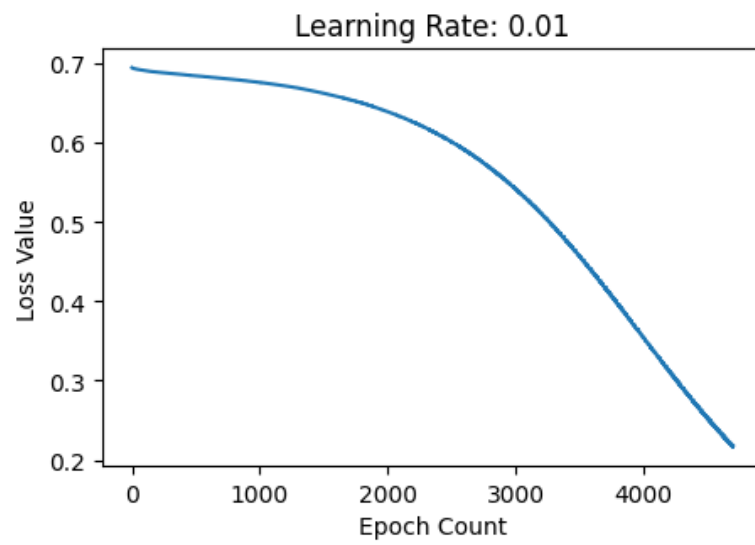
learning_rates = [0.01, 0.1, 1.0]

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.

Epoch Count Analysis

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')

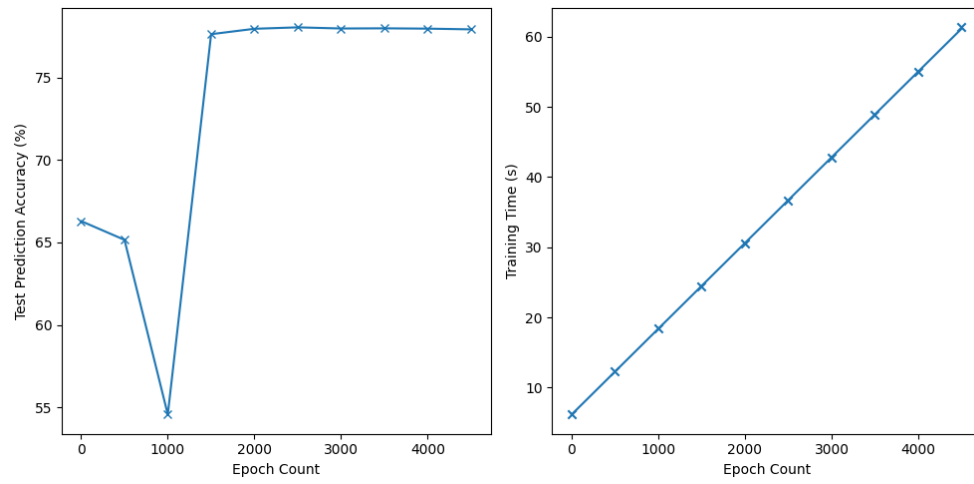
# Plot regression line
axis[1].plot(epoch_counts, m * epoch_counts + c)

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.

Train Dataset Size Analysis

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)

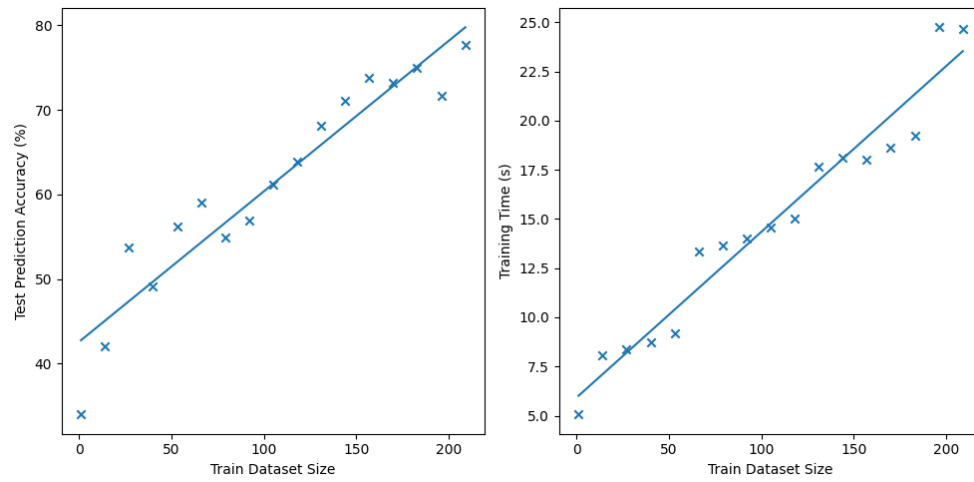
plt.tight_layout()
plt.show()

```

'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.

Layer Count Analysis

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')

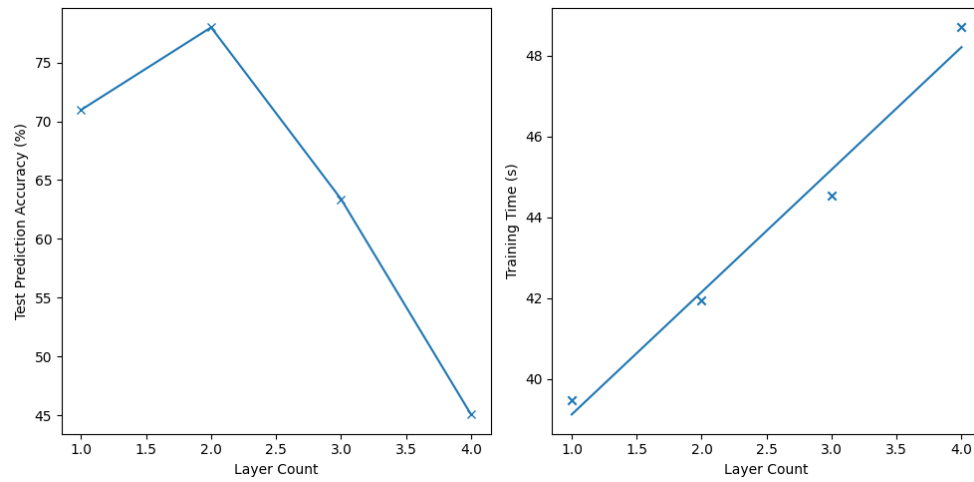
# Plot regression line
axis[1].plot(layer_counts, m * layer_counts + c)

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.

Neuron Count Analysis

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')

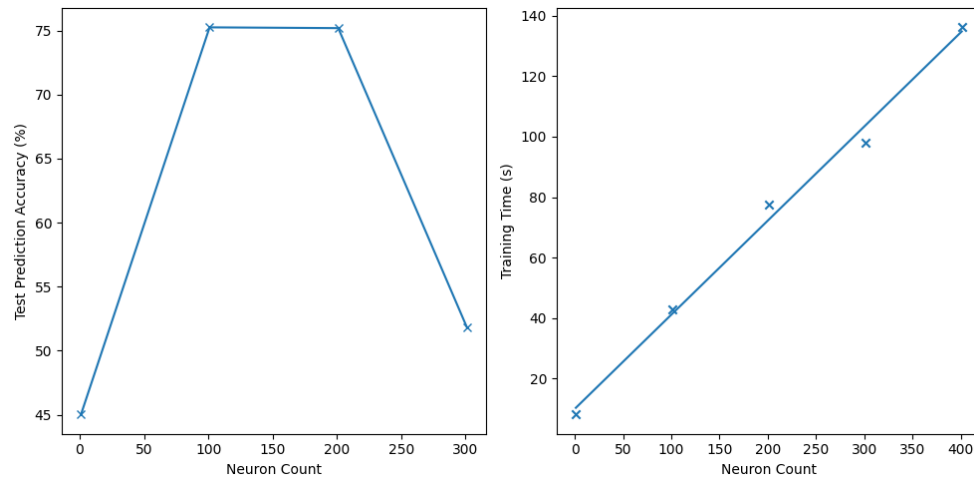
# Plot regression line
axis[1].plot(neuron_counts, m * neuron_counts + c)

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.

ReLu Analysis

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))

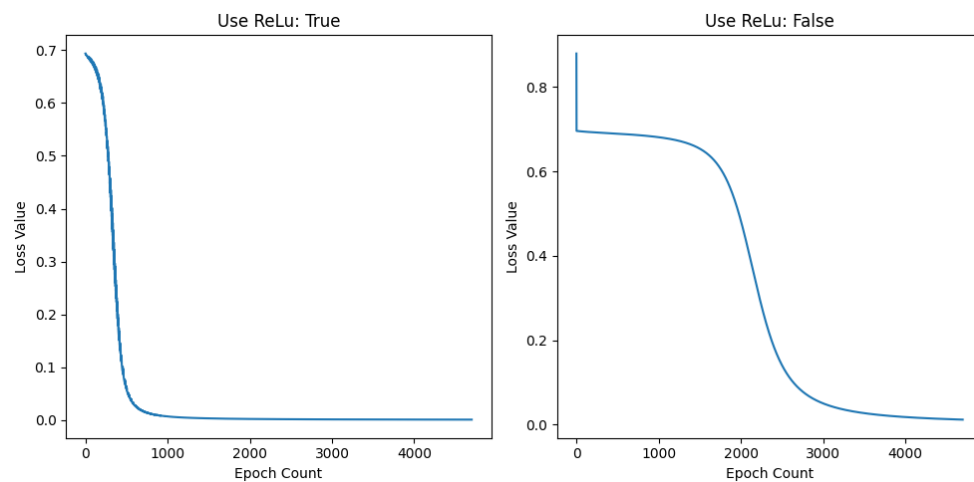
model = Model(hidden_layers_shape=[100, 100],
               train_dataset_size=4,
               learning_rate=0.1,
               use_relu=False)
model.create_model_values()
model.train(epoch_count=4_700)
model.test()
```

```

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.

CPU vs GPU Analysis

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

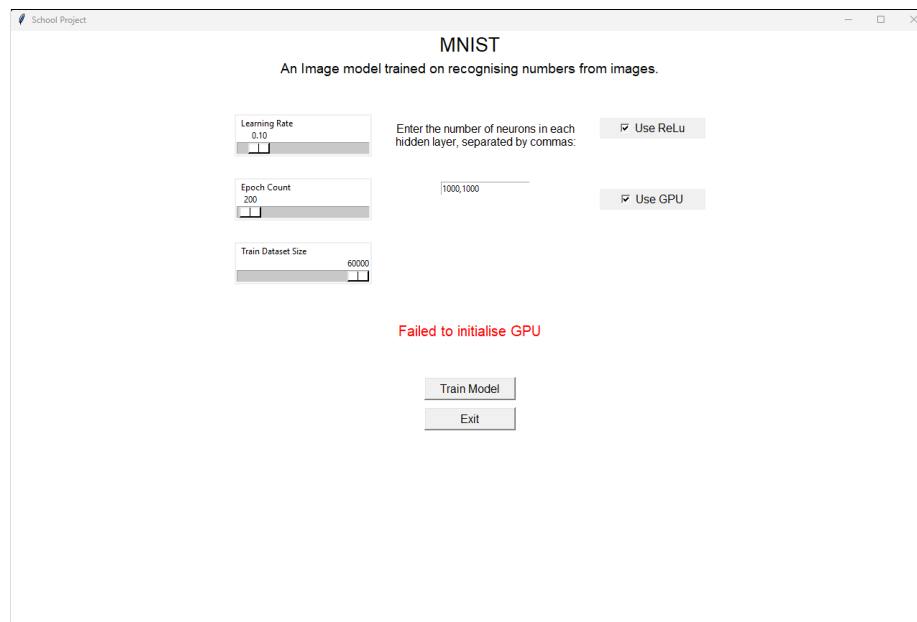
As shown above, the GPU is almost four times faster at training the model than the CPU, showing how beneficial it is to utilise the parallel computations of the GPU

4.2 Manual Testing

4.2.1 Input Validation Testing

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

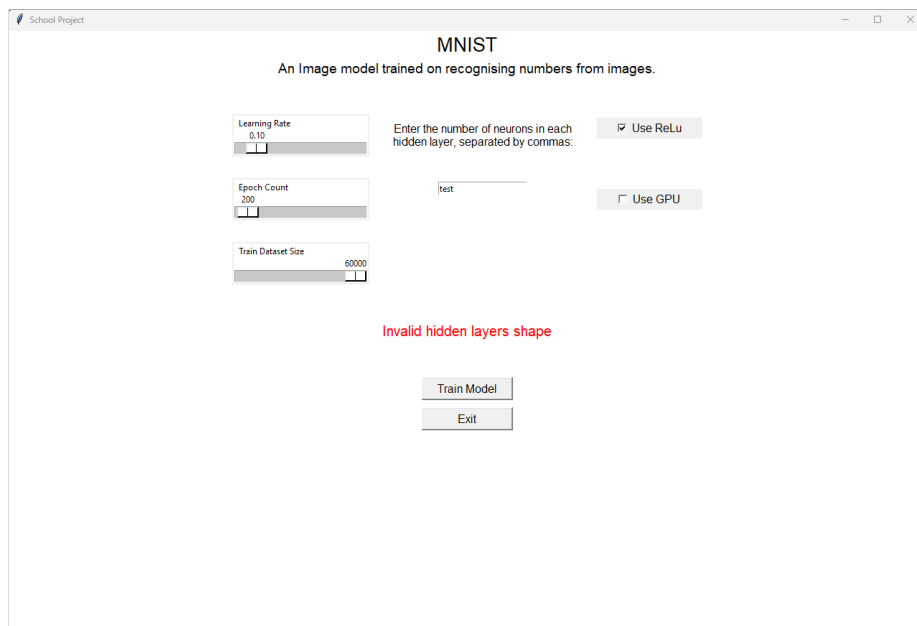
- Hyper Parameter Frame:
 - Use GPU Validation:
 - * Description: Select Use GPU checkbox without a GPU present.
 - * Expected Result: The exception should be handled and a useful error message should be displayed.
 - * Actual Result: Expected Result
 - * Test Status: Pass
 - * Evidence:



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

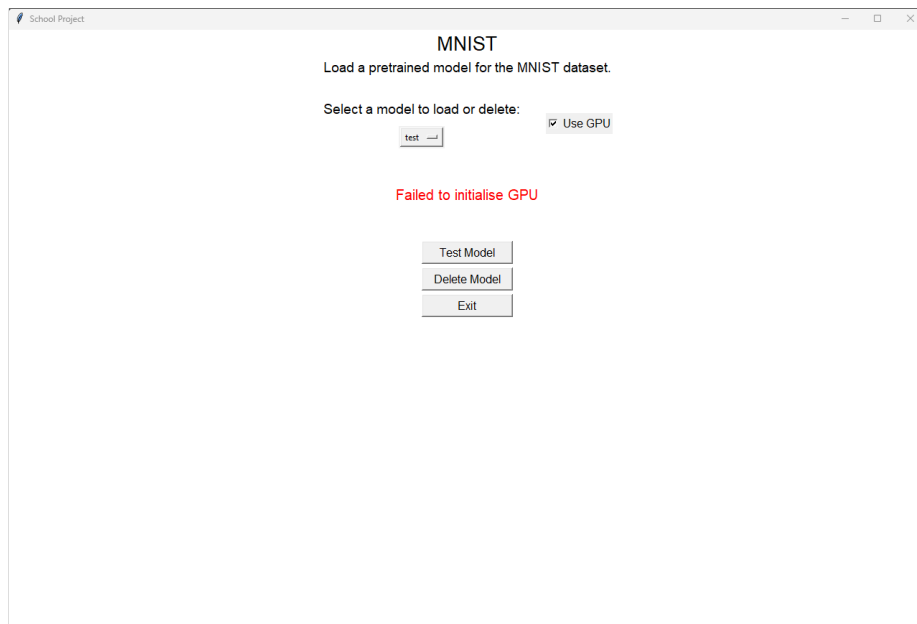
– Hidden Layers Shape Validation:

- * Description: Enter an invalid 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:



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

- 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:

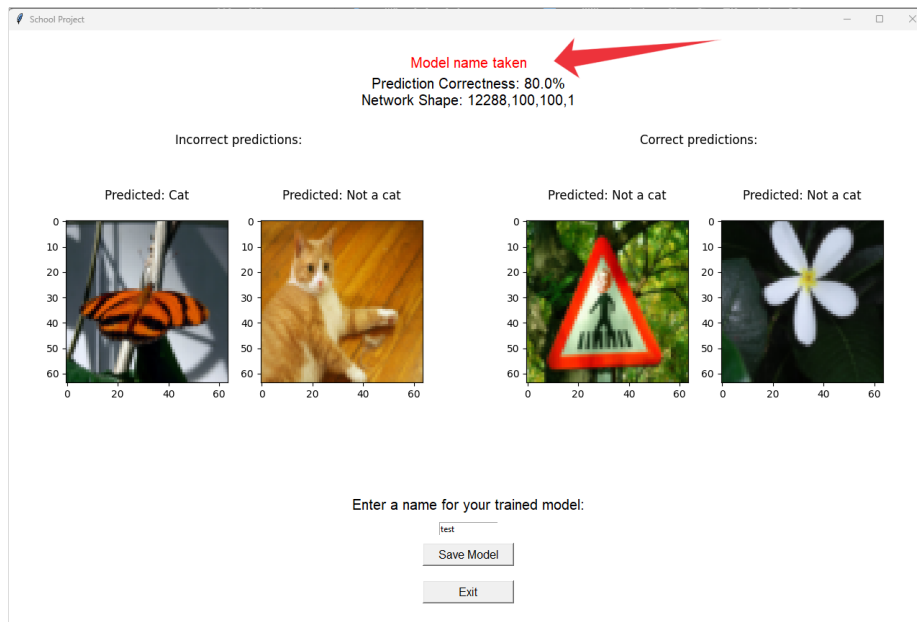


Link to video evidence: <https://github.com/mcttn22/school-project/blob/main/project-report/input-validation-testing-videos.md/#hidden-layers-shape-validation>

- Test Frames:

- Taken Trained Model Name Validation:

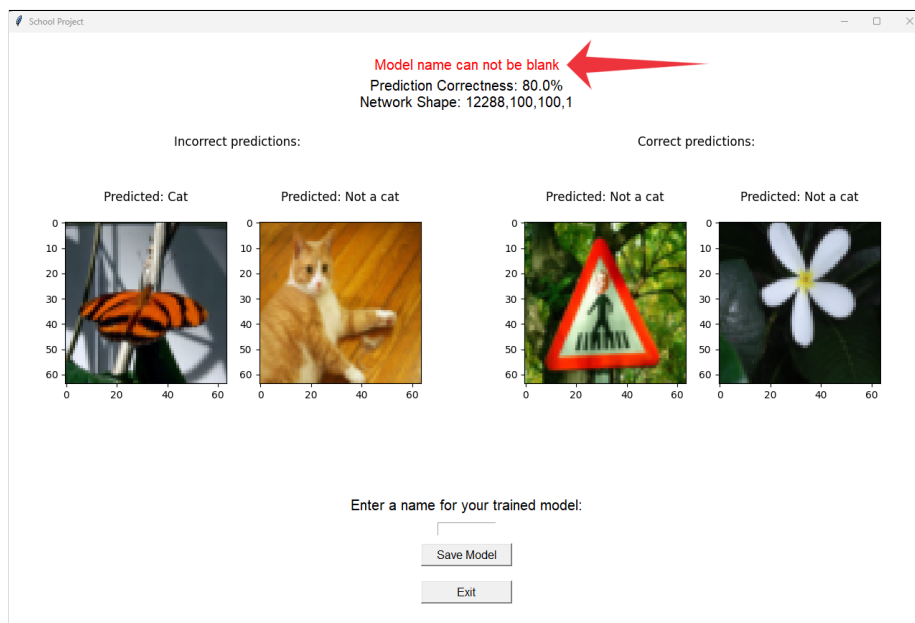
- * Description: Try to save a trained model with an already taken name.
 - * Data Value: "test"
 - * Data Type: Erroneous
 - * Expected Result: The exception should be handled and a useful error message should be displayed.
 - * Actual Result: Expected Result
 - * Test Status: Pass
 - * Evidence:



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

– Empty Trained Model Name Validation:

- * Description: Try to save a trained model with blank name.
- * Data Value: ""
- * 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:



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

4.3 Automated Testing

4.3.1 Unit Tests

Within the test package, I have written the following 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 only worth showing the code for the cpu version as both are very similar in functionality.

- test_model.py module:

```
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 chosen."""
20         train_dataset_size = 4
21         model = XORModel(hidden_layers_shape = [100, 100],
22                           train_dataset_size = train_dataset_size,
23                           learning_rate = 0.1,
24                           use_relu = True)
25         model.create_model_values()
26         model.train(epoch_count=1)
27         self.assertEqual(first=model.layers.head.input.shape[1],
28                           second=train_dataset_size)
29
30     def test_network_shape(self) -> None:
31         """Test the neuron count of each layer to match the set shape of
↪ the
32         network."""
33         layers_shape = [2, 100, 100, 1]
34         model = XORModel(hidden_layers_shape = [100, 100],
35                           train_dataset_size = 4,
36                           learning_rate = 0.1,
37                           use_relu = True)
38         model.create_model_values()
39         model.train(epoch_count=1)
40         for count, layer in enumerate(model.layers):
41             self.assertEqual(first=layer.input_neuron_count,
42                               second=layers_shape[count])
43
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,
↪             second=learning_rate)
55
56     def test_relu_model_transfer_types(self) -> None:
57         """Test transfer type of each layer to match whats set."""
58         transfer_types = ['relu', 'relu', 'sigmoid']
59         model = XORModel(hidden_layers_shape = [100, 100],
60                           train_dataset_size = 4,
61                           learning_rate = 0.1,
62                           use_relu = True)
63         model.create_model_values()
64         model.train(epoch_count=1)
65         for count, layer in enumerate(model.layers):

```

```

66         self.assertEqual(first=layer.transfer_type,
67                           second=transfer_types[count])
68
69     def test_sigmoid_model_transfer_types(self) -> None:
70         """Test transfer type of each layer to match whats set."""
71         transfer_types = ['sigmoid', 'sigmoid', 'sigmoid']
72         model = XORModel(hidden_layers_shape = [100, 100],
73                           train_dataset_size = 4,
74                           learning_rate = 0.1,
75                           use_relu = False)
76         model.create_model_values()
77         model.train(epoch_count=1)
78         for count, layer in enumerate(model.layers):
79             self.assertEqual(first=layer.transfer_type,
80                             second=transfer_types[count])
81
82     def test_weight_matrice_shapes(self) -> None:
83         """Test that each layer's weight matrix has the same number of
84 ↪ columns
85 as the layer's input matrix's number of rows, for the matrix
86 multiplication."""
87         model = XORModel(hidden_layers_shape = [100, 100],
88                           train_dataset_size = 4,
89                           learning_rate = 0.1,
90                           use_relu = True)
91         model.create_model_values()
92         model.train(epoch_count=1)
93         for layer in model.layers:
94             self.assertEqual(first=layer.weights.shape[1],
95                             second=layer.input.shape[0])
96
97     def test_bias_matrice_shapes(self) -> None:
98         """Test that each layer's bias matrix has the same number of rows
99 as the result of the layer's weights and input multiplication, for
100 element-wise addition of the biases."""
101         model = XORModel(hidden_layers_shape = [100, 100],
102                           train_dataset_size = 4,
103                           learning_rate = 0.1,
104                           use_relu = True)
105         model.create_model_values()
106         model.train(epoch_count=1)
107         for layer in model.layers:
108             self.assertEqual(first=layer.biases.shape[0],
109                             second=layer.weights.shape[0])
110
111     def test_layer_output_shapes(self) -> None:
112         """Test the shape of each layer's activation function's 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 correctly."""

```

```

126         initial_model = XORModel(hidden_layers_shape = [100, 100],
127                                   train_dataset_size = 4,
128                                   learning_rate = 0.1,
129                                   use_relu = True)
130     initial_model.create_model_values()
131     initial_model.train(epoch_count=1)
132
133     # Save model values
134     file_location =
135     ↪ f"school_project/saved-models/{uuid.uuid4().hex}.npz"
136     initial_model.save_model_values(file_location=file_location)
137
138     # Create model from the saved values
139     loaded_model = XORModel(hidden_layers_shape = [100, 100],
140                              train_dataset_size = 4,
141                              learning_rate = 0.1,
142                              use_relu = True)
143     loaded_model.load_model_values(file_location=file_location)
144
145     # Remove the saved model values
146     os.remove(path=file_location)
147
148     # Compare initial and loaded model values
149     for layer1, layer2 in zip(initial_model.layers,
150                               ↪ loaded_model.layers):
151         self.assertTrue(np.array_equal(a1=layer1.weights,
152                                         a2=layer2.weights))
153         self.assertTrue(np.array_equal(a1=layer1.biases,
154                                         a2=layer2.biases))
155
156 if __name__ == '__main__':
157     unittest.main()

```

- test_tools.py module:

```

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()

```

4.3.2 GitHub Automated Testing

With the following configuration programmed 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

```

4.3.3 Docker

I also provide a basic Dockerfile and instructions for its use in the README.md file, so that the project can 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"]

```

5 Evaluation TODO

5.1 Project Objectives Evaluation

5.1.1 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

5.1.2 Project Objective Evaluations

Objective ID	Evaluation	Status
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
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
2.1	The Artificial Neural Networks allow the use of a graphics card where applicable.	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
3	A Graphical User Interface allowing configuration of all hyper-parameters, loading and saving of trained models and testing has been developed.	Fully met
3.1	The Graphical User Interface allows user configuration of all utilised model hyper-parameters.	Fully met
3.2	The model predictions can be compared in terms of both learning rate and overall accuracy.	Fully met

5.2 Third Party Feedback TODO

5.3 Future Improvements TODO