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

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 usefull free resources on the internet to use. I particullarly 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

- Learn how Artificial Neural Networks work and develop them from first principles

- Implement the Artificial Neural Networks by creating trained models on image datasets

  – Allow use of Graphics Cards for faster training

  – Allow for the saving of trained models

- Develop a Graphical User Interface

  – Provide controls for hyper-parameters of models

  – Display and compare the results each model's predictions

1.4 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.
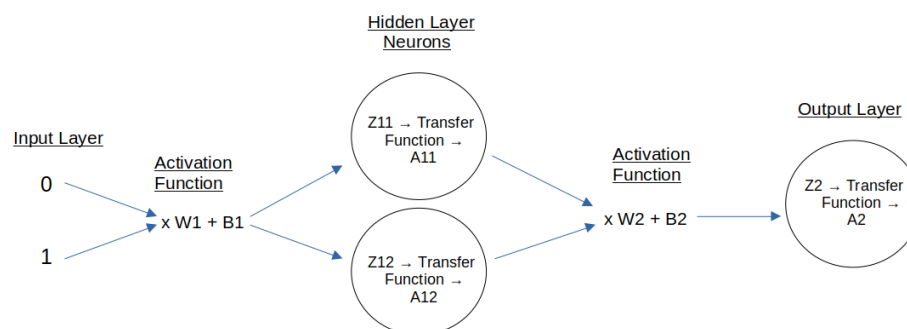
1.4.1 Structure

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

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.

To implement an Artificial Neural Network, matrices are used to represent the layers, where each layer is a matrice 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 matrice 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 usefull 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.

1.4.2 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 $u = g(x)$, $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} * \frac{\partial u}{\partial x}$

      · For a matrice of $f(x)$ values, the matrice 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

    * However, during backpropagation some issues can occur, such as the following:

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

- · 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.

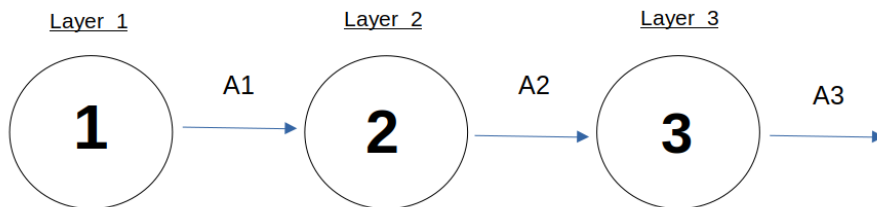## 1.5 Theory Behind Deep Artificial Neural Networks

1.5.1 Setup

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

- 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 intialised 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.
    * Where $sigmoid(Z) = \frac{1}{1+e^{-z}}$
  - 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 largers values of $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial B}$ for Gradient Descent than Sigmoid, so updates at a quicker rate.
    * Where $relu(Z) = max(0, Z)$

1.5.2   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.

1.5.3   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. $Z = W * X + B$

8

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, $\frac{\partial Z3}{\partial A2} = W3$

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

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

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

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

— $\frac{\partial L}{\partial B2} = \frac{\partial L}{\partial A2} * \frac{\partial A2}{\partial Z2} * \frac{\partial Z2}{\partial B2}$

By using function 1, $\frac{\partial Z2}{\partial B2} = 1$

$=> \frac{\partial L}{\partial W2} = \frac{\partial L}{\partial A2} * \frac{\partial A2}{\partial Z2} * 1$

- 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 derviative 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 $Z >= 0$ 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 caluculations, so instead it can be found with the derivative of the Log-Loss function, as shown in the following:

    – Using function 4, $\frac{\partial L}{\partial A} = (-\frac{1}{m})(\frac{Y-A}{A*(1-A)})$

1.6 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).

1.6.1 Datasets

- MNIST dataset

  - The MNIST dataset is a famouse 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 matrice 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 matrice 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 usefull for quickly comparing different hyper-parameters of a Network, whilst still not being linearly seperable.

1.6.2 Theory behind using Graphics Cards to train Artificial Neural Networks

Graphics Cards consist of many Tensor cores which are processing units specialiased 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
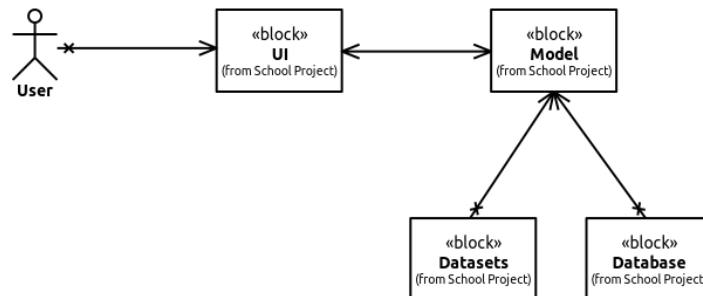
2.3 Class Diagrams

2.3.1 UI Class Diagram

bdd [package] School Project [UI Class Diagram]
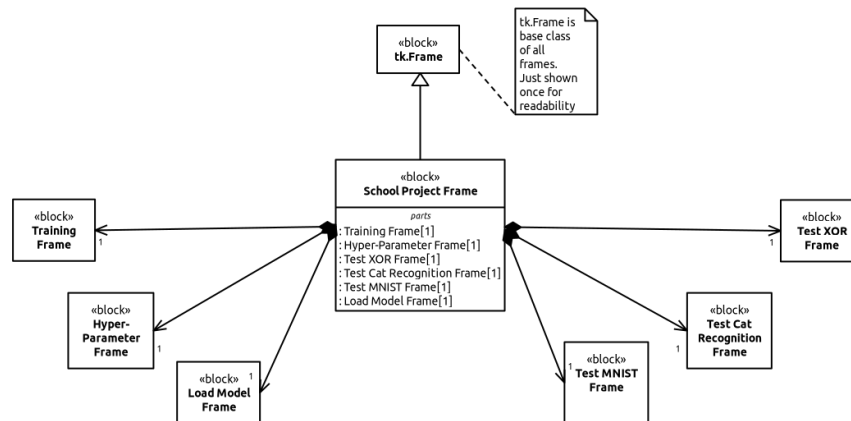
2.3.2 Model Class Diagram

bdd [package] School Project [Model Class Diagram]
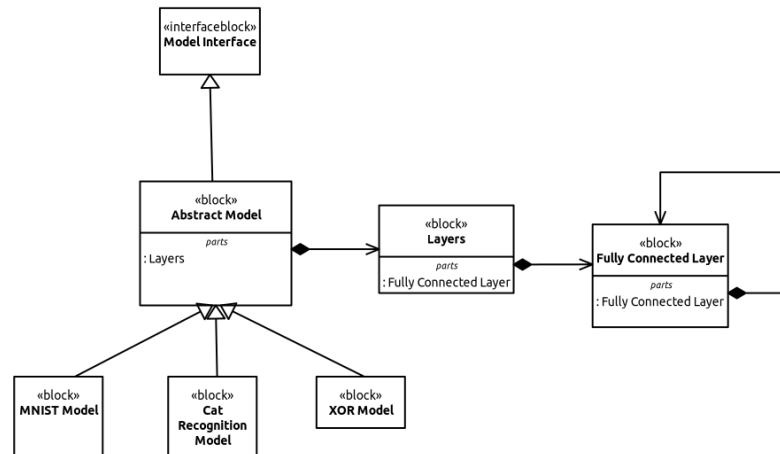
2.4   System Flow chart
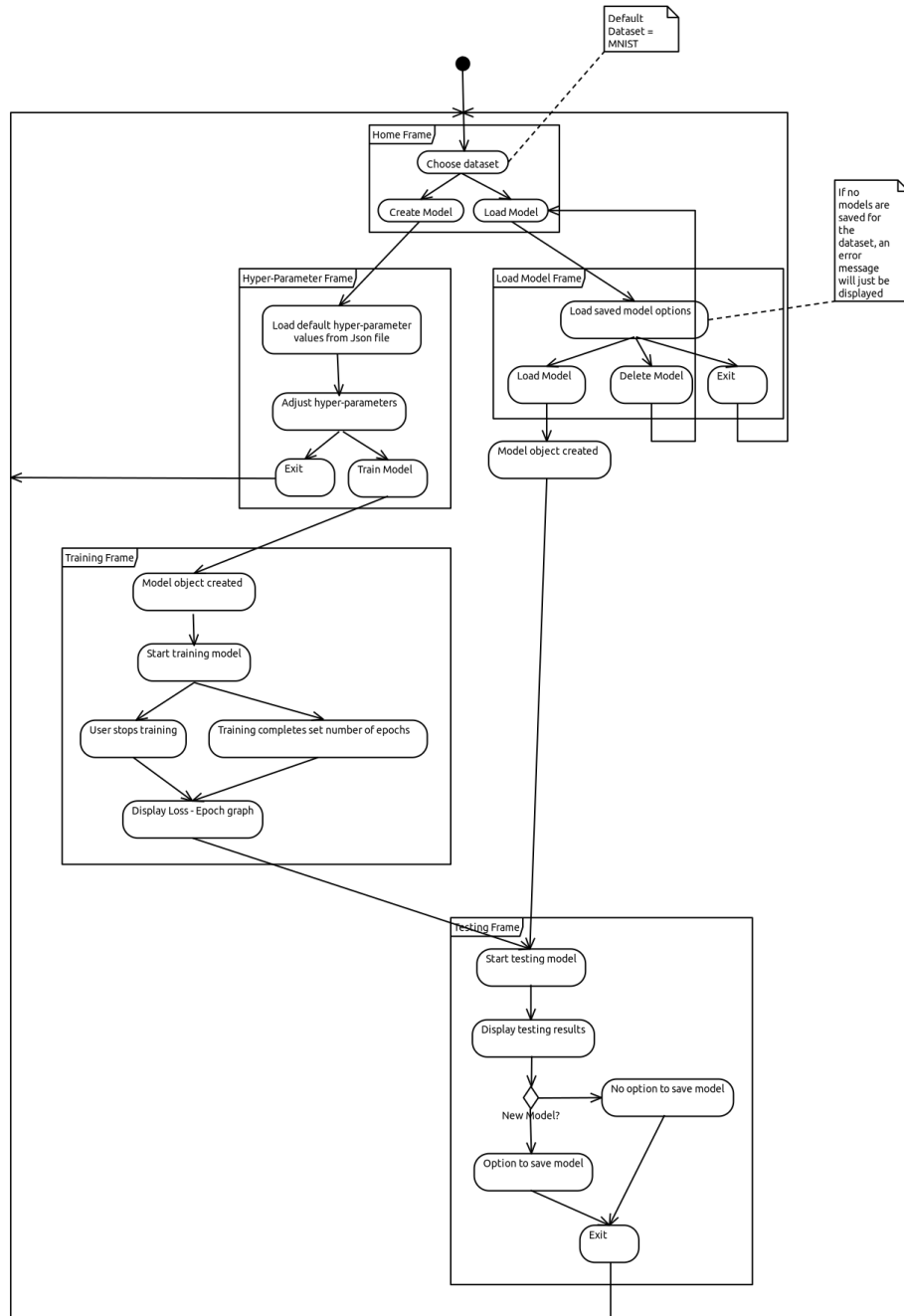
2.5   Algorithms

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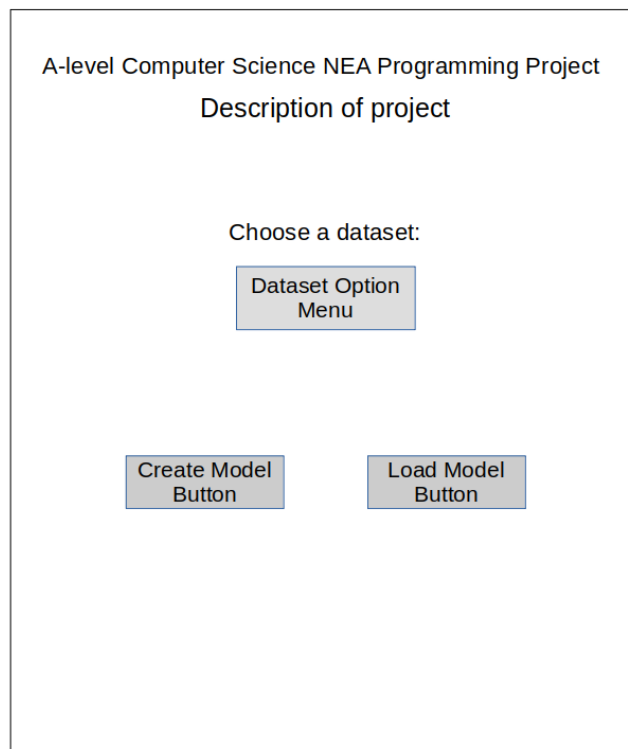
- Hyper-Parameter Frame design:
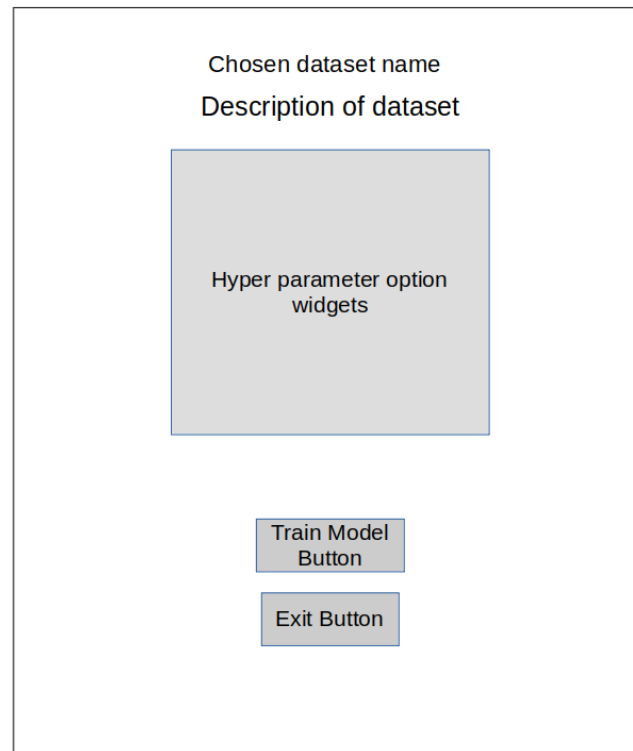
- Training Frame design:
  - During training, the following is displayed on the Training Frame:

Training weights and biases...

Epoch x / y

Stop Training Model
Button
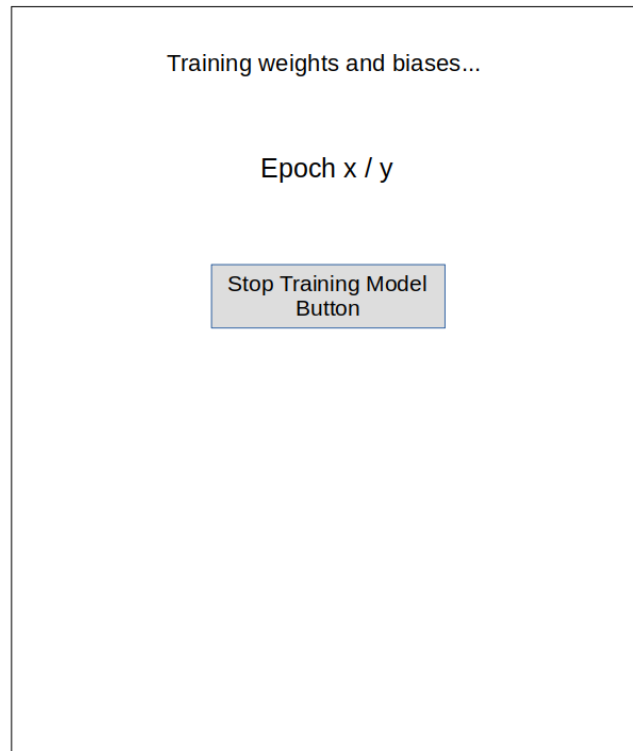
– Once training has finished, the following is displayed on the Training Frame:
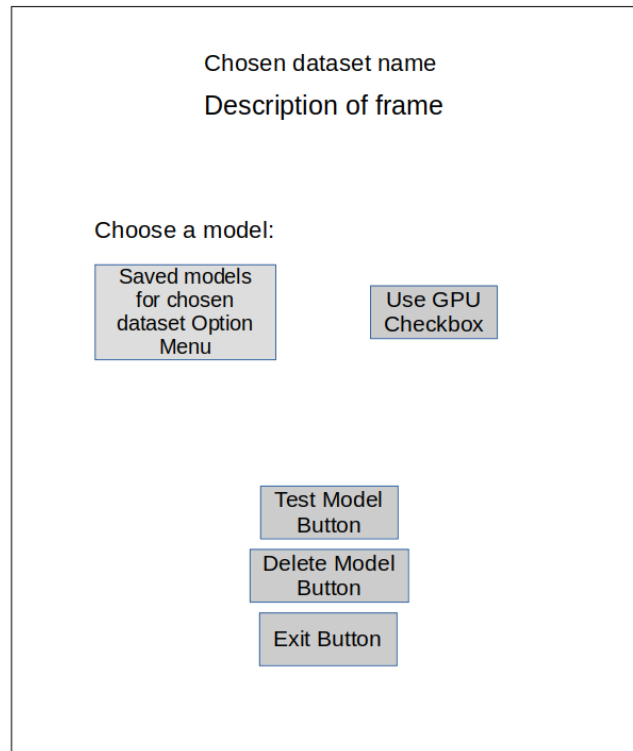
Training weights and biases...

Epoch x / y

Stop Training Model Button

• Load Model Frame design:

Chosen dataset name

Description of frame

Choose a model:

Saved models
for chosen
dataset Option
Menu

Use GPU
Checkbox

Test Model
Button

Delete Model
Button

Exit Button

- Test Frame design:

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 computationaly powerfull 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 seperately.

- 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 TODO

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.

```
.
|-- .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, 48 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,
    ↪   TestXORFrame
6
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='1.0.0',
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                          'pympler'
17      ],
18  )
```

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       ```
```

```
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 Nvdia drivers must be installed and the CUDA Toolkit must
 be installed from
<a href="https://developer.nvidia.com/cuda-downloads">here</a>.*

## 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/
  ```

Compile Project Report PDF with:
```
make all
```
*Note: This requires the Latexmk library*
```

– Which will generate the following:

A-level Computer Science NEA Programming Project

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

Installation

1. Download the Repository with:

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

    ◦ Or by downloading as a ZIP file

2. Create a virtual environment (venv) with:

    ◦ Windows:
    ```
    python -m venv {venv name}
    ```

    ◦ Linux:
    ```
    python3 -m venv {venv name}
    ```

3. Enter the venv with:

   ○ Windows:

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

   ○ Linux:

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

4. Enter the project directory with:

   ```
   cd school-project/
   ```

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

   ○ Windows:

   ```
   python setup.py install
   ```

   ○ Linux:

   ```
   python3 setup.py install
   ```

Note: In order to use an Nvidia GPU for training the networks, the latest Nvdia 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/
```

Compile Project Report PDF with:

```
make all
```

Note: This requires the Latexmk library

- 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:

```python
"""Helper functions and ModelInterface class for model module."""

from abc import ABC, abstractmethod

import numpy as np

class ModelInterface(ABC):
    """Interface for ANN models."""
    @abstractmethod
    def _setup_layers(setup_values: callable) -> None:
        """Decorator that sets up model layers and sets up values of each
        layer
            with the method given.

        Args:
            setup_values (callable): the method that sets up the values of
        each
                layer.
        Raises:
            NotImplementedError: if this method is not implemented.

        """
        raise NotImplementedError

    @abstractmethod
    def create_model_values(self) -> None:
        """Create weights and bias/biases

        Raises:
            NotImplementedError: if this method is not implemented.

        """
        raise NotImplementedError

    @abstractmethod
    def load_model_values(self, file_location: str) -> None:
        """Load weights and bias/biases from .npz file.

        Args:
            file_location (str): the location of the file to load from.
        Raises:
            NotImplementedError: if this method is not implemented.

        """
        raise NotImplementedError

    @abstractmethod
```

```python
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 |
        ↪   float:
169         """Calculate derivative of sigmoid Transfer function with respect to z.
170
171         Args:
172             output (numpy.ndarray | int | float):
173             the numpy.ndarray | int | float output of the sigmoid transfer
        ↪   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 *
        ↪   np.log(prediction) + (1 - outputs) * np.log(1 - prediction)))
202
203     def calculate_prediction_accuracy(prediction: np.ndarray,
204                                       outputs: np.ndarray) -> float:
205         """Calculate the percentage accuracy of the predictions.
206
207         Args:
208             prediction (np.ndarray): the array of prediction values.
209             outputs (np.ndarray):
210             the train/test outputs array to compare with the prediction.
211         Returns:
212             float prediction accuracy
213
214         """
215         return 100 - np.mean(np.abs(prediction - outputs)) * 100
```

- model.py module:

```
1   """Provides an abstract class for Artificial Neural Network models."""
2
3   import time
```

```python
import numpy as np

from .tools import (
                    ModelInterface,
                    relu,
                    relu_derivative,
                    sigmoid,
                    sigmoid_derivative,
                    calculate_loss,
                    calculate_prediction_accuracy
                    )

class _Layers():
    """Manages linked list of layers."""
    def __init__(self) -> None:
        """Initialise linked list."""
        self.head = None
        self.tail = None

    def __iter__(self) -> None:
        """Iterate forward through the network."""
        current_layer = self.head
        while True:
            yield current_layer
            if current_layer.next_layer is not None:
                current_layer = current_layer.next_layer
            else:
                break

    def __reversed__(self) -> None:
        """Iterate back through the network."""
        current_layer = self.tail
        while True:
            yield current_layer
            if current_layer.previous_layer is not None:
                current_layer = current_layer.previous_layer
            else:
                break

class _FullyConnectedLayer():
    """Fully connected layer for Deep ANNs,
       represented as a node of a Doubly linked list."""
    def __init__(self, learning_rate: float, input_neuron_count: int,
                 output_neuron_count: int, transfer_type: str) -> None:
        """Initialise layer values.

        Args:
            learning_rate (float): the learning rate of the model.
            input_neuron_count (int):
            the number of input neurons into the layer.
            output_neuron_count (int):
            the number of output neurons into the layer.
            transfer_type (str): the transfer function type
            ('sigmoid' or 'relu')

        """
        # Setup layer attributes
        self.previous_layer = None
        self.next_layer = None
        self.input_neuron_count = input_neuron_count
        self.output_neuron_count = output_neuron_count
```

```python
66              self.transfer_type = transfer_type
67              self.input: np.ndarray
68              self.output: np.ndarray
69
70              # Setup weights and biases
71              self.weights: np.ndarray
72              self.biases: np.ndarray
73              self.learning_rate = learning_rate
74
75          def __repr__(self) -> str:
76              """Read values of the layer.
77
78              Returns:
79                  a string description of the layers's
80                  weights, bias and learning rate values.
81
82              """
83              return (f"Weights: {self.weights.tolist()}\n" +
84                      f"Biases: {self.biases.tolist()}\n")
85
86          def init_layer_values_random(self) -> None:
87              """Initialise weights to random values and biases to 0s"""
88              np.random.seed(1)  # Sets up pseudo random values for layer weight
             ↪   arrays
89              self.weights = np.random.rand(self.output_neuron_count,
             ↪   self.input_neuron_count) - 0.5
90              self.biases = np.zeros(shape=(self.output_neuron_count, 1))
91
92          def init_layer_values_zeros(self) -> None:
93              """Initialise weights to 0s and biases to 0s"""
94              self.weights = np.zeros(shape=(self.output_neuron_count,
             ↪   self.input_neuron_count))
95              self.biases = np.zeros(shape=(self.output_neuron_count, 1))
96
97          def back_propagation(self, dloss_doutput) -> np.ndarray:
98              """Adjust the weights and biases via gradient descent.
99
100             Args:
101                 dloss_doutput (numpy.ndarray): the derivative of the loss of
     ↪   the
102                 layer's output, with respect to the layer's output.
103             Returns:
104                 a numpy.ndarray derivative of the loss of the layer's input,
105                 with respect to the layer's input.
106             Raises:
107                 ValueError:
108                 if dloss_doutput
109                 is not a suitable multiplier with the weights
110                 (incorrect shape)
111
112             """
113             match self.transfer_type:
114                 case 'sigmoid':
115                     dloss_dz = dloss_doutput *
                     ↪   sigmoid_derivative(output=self.output)
116                 case 'relu':
117                     dloss_dz = dloss_doutput *
                     ↪   relu_derivative(output=self.output)
118
119             dloss_dweights = np.dot(dloss_dz, self.input.T)
120             dloss_dbiases = np.sum(dloss_dz)
121
```

```python
122                assert dloss_dweights.shape == self.weights.shape
123
124                dloss_dinput = np.dot(self.weights.T, dloss_dz)
125
126                # Update weights and biases
127                self.weights -= self.learning_rate * dloss_dweights
128                self.biases -= self.learning_rate * dloss_dbiases
129
130                return dloss_dinput
131
132        def forward_propagation(self, inputs) -> np.ndarray:
133            """Generate a layer output with the weights and biases.
134
135            Args:
136                inputs (np.ndarray): the input values to the layer.
137            Returns:
138                a numpy.ndarray of the output values.
139
140            """
141            self.input = inputs
142            z = np.dot(self.weights, self.input) + self.biases
143            if self.transfer_type == 'sigmoid':
144                self.output = sigmoid(z)
145            elif self.transfer_type == 'relu':
146                self.output = relu(z)
147            return self.output
148
149    class AbstractModel(ModelInterface):
150        """ANN model with variable number of hidden layers"""
151        def __init__(self,
152                     hidden_layers_shape: list[int],
153                     train_dataset_size: int,
154                     learning_rate: float,
155                     use_relu: bool) -> None:
156            """Initialise model values.
157
158            Args:
159                hidden_layers_shape (list[int]):
160                list of the number of neurons in each hidden layer.
161                train_dataset_size (int): the number of train dataset inputs to
    ↪   use.
162                learning_rate (float): the learning rate of the model.
163                use_relu (bool): True or False whether the ReLu Transfer
    ↪   function
164                should be used.
165
166            """
167            # Setup model data
168            self.train_inputs, self.train_outputs,\
169            self.test_inputs, self.test_outputs = self.load_datasets(
170
                                                ↪   train_dataset_size=train_dataset_size
171                                                )
172            self.train_losses: list[float]
173            self.test_prediction: np.ndarray
174            self.test_prediction_accuracy: float
175            self.training_progress = ""
176            self.training_time: float
177
178            # Setup model attributes
179            self.__running = True
180            self.input_neuron_count: int = self.train_inputs.shape[0]
```

```python
181            self.input_count = self.train_inputs.shape[1]
182            self.hidden_layers_shape = hidden_layers_shape
183            self.output_neuron_count = self.train_outputs.shape[0]
184            self.layers_shape = [f'{layer}' for layer in (
185                            [self.input_neuron_count] +
186                            self.hidden_layers_shape +
187                            [self.output_neuron_count]
188                            )]
189            self.use_relu = use_relu
190
191            # Setup model values
192            self.layers = _Layers()
193            self.learning_rate = learning_rate
194
195        def __repr__(self) -> str:
196            """Read current state of model.
197
198            Returns:
199                a string description of the model's shape,
200                weights, bias and learning rate values.
201
202            """
203            return (f"Layers Shape: {','.join(self.layers_shape)}\n" +
204                    f"Learning Rate: {self.learning_rate}")
205
206        def set_running(self, value: bool) -> None:
207            """Set the running attribute to the given value.
208
209            Args:
210                value (bool): the value to set the running attribute to.
211
212            """
213            self.__running = value
214
215        def _setup_layers(setup_values: callable) -> None:
216            """Decorator that sets up model layers and sets up values of each
    ↪   layer
217                with the method given.
218
219            Args:
220                setup_values (callable): the method that sets up the values of
    ↪   each
221                layer.
222
223            """
224            def decorator(self, *args, **kwargs) -> None:
225                # Check if setting up Deep Network
226                if len(self.hidden_layers_shape) > 0:
227                    if self.use_relu:
228
229                        # Add input layer
230                        self.layers.head = _FullyConnectedLayer(
231
    ↪   learning_rate=self.learning_rate,
232
    ↪   input_neuron_count=self.input_neuron_count,
233
    ↪   output_neuron_count=self.hidden_layers_shape[0],
234                            transfer_type='relu'
235                            )
236                    current_layer = self.layers.head
237
```

```python
238                     # Add hidden layers
239                     for layer in range(len(self.hidden_layers_shape) - 1):
240                         current_layer.next_layer = _FullyConnectedLayer(
241                                     learning_rate=self.learning_rate,
242
                                     ↪  input_neuron_count=self.hidden_layers_shape[layer],
243
                                     ↪  output_neuron_count=self.hidden_layers_shape[layer
                                     ↪  + 1],
244                                     transfer_type='relu'
245                                     )
246                         current_layer.next_layer.previous_layer =
                         ↪  current_layer
247                         current_layer = current_layer.next_layer
248             else:
249
250                     # Add input layer
251                     self.layers.head = _FullyConnectedLayer(
252
                                         ↪  learning_rate=self.learning_rate,
253
                                         ↪  input_neuron_count=self.input_neuron_count,
254
                                         ↪  output_neuron_count=self.hidden_layers_shape[0],
255                                     transfer_type='sigmoid'
256                                     )
257                     current_layer = self.layers.head
258
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
                                     ↪  + 1],
265                                     transfer_type='sigmoid'
266                                     )
267                         current_layer.next_layer.previous_layer =
                         ↪  current_layer
268                         current_layer = current_layer.next_layer
269
270             # Add output layer
271             current_layer.next_layer = _FullyConnectedLayer(
272                                 learning_rate=self.learning_rate,
273
                                 ↪  input_neuron_count=self.hidden_layers_shape[-1],
274
                                 ↪  output_neuron_count=self.output_neuron_count,
275                                 transfer_type='sigmoid'
276                                 )
277             current_layer.next_layer.previous_layer = current_layer
278             self.layers.tail = current_layer.next_layer
279
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,
```

```python
286                                             transfer_type='sigmoid'
287                                         )
288                     self.layers.tail = self.layers.head
289
290                 setup_values(self, *args, **kwargs)
291
292             return decorator
293
294         @_setup_layers
295         def create_model_values(self) -> None:
296             """Create weights and bias/biases"""
297             # Check if setting up Deep Network
298             if len(self.hidden_layers_shape) > 0:
299
300                 # Initialise Layer values to random values
301                 for layer in self.layers:
302                     layer.init_layer_values_random()
303
304             # Setup Perceptron Network
305             else:
306
307                 # Initialise Layer values to zeros
308                 for layer in self.layers:
309                     layer.init_layer_values_zeros()
310
311         @_setup_layers
312         def load_model_values(self, file_location: str) -> None:
313             """Load weights and bias/biases from .npz file.
314
315             Args:
316                 file_location (str): the location of the file to load from.
317
318             """
319             data: dict[str, np.ndarray] = np.load(file=file_location)
320
321             # Initialise Layer values
322             i = 0
323             keys = list(data.keys())
324             for layer in self.layers:
325                 layer.weights = data[keys[i]]
326                 layer.biases = data[keys[i + 1]]
327                 i += 2
328
329         def back_propagation(self, dloss_doutput) -> None:
330             """Train each layer's weights and biases.
331
332             Args:
333                 dloss_doutput (np.ndarray): the derivative of the loss of the
334                 output layer's output, with respect to the output layer's
     ↪  output.
335
336             """
337             for layer in reversed(self.layers):
338                 dloss_doutput =
                     ↪  layer.back_propagation(dloss_doutput=dloss_doutput)
339
340         def forward_propagation(self) -> np.ndarray:
341             """Generate a prediction with the layers.
342
343             Returns:
344                 a numpy.ndarray of the prediction values.
345
```

41

```python
346            """
347            output = self.train_inputs
348            for layer in self.layers:
349                output = layer.forward_propagation(inputs=output)
350            return output
351
352        def test(self) -> None:
353            """Test the layers' trained weights and biases."""
354            output = self.test_inputs
355            for layer in self.layers:
356                output = layer.forward_propagation(inputs=output)
357            self.test_prediction = output
358
359            # Calculate performance of model
360            self.test_prediction_accuracy = calculate_prediction_accuracy(
361
                                                        ↪  prediction=self.test_prediction,
362                                                        outputs=self.test_outputs
363                                                        )
364
365        def train(self, epoch_count: int) -> None:
366            """Train layers' weights and biases.
367
368               Args:
369                   epoch_count (int): the number of training epochs.
370
371            """
372            self.layers_shape = [f'{layer}' for layer in (
373                                [self.input_neuron_count] +
374                                self.hidden_layers_shape +
375                                [self.output_neuron_count]
376                                )]
377            self.train_losses = []
378            training_start_time = time.time()
379            for epoch in range(epoch_count):
380                if not self.__running:
381                    break
382                self.training_progress = f"Epoch {epoch} / {epoch_count}"
383                prediction = self.forward_propagation()
384                loss = calculate_loss(input_count=self.input_count,
385                                      outputs=self.train_outputs,
386                                      prediction=prediction)
387                self.train_losses.append(loss)
388                if not self.__running:
389                    break
390                dloss_doutput = -(1/self.input_count) * ((self.train_outputs -
                ↪  prediction)/(prediction * (1 - prediction)))
391                self.back_propagation(dloss_doutput=dloss_doutput)
392            self.training_time = round(number=time.time() -
            ↪  training_start_time,
393                                       ndigits=2)
394
395        def save_model_values(self, file_location: str) -> None:
396            """Save the model by saving the weights then biases of each layer
        ↪  to
397            a .npz file with a given file location.
398
399               Args:
400                   file_location (str): the file location to save the model to.
401
402            """
403            saved_model: list[np.ndarray] = []
```

```
404             for layer in self.layers:
405                 saved_model.append(layer.weights)
406                 saved_model.append(layer.biases)
407             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          """Load image input and output datasets.
34
35          Args:
36              train_dataset_size (int): the number of train dataset inputs to
↪   use.
37          Returns:
38              tuple of image train_inputs, train_outputs,
39              test_inputs and test_outputs numpy.ndarrys.
40
41          Raises:
```

```
42              FileNotFoundError: if file does not exist.

43
44          """
45          # Load datasets from h5 files
46          # (h5 files stores large amount of data with quick access)
47          train_dataset: h5py.File = h5py.File(
48              r'school_project/models/datasets/train-cat.h5',
49              'r'
50              )
51          test_dataset: h5py.File = h5py.File(
52              r'school_project/models/datasets/test-cat.h5',
53              'r'
54              )

55
56          # Load input arrays,
57          # containing the RGB values for each pixel in each 64x64 pixel
            ↪   image,
58          # for 209 images
59          train_inputs: np.ndarray =
            ↪   np.array(train_dataset['train_set_x'][:])
60          test_inputs: np.ndarray = np.array(test_dataset['test_set_x'][:])

61
62          # Load output arrays of 1s for cat and 0s for not cat
63          train_outputs: np.ndarray =
            ↪   np.array(train_dataset['train_set_y'][:])
64          test_outputs: np.ndarray = np.array(test_dataset['test_set_y'][:])

65
66          # Reshape input arrays into 1 dimension (flatten),
67          # then divide by 255 (RGB)
68          # to standardize them to a number between 0 and 1
69          train_inputs = train_inputs.reshape((train_inputs.shape[0],
70                                               -1)).T / 255
71          test_inputs = test_inputs.reshape((test_inputs.shape[0], -1)).T /
            ↪   255

72
73          # Reshape output arrays into a 1 dimensional list of outputs
74          train_outputs = train_outputs.reshape((1, train_outputs.shape[0]))
75          test_outputs = test_outputs.reshape((1, test_outputs.shape[0]))

76
77          # Reduce train datasets' sizes to train_dataset_size
78          train_inputs = (train_inputs.T[:train_dataset_size]).T
79          train_outputs = (train_outputs.T[:train_dataset_size]).T

80
81          return train_inputs, train_outputs, test_inputs, test_outputs
```

- mnist.py module:

```
1   """Implementation of Artificial Neural Network model on MNIST dataset."""
2
3   import pickle
4   import gzip
5
6   import numpy as np
7
8   from .utils.model import AbstractModel
9
10  class MNISTModel(AbstractModel):
11      """ANN model that trains to predict Numbers from images."""
12      def __init__(self, hidden_layers_shape: list[int],
13                   train_dataset_size: int,
14                   learning_rate: float,
```

```python
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
    ↪   use.
22                learning_rate (float): the learning rate of the model.
23                use_relu (bool): True or False whether the ReLu Transfer
    ↪   function
24                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,
    ↪   np.ndarray,
33                                                                  np.ndarray,
                                                                  ↪   np.ndarray]:
34            """Load image input and output datasets.
35            Args:
36                train_dataset_size (int): the number of dataset inputs to use.
37            Returns:
38                tuple of image train_inputs, train_outputs,
39                test_inputs and test_outputs numpy.ndarrys.
40
41            Raises:
42                FileNotFoundError: if file does not exist.
43
44            """
45            # Load datasets from pkl.gz file
46            with gzip.open(
47                    'school_project/models/datasets/mnist.pkl.gz',
48                    'rb'
49                    ) as mnist:
50                (train_inputs, train_outputs),\
51                (test_inputs, test_outputs) = pickle.load(mnist,
                    ↪   encoding='bytes')
52
53            # Reshape input arrays into 1 dimension (flatten),
54            # then divide by 255 (RGB)
55            # to standardize them to a number between 0 and 1
56            train_inputs =
    ↪   np.array(train_inputs.reshape((train_inputs.shape[0],
57                                           -1)).T / 255)
58            test_inputs = np.array(test_inputs.reshape(test_inputs.shape[0],
                ↪   -1).T / 255)
59
60            # Represent number values
61            # with a one at the matching index of an array of zeros
62            train_outputs = np.eye(np.max(train_outputs) + 1)[train_outputs].T
63            test_outputs = np.eye(np.max(test_outputs) + 1)[test_outputs].T
64
65            # Reduce train datasets' sizes to train_dataset_size
66            train_inputs = (train_inputs.T[:train_dataset_size]).T
67            train_outputs = (train_outputs.T[:train_dataset_size]).T
68
69            return train_inputs, train_outputs, test_inputs, test_outputs
```

- xor.py module

```python
"""Implementation of Artificial Neural Network model on XOR dataset."""

import numpy as np

from .utils.model import AbstractModel


class XORModel(AbstractModel):
    """ANN model that trains to predict the output of a XOR gate with two
        inputs."""
    def __init__(self,
                 hidden_layers_shape: list[int],
                 train_dataset_size: int,
                 learning_rate: float,
                 use_relu: bool) -> None:
        """Initialise Model's Base class.

        Args:
            hidden_layers_shape (list[int]):
            list of the number of neurons in each hidden layer.
            train_dataset_size (int): the number of train dataset inputs to
    use.
            learning_rate (float): the learning rate of the model.
            use_relu (bool): True or False whether the ReLu Transfer
    function
            should be used.

        """
        super().__init__(hidden_layers_shape=hidden_layers_shape,
                         train_dataset_size=train_dataset_size,
                         learning_rate=learning_rate,
                         use_relu=use_relu)

    def load_datasets(self, train_dataset_size: int) -> tuple[np.ndarray,
    np.ndarray,
                                                              np.ndarray,
                                                              np.ndarray]:
        """Load XOR input and output datasets.

        Args:
            train_dataset_size (int): the number of dataset inputs to use.
        Returns:
            tuple of XOR train_inputs, train_outputs,
            test_inputs and test_outputs numpy.ndarrys.

        """
        inputs: np.ndarray = np.array([[0, 0, 1, 1],
                                       [0, 1, 0, 1]])
        outputs: np.ndarray = np.array([[0, 1, 1, 0]])

        # Reduce train datasets' sizes to train_dataset_size
        inputs = (inputs.T[:train_dataset_size]).T
        outputs = (outputs.T[:train_dataset_size]).T

        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:

```json
{
    "MNIST": {
        "description": "An Image model trained on recognising numbers from
          images.",
        "epochCount": 150,
        "hiddenLayersShape": [1000, 1000],
        "minTrainDatasetSize": 1,
        "maxTrainDatasetSize": 60000,
        "maxLearningRate": 1
    },
    "Cat Recognition": {
        "description": "An Image model trained on recognising if an image
          is a cat or not.",
        "epochCount": 3500,
        "hiddenLayersShape": [100, 100],
        "minTrainDatasetSize": 1,
        "maxTrainDatasetSize": 209,
        "maxLearningRate": 0.3
    },
    "XOR": {
        "description": "For experimenting with Artificial Neural Networks,
          a XOR gate model has been used for its lesser computation time.",
        "epochCount": 4700,
        "hiddenLayersShape": [100, 100],
        "minTrainDatasetSize":  2,
        "maxTrainDatasetSize": 4,
        "maxLearningRate": 1
    }
}
```

- create_model.py module:

```python
"""Tkinter frames for creating an Artificial Neural Network model."""

import json
import threading
import tkinter as tk
import tkinter.font as tkf

from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import numpy as np

class HyperParameterFrame(tk.Frame):
    """Frame for hyper-parameter page."""
    def __init__(self, root: tk.Tk, width: int,
                 height: int, bg: str, dataset: str) -> None:
        """Initialise hyper-parameter frame widgets.

        Args:
            root (tk.Tk): the widget object that contains this widget.
```

47

```python
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
                                                                     ↪   dataset=dataset
40                                                                     )
41
42          # Setup widgets
43          self.title_label = tk.Label(master=self,
44                                      bg=self.BG,
45                                      font=('Arial', 20),
46                                      text=dataset)
47          self.about_label = tk.Label(
48                              master=self,
49                              bg=self.BG,
50                              font=('Arial', 14),
51
                                  ↪   text=self.default_hyper_parameters['description']
52                              )
53          self.learning_rate_scale = tk.Scale(
54                              master=self,
55                              bg=self.BG,
56                              orient='horizontal',
57                              label="Learning Rate",
58                              length=185,
59                              from_=0,
60
                                  ↪   to=self.default_hyper_parameters['maxLearningRate'],
61                              resolution=0.01
62                              )
63          self.learning_rate_scale.set(value=0.1)
64          self.epoch_count_scale = tk.Scale(master=self,
65                                            bg=self.BG,
66                                            orient='horizontal',
67                                            label="Epoch Count",
68                                            length=185,
69                                            from_=0,
70                                            to=10_000,
71                                            resolution=100)
72          self.epoch_count_scale.set(
73
                                  ↪   value=self.default_hyper_parameters['epochCount']
74                              )
75          self.train_dataset_size_scale = tk.Scale(
```

48

```python
76                        master=self,
77                        bg=self.BG,
78                        orient='horizontal',
79                        label="Train Dataset Size",
80                        length=185,
81
                          ↪  from_=self.default_hyper_parameters['minTrainDatasetSize'],
82                        to=self.default_hyper_parameters['maxTrainDatasetSize'],
83                        resolution=1
84                        )
85            self.train_dataset_size_scale.set(
86
                             ↪  value=self.default_hyper_parameters['maxTrainDatasetSize']
87                        )
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" +
93                                    "hidden layer, separated by
                                    ↪  commas:"
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] |
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
    ↪   hyper-parameters
141                 for. ('MNIST', 'Cat Recognition' or 'XOR')
142             Returns:
143                 a dictionary of default hyper-parameter values.
144         """
145         with open('school_project/frames/hyper-parameter-defaults.json') as
           ↪   f:
146             return json.load(f)[dataset]
147
148     def create_model(self) -> object:
149         """Create and return a Model using the hyper-parameters set.
150
151             Returns:
152                 a Model object.
153         """
154         self.use_gpu = self.use_gpu_check_button_var.get()
155
156         # Validate hidden layers shape input
157         hidden_layers_shape_input = [layer for layer in
           ↪   self.hidden_layers_shape_entry.get().replace(' ',
           ↪   '').split(',') if layer != '']
158         for layer in hidden_layers_shape_input:
159             if not layer.isdigit():
160                 self.model_status_label.configure(
161                                      text="Invalid hidden layers shape",
162                                      fg='red'
163                                      )
164                 raise ValueError
165
166         # Create Model
167         if not self.use_gpu:
168             if self.dataset == "MNIST":
169                 from school_project.models.cpu.mnist import MNISTModel as
                   ↪   Model
170             elif self.dataset == "Cat Recognition":
171                 from school_project.models.cpu.cat_recognition import
                   ↪   CatRecognitionModel as Model
172             elif self.dataset == "XOR":
173                 from school_project.models.cpu.xor import XORModel as Model
174             model = Model(
175                 hidden_layers_shape = [int(neuron_count) for neuron_count
                   ↪   in hidden_layers_shape_input],
176                 train_dataset_size = self.train_dataset_size_scale.get(),
177                 learning_rate = self.learning_rate_scale.get(),
178                 use_relu = self.use_relu_check_button_var.get()
179                 )
180             model.create_model_values()
181
182         else:
183             try:
184                 if self.dataset == "MNIST":
```

```
185                          from school_project.models.gpu.mnist import MNISTModel
                          ↪  as Model
186                    elif self.dataset == "Cat Recognition":
187                        from school_project.models.gpu.cat_recognition import
                          ↪  CatRecognitionModel as Model
188                    elif self.dataset == "XOR":
189                        from school_project.models.gpu.xor import XORModel as
                          ↪  Model
190                    model = Model(hidden_layers_shape = [int(neuron_count) for
                      ↪  neuron_count in hidden_layers_shape_input],
191                                  train_dataset_size =
                                  ↪  self.train_dataset_size_scale.get(),
192                                  learning_rate =
                                  ↪  self.learning_rate_scale.get(),
193                                  use_relu =
                                  ↪  self.use_relu_check_button_var.get())
194                    model.create_model_values()
195            except ImportError as ie:
196                self.model_status_label.configure(
197                                        text="Failed to initialise GPU",
198                                        fg='red'
199                                        )
200                raise ImportError
201        return model
202
203  class TrainingFrame(tk.Frame):
204      """Frame for training page."""
205      def __init__(self, root: tk.Tk, width: int,
206                   height: int, bg: str,
207                   model: object, epoch_count: int) -> None:
208          """Initialise training frame widgets.
209
210          Args:
211              root (tk.Tk): the widget object that contains this widget.
212              width (int): the pixel width of the frame.
213              height (int): the pixel height of the frame.
214              bg (str): the hex value or name of the frame's background
     ↪  colour.
215              model (object): the Model object to be trained.
216              epoch_count (int): the number of training epochs.
217          Raises:
218              TypeError: if root, width or height are not of the correct
     ↪  type.
219
220          """
221          super().__init__(master=root, width=width, height=height, bg=bg)
222          self.root = root
223          self.WIDTH = width
224          self.HEIGHT = height
225          self.BG = bg
226
227          # Setup widgets
228          self.model_status_label = tk.Label(master=self,
229                                             bg=self.BG,
230                                             font=('Arial', 15))
231          self.training_progress_label = tk.Label(master=self,
232                                             bg=self.BG,
233                                             font=('Arial', 15))
234          self.loss_figure: Figure = Figure()
235          self.loss_canvas: FigureCanvasTkAgg = FigureCanvasTkAgg(
236
                                                      ↪  figure=self.loss_figure,
```

```
237                                                              master=self
238                                                              )
239
240            # Pack widgets
241            self.model_status_label.pack(pady=(30,0))
242            self.training_progress_label.pack(pady=30)
243
244            # Start training thread
245            self.model_status_label.configure(
246                                    text="Training weights and
                                    ↪   biases...",
247                                    fg='red'
248                                    )
249            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
                        ↪   {model.training_time}s",
264                    fg='green'
265                    )
266            graph: Figure.axes = self.loss_figure.add_subplot(111)
267            graph.set_title("Learning rate: " +
268                        f"{model.learning_rate}")
269            graph.set_xlabel("Epochs")
270            graph.set_ylabel("Loss Value")
271            graph.plot(np.squeeze(model.train_losses))
272            self.loss_canvas.get_tk_widget().pack()
```

Which outputs the following for the hyper-parameter frame:
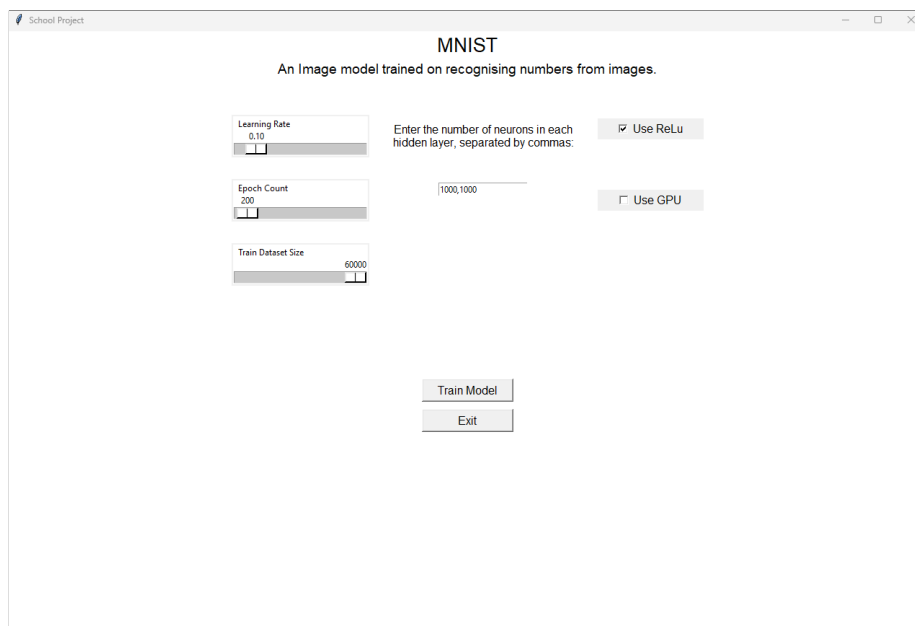
And outputs the following for the training frame during training:
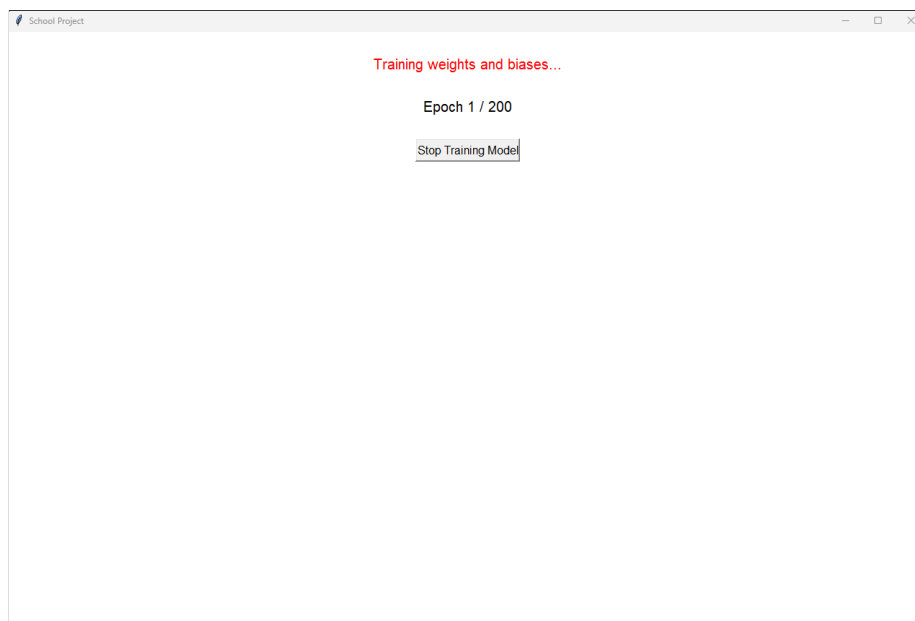
And outputs the following for the training frame once training has completed:
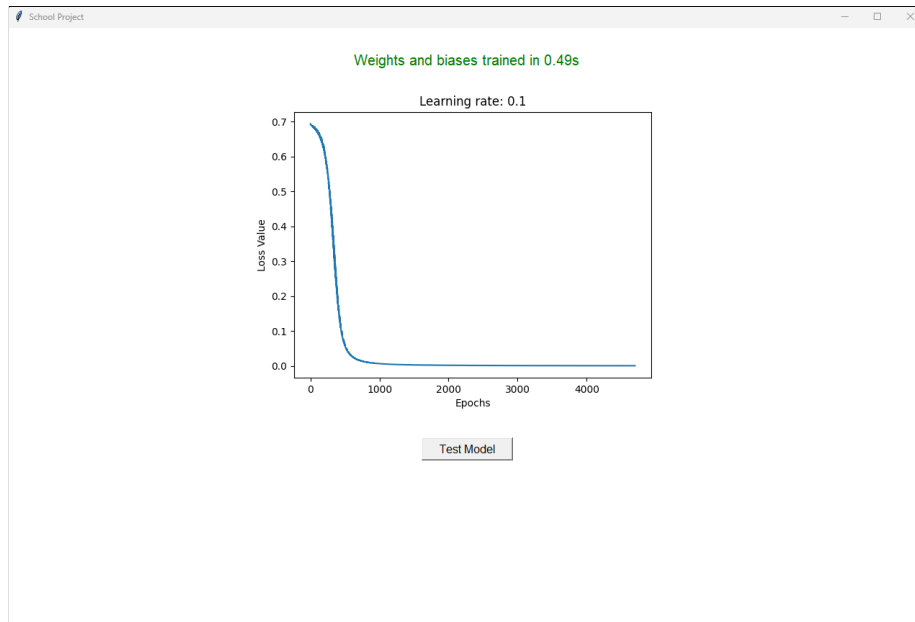
- load_model.py module:

```python
"""Tkinter frames for loading a saved Artificial Neural Network Model."""

import sqlite3
import tkinter as tk
import tkinter.font as tkf

class LoadModelFrame(tk.Frame):
    """Frame for load model page."""
    def __init__(self, root: tk.Tk,
                 width: int, height: int,
                 bg: str, connection: sqlite3.Connection,
                 cursor: sqlite3.Cursor, dataset: str) -> None:
        """Initialise load model frame widgets.

        Args:
            root (tk.Tk): the widget object that contains this widget.
            width (int): the pixel width of the frame.
            height (int): the pixel height of the frame.
            bg (str): the hex value or name of the frame's background
    ↪   colour.
            connection (sqlite3.Connection): the database connection
    ↪   object.
            cursor (sqlite3.Cursor): the database cursor object.
            dataset (str): the name of the dataset to use
            ('MNIST', 'Cat Recognition' or 'XOR')
        Raises:
            TypeError: if root, width or height are not of the correct
    ↪   type.

        """
```

```python
            super().__init__(master=root, width=width, height=height, bg=bg)
            self.root = root
            self.WIDTH = width
            self.HEIGHT = height
            self.BG = bg

            # Setup load model frame variables
            self.connection = connection
            self.cursor = cursor
            self.dataset = dataset
            self.use_gpu: bool
            self.model_options = self.load_model_options()

            # Setup widgets
            self.title_label = tk.Label(master=self,
                                        bg=self.BG,
                                        font=('Arial', 20),
                                        text=dataset)
            self.about_label = tk.Label(
                        master=self,
                        bg=self.BG,
                        font=('Arial', 14),
                        text=f"Load a pretrained model for the {dataset}
                        ↪  dataset."
                        )
            self.model_status_label = tk.Label(master=self,
                                               bg=self.BG,
                                               font=('Arial', 15))

            # Don't give loaded model options if no models have been saved for
            ↪  the
            # dataset.
            if len(self.model_options) > 0:
                self.model_option_menu_label = tk.Label(
                                                master=self,
                                                bg=self.BG,
                                                font=('Arial', 14),
                                                text="Select a model to
                                                ↪  load or delete:"
                                                )
                self.model_option_menu_var = tk.StringVar(
                                                master=self,

                                                ↪  value=self.model_options[0]
                                                )
                self.model_option_menu = tk.OptionMenu(
                                                self,

                                                ↪  self.model_option_menu_var,
                                                *self.model_options
                                                )
                self.use_gpu_check_button_var = tk.BooleanVar()
                self.use_gpu_check_button = tk.Checkbutton(
                                                master=self,
                                                width=7, height=1,
                                                font=tkf.Font(size=12),
                                                text="Use GPU",

                                                ↪  variable=self.use_gpu_check_button_var
                                                )
            else:
                self.model_status_label.configure(
```

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

56

```python
139                    elif self.dataset == "Cat Recognition":
140                        from school_project.models.cpu.cat_recognition import
        ↪   CatRecognitionModel as Model
141                    elif self.dataset == "XOR":
142                        from school_project.models.cpu.xor import XORModel as Model
143                    model = Model(
144                        hidden_layers_shape=[int(neuron_count) for neuron_count in
        ↪   hidden_layers_shape_input],
145                        train_dataset_size=data[6],
146                        learning_rate=data[4],
147                        use_relu=data[7]
148                        )
149                    model.load_model_values(file_location=data[2])

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

Which outputs the following for the load model frame when models have
been saved for the dataset:
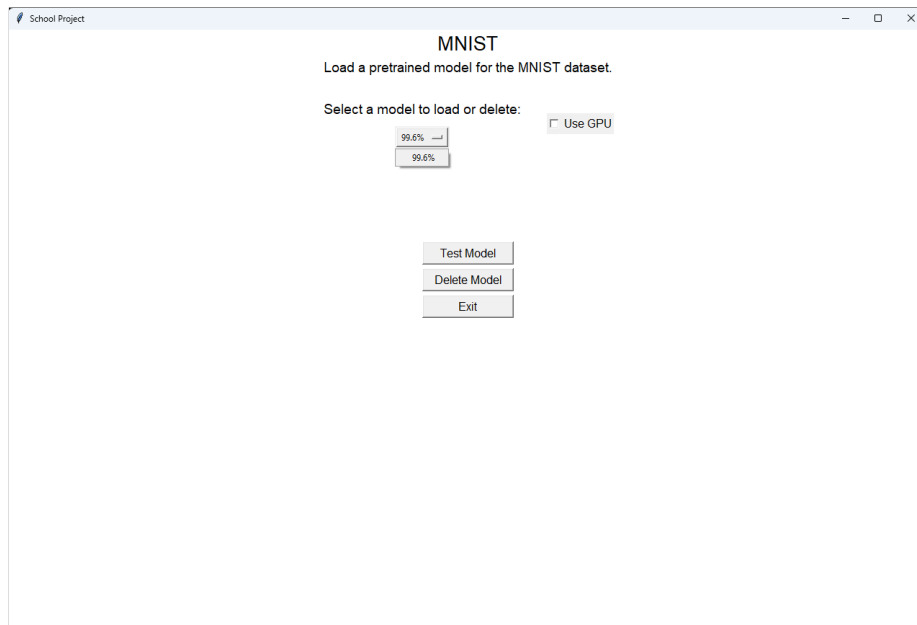
And outputs the following for the load model frame when no models have been saved for the dataset:
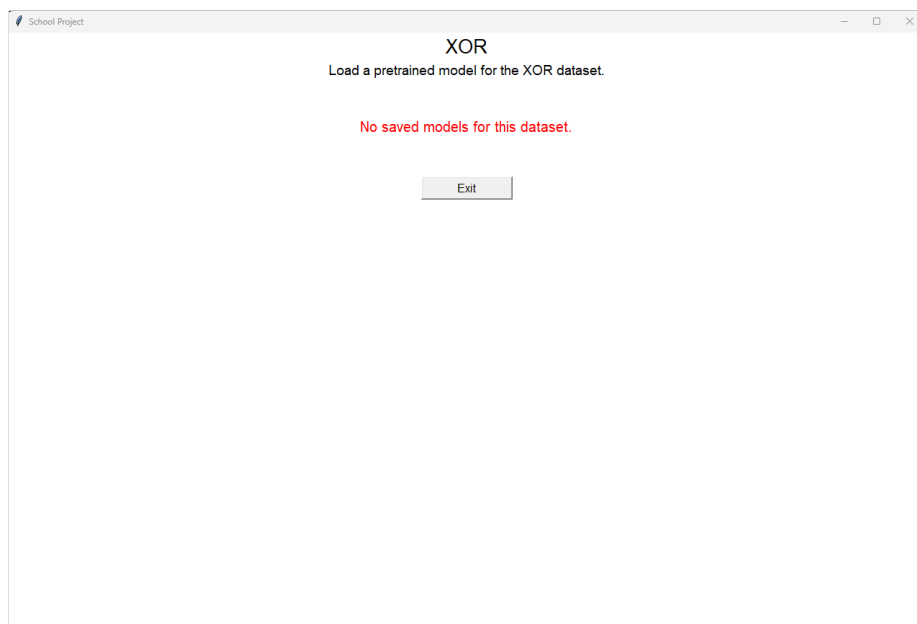
3.4   __main__.py module

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

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

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

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

```
187
188            """
189            connection = sqlite3.connect(
190                              database='school_project/saved_models.db'
191                              )
192            cursor = connection.cursor()
193            cursor.execute("""
194        CREATE TABLE IF NOT EXISTS Models
195        (Model_ID INTEGER PRIMARY KEY,
196        Dataset TEXT,
197        File_Location TEXT,
198        Hidden_Layers_Shape TEXT,
199        Learning_Rate FLOAT,
200        Name TEXT,
201        Train_Dataset_Size INTEGER,
202        Use_ReLu INTEGER,
203        UNIQUE (Dataset, Name))
204        """)
205            return (connection, cursor)

207    def enter_hyper_parameter_frame(self) -> None:
208        """Unpack home frame and pack hyper-parameter frame."""
209        self.home_frame.pack_forget()
210        self.hyper_parameter_frame = HyperParameterFrame(
211                              root=self,
212                              width=self.WIDTH,
213                              height=self.HEIGHT,
214                              bg=self.BG,
215                              dataset=self.dataset_option_menu_var.get()
216                              )
217        self.hyper_parameter_frame.pack()
218        self.train_button.pack()
219        self.exit_hyper_parameter_frame_button.pack(pady=(10,0))

221    def enter_load_model_frame(self) -> None:
222        """Unpack home frame and pack load model frame."""
223        self.home_frame.pack_forget()
224        self.load_model_frame = LoadModelFrame(
225                              root=self,
226                              width=self.WIDTH,
227                              height=self.HEIGHT,
228                              bg=self.BG,
229                              connection=self.connection,
230                              cursor=self.cursor,
231                              dataset=self.dataset_option_menu_var.get()
232                              )
233        self.load_model_frame.pack()

235        # Don't give option to test loaded model if no models have been saved
236        # for the dataset.
237        if len(self.load_model_frame.model_options) > 0:
238            self.test_loaded_model_button.pack()
239            self.delete_loaded_model_button.pack(pady=(5,0))

241        self.exit_load_model_frame_button.pack(pady=(5,0))

243    def exit_hyper_parameter_frame(self) -> None:
244        """Unpack hyper-parameter frame and pack home frame."""
245        self.hyper_parameter_frame.pack_forget()
246        self.train_button.pack_forget()
247        self.exit_hyper_parameter_frame_button.pack_forget()
248        self.home_frame.pack()
```

```python
249
250        def exit_load_model_frame(self) -> None:
251            """Unpack load model frame and pack home frame."""
252            self.load_model_frame.pack_forget()
253            self.test_loaded_model_button.pack_forget()
254            self.delete_loaded_model_button.pack_forget()
255            self.exit_load_model_frame_button.pack_forget()
256            self.home_frame.pack()
257
258        def enter_training_frame(self) -> None:
259            """Load untrained model from hyper parameter frame,
260               unpack hyper-parameter frame, pack training frame
261               and begin managing the training thread.
262            """
263            try:
264                self.model = self.hyper_parameter_frame.create_model()
265            except (ValueError, ImportError) as e:
266                return
267            self.hyper_parameter_frame.pack_forget()
268            self.train_button.pack_forget()
269            self.exit_hyper_parameter_frame_button.pack_forget()
270            self.training_frame = TrainingFrame(
271                    root=self,
272                    width=self.WIDTH,
273                    height=self.HEIGHT,
274                    bg=self.BG,
275                    model=self.model,
276                    epoch_count=self.hyper_parameter_frame.epoch_count_scale.get()
277                    )
278            self.training_frame.pack()
279            self.stop_training_button.pack()
280            self.manage_training(train_thread=self.training_frame.train_thread)
281
282        def manage_training(self, train_thread: threading.Thread) -> None:
283            """Wait for model training thread to finish,
284               then plot training losses on training frame.
285
286            Args:
287                train_thread (threading.Thread):
288                the thread running the model's train() method.
289            Raises:
290                TypeError: if train_thread is not of type threading.Thread.
291
292            """
293            if not train_thread.is_alive():
294                self.training_frame.training_progress_label.pack_forget()
295                self.training_frame.plot_losses(model=self.model)
296                self.stop_training_button.pack_forget()
297                self.test_created_model_button.pack(pady=(30,0))
298            else:
299                self.training_frame.training_progress_label.configure(
300                                                text=self.model.training_progress
301                                                )
302                self.after(100, self.manage_training, train_thread)
303
304        def test_created_model(self) -> None:
305            """Unpack training frame, pack test frame for the dataset
306               and begin managing the test thread."""
307            self.saving_model = True
308            self.training_frame.pack_forget()
309            self.test_created_model_button.pack_forget()
310            if self.hyper_parameter_frame.dataset == "MNIST":
```

```python
                    self.test_frame = TestMNISTFrame(
                                          root=self,
                                          width=self.WIDTH,
                                          height=self.HEIGHT,
                                          bg=self.BG,
                                          use_gpu=self.hyper_parameter_frame.use_gpu,
                                          model=self.model
                                          )
            elif self.hyper_parameter_frame.dataset == "Cat Recognition":
                    self.test_frame = TestCatRecognitionFrame(
                                          root=self,
                                          width=self.WIDTH,
                                          height=self.HEIGHT,
                                          bg=self.BG,
                                          use_gpu=self.hyper_parameter_frame.use_gpu,
                                          model=self.model
                                          )
            elif self.hyper_parameter_frame.dataset == "XOR":
                    self.test_frame = TestXORFrame(root=self,
                                          width=self.WIDTH,
                                          height=self.HEIGHT,
                                          bg=self.BG,
                                          model=self.model)
            self.test_frame.pack()
            self.manage_testing(test_thread=self.test_frame.test_thread)

    def test_loaded_model(self) -> None:
        """Load saved model from load model frame, unpack load model frame,
            pack test frame for the dataset and begin managing the test thread."""
        self.saving_model = False
        try:
            self.model = self.load_model_frame.load_model()
        except (ValueError, ImportError) as e:
            return
        self.load_model_frame.pack_forget()
        self.test_loaded_model_button.pack_forget()
        self.delete_loaded_model_button.pack_forget()
        self.exit_load_model_frame_button.pack_forget()
        if self.load_model_frame.dataset == "MNIST":
            self.test_frame = TestMNISTFrame(
                                  root=self,
                                  width=self.WIDTH,
                                  height=self.HEIGHT,
                                  bg=self.BG,
                                  use_gpu=self.load_model_frame.use_gpu,
                                  model=self.model
                                  )
        elif self.load_model_frame.dataset == "Cat Recognition":
            self.test_frame = TestCatRecognitionFrame(
                                  root=self,
                                  width=self.WIDTH,
                                  height=self.HEIGHT,
                                  bg=self.BG,
                                  use_gpu=self.load_model_frame.use_gpu,
                                  model=self.model
                                  )
        elif self.load_model_frame.dataset == "XOR":
            self.test_frame = TestXORFrame(root=self,
                                  width=self.WIDTH,
                                  height=self.HEIGHT,
                                  bg=self.BG,
                                  model=self.model)
```

```python
373                  self.test_frame.pack()
374                  self.manage_testing(test_thread=self.test_frame.test_thread)
375
376          def manage_testing(self, test_thread: threading.Thread) -> None:
377              """Wait for model test thread to finish,
378                 then plot results on test frame.
379
380              Args:
381                  test_thread (threading.Thread):
382                  the thread running the model's predict() method.
383              Raises:
384                  TypeError: if test_thread is not of type threading.Thread.
385
386              """
387              if not test_thread.is_alive():
388                  self.test_frame.plot_results(model=self.model)
389                  if self.saving_model:
390                      self.save_model_label.pack(pady=(30,0))
391                      self.save_model_name_entry.pack(pady=10)
392                      self.save_model_button.pack()
393                  self.exit_button.pack(pady=(20,0))
394              else:
395                  self.after(1_000, self.manage_testing, test_thread)
396
397          def save_model(self) -> None:
398              """Save the model, save the model information to the database, then
399                 enter the home frame."""
400              model_name = self.save_model_name_entry.get()
401
402              # Check if model name is empty
403              if model_name == '':
404                  self.test_frame.model_status_label.configure(
405                                                  text="Model name can not be blank",
406                                                  fg='red'
407                                                  )
408                  return
409
410              # Check if model name has already been taken
411              dataset = self.dataset_option_menu_var.get().replace(" ", "_")
412              sql = """
413              SELECT Name FROM Models WHERE Dataset=?
414              """
415              parameters = (dataset,)
416              self.cursor.execute(sql, parameters)
417              for saved_model_name in self.cursor.fetchall():
418                  if saved_model_name[0] == model_name:
419                      self.test_frame.model_status_label.configure(
420                                                      text="Model name taken",
421                                                      fg='red'
422                                                      )
423                      return
424
425              # Save model to random hex file name
426              file_location = f"school_project/saved-models/{uuid.uuid4().hex}.npz"
427              self.model.save_model_values(file_location=file_location)
428
429              # Save the model information to the database
430              sql = """
431              INSERT INTO Models
432              (Dataset, File_Location, Hidden_Layers_Shape, Learning_Rate, Name,
     ↪ Train_Dataset_Size, Use_ReLu)
433              VALUES (?, ?, ?, ?, ?, ?, ?)
```

```
434                """
435            parameters = (
436                        dataset,
437                        file_location,
438                        self.hyper_parameter_frame.hidden_layers_shape_entry.get(),
439                        self.hyper_parameter_frame.learning_rate_scale.get(),
440                        model_name,
441                        self.hyper_parameter_frame.train_dataset_size_scale.get(),
442                        self.hyper_parameter_frame.use_relu_check_button_var.get()
443                        )
444            self.cursor.execute(sql, parameters)
445            self.connection.commit()
446
447            self.enter_home_frame()
448
449        def delete_loaded_model(self) -> None:
450            """Delete saved model file and model data from the database."""
451            dataset = self.dataset_option_menu_var.get().replace(" ", "_")
452            model_name = self.load_model_frame.model_option_menu_var.get()
453
454            # Delete saved model
455            sql = f"""SELECT File_Location FROM Models WHERE Dataset=? AND Name=?"""
456            parameters = (dataset, model_name)
457            self.cursor.execute(sql, parameters)
458            os.remove(self.cursor.fetchone()[0])
459
460            # Remove model data from database
461            sql = """DELETE FROM Models WHERE Dataset=? AND Name=?"""
462            parameters = (dataset, model_name)
463            self.cursor.execute(sql, parameters)
464            self.connection.commit()
465
466            # Reload load model frame with new options
467            self.exit_load_model_frame()
468            self.enter_load_model_frame()
469
470        def enter_home_frame(self) -> None:
471            """Unpack test frame and pack home frame."""
472            self.model = None  # Free up trained Model from memory
473            self.test_frame.pack_forget()
474            if self.saving_model:
475                self.save_model_label.pack_forget()
476                self.save_model_name_entry.delete(0, tk.END)  # Clear entry's text
477                self.save_model_name_entry.pack_forget()
478                self.save_model_button.pack_forget()
479            self.exit_button.pack_forget()
480            self.home_frame.pack()
481            summary_tracker.create_summary()  # BUG: Object summary seems to reduce
482                                              # memory leak greatly
483
484    def main() -> None:
485        """Entrypoint of project."""
486        root = tk.Tk()
487        school_project_frame = SchoolProjectFrame(root=root, width=1280,
488                                                  height=835, bg='white')
489        school_project_frame.pack(side='top', fill='both', expand=True)
490        root.mainloop()
491
492        # Stop model training when GUI closes
493        if school_project_frame.model is not None:
494            school_project_frame.model.set_running(value=False)
495
```
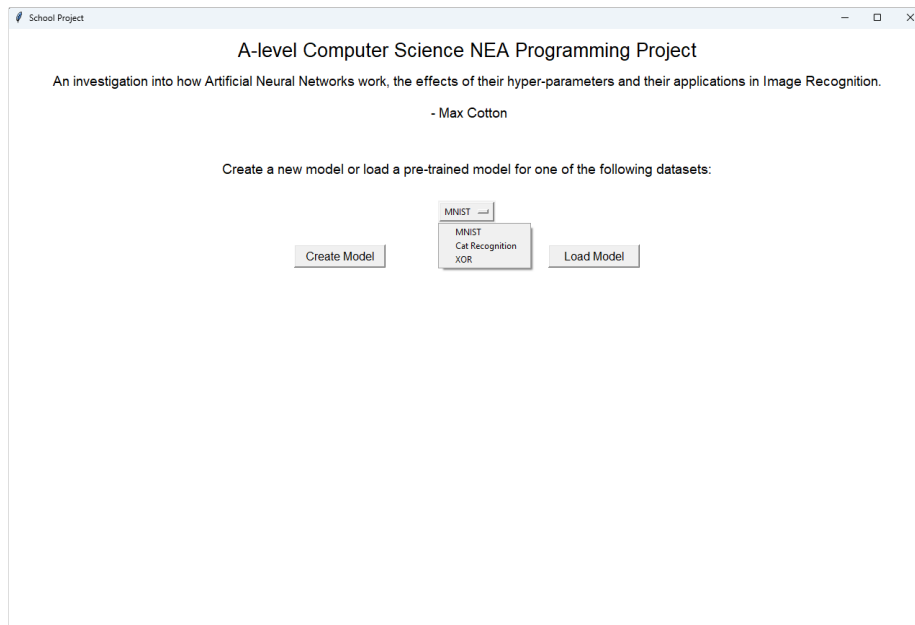
```
496  if __name__ == "__main__":
497      summary_tracker = tracker.SummaryTracker()   # Setup object tracker
498      main()
```

Which outputs the following for the home frame:

4   Testing TODO

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. 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,
```

```python
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:
```

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

```python
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])) ==
       ↪    test_outputs[i].index(max(test_outputs[i])):
140                    if image_count == 2:
141                        break
142                    elif image_count == 0:
143                        image = self.correct_prediction_figure.add_subplot(121)
144                    elif image_count == 1:
145                        image = self.correct_prediction_figure.add_subplot(122)
146                    image.set_title(f"Predicted:
       ↪    {test_prediction[i].index(max(test_prediction[i]))}")
147                    image.imshow(test_inputs[i].reshape((28,28)))
148                    image_count += 1
149
150    class TestCatRecognitionFrame(tk.Frame):
151        """Frame for Testing Cat Recognition page."""
152        def __init__(self, root: tk.Tk, width: int,
153                     height: int, bg: str,
154                     use_gpu: bool, model: object) -> None:
155            """Initialise test cat recognition frame widgets.
156
157            Args:
158                root (tk.Tk): the widget object that contains this widget.
159                width (int): the pixel width of the frame.
160                height (int): the pixel height of the frame.
161                bg (str): the hex value or name of the frame's background colour.
162                use_gpu (bool): True or False whether the GPU should be used.
163                model (object): the Model object to be tested.
164            Raises:
165                TypeError: if root, width or height are not of the correct type.
166
167            """
168            super().__init__(master=root, width=width, height=height, bg=bg)
169            self.root = root
170            self.WIDTH = width
171            self.HEIGHT = height
172            self.BG = bg
173
174            # Setup image recognition frame variables
175            self.use_gpu = use_gpu
176
177            # Setup widgets
178            self.model_status_label = tk.Label(master=self,
179                                               bg=self.BG,
180                                               font=('Arial', 15))
181            self.results_label = tk.Label(master=self,
182                                          bg=self.BG,
183                                          font=('Arial', 15))
184            self.correct_prediction_figure = Figure()
185            self.correct_prediction_canvas = FigureCanvasTkAgg(
186                                            figure=self.correct_prediction_figure,
187                                            master=self
188                                            )
189            self.incorrect_prediction_figure = Figure()
190            self.incorrect_prediction_canvas = FigureCanvasTkAgg(
191                                            figure=self.incorrect_prediction_figure,
```

70

```
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() -
                 ↪  model.test_outputs)) * 100, ndigits=1)}%\n" +
219              f"Network Shape: " +
220              f"{','.join(model.layers_shape)}\n"
221              )
222
223             # Randomly shuffle order of test_inputs, test_outputs and
                ↪  test_prediciton
224             # whilst maintaining order between them
225             test_data = list(zip(model.test_inputs.T,
226                                  np.squeeze(model.test_outputs).T.tolist(),
227
                                     ↪  np.squeeze(model.test_prediction.round()).T.tolist()))
228             random.shuffle(test_data)
229             (test_inputs,
230              test_outputs,
231              test_prediction) = map(lambda arr: np.array(arr).T,
232                                     zip(*test_data))
233
234         elif self.use_gpu:
235
236             import cupy as cp
237
238             self.results_label.configure(
239              text="Prediction Correctness: " +
240              f"{round(number=100 -
                 ↪  np.mean(np.abs(cp.asnumpy(model.test_prediction).round() -
                 ↪  cp.asnumpy(model.test_outputs))) * 100, ndigits=1)}%\n" +
241              f"Network Shape: " +
242              f"{','.join(model.layers_shape)}\n"
243              )
244
245             # Randomly shuffle order of test_inputs, test_outputs and
                ↪  test_prediciton
246             # whilst maintaining order between them
247             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
                        ↪  else 'Not a cat'}\n")
268                        image.imshow(test_inputs[:,i].reshape((64,64,3)))
269                        image_count += 1
270
271                # Setup correct prediction figure
272                self.correct_prediction_figure.suptitle("Correct predictions:")
273                image_count = 0
274                for i in range(len(test_prediction)):
275                    if test_prediction[i] == test_outputs[i]:
276                        if image_count == 2:
277                            break
278                        elif image_count == 0:
279                            image = self.correct_prediction_figure.add_subplot(121)
280                        elif image_count == 1:
281                            image = self.correct_prediction_figure.add_subplot(122)
282                        image.set_title(f"Predicted: {'Cat' if test_prediction[i] == 1
                        ↪  else 'Not a cat'}\n")
283                        image.imshow(test_inputs[:,i].reshape((64,64,3)))
284                        image_count += 1
285
286     class TestXORFrame(tk.Frame):
287         """Frame for Testing XOR page."""
288         def __init__(self, root: tk.Tk, width: int,
289                      height: int, bg: str, model: object) -> None:
290             """Initialise test XOR frame widgets.
291
292             Args:
293                 root (tk.Tk): the widget object that contains this widget.
294                 width (int): the pixel width of the frame.
295                 height (int): the pixel height of the frame.
296                 bg (str): the hex value or name of the frame's background colour.
297                 model (object): the Model object to be tested.
298             Raises:
299                 TypeError: if root, width or height are not of the correct type.
300
301             """
302             super().__init__(master=root, width=width, height=height, bg=bg)
303             self.root = root
304             self.WIDTH = width
305             self.HEIGHT = height
```

```
306            self.BG = bg
307
308            # Setup widgets
309            self.model_status_label = tk.Label(master=self,
310                                               bg=self.BG,
311                                               font=('Arial', 15))
312            self.results_label = tk.Label(master=self,
313                                          bg=self.BG,
314                                          font=('Arial', 20))
315
316            # Pack widgets
317            self.model_status_label.pack(pady=(30,0))
318
319            # Start test thread
320            self.model_status_label.configure(text="Testing trained model...",
321                                              fg='red')
322            self.test_thread = threading.Thread(target=model.test)
323            self.test_thread.start()
324
325        def plot_results(self, model: object):
326            """Plot results of Model test.
327
328               Args:
329                   model (object): the Model object thats been tested.
330
331            """
332            self.model_status_label.configure(text="Testing Results:", fg='green')
333            results = (
334                f"Prediction Accuracy: " +
335                f"{round(number=model.test_prediction_accuracy, ndigits=1)}%\n" +
336                f"Network Shape: " +
337                f"{','.join(model.layers_shape)}\n"
338                )
339            for i in range(model.test_inputs.shape[1]):
340                results += f"{model.test_inputs[0][i]},"
341                results += f"{model.test_inputs[1][i]} = "
342                if np.squeeze(model.test_prediction)[i] >= 0.5:
343                    results += "1\n"
344                else:
345                    results += "0\n"
346            self.results_label.configure(text=results)
347            self.results_label.pack()
```
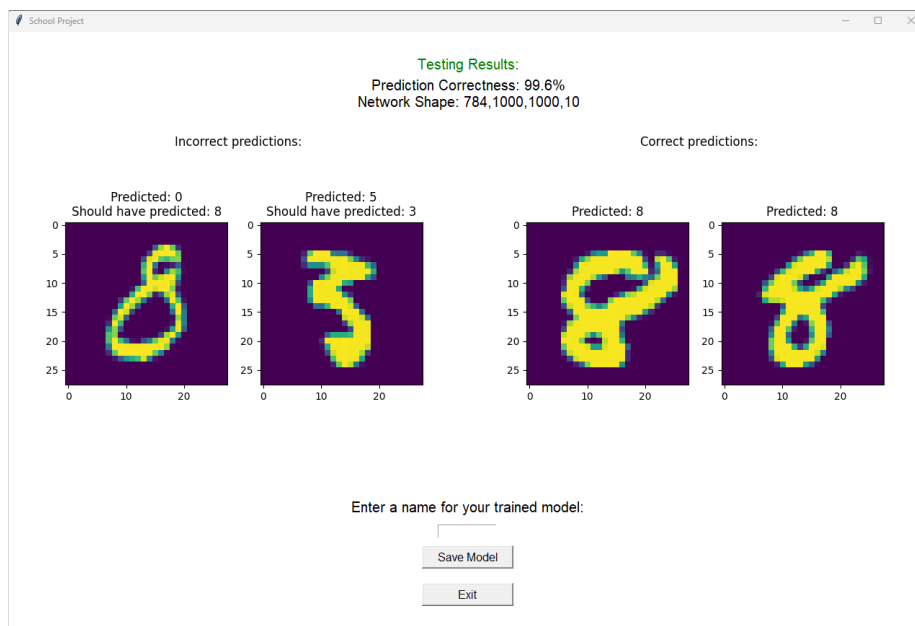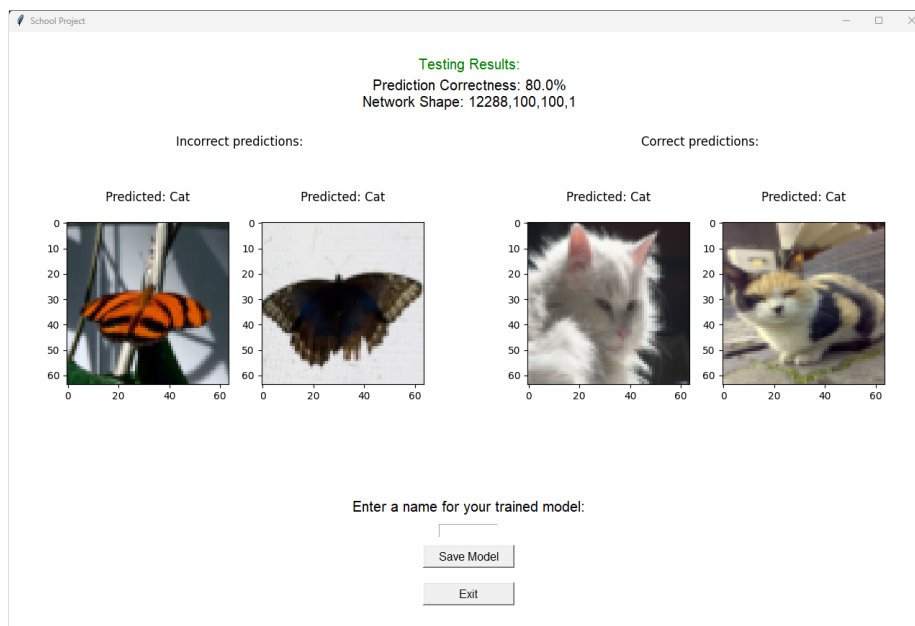
Which outputs the following for the MNIST dataset:

And outputs the following for the Cat Recognition dataset:

And outputs the following for the XOR dataset:
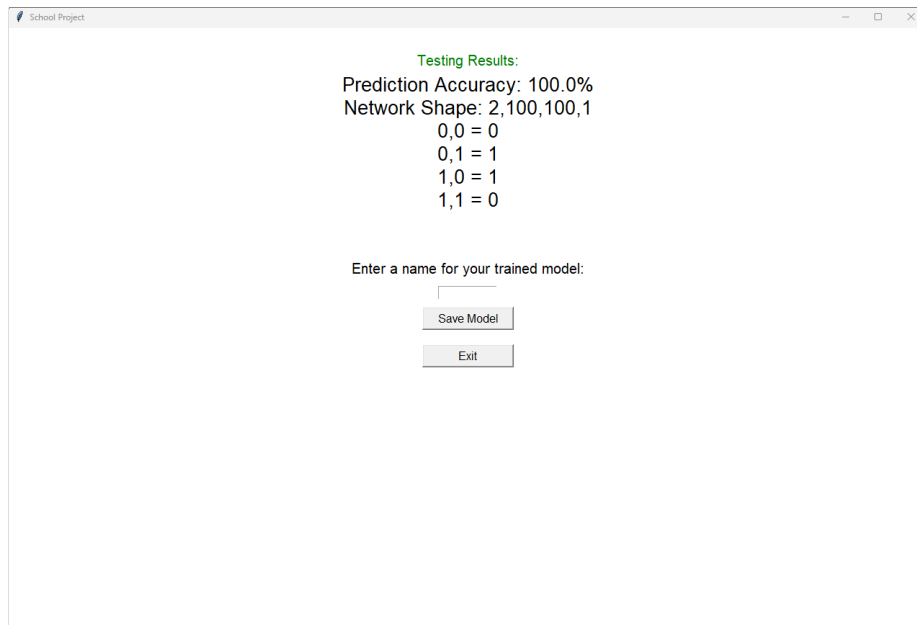
4.1.2 Effects of Hyper-Parameters

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.

```python
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()
```

1
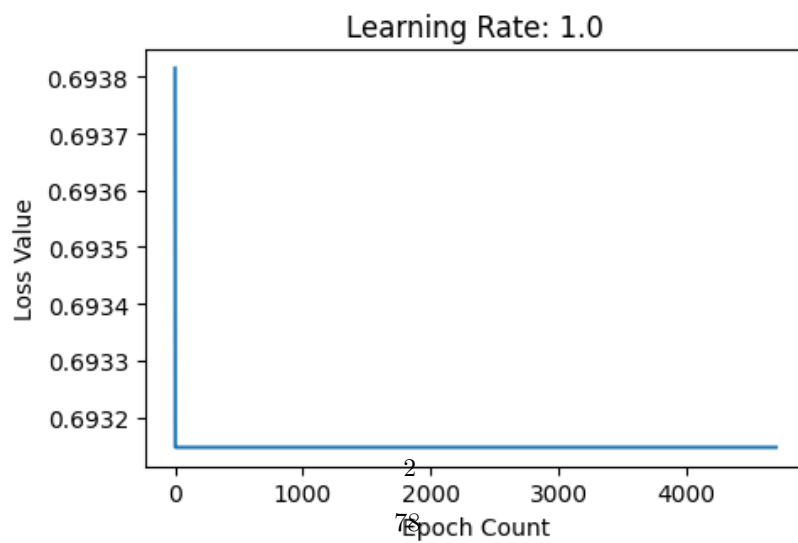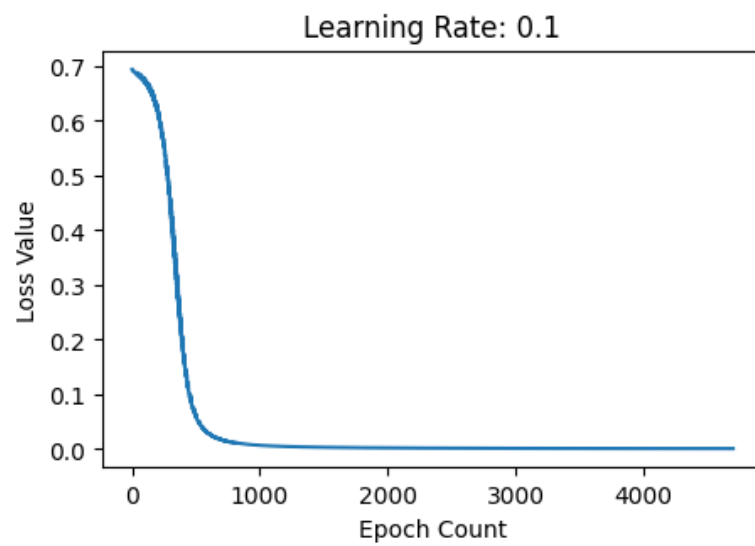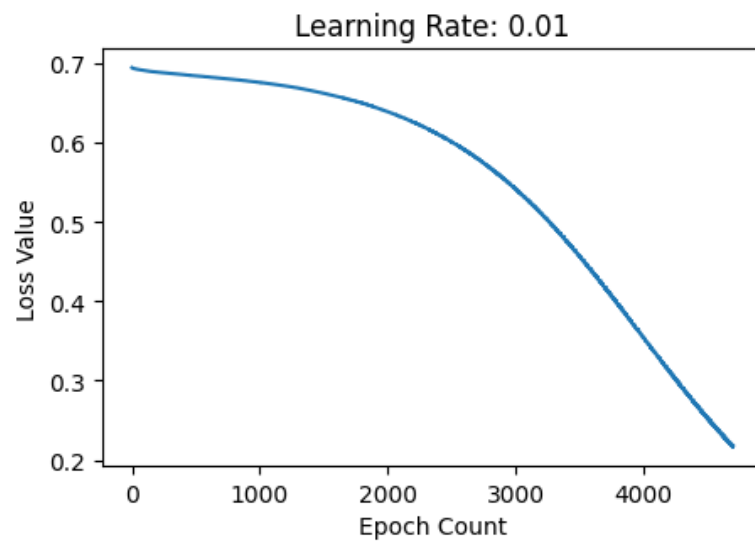
As shown above, if the learning rate is set to too low of a value (0.01 in this case) the model will take more epochs to reduce the loss value, and may even get stuck in unwanted local minimums. If the learning rate is set to an optimal value (0.1 in this case) the model reduces the loss value efficiently 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.

3

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.

```python
[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)}%")
```

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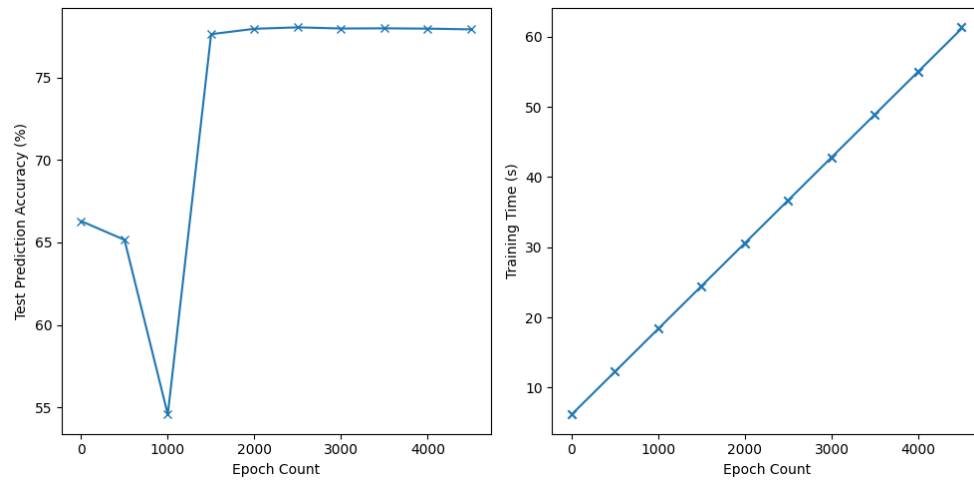
As shown above, as the epoch count increases so does both the test prediction accuracy and the training time taken.

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.

```python
[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()
```

1

```python
    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

2
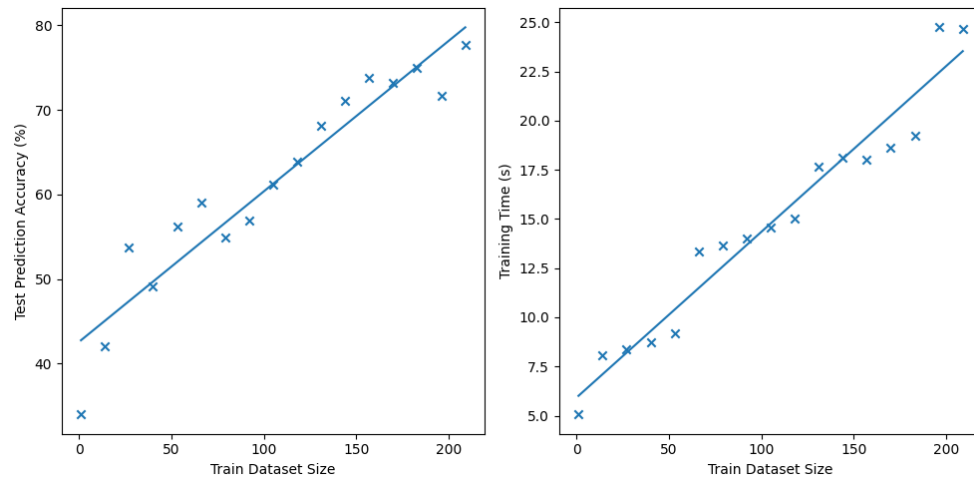
As shown above, as the train dataset size increases do does both the prediction accuracy and the training time taken. Therefore, I can predict that if I increase the size of the Cat Recognition dataset, I could improve the accuracy of the model trained on the dataset.

3

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.

```python
[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()
```

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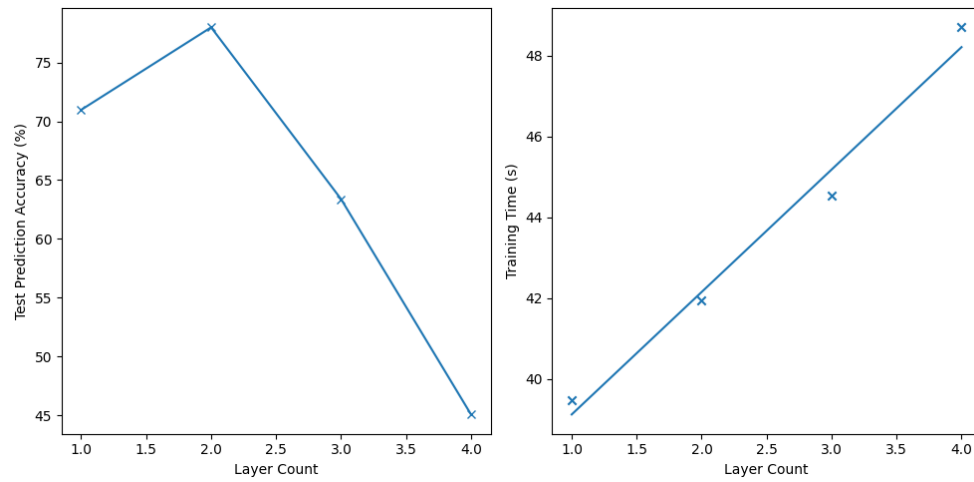
As shown above, as the layer count increases so does the training time taken and the test prediction accuracy at first. However, as the layer count continued to increase the prediction accuracy began to drop greatly (after 2 layers in this case). This is most likely due to the model overfitting and learning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

3

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.

```python
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()
```

1

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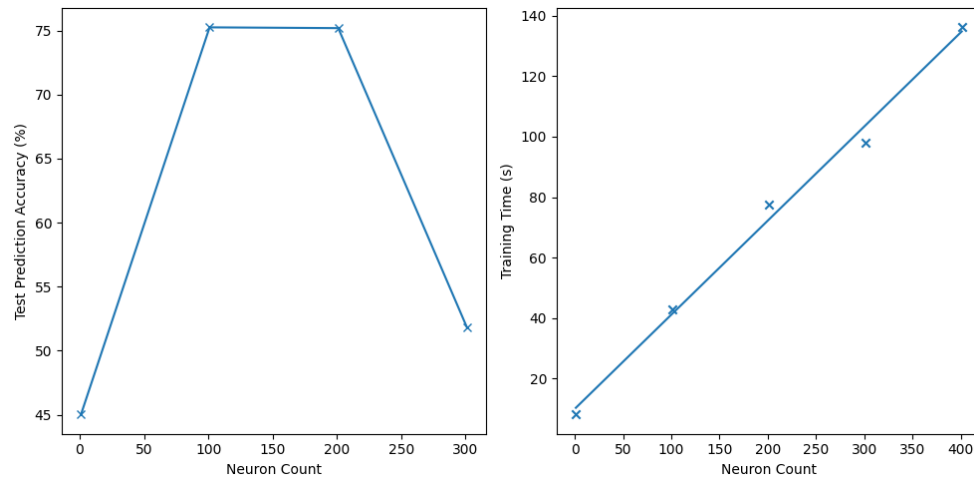
As shown above, as the neuron count of each layer increases so does the training time taken and the test prediction accuracy at first. However, as the neuron count continued to increase the prediction accuracy began to drop greatly (after 200 neurons in this case). This is most likely due to the model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

3

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.

```python
[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()
```
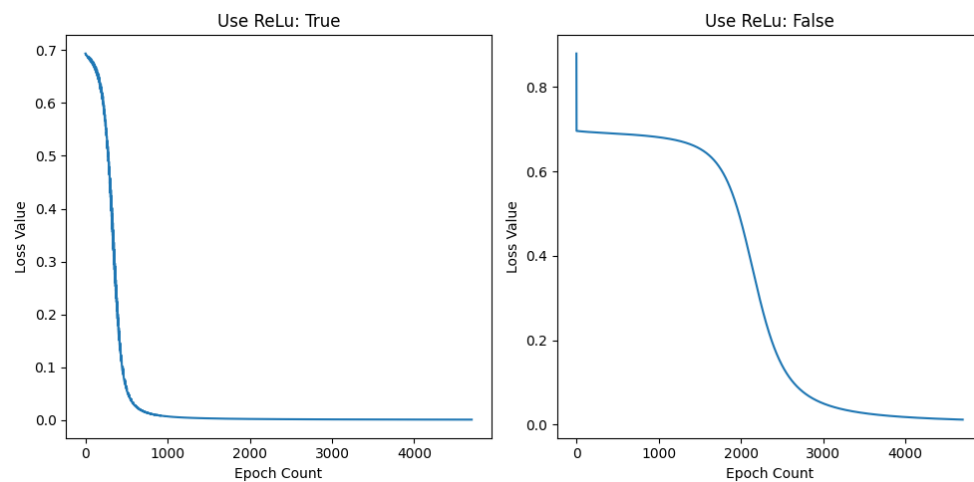

As shown above, when using the ReLu transfer function along with the Sigmoid transfer function, the loss value decreases at a much faster rate than without. The model without the ReLu transfer function does reach the same accuracy but takes far more training epochs to do so.

2

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.

```python
[2]: 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}")

     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}")
```

```
CPU Training Time: 160.45
GPU Training Time: 42.58
```

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

1

4.2 Manual Testing

4.2.1 Input Validation Testing TODO

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:

```python
"""Unit tests for model module."""

import unittest

# Test XOR implementation of Model for its lesser computation time
from school_project.models.cpu.xor import XORModel

class TestModel(unittest.TestCase):
    """Unit tests for model module."""
    def __init__(self, *args, **kwargs) -> None:
        """Initialise unit tests and inputs."""
        super(TestModel, self).__init__(*args, **kwargs)

    def test_train_dataset_size(self) -> None:
        """Test the size of training dataset to be value chosen."""
        train_dataset_size = 4
        model = XORModel(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=1)
        self.assertEqual(first=model.layers.head.input.shape[1],
                         second=train_dataset_size)

    def test_network_shape(self) -> None:
        """Test the neuron count of each layer to match the set shape of
        the
            network."""
        layers_shape = [2, 100, 100, 1]
        model = XORModel(hidden_layers_shape = [100, 100],
                         train_dataset_size = 4,
                         learning_rate = 0.1,
                         use_relu = True)
        model.create_model_values()
        model.train(epoch_count=1)
        for count, layer in enumerate(model.layers):
            self.assertEqual(first=layer.input_neuron_count,
                             second=layers_shape[count])

    def test_learning_rates(self) -> None:
        """Test learning rate of each layer to be the same."""
        learning_rate = 0.1
        model = XORModel(hidden_layers_shape = [100, 100],
                         train_dataset_size = 4,
                         learning_rate = learning_rate,
```

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

```python
106        def test_layer_output_shapes(self) -> None:
107            """Test the shape of each layer's activation function's output."""
108            model = XORModel(hidden_layers_shape = [100, 100],
109                             train_dataset_size = 4,
110                             learning_rate = 0.1,
111                             use_relu = True)
112            model.create_model_values()
113            model.train(epoch_count=1)
114            for layer in model.layers:
115                self.assertEqual(
116                        first=(layer.weights.shape[0],
117                        ↪  layer.input.shape[1]),
117                        second=layer.output.shape
118                        )
119
120 if __name__ == '__main__':
121     unittest.main()
```

- test_tools.py module:

```python
"""Unit tests for tools module."""

import unittest

from school_project.models.cpu.utils import tools

class TestTools(unittest.TestCase):
    """Unit tests for the tools module."""
    def __init__(self, *args, **kwargs) -> None:
        """Initialise unit tests."""
        super(TestTools, self).__init__(*args, **kwargs)

    def test_relu(self) -> None:
        """Test ReLu output range to be >=0."""
        test_inputs = [-100, 0, 100]
        for test_input in test_inputs:
            output = tools.relu(z=test_input)
            self.assertGreaterEqual(a=output, b=0)

    def test_sigmoid(self) -> None:
        """Test sigmoid output range to be within 0-1."""
        test_inputs = [-100, 0, 100]
        for test_input in test_inputs:
            output = tools.sigmoid(z=test_input)
            self.assertTrue(expr=output >= 0 and output <= 1)

if __name__ == '__main__':
    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.

```yaml
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
```