

# 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

<b>1</b>	<b>Analysis</b>	<b>2</b>
1.1	About . . . . .	2
1.2	Interview . . . . .	2
1.3	Project Objectives . . . . .	3
1.4	Theory behind Artificial Neural Networks . . . . .	4
1.4.1	Structure . . . . .	4
1.4.2	How Artificial Neural Networks learn . . . . .	5
1.5	Theory Behind Deep Artificial Neural Networks . . . . .	6
1.5.1	Setup . . . . .	6
1.5.2	Forward Propagation: . . . . .	8
1.5.3	Back Propagation: . . . . .	8
1.6	Theory behind training the Artificial Neural Networks . . . . .	9
1.6.1	Datasets . . . . .	9
1.6.2	Theory behind using Graphics Cards to train Artificial Neural Networks . . . . .	10
<b>2</b>	<b>Design</b>	<b>11</b>
2.1	Introduction . . . . .	11
2.2	System Architecture . . . . .	11
2.3	Class Diagrams . . . . .	12
2.3.1	UI Class Diagram . . . . .	12
2.3.2	Model Class Diagram . . . . .	12
2.4	System Flow chart . . . . .	13
2.5	Algorithms . . . . .	13
2.6	Data Structures . . . . .	14
2.7	File Structure . . . . .	14
2.8	Database Design . . . . .	15
2.9	Queries . . . . .	15
2.10	Human-Computer Interaction TODO . . . . .	16
2.11	Hardware Design . . . . .	16
2.12	Workflow and source control . . . . .	16

<b>3</b>	<b>Technical Solution TODO</b>	<b>16</b>
3.1	Setup . . . . .	16
3.1.1	File Structure . . . . .	16
3.1.2	Dependencies . . . . .	18
3.1.3	Git and Github files . . . . .	18
3.1.4	Organisation . . . . .	23
3.2	models package . . . . .	23
3.2.1	utils subpackage . . . . .	24
3.2.2	Artificial Neural Network implementations . . . . .	34
3.3	frames package . . . . .	38
3.4	__main__.py module . . . . .	47

# 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 particularly like the platform 'Medium' which offers many scientific articles as well as more obvious resources such as IBMs'."
- Q:"What do you think would be a good goal for my project?"  
A:"I think it would be great to aim for applying the Neural Networks on Image Recognition for some famous datasets. For you, I would recommend the MNIST dataset as a goal."

- Q: "What features of the Artificial Neural Networks would you like to be able to experiment with?"  
A: "I'd like to be able to experiment with the number of layers and the number of neurons in each layer, and then be able to see how these changes effect the performance of the model. I can see that you've utilised the Sigmoid transfer function and I would recommend having the option to test alternatives such as the ReLu transfer function, which will help stop issues such as a vanishing gradient."
- Q: "What are some practical constraints of AI?"  
A: "Training AI models can require a large amount of computing power, also large datasets are needed for training models to a high accuracy which can be hard to obtain."
- Q: "What would you say increases the computing power required the most?"  
A: "The number of layers and neurons in each layer will have the greatest effect on the computing power required. This is another reason why I recommend adding the ReLu transfer function as it updates the values of the weights and biases faster than the Sigmoid transfer function."
- Q: "Do you think I should explore other computer architectures for training the models?"  
A: "Yes, it would be great to add support for using graphics cards for training models, as this would be a vast improvement in training time compared to using just CPU power."
- Q: "I am also creating a user interface for the program, what hyper-parameters would you like to be able to control through this?"  
A: "It would be nice to control the transfer functions used, as well as the general hyper-parameters of the model. I also think you could add a progress tracker to be displayed during training for the user."
- Q: "How do you think I should measure the performance of models?"  
A: "You should show the accuracy of the model's predictions, as well as example incorrect and correct prediction results for the trained model. Additionally, you could compare how the size of the training dataset effects the performance of the model after training, to see if a larger dataset would seem beneficial."
- Q: "Are there any other features you would like add?"  
A: "Yes, it would be nice to be able to save a model after training and have the option to load in a trained model for testing."

### 1.3 Project Objectives

- 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

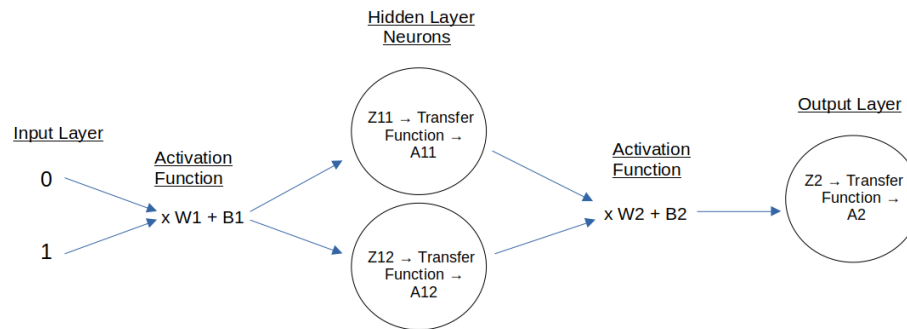


Figure 1: This shows an Artificial Neural Network with one single hidden layer and is known as a Shallow Neural Network.

I have focused on Feed-Forward Artificial Neural Networks, where values are entered to the input layer and passed forwards repetitively to the next layer until reaching the output layer. Within this, I have 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 matrix of the layer's neuron's values. In order to use matrices for this, the following basic theory must be known about them:

- When Adding two matrices, both matrices must have the same number of rows and columns. Or one of the matrices can have the same number of rows but only one column, then be added by element-wise addition where each element is added to all of the elements of the other matrix in the same row.
- When multiplying 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.
- In order to multiply matrices, I take the 'dot product' of the matrices, which multiplies the row of one matrix with the column of the other, by multiplying matching members and then summing up.
- 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  $u = g(x)$ ,  $\frac{\partial y}{\partial u} * \frac{\partial u}{\partial x} =$
      - For a matrix of  $f(x)$  values, the matrix of  $\frac{\partial f(x)}{\partial x}$  values is known as the Jacobian matrix
    - \* This repetitive process will continue to reduce the Loss to a minimum, if the learning rate is set to an appropriate value
    - \* However, during backpropagation some issues can occur, such as the following:
      - Finding a false local minimum rather than the global minimum of the function
      - Having an 'Exploding Gradient', where the gradient value grows exponentially to the point of overflow errors
      - Having a 'Vanishing Gradient', where the gradient value decreases to a very small value or zero, resulting in a lack of updating values during training.

## 1.5 Theory Behind Deep Artificial Neural Networks

### 1.5.1 Setup

- Where a layer takes the previous layer's output as its input X



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

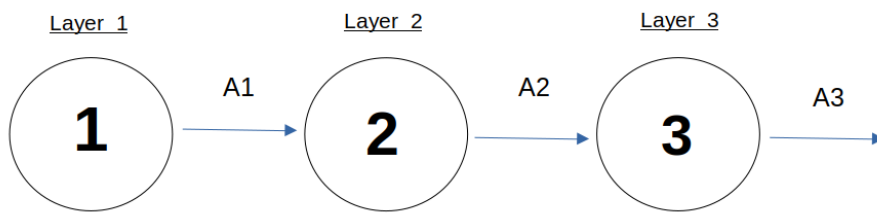


Figure 3: This shows an abstracted view of an Artificial Neural Network with multiple hidden layers and is known as a Deep Neural Network.

- Then it applies an Activation function to  $X$  to obtain  $Z$ , by taking the dot product of  $X$  with a weight matrix  $W$ , then sums the result with a bias matrix  $B$ . At first the weights are initialised to random values and the biases are set to zeros.

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

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

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

- However, for the input layer and the hidden layers, another transfer function known as ReLu (Rectified Linear Unit) can be better suited as it produces larger values of  $\frac{\partial L}{\partial W}$  and  $\frac{\partial L}{\partial B}$  for Gradient Descent than Sigmoid, so updates at a quicker rate.
- \* 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.  $Z = W * X + B$
2.  $A_{relu} = \max(0, Z)$
3.  $A_{sigmoid} = \frac{1}{1 + e^{-Z}}$
4.  $L = -(\frac{1}{m}) * \sum(Y * \log(A) + (1 - Y) * \log(1 - A))$

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

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

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

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

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

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

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

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

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



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

## 1.6 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 famous dataset of images of handwritten digits from zero to ten and is commonly used to test the performance of an Artificial Neural Network.
  - The dataset consists of 60,000 input images, made up from 28x28 pixels and each pixel has an RGB value from 0 to 255
  - To format the images into a suitable format to be inputted into the Artificial Neural Networks, each image's matrix of RGB values are 'flattened' into a 1 dimensional matrix of values, where each element is also divided by 255 (the max RGB value) to a number between 0 and 1, to standardize the dataset.
  - The output dataset is also loaded, where each output for each image is an array, where the index represents the number of the image, by having a 1 in the index that matches the number represented and zeros for all other indexes.

- To create a trained Artificial Neural Network model on this dataset, the model will require 10 output neurons (one for each digit), then by using the Sigmoid Transfer function to output a number between one and zero to each neuron, whichever neuron has the highest value is predicted. This is multi-class classification, where the model must predict one of 10 classes (in this case, each class is one of the digits from zero to ten).
- Cat dataset
  - I will also use a dataset of images sourced from <https://github.com/marcopeix>, where each image is either a cat or not a cat.
  - The dataset consists of 209 input images, made up from 64x64 pixels and each pixel has an RGB value from 0 to 255
  - To format the images into a suitable format to be inputted into the Artificial Neural Networks, each image's matrix of RGB values are 'flattened' into a 1 dimensional array of values, where each element is also divided by 255 (the max RGB value) to a number between 0 and 1, to standardize the dataset.
  - The output dataset is also loaded, and is reshaped into a 1 dimensional array of 1s and 0s, to store the output of each image (1 for cat, 0 for non cat)
  - To create a trained Artificial Neural Network model on this dataset, the model will require only 1 output neuron, then by using the Sigmoid Transfer function to output a number between one and zero for the neuron, if the neuron's value is closer to 1 it predicts cat, otherwise it predicts not a cat. This is binary classification, where the model must use logistic regression to predict whether it is a cat or not a cat.
- XOR dataset
  - For experimenting with Artificial Neural Networks, I solve the XOR gate problem, where the Neural Network is fed input pairs of zeros and ones and learns to predict the output of a XOR gate used in circuits.
  - This takes much less computation time than image datasets, so is usefull for quickly comparing different hyper-parameters of a Network.

### 1.6.2 Theory behind using Graphics Cards to train Artificial Neural Networks

Graphics Cards consist of many Tensor cores which are processing units specialised for matrix operations for calculating the co-ordinates of 3D graphics, however they can be used here for operating on the matrices in the network at a much faster speed compared to CPUs. GPUs also include CUDA cores which act as an API to the GPU's computing to be used for any operations (in this case training the Artificial Neural Networks).

## 2 Design

### 2.1 Introduction

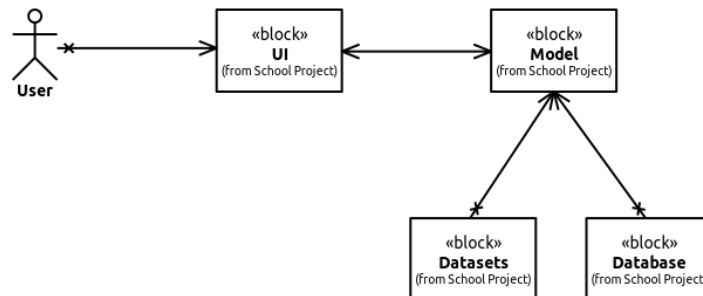
The following design focuses have been made for the project:

- The program will support multiple platforms to run on, including Windows and Linux.
- The program will use python3 as its main programming language.
- I will take an object-orientated approach to the project.
- I will give an option to use either a Graphics Card or a CPU to train and test the Artificial Neural Networks.

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

### 2.2 System Architecture

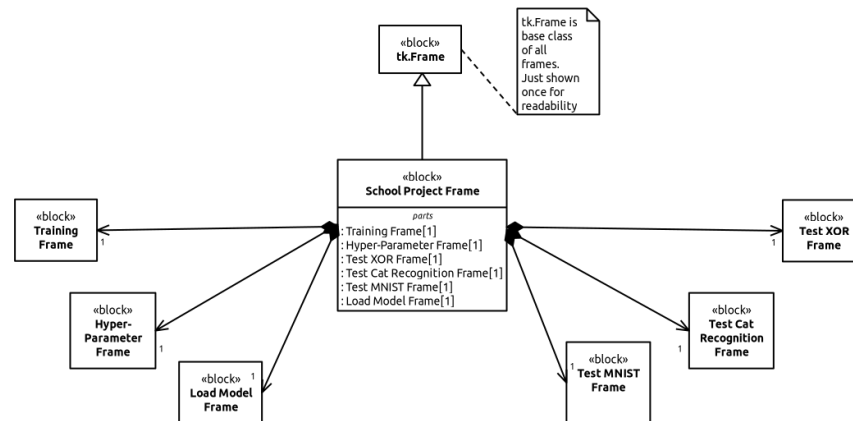
bdd [block] School Project Frame [System Architecture Diagram]



## 2.3 Class Diagrams

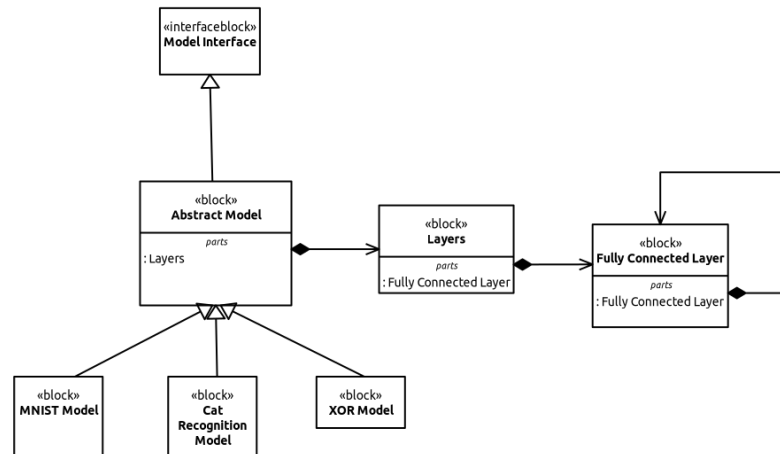
### 2.3.1 UI Class Diagram

bdd [package] School Project [UI Class Diagram]



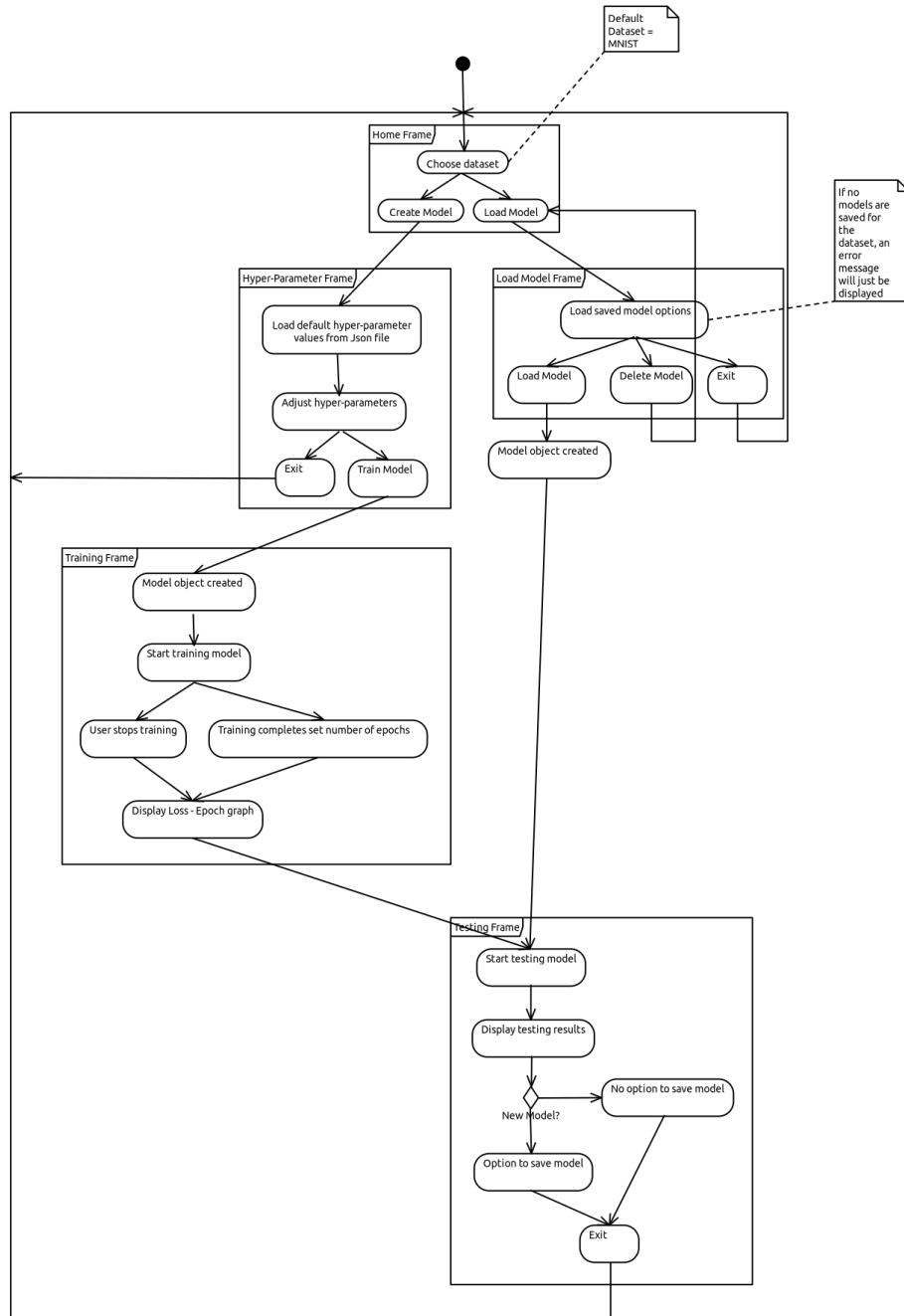
### 2.3.2 Model Class Diagram

bdd [package] School Project [Model Class Diagram]



## 2.4 System Flow chart

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



## 2.5 Algorithms

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

## 2.6 Data Structures

I will use the following data structures in the program:

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

## 2.7 File Structure

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

- A JSON file for storing the default hyper-parameters for creating a new model for each dataset.
- I will store the image dataset files in a 'datasets' directory. The dataset files will either be a compressed archive file (such as .pkl.gz files) or of the Hierarchical Data Format (such as .h5) for storing large datasets with fast retrieval.
- I will save the weights and biases of saved models as numpy arrays in .npz files (a zipped archive file format) in a 'saved-models' directory, due to their compatibility with the numpy library.

## 2.8 Database Design

I will use the following Relational database design for saving models, where the dataset, name and features of the saved model (including the location of the saved models' weights and biases and the saved models' hyper-parameters) are saved:

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

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

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

## 2.9 Queries

Here are some example queries for interacting with the database:

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

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

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

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

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

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

## 2.10 Human-Computer Interaction TODO

- Labeled screenshots of UI

## 2.11 Hardware Design

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

## 2.12 Workflow and source control

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

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

# 3 Technical Solution 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.

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



```

.
|-- .github
|   |-- workflows
|   |-- tests.yml
|-- .gitignore
|-- LICENSE
|-- 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

```

17 directories, 41 files

### 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     ```
39
40 4. Enter the project directory with:
41     ```
42     cd school-project/
43     ```
44
45 5. For normal use, install the dependencies and the project to the
  ↳ venv with:
46 - Windows:
47     ```
48     python setup.py install
49     ```
50 - Linux:
51     ```
52     python3 setup.py install
53     ```
54
55 *Note: In order to use an Nvidia GPU for training the networks, the
  ↳ latest Nvidia drivers must be installed and the CUDA Toolkit must
  ↳ be installed from
56 <a href="https://developer.nvidia.com/cuda-downloads">here</a>.*
57
58 ## Usage

```

```

59
60 Run with:
61 - Windows:
62     ```
63     python school_project
64     ```
65 - Linux:
66     ```
67     python3 school_project
68     ```
69
70 ## Development
71
72 Install the dependencies and the project to the venv in developing
73 ↪ mode with:
74 - Windows:
75     ```
76     python setup.py develop
77     ```
78 - Linux:
79     ```
80     python3 setup.py develop
81     ```
82
83 Run Tests with:
84 - Windows:
85     ```
86     python -m unittest discover .\school_project\test\
87     ```
88 - Linux:
89     ```
90     python3 -m unittest discover ./school_project/test/
91     ```
92
93 Compile Project Report PDF with:
94     ```
95     make all
96     ```
97
98 *Note: This requires the Latexmk library*

```

---

- Which will generate the following:

Tests passing Python 3

## A-level Computer Science NEA Programming Project

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

### Installation

1. Download the Repository with:

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



- Or by downloading as a ZIP file

2. Create a virtual environment (venv) with:

- Windows:

```
python -m venv {venv name}
```



- Linux:

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



3. Enter the venv with:

◦ Windows:

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



◦ Linux:

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



4. Enter the project directory with:

```
cd school-project/
```



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

◦ Windows:

```
python setup.py install
```



◦ Linux:

```
python3 setup.py install
```



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

## Usage

Run with:

• Windows:

```
python school_project
```



- Linux:

```
python3 school_project
```



## Development

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

- Windows:

```
python setup.py develop
```



- Linux:

```
python3 setup.py develop
```



Run Tests with:

- Windows:

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



- Linux:

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



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:

---

```

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

```



```

43         Returns:
44             tuple of train_inputs, train_outputs,
45             test_inputs and test_outputs.
46         Raises:
47             NotImplementedError: if this method is not implemented.
48
49         """
50         raise NotImplementedError
51
52     @abstractmethod
53     def back_propagation(self, prediction: np.ndarray) -> None:
54         """Adjust the weights and bias/biases via gradient descent.
55
56         Args:
57             prediction (numpy.ndarray): the matrice of prediction values
58         Raises:
59             NotImplementedError: if this method is not implemented.
60
61         """
62         raise NotImplementedError
63
64     @abstractmethod
65     def forward_propagation(self) -> np.ndarray:
66         """Generate a prediction with the weights and bias/biases.
67
68         Returns:
69             numpy.ndarray of prediction values.
70         Raises:
71             NotImplementedError: if this method is not implemented.
72
73         """
74         raise NotImplementedError
75
76     @abstractmethod
77     def test(self) -> None:
78         """Test trained weights and bias/biases.
79
80         Raises:
81             NotImplementedError: if this method is not implemented.
82
83         """
84         raise NotImplementedError
85
86     @abstractmethod
87     def train(self, epochs: int) -> None:
88         """Train weights and bias/biases.
89
90         Args:
91             epochs (int): the number of forward and back propagations to
↪ do.
92         Raises:
93             NotImplementedError: if this method is not implemented.
94
95         """
96         raise NotImplementedError
97
98     @abstractmethod
99     def save_model_values(self, file_location: str) -> None:
100 ↪ """Save the model by saving the weights then biases of each layer to
101         a .npz file with a given file location.
102

```

```

103         Args:
104             file_location (str): the file location to save the model to.
105
106         """
107         raise NotImplementedError
108
109     def relu(z: np.ndarray | int | float) -> np.ndarray | float:
110         """Transfer function, transform input to max number between 0 and z.
111
112         Args:
113             z (numpy.ndarray | int | float):
114                 the numpy.ndarray | int | float to be transferred.
115         Returns:
116             numpy.ndarray | float,
117                 with all values | the value transferred to max number between 0-z.
118         Raises:
119             TypeError: if z is not of type numpy.ndarray | int | float.
120
121         """
122         return np.maximum(0.1*z, 0) # Divide by 10 to stop overflow errors
123
124     def relu_derivative(output: np.ndarray | int | float) -> np.ndarray |
125     ↪ float:
126         """Calculate derivative of ReLu Transfer function with respect to z.
127
128         Args:
129             output (numpy.ndarray | int | float):
130                 the numpy.ndarray | int | float output of the ReLu transfer
131                 ↪ function.
132         Returns:
133             numpy.ndarray | float,
134                 derivative of the ReLu transfer function with respect to z.
135         Raises:
136             TypeError: if output is not of type numpy.ndarray | int | float.
137
138         """
139         output[output <= 0] = 0
140         output[output > 0] = 1
141
142         return output
143
144     def sigmoid(z: np.ndarray | int | float) -> np.ndarray | float:
145         """Transfer function, transform input to number between 0 and 1.
146
147         Args:
148             z (numpy.ndarray | int | float):
149                 the numpy.ndarray | int | float to be transferred.
150         Returns:
151             numpy.ndarray | float,
152                 with all values | the value transferred to a number between 0-1.
153         Raises:
154             TypeError: if z is not of type numpy.ndarray | int | float.
155
156         """
157         return 1 / (1 + np.exp(-z))
158
159     def sigmoid_derivative(output: np.ndarray | int | float) -> np.ndarray |
160     ↪ float:
161         """Calculate derivative of sigmoid Transfer function with respect to z.
162
163         Args:
164             output (numpy.ndarray | int | float):

```

```

162         the numpy.ndarray / int / float output of the sigmoid transfer
↪ function.
163     Returns:
164         numpy.ndarray / float,
165         derivative of the sigmoid transfer function with respect to z.
166     Raises:
167         TypeError: if output is not of type numpy.ndarray / int / float.
168
169     """
170     return output * (1 - output)
171
172 def calculate_loss(input_count: int,
173                   outputs: np.ndarray,
174                   prediction: np.ndarray) -> float:
175     """Calculate average loss/error of the prediction to the outputs.
176
177     Args:
178         input_count (int): the number of inputs.
179         outputs (np.ndarray):
180             the train/test outputs array to compare with the prediction.
181         prediction (np.ndarray): the array of prediction values.
182     Returns:
183         float loss.
184     Raises:
185         ValueError:
186             if outputs is not a suitable multiplier with the prediction
187             (incorrect shapes)
188
189     """
190     ↪ return np.squeeze(- (1/input_count) * np.sum(outputs *
191     ↪ np.log(prediction) + (1 - outputs) * np.log(1 - prediction)))
192
193 def calculate_prediction_accuracy(prediction: np.ndarray,
194                                  outputs: np.ndarray) -> float:
195     """Calculate the percentage accuracy of the predictions.
196
197     Args:
198         prediction (np.ndarray): the array of prediction values.
199         outputs (np.ndarray):
200             the train/test outputs array to compare with the prediction.
201     Returns:
202         float prediction accuracy
203
204     """
205     return 100 - np.mean(np.abs(prediction - outputs)) * 100

```

---

- model.py module:

---

```

1 import time
2
3 import numpy as np
4
5 from school_project.models.cpu.utils.tools import (
6     ModelInterface,
7     relu,
8     relu_derivative,
9     sigmoid,
10    sigmoid_derivative,
11    calculate_loss,
12    calculate_prediction_accuracy
13 )

```

```

14
15 class _Layers():
16     """Manages linked list of layers."""
17     def __init__(self):
18         """Initialise linked list."""
19         self.head = None
20         self.tail = None
21
22     def __iter__(self):
23         """Iterate forward through the network."""
24         current_layer = self.head
25         while True:
26             yield current_layer
27             if current_layer.next_layer != None:
28                 current_layer = current_layer.next_layer
29             else:
30                 break
31
32     def __reversed__(self):
33         """Iterate back through the network."""
34         current_layer = self.tail
35         while True:
36             yield current_layer
37             if current_layer.previous_layer != None:
38                 current_layer = current_layer.previous_layer
39             else:
40                 break
41
42 class _FullyConnectedLayer():
43     """Fully connected layer for Deep ANNs,
44     represented as a node of a Doubly linked list."""
45     def __init__(self, learning_rate: float, input_neuron_count: int,
46                 output_neuron_count: int, transfer_type: str) -> None:
47         """Initialise layer values.
48
49         Args:
50             learning_rate (float): the learning rate of the model.
51             input_neuron_count (int):
52                 the number of input neurons into the layer.
53             output_neuron_count (int):
54                 the number of output neurons into the layer.
55             transfer_type (str): the transfer function type
56                 ('sigmoid' or 'relu')
57
58         """
59         # Setup layer attributes
60         self.previous_layer = None
61         self.next_layer = None
62         self.input_neuron_count = input_neuron_count
63         self.output_neuron_count = output_neuron_count
64         self.transfer_type = transfer_type
65         self.input: np.ndarray
66         self.output: np.ndarray
67
68         # Setup weights and biases
69         self.weights: np.ndarray
70         self.biases: np.ndarray
71         self.learning_rate = learning_rate
72
73     def __repr__(self) -> str:
74         """Read values of the layer.
75

```

```

76         Returns:
77             a string description of the layers's
78             weights, bias and learning rate values.
79
80         """
81         return (f"Weights: {self.weights.tolist()}\n" +
82                 f"Biases: {self.biases.tolist()}\n")
83
84     def init_layer_values_random(self) -> None:
85         """Initialise weights to random values and biases to 0s"""
86         np.random.seed(1) # Sets up pseudo random values for layer weight
87                             ↪ arrays
88         self.weights = np.random.rand(self.output_neuron_count,
89                                       ↪ 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,
95                                       ↪ self.input_neuron_count))
96         self.biases = np.zeros(shape=(self.output_neuron_count, 1))
97
98     def back_propagation(self, dloss_doutput) -> np.ndarray:
99         """Adjust the weights and biases via gradient descent.
100
101         Args:
102             dloss_doutput (numpy.ndarray): the derivative of the loss of the
103
104             ↪ layer's output, with respect to the layer's output.
105         Returns:
106             a numpy.ndarray derivative of the loss of the layer's input,
107             with respect to the layer's input.
108         Raises:
109             ValueError:
110                 if dloss_doutput
111                 is not a suitable multiplier with the weights
112                 (incorrect shape)
113
114         """
115         match self.transfer_type:
116             case 'sigmoid':
117                 dloss_dz = dloss_doutput *
118                             ↪ sigmoid_derivative(output=self.output)
119             case 'relu':
120                 dloss_dz = dloss_doutput *
121                             ↪ relu_derivative(output=self.output)
122
123         dloss_dweights = np.dot(dloss_dz, self.input.T)
124         dloss_dbias = np.sum(dloss_dz)
125
126         assert dloss_dweights.shape == self.weights.shape
127
128         dloss_dinput = np.dot(self.weights.T, dloss_dz)
129
130         # Update weights and biases
131         self.weights -= self.learning_rate * dloss_dweights
132         self.biases -= self.learning_rate * dloss_dbias
133
134         return dloss_dinput
135
136     def forward_propagation(self, inputs) -> np.ndarray:
137         """Generate a layer output with the weights and biases.

```

```

132
133     Args:
134         inputs (np.ndarray): the input values to the layer.
135     Returns:
136         a numpy.ndarray of the output values.
137
138     """
139     self.input = inputs
140     z = np.dot(self.weights, self.input) + self.biases
141     if self.transfer_type == 'sigmoid':
142         self.output = sigmoid(z)
143     elif self.transfer_type == 'relu':
144         self.output = relu(z)
145     return self.output
146
147 class AbstractModel(ModelInterface):
148     """ANN model with variable number of hidden layers"""
149     def __init__(self,
150                 hidden_layers_shape: list[int],
151                 train_dataset_size: int,
152                 learning_rate: float,
153                 use_relu: bool) -> None:
154         """Initialise model values.
155
156         Args:
157             hidden_layers_shape (list[int]):
158                 list of the number of neurons in each hidden layer.
159             train_dataset_size (int): the number of train dataset inputs to
160     ↪ use.
161             learning_rate (float): the learning rate of the model.
162             use_relu (bool): True or False whether the ReLu Transfer
163     ↪ 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
171             ↪ train_dataset_size=train_dataset_size
172             )
173         self.train_losses: list[float]
174         self.test_prediction: np.ndarray
175         self.test_prediction_accuracy: float
176         self.training_progress = ""
177         self.training_time: float
178
179         # Setup model attributes
180         self._running = True
181         self.input_neuron_count: int = self.train_inputs.shape[0]
182         self.input_count = self.train_inputs.shape[1]
183         self.hidden_layers_shape = hidden_layers_shape
184         self.output_neuron_count = self.train_outputs.shape[0]
185         self.layers_shape = [f'{layer}' for layer in (
186             [self.input_neuron_count] +
187             self.hidden_layers_shape +
188             [self.output_neuron_count]
189         )]
190         self.use_relu = use_relu
191
192         # Setup model values
193         self.layers = _Layers()

```

```

191         self.learning_rate = learning_rate
192
193     def __repr__(self) -> str:
194         """Read current state of model.
195
196         Returns:
197             a string description of the model's shape,
198             weights, bias and learning rate values.
199
200         """
201         return (f"Layers Shape: {'.'.join(self.layers_shape)}\n" +
202                 f"Learning Rate: {self.learning_rate}")
203
204     def set_running(self, value:bool):
205         self._running = value
206
207     def _setup_layers(setup_values: callable) -> None:
208         """Setup model layers"""
209         def decorator(self, *args, **kwargs):
210             # Check if setting up Deep Network
211             if len(self.hidden_layers_shape) > 0:
212                 if self.use_relu:
213
214                     # Add input layer
215                     self.layers.head = _FullyConnectedLayer(
216
217                                     ↪ learning_rate=self.learning_rate,
218
219                                     ↪ input_neuron_count=self.input_neuron_count,
220
221                                     ↪ output_neuron_count=self.hidden_layers_shape[0],
222                                     transfer_type='relu'
223                                 )
224                     current_layer = self.layers.head
225
226                     # Add hidden layers
227                     for layer in range(len(self.hidden_layers_shape) - 1):
228                         current_layer.next_layer = _FullyConnectedLayer(
229                             learning_rate=self.learning_rate,
230
231                             ↪ input_neuron_count=self.hidden_layers_shape[layer],
232
233                             ↪ output_neuron_count=self.hidden_layers_shape[layer
234                             ↪ + 1],
235                             transfer_type='relu'
236                         )
237                         current_layer.next_layer.previous_layer =
238                             ↪ current_layer
239                         current_layer = current_layer.next_layer
240
241             else:
242
243                 # Add input layer
244                 self.layers.head = _FullyConnectedLayer(
245
246                                     ↪ learning_rate=self.learning_rate,
247
248                                     ↪ input_neuron_count=self.input_neuron_count,
249
250                                     ↪ output_neuron_count=self.hidden_layers_shape[0],
251                                     transfer_type='sigmoid'
252                                 )
253                 current_layer = self.layers.head

```

```

243
244         # Add hidden layers
245         for layer in range(len(self.hidden_layers_shape) - 1):
246             current_layer.next_layer = _FullyConnectedLayer(
247                 learning_rate=self.learning_rate,
248
249                 ↪ input_neuron_count=self.hidden_layers_shape[layer],
250
251                 ↪ output_neuron_count=self.hidden_layers_shape[layer
252                 ↪ + 1],
253                 transfer_type='sigmoid'
254             )
255             current_layer.next_layer.previous_layer =
256             ↪ current_layer
257             current_layer = current_layer.next_layer
258
259         # Add output layer
260         current_layer.next_layer = _FullyConnectedLayer(
261             learning_rate=self.learning_rate,
262
263             ↪ input_neuron_count=self.hidden_layers_shape[-1],
264
265             ↪ output_neuron_count=self.output_neuron_count,
266             transfer_type='sigmoid'
267         )
268         current_layer.next_layer.previous_layer = current_layer
269         self.layers.tail = current_layer.next_layer
270
271         # Setup Perceptron Network
272         else:
273             self.layers.head = _FullyConnectedLayer(
274                 learning_rate=self.learning_rate,
275
276                 ↪ input_neuron_count=self.input_neuron_count,
277
278                 ↪ output_neuron_count=self.output_neuron_count,
279                 transfer_type='sigmoid'
280             )
281             self.layers.tail = self.layers.head
282
283         setup_values(self, *args, **kwargs)
284
285         return decorator
286
287     @setup_layers
288     def create_model_values(self) -> None:
289         """Create weights and bias/biases"""
290         # Check if setting up Deep Network
291         if len(self.hidden_layers_shape) > 0:
292
293             # Initialise Layer values to random values
294             for layer in self.layers:
295                 layer.init_layer_values_random()
296
297         # Setup Perceptron Network
298         else:
299
300             # Initialise Layer values to zeros
301             for layer in self.layers:
302                 layer.init_layer_values_zeros()
303
304     @setup_layers

```



```

297 def load_model_values(self, file_location: str) -> None:
298     """Load weights and bias/biases from .npz file.
299
300     Args:
301         file_location (str): the location of the file to load from.
302
303     """
304     data: dict[str, np.ndarray] = np.load(file=file_location)
305
306     # Initialise Layer values
307     i = 0
308     keys = list(data.keys())
309     for layer in self.layers:
310         layer.weights = data[keys[i]]
311         layer.biases = data[keys[i + 1]]
312         i += 2
313
314 def back_propagation(self, dloss_doutput) -> None:
315     """Train each layer's weights and biases.
316
317     Args:
318         dloss_doutput (np.ndarray): the derivative of the loss of the
319         output layer's output, with respect to the output layer's
320     ↪ output.
321
322     """
323     for layer in reversed(self.layers):
324         dloss_doutput =
325         ↪ layer.back_propagation(dloss_doutput=dloss_doutput)
326
327 def forward_propagation(self) -> np.ndarray:
328     """Generate a prediction with the layers.
329
330     Returns:
331         a numpy.ndarray of the prediction values.
332
333     """
334     output = self.train_inputs
335     for layer in self.layers:
336         output = layer.forward_propagation(inputs=output)
337     return output
338
339 def test(self) -> None:
340     """Test the layers' trained weights and biases."""
341     output = self.test_inputs
342     for layer in self.layers:
343         output = layer.forward_propagation(inputs=output)
344     self.test_prediction = output
345
346     # Calculate performance of model
347     self.test_prediction_accuracy = calculate_prediction_accuracy(
348         ↪ prediction=self.test_prediction,
349         outputs=self.test_outputs
350     )
351
352 def train(self, epoch_count: int) -> None:
353     """Train layers' weights and biases.
354
355     Args:
356         epoch_count (int): the number of training epochs.

```

```

356         """
357         self.layers_shape = [f'{layer}' for layer in (
358             [self.input_neuron_count] +
359             self.hidden_layers_shape +
360             [self.output_neuron_count]
361         )]
362         self.train_losses = []
363         training_start_time = time.time()
364         for epoch in range(epoch_count):
365             if not self.__running:
366                 break
367             self.training_progress = f"Epoch {epoch} / {epoch_count}"
368             prediction = self.forward_propagation()
369             loss = calculate_loss(input_count=self.input_count,
370                                 outputs=self.train_outputs,
371                                 prediction=prediction)
372             self.train_losses.append(loss)
373             if not self.__running:
374                 break
375             dloss_doutput = -(1/self.input_count) * ((self.train_outputs -
376                 ↪ prediction)/(prediction * (1 - prediction)))
377             self.back_propagation(dloss_doutput=dloss_doutput)
378             self.training_time = round(number=time.time() -
379                 ↪ training_start_time,
380                                     ndigits=2)
381
382         def save_model_values(self, file_location: str) -> None:
383             """Save the model by saving the weights then biases of each layer to
384             ↪ a .npz file with a given file location.
385
386             Args:
387                 file_location (str): the file location to save the model to.
388
389             """
390             saved_model: list[np.ndarray] = []
391             for layer in self.layers:
392                 saved_model.append(layer.weights)
393                 saved_model.append(layer.biases)
394             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  import h5py
2  import numpy as np
3
4  from school_project.models.cpu.utils.model import AbstractModel
5
6  class CatRecognitionModel(AbstractModel):
7      """ANN model that trains to predict if an image is a cat or not a
8      ↪ cat."""
9      def __init__(self,
10                  hidden_layers_shape: list[int],
11                  train_dataset_size: int,
12                  learning_rate: float,

```

```

12         use_relu: bool) -> None:
13         """Initialise Model's Base class.
14
15     Args:
16         hidden_layers_shape (list[int]):
17             list of the number of neurons in each hidden layer.
18         train_dataset_size (int): the number of train dataset inputs to
19 ↪ use.
20         learning_rate (float): the learning rate of the model.
21         use_relu (bool): True or False whether the ReLu Transfer
22 ↪ 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,
32 ↪ np.ndarray,
33                               np.ndarray,
34                               ↪ np.ndarray]:
35
36         """Load image input and output datasets.
37
38     Args:
39         train_dataset_size (int): the number of train dataset inputs to
40 ↪ use.
41
42     Returns:
43         tuple of image train_inputs, train_outputs,
44         test_inputs and test_outputs numpy.ndarrays.
45
46     Raises:
47         FileNotFoundError: if file does not exist.
48
49     """
50     # Load datasets from h5 files
51     # (h5 files stores large amount of data with quick access)
52     train_dataset: h5py.File = h5py.File(
53         r'school_project/models/datasets/train-cat.h5',
54         'r'
55     )
56     test_dataset: h5py.File = h5py.File(
57         r'school_project/models/datasets/test-cat.h5',
58         'r'
59     )
60
61     # Load input arrays,
62     # containing the RGB values for each pixel in each 64x64 pixel
63     ↪ image,
64     # for 209 images
65     train_inputs: np.ndarray =
66     ↪ np.array(train_dataset['train_set_x'][:])
67     test_inputs: np.ndarray = np.array(test_dataset['test_set_x'][:])
68
69     # Load output arrays of 1s for cat and 0s for not cat
70     train_outputs: np.ndarray =
71     ↪ np.array(train_dataset['train_set_y'][:])
72     test_outputs: np.ndarray = np.array(test_dataset['test_set_y'][:])
73
74     # Reshape input arrays into 1 dimension (flatten),
75     # then divide by 255 (RGB)

```

```

66     # to standardize them to a number between 0 and 1
67     train_inputs = train_inputs.reshape((train_inputs.shape[0],
68                                         -1)).T / 255
69     test_inputs = test_inputs.reshape((test_inputs.shape[0], -1)).T /
    ↪ 255
70
71     # Reshape output arrays into a 1 dimensional list of outputs
72     train_outputs = train_outputs.reshape((1, train_outputs.shape[0]))
73     test_outputs = test_outputs.reshape((1, test_outputs.shape[0]))
74
75     # Reduce train datasets' sizes to train_dataset_size
76     train_inputs = (train_inputs.T[:train_dataset_size]).T
77     train_outputs = (train_outputs.T[:train_dataset_size]).T
78
79     return train_inputs, train_outputs, test_inputs, test_outputs

```

---

- mnist.py module:

---

```

1  import pickle
2  import gzip
3
4  import numpy as np
5
6  from school_project.models.cpu.utils.model import (
7                                          AbstractModel
8                                          )
9
10 class MNISTModel(AbstractModel):
11     """ANN model that trains to predict Numbers from images."""
12     def __init__(self, hidden_layers_shape: list[int],
13                 train_dataset_size: int,
14                 learning_rate: float,
15                 use_relu: bool) -> None:
16         """Initialise Model's Base class.
17
18         Args:
19             hidden_layers_shape (list[int]):
20                 list of the number of neurons in each hidden layer.
21             train_dataset_size (int): the number of train dataset inputs to
    ↪ 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
35         """Load image input and output datasets.
36         Args:
37             train_dataset_size (int): the number of dataset inputs to use.
38         Returns:
39             tuple of image train_inputs, train_outputs,
40             test_inputs and test_outputs numpy.ndarrays.

```

```

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,
52         ↪ encoding='bytes')
53
54     # Reshape input arrays into 1 dimension (flatten),
55     # then divide by 255 (RGB)
56     # to standardize them to a number between 0 and 1
57     train_inputs =
58     ↪ np.array(train_inputs.reshape((train_inputs.shape[0],
59     ↪ -1)).T / 255)
60     test_inputs = np.array(test_inputs.reshape(test_inputs.shape[0],
61     ↪ -1)).T / 255)
62
63     # Represent number values
64     # with a one at the matching index of an array of zeros
65     train_outputs = np.eye(np.max(train_outputs) + 1)[train_outputs].T
66     test_outputs = np.eye(np.max(test_outputs) + 1)[test_outputs].T
67
68     # Reduce train datasets' sizes to train_dataset_size
69     train_inputs = (train_inputs.T[:train_dataset_size]).T
70     train_outputs = (train_outputs.T[:train_dataset_size]).T
71
72     return train_inputs, train_outputs, test_inputs, test_outputs

```

- xor.py module

---

```

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

```

```

24         super().__init__(hidden_layers_shape=hidden_layers_shape,
25                           train_dataset_size=train_dataset_size,
26                           learning_rate=learning_rate,
27                           use_relu=use_relu)
28
29     def load_datasets(self, train_dataset_size: int) -> tuple[np.ndarray,
30                                                            np.ndarray,
31                                                            np.ndarray]:
32
33         """Load XOR input and output datasets.
34
35         Args:
36             train_dataset_size (int): the number of dataset inputs to use.
37
38         Returns:
39             tuple of XOR train_inputs, train_outputs,
40             test_inputs and test_outputs numpy.ndarrays.
41
42         """
43         inputs: np.ndarray = np.array([[0, 0, 1, 1],
44                                         [0, 1, 0, 1]])
45         outputs: np.ndarray = np.array([[0, 1, 1, 0]])
46
47         # Reduce train datasets' sizes to train_dataset_size
48         inputs = (inputs.T[:train_dataset_size]).T
49         outputs = (outputs.T[:train_dataset_size]).T
50
51         return inputs, outputs, inputs, outputs

```

---

### 3.3 frames package

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

- hyper-parameter-defaults.json file contents:

```

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

```

```

21         "hiddenLayersShape": [100, 100],
22         "minTrainDatasetSize": 2,
23         "maxTrainDatasetSize": 4,
24         "maxLearningRate": 1
25     }
26 }

```

---

- create\_model.py module:

---

```

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

```

```

50         )
51     self.learning_rate_scale = tk.Scale(
52         master=self,
53         bg=self.BG,
54         orient='horizontal',
55         label="Learning Rate",
56         length=185,
57         from_=0,
58
59         ↪ to=self.default_hyper_parameters['maxLearningRate'],
60         resolution=0.01
61     )
62     self.learning_rate_scale.set(value=0.1)
63     self.epoch_count_scale = tk.Scale(master=self,
64         bg=self.BG,
65         orient='horizontal',
66         label="Epoch Count",
67         length=185,
68         from_=0,
69         to=10_000,
70         resolution=100)
71     self.epoch_count_scale.set(
72         ↪ value=self.default_hyper_parameters['epochCount']
73     )
74     self.train_dataset_size_scale = tk.Scale(
75         master=self,
76         bg=self.BG,
77         orient='horizontal',
78         label="Train Dataset Size",
79         length=185,
80
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
93         ↪ each\n" +
94         ↪ "hidden layer, separated by
95         ↪ commas:"
96     )
97     self.hidden_layers_shape_entry = tk.Entry(master=self)
98     self.hidden_layers_shape_entry.insert(0, ",".join(
99         f"{neuron_count}" for neuron_count in
100         ↪ self.default_hyper_parameters['hiddenLayersShape']
101     ))
102     self.use_relu_check_button_var = tk.BooleanVar(value=True)
103     self.use_relu_check_button = tk.Checkbutton(
104         master=self,
105         width=13, height=1,
106         font=tkf.Font(size=12),
107         text="Use ReLu",
108
109         ↪ variable=self.use_relu_check_button_var

```



```

104         )
105     self.use_gpu_check_button_var = tk.BooleanVar()
106     self.use_gpu_check_button = tk.Checkbutton(
107         master=self,
108         width=13, height=1,
109         font=tkf.Font(size=12),
110         text="Use GPU",
111         ↪ variable=self.use_gpu_check_button_var
112     )
113     self.model_status_label = tk.Label(master=self,
114         bg=self.BG,
115         font=('Arial', 15))
116
117     # Pack widgets
118     self.title_label.grid(row=0, column=0, columnspan=3)
119     self.about_label.grid(row=1, column=0, columnspan=3)
120     self.learning_rate_scale.grid(row=2, column=0, pady=(50,0))
121     self.epoch_count_scale.grid(row=3, column=0, pady=(30,0))
122     self.train_dataset_size_scale.grid(row=4, column=0, pady=(30,0))
123     self.hidden_layers_shape_label.grid(row=2, column=1,
124         padx=30, pady=(50,0))
125     self.hidden_layers_shape_entry.grid(row=3, column=1, padx=30)
126     self.use_relu_check_button.grid(row=2, column=2, pady=(30, 0))
127     self.use_gpu_check_button.grid(row=3, column=2, pady=(30, 0))
128     self.model_status_label.grid(row=5, column=0,
129         columnspan=3, pady=50)
130
131     def load_default_hyper_parameters(self, dataset: str) -> dict[
132         str,
133         str | int | list[int] |
134         ↪ float
135         ]:
136         """Load the dataset's default hyper-parameters from the json file.
137
138         Args:
139             dataset (str): the name of the dataset to load
140             ↪ 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
146             ↪ f:
147             return json.load(f)[dataset]
148
149     def create_model(self) -> object:
150         """Create and return a Model using the hyper-parameters set.
151
152         Returns:
153             a Model object.
154         """
155         self.use_gpu = self.use_gpu_check_button_var.get()
156
157         # Validate hidden layers shape input
158         hidden_layers_shape_input = [layer for layer in
159             ↪ self.hidden_layers_shape_entry.get().replace(' ',
160             ↪ '').split(',') if layer != '']
161         for layer in hidden_layers_shape_input:
162             if not layer.isdigit():
163                 self.model_status_label.configure(
164                     text="Invalid hidden layers shape",

```

```

160                                     fg='red'
161                                     )
162         raise ValueError
163
164     # Create Model
165     if not self.use_gpu:
166         if self.dataset == "MNIST":
167             from school_project.models.cpu.mnist import MNISTModel as
168                 ↪ Model
169         elif self.dataset == "Cat Recognition":
170             from school_project.models.cpu.cat_recognition import
171                 ↪ CatRecognitionModel as Model
172         elif self.dataset == "XOR":
173             from school_project.models.cpu.xor import XORModel as Model
174     model = Model(hidden_layers_shape = [int(neuron_count) for
175         ↪ neuron_count in hidden_layers_shape_input],
176                 train_dataset_size =
177                     ↪ self.train_dataset_size_scale.get(),
178                 learning_rate = self.learning_rate_scale.get(),
179                 use_relu = self.use_relu_check_button_var.get())
180     model.create_model_values()
181
182     else:
183         try:
184             if self.dataset == "MNIST":
185                 from school_project.models.gpu.mnist import MNISTModel
186                 ↪ as Model
187             elif self.dataset == "Cat Recognition":
188                 from school_project.models.gpu.cat_recognition import
189                 ↪ CatRecognitionModel as Model
190             elif self.dataset == "XOR":
191                 from school_project.models.gpu.xor import XORModel as
192                 ↪ Model
193             model = Model(hidden_layers_shape = [int(neuron_count) for
194                 ↪ neuron_count in hidden_layers_shape_input],
195                     train_dataset_size =
196                         ↪ self.train_dataset_size_scale.get(),
197                     learning_rate =
198                         ↪ self.learning_rate_scale.get(),
199                     use_relu =
200                         ↪ self.use_relu_check_button_var.get())
201             model.create_model_values()
202         except ImportError as ie:
203             self.model_status_label.configure(
204                 text="Failed to initialise GPU",
205                 fg='red'
206                 )
207             raise ImportError
208     return model
209
210 class TrainingFrame(tk.Frame):
211     """Frame for training page."""
212     def __init__(self, root: tk.Tk, width: int,
213                 height: int, bg: str,
214                 model: object, epoch_count: int) -> None:
215         """Initialise training frame widgets.
216
217         Args:
218             root (tk.Tk): the widget object that contains this widget.
219             width (int): the pixel width of the frame.
220             height (int): the pixel height of the frame.
221             bg (str): the hex value or name of the frame's background
222             ↪ colour.

```

```

211         model (object): the Model object to be trained.
212         epoch_count (int): the number of training epochs.
213     Raises:
214         TypeError: if root, width or height are not of the correct
↪ type.
215
216     """
217     super().__init__(master=root, width=width, height=height, bg=bg)
218     self.root = root
219     self.WIDTH = width
220     self.HEIGHT = height
221     self.BG = bg
222
223     # Setup widgets
224     self.model_status_label = tk.Label(master=self,
225                                       bg=self.BG,
226                                       font=('Arial', 15))
227     self.training_progress_label = tk.Label(master=self,
228                                             bg=self.BG,
229                                             font=('Arial', 15))
230     self.loss_figure: Figure = Figure()
231     self.loss_canvas: FigureCanvasTkAgg = FigureCanvasTkAgg(
232
233         ↪ figure=self.loss_figure,
234         master=self
235     )
236
237     # Pack widgets
238     self.model_status_label.pack(pady=(30,0))
239     self.training_progress_label.pack(pady=30)
240
241     # Start training thread
242     self.model_status_label.configure(
243         text="Training weights and
244         ↪ biases...",
245         fg='red'
246     )
247     self.train_thread: threading.Thread = threading.Thread(
248
249         ↪ target=model.train,
250
251         ↪ args=(epoch_count,)
252     )
253     self.train_thread.start()
254
255     def plot_losses(self, model: object) -> None:
256         """Plot losses of Model training.
257
258         Args:
259             model (object): the Model object thats been trained.
260
261         """
262         self.model_status_label.configure(
263             text=f"Weights and biases trained in
264             ↪ {model.training_time}s",
265             fg='green'
266         )
267         graph: Figure.axes = self.loss_figure.add_subplot(111)
268         graph.set_title("Learning rate: " +
269                        f"{model.learning_rate}")
270         graph.set_xlabel("Epochs")
271         graph.set_ylabel("Loss Value")

```

```

267         graph.plot(np.squeeze(model.train_losses))
268         self.loss_canvas.get_tk_widget().pack()

```

---

- load\_model.py module:

---

```

1  import sqlite3
2  import tkinter as tk
3  import tkinter.font as tkf
4
5  class LoadModelFrame(tk.Frame):
6      """Frame for load model page."""
7      def __init__(self, root: tk.Tk,
8                  width: int, height: int,
9                  bg: str, connection: sqlite3.Connection,
10                 cursor: sqlite3.Cursor, dataset: str) -> None:
11         """Initialise load model frame widgets.
12
13         Args:
14             root (tk.Tk): the widget object that contains this widget.
15             width (int): the pixel width of the frame.
16             height (int): the pixel height of the frame.
17             bg (str): the hex value or name of the frame's background
18         ↪ colour.
19             connection (sqlite3.Connection): the database connection
20         ↪ object.
21             cursor (sqlite3.Cursor): the database cursor object.
22             dataset (str): the name of the dataset to use
23             ('MNIST', 'Cat Recognition' or 'XOR')
24         Raises:
25             TypeError: if root, width or height are not of the correct
26         ↪ type.
27
28         """
29         super().__init__(master=root, width=width, height=height, bg=bg)
30         self.root = root
31         self.WIDTH = width
32         self.HEIGHT = height
33         self.BG = bg
34
35         # Setup load model frame variables
36         self.connection = connection
37         self.cursor = cursor
38         self.dataset = dataset
39         self.use_gpu: bool
40         self.model_options = self.load_model_options()
41
42         # Setup widgets
43         self.title_label = tk.Label(master=self,
44                                     bg=self.BG,
45                                     font=('Arial', 20),
46                                     text=dataset)
47
48         self.about_label = tk.Label(
49             master=self,
50             bg=self.BG,
51             font=('Arial', 14),
52             text=f"Load a pretrained model for the {dataset}"
53             ↪ dataset."
54         )
55
56         self.model_status_label = tk.Label(master=self,
57                                             bg=self.BG,
58                                             font=('Arial', 15))

```

```

53
54     # Don't give loaded model options if no models have been saved for
55     ↳ the
56     # dataset.
57     if len(self.model_options) > 0:
58         self.model_option_menu_label = tk.Label(
59             master=self,
60             bg=self.BG,
61             font=('Arial', 14),
62             text="Select a model to
63                 ↳ load or delete:"
64         )
65         self.model_option_menu_var = tk.StringVar(
66             master=self,
67             ↳ value=self.model_options[0]
68         )
69         self.model_option_menu = tk.OptionMenu(
70             self,
71             ↳ self.model_option_menu_var,
72             *self.model_options
73         )
74         self.use_gpu_check_button_var = tk.BooleanVar()
75         self.use_gpu_check_button = tk.Checkbutton(
76             master=self,
77             width=7, height=1,
78             font=tkf.Font(size=12),
79             text="Use GPU",
80             ↳ variable=self.use_gpu_check_button_var
81         )
82     else:
83         self.model_status_label.configure(
84             text='No saved models for this
85                 ↳ dataset.',
86             fg='red'
87         )
88
89     # Pack widgets
90     self.title_label.grid(row=0, column=0, columnspan=3)
91     self.about_label.grid(row=1, column=0, columnspan=3)
92     if len(self.model_options) > 0: # Check if options should be given
93         self.model_option_menu_label.grid(row=2, column=0, padx=(0,30),
94             ↳ pady=(30,0))
95         self.use_gpu_check_button.grid(row=2, column=2, rowspan=2,
96             ↳ pady=(30,0))
97         self.model_option_menu.grid(row=3, column=0, padx=(0,30),
98             ↳ pady=(10,0))
99         self.model_status_label.grid(row=4, column=0,
100             columnspan=3, pady=50)
101
102     def load_model_options(self) -> list[str]:
103         """Load the model options from the database.
104         Returns:
105             a list of the model options.
106         """
107         sql = f"""
108         SELECT Name FROM Models WHERE Dataset=?
109         """
110         parameters = (self.dataset.replace(" ", "_"),)

```

```

106         self.cursor.execute(sql, parameters)
107
108         # Save the string value contained within the tuple of each row
109         model_options = []
110         for model_option in self.cursor.fetchall():
111             model_options.append(model_option[0])
112
113         return model_options
114
115     def load_model(self) -> object:
116         """Create model using saved weights and biases.
117
118         Returns:
119         a Model object.
120
121         """
122         self.use_gpu = self.use_gpu_check_button_var.get()
123
124         # Query data of selected saved model from database
125         sql = """
126         SELECT * FROM Models WHERE Dataset=? AND Name=?
127         """
128         parameters = (self.dataset.replace(" ", "_"),
129             ↪ self.model_option_menu_var.get())
130         self.cursor.execute(sql, parameters)
131         data = self.cursor.fetchone()
132         hidden_layers_shape_input = [layer for layer in data[3].replace('
133             ↪ ', '').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
139                 ↪ Model
140             elif self.dataset == "Cat Recognition":
141                 from school_project.models.cpu.cat_recognition import
142                 ↪ CatRecognitionModel as Model
143             elif self.dataset == "XOR":
144                 from school_project.models.cpu.xor import XORModel as Model
145             model = Model(
146                 hidden_layers_shape=[int(neuron_count) for neuron_count in
147                 ↪ hidden_layers_shape_input],
148                 train_dataset_size=data[6],
149                 learning_rate=data[4],
150                 use_relu=data[7]
151             )
152             model.load_model_values(file_location=data[2])
153
154         else:
155             try:
156                 if self.dataset == "MNIST":
157                     from school_project.models.gpu.mnist import MNISTModel
158                     ↪ as Model
159                 elif self.dataset == "Cat Recognition":
160                     from school_project.models.gpu.cat_recognition import
161                     ↪ CatRecognitionModel as Model
162                 elif self.dataset == "XOR":
163                     from school_project.models.gpu.xor import XORModel as
164                     ↪ Model
165                 model = Model(
166                     hidden_layers_shape=[int(neuron_count) for neuron_count
167                     ↪ in hidden_layers_shape_input],

```

---

```

159         train_dataset_size=data[6],
160         learning_rate=data[4],
161         use_relu=data[7]
162     )
163     model.load_model_values(file_location=data[2])
164 except ImportError as ie:
165     self.model_status_label.configure(
166         text="Failed to initialise
167             ↪ GPU",
168         fg='red'
169     )
170     raise ImportError
171 return model

```

---

### 3.4 \_\_main\_\_.py module

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

---

```

1  import os
2  import sqlite3
3  import threading
4  import tkinter as tk
5  import tkinter.font as tkf
6  import uuid
7
8  import pympler.tracker as tracker
9
10 from school_project.frames.create_model import (HyperParameterFrame,
11                                                  TrainingFrame)
12 from school_project.frames.load_model import LoadModelFrame
13 from school_project.frames.test_model import (TestMNISTFrame,
14                                               TestCatRecognitionFrame,
15                                               TestXORFrame)
16
17 class SchoolProjectFrame(tk.Frame):
18     """Main frame of school project."""
19     def __init__(self, root: tk.Tk, width: int, height: int, bg: str) -> None:
20         """Initialise school project pages.
21
22         Args:
23             root (tk.Tk): the widget object that contains this widget.
24             width (int): the pixel width of the frame.
25             height (int): the pixel height of the frame.
26             bg (str): the hex value or name of the frame's background colour.
27
28         Raises:
29             TypeError: if root, width or height are not of the correct type.
30
31         """
32         super().__init__(master=root, width=width, height=height, bg=bg)
33         self.root = root.title("School Project")
34         self.WIDTH = width
35         self.HEIGHT = height
36         self.BG = bg
37
38         # Setup school project frame variables
39         self.hyper_parameter_frame: HyperParameterFrame
40         self.training_frame: TrainingFrame
41         self.load_model_frame: LoadModelFrame
42         self.test_frame: TestMNISTFrame | TestCatRecognitionFrame | TestXORFrame

```

```

42     self.connection, self.cursor = self.setup_database()
43     self.model = None
44
45     # Record if the model should be saved after testing,
46     # as only newly created models should be given the option to be saved.
47     self.saving_model: bool
48
49     # Setup school project frame widgets
50     self.exit_hyper_parameter_frame_button = tk.Button(
51         master=self,
52         width=13,
53         height=1,
54         font=tkf.Font(size=12),
55         text="Exit",
56         command=self.exit_hyper_parameter_frame
57     )
58     self.exit_load_model_frame_button = tk.Button(
59         master=self,
60         width=13,
61         height=1,
62         font=tkf.Font(size=12),
63         text="Exit",
64         command=self.exit_load_model_frame
65     )
66     self.train_button = tk.Button(master=self,
67         width=13,
68         height=1,
69         font=tkf.Font(size=12),
70         text="Train Model",
71         command=self.enter_training_frame)
72     self.stop_training_button = tk.Button(
73         master=self,
74         width=15, height=1,
75         font=tkf.Font(size=12),
76         text="Stop Training Model",
77         command=lambda: self.model.set_running(
78             value=False
79         )
80     )
81     self.test_created_model_button = tk.Button(
82         master=self,
83         width=13, height=1,
84         font=tkf.Font(size=12),
85         text="Test Model",
86         command=self.test_created_model
87     )
88     self.test_loaded_model_button = tk.Button(
89         master=self,
90         width=13, height=1,
91         font=tkf.Font(size=12),
92         text="Test Model",
93         command=self.test_loaded_model
94     )
95     self.delete_loaded_model_button = tk.Button(
96         master=self,
97         width=13, height=1,
98         font=tkf.Font(size=12),
99         text="Delete Model",
100        command=self.delete_loaded_model
101    )
102    self.save_model_label = tk.Label(
103        master=self,

```



```

104         text="Enter a name for your trained model:",
105         bg=self.BG,
106         font=('Arial', 15)
107     )
108     self.save_model_name_entry = tk.Entry(master=self, width=13)
109     self.save_model_button = tk.Button(master=self,
110         width=13,
111         height=1,
112         font=tkf.Font(size=12),
113         text="Save Model",
114         command=self.save_model)
115     self.exit_button = tk.Button(master=self,
116         width=13, height=1,
117         font=tkf.Font(size=12),
118         text="Exit",
119         command=self.enter_home_frame)
120
121     # Setup home frame
122     self.home_frame = tk.Frame(master=self,
123         width=self.WIDTH,
124         height=self.HEIGHT,
125         bg=self.BG)
126     self.title_label = tk.Label(
127         master=self.home_frame,
128         bg=self.BG,
129         font=('Arial', 20),
130         text="A-level Computer Science NEA Programming Project"
131     )
132     self.about_label = tk.Label(
133         master=self.home_frame,
134         bg=self.BG,
135         font=('Arial', 14),
136         text="An investigation into how Artificial Neural Networks work, " +
137         "the effects of their hyper-parameters and their applications " +
138         "in Image Recognition.\n\n" +
139         " - Max Cotton"
140     )
141     self.model_menu_label = tk.Label(master=self.home_frame,
142         bg=self.BG,
143         font=('Arial', 14),
144         text="Create a new model " +
145         "or load a pre-trained model "
146         "for one of the following datasets:")
147     self.dataset_option_menu_var = tk.StringVar(master=self.home_frame,
148         value="MNIST")
149     self.dataset_option_menu = tk.OptionMenu(self.home_frame,
150         self.dataset_option_menu_var,
151         "MNIST",
152         "Cat Recognition",
153         "XOR")
154     self.create_model_button = tk.Button(
155         master=self.home_frame,
156         width=13, height=1,
157         font=tkf.Font(size=12),
158         text="Create Model",
159         command=self.enter_hyper_parameter_frame
160     )
161     self.load_model_button = tk.Button(master=self.home_frame,
162         width=13, height=1,
163         font=tkf.Font(size=12),
164         text="Load Model",
165         command=self.enter_load_model_frame)

```

```

166
167     # Grid home frame widgets
168     self.title_label.grid(row=0, column=0, columnspan=4, pady=(10,0))
169     self.about_label.grid(row=1, column=0, columnspan=4, pady=(10,50))
170     self.model_menu_label.grid(row=2, column=0, columnspan=4)
171     self.dataset_option_menu.grid(row=3, column=0, columnspan=4, pady=30)
172     self.create_model_button.grid(row=4, column=1)
173     self.load_model_button.grid(row=4, column=2)
174
175     self.home_frame.pack()
176
177     # Setup frame attributes
178     self.grid_propagate(flag=False)
179     self.pack_propagate(flag=False)
180
181     @staticmethod
182     def setup_database() -> tuple[sqlite3.Connection, sqlite3.Cursor]:
183         """Create a connection to the pretrained_models database file and
184             setup base table if needed.
185
186         Returns:
187             a tuple of the database connection and the cursor for it.
188
189         """
190         connection = sqlite3.connect(
191             database='school_project/saved_models.db'
192         )
193         cursor = connection.cursor()
194         cursor.execute("""
195             CREATE TABLE IF NOT EXISTS Models
196             (Model_ID INTEGER PRIMARY KEY,
197             Dataset TEXT,
198             File_Location TEXT,
199             Hidden_Layers_Shape TEXT,
200             Learning_Rate FLOAT,
201             Name TEXT,
202             Train_Dataset_Size INTEGER,
203             Use_ReLu INTEGER,
204             UNIQUE (Dataset, Name))
205         """)
206         return (connection, cursor)
207
208     def enter_hyper_parameter_frame(self) -> None:
209         """Unpack home frame and pack hyper-parameter frame."""
210         self.home_frame.pack_forget()
211         self.hyper_parameter_frame = HyperParameterFrame(
212             root=self,
213             width=self.WIDTH,
214             height=self.HEIGHT,
215             bg=self.BG,
216             dataset=self.dataset_option_menu_var.get()
217         )
218         self.hyper_parameter_frame.pack()
219         self.train_button.pack()
220         self.exit_hyper_parameter_frame_button.pack(pady=(10,0))
221
222     def enter_load_model_frame(self) -> None:
223         """Unpack home frame and pack load model frame."""
224         self.home_frame.pack_forget()
225         self.load_model_frame = LoadModelFrame(
226             root=self,
227             width=self.WIDTH,

```

```

228             height=self.HEIGHT,
229             bg=self.BG,
230             connection=self.connection,
231             cursor=self.cursor,
232             dataset=self.dataset_option_menu_var.get()
233         )
234     self.load_model_frame.pack()
235
236     # Don't give option to test loaded model if no models have been saved
237     # for the dataset.
238     if len(self.load_model_frame.model_options) > 0:
239         self.test_loaded_model_button.pack()
240         self.delete_loaded_model_button.pack(pady=(5,0))
241
242     self.exit_load_model_frame_button.pack(pady=(5,0))
243
244     def exit_hyper_parameter_frame(self) -> None:
245         """Unpack hyper-parameter frame and pack home frame."""
246         self.hyper_parameter_frame.pack_forget()
247         self.train_button.pack_forget()
248         self.exit_hyper_parameter_frame_button.pack_forget()
249         self.home_frame.pack()
250
251     def exit_load_model_frame(self) -> None:
252         """Unpack load model frame and pack home frame."""
253         self.load_model_frame.pack_forget()
254         self.test_loaded_model_button.pack_forget()
255         self.delete_loaded_model_button.pack_forget()
256         self.exit_load_model_frame_button.pack_forget()
257         self.home_frame.pack()
258
259     def enter_training_frame(self) -> None:
260         """Load untrained model from hyper parameter frame,
261         unpack hyper-parameter frame, pack training frame
262         and begin managing the training thread.
263         """
264         try:
265             self.model = self.hyper_parameter_frame.create_model()
266         except (ValueError, ImportError) as e:
267             return
268         self.hyper_parameter_frame.pack_forget()
269         self.train_button.pack_forget()
270         self.exit_hyper_parameter_frame_button.pack_forget()
271         self.training_frame = TrainingFrame(
272             root=self,
273             width=self.WIDTH,
274             height=self.HEIGHT,
275             bg=self.BG,
276             model=self.model,
277             epoch_count=self.hyper_parameter_frame.epoch_count_scale.get()
278         )
279         self.training_frame.pack()
280         self.stop_training_button.pack()
281         self.manage_training(train_thread=self.training_frame.train_thread)
282
283     def manage_training(self, train_thread: threading.Thread) -> None:
284         """Wait for model training thread to finish,
285         then plot training losses on training frame.
286
287     Args:
288         train_thread (threading.Thread):
289             the thread running the model's train() method.

```

```

290         Raises:
291         TypeError: if train_thread is not of type threading.Thread.
292
293         """
294         if not train_thread.is_alive():
295             self.training_frame.training_progress_label.pack_forget()
296             self.training_frame.plot_losses(model=self.model)
297             self.stop_training_button.pack_forget()
298             self.test_created_model_button.pack(pady=(30,0))
299         else:
300             self.training_frame.training_progress_label.configure(
301                 text=self.model.training_progress
302             )
303             self.after(100, self.manage_training, train_thread)
304
305     def test_created_model(self) -> None:
306         """Unpack training frame, pack test frame for the dataset
307         and begin managing the test thread."""
308         self.saving_model = True
309         self.training_frame.pack_forget()
310         self.test_created_model_button.pack_forget()
311         if self.hyper_parameter_frame.dataset == "MNIST":
312             self.test_frame = TestMNISTFrame(
313                 root=self,
314                 width=self.WIDTH,
315                 height=self.HEIGHT,
316                 bg=self.BG,
317                 use_gpu=self.hyper_parameter_frame.use_gpu,
318                 model=self.model
319             )
320         elif self.hyper_parameter_frame.dataset == "Cat Recognition":
321             self.test_frame = TestCatRecognitionFrame(
322                 root=self,
323                 width=self.WIDTH,
324                 height=self.HEIGHT,
325                 bg=self.BG,
326                 use_gpu=self.hyper_parameter_frame.use_gpu,
327                 model=self.model
328             )
329         elif self.hyper_parameter_frame.dataset == "XOR":
330             self.test_frame = TestXORFrame(root=self,
331                 width=self.WIDTH,
332                 height=self.HEIGHT,
333                 bg=self.BG,
334                 model=self.model)
335         self.test_frame.pack()
336         self.manage_testing(test_thread=self.test_frame.test_thread)
337
338     def test_loaded_model(self) -> None:
339         """Load saved model from load model frame, unpack load model frame,
340         pack test frame for the dataset and begin managing the test thread."""
341         self.saving_model = False
342         try:
343             self.model = self.load_model_frame.load_model()
344         except (ValueError, ImportError) as e:
345             return
346         self.load_model_frame.pack_forget()
347         self.test_loaded_model_button.pack_forget()
348         self.delete_loaded_model_button.pack_forget()
349         self.exit_load_model_frame_button.pack_forget()
350         if self.load_model_frame.dataset == "MNIST":
351             self.test_frame = TestMNISTFrame(

```

```

352         root=self,
353         width=self.WIDTH,
354         height=self.HEIGHT,
355         bg=self.BG,
356         use_gpu=self.load_model_frame.use_gpu,
357         model=self.model
358     )
359     elif self.load_model_frame.dataset == "Cat Recognition":
360         self.test_frame = TestCatRecognitionFrame(
361             root=self,
362             width=self.WIDTH,
363             height=self.HEIGHT,
364             bg=self.BG,
365             use_gpu=self.load_model_frame.use_gpu,
366             model=self.model
367         )
368     elif self.load_model_frame.dataset == "XOR":
369         self.test_frame = TestXORFrame(root=self,
370             width=self.WIDTH,
371             height=self.HEIGHT,
372             bg=self.BG,
373             model=self.model)
374     self.test_frame.pack()
375     self.manage_testing(test_thread=self.test_frame.test_thread)
376
377 def manage_testing(self, test_thread: threading.Thread) -> None:
378     """Wait for model test thread to finish,
379     then plot results on test frame.
380
381     Args:
382         test_thread (threading.Thread):
383             the thread running the model's predict() method.
384     Raises:
385         TypeError: if test_thread is not of type threading.Thread.
386
387     """
388     if not test_thread.is_alive():
389         self.test_frame.plot_results(model=self.model)
390         if self.saving_model:
391             self.save_model_label.pack(pady=(30,0))
392             self.save_model_name_entry.pack(pady=10)
393             self.save_model_button.pack()
394             self.exit_button.pack(pady=(20,0))
395         else:
396             self.after(1_000, self.manage_testing, test_thread)
397
398 def save_model(self) -> None:
399     """Save the model, save the model information to the database, then
400     enter the home frame."""
401     model_name = self.save_model_name_entry.get()
402
403     # Check if model name is empty
404     if model_name == '':
405         self.test_frame.model_status_label.configure(
406             text="Model name can not be blank",
407             fg='red'
408         )
409         return
410
411     # Check if model name has already been taken
412     dataset = self.dataset_option_menu_var.get().replace(" ", "_")
413     sql = """

```

```

414         SELECT Name FROM Models WHERE Dataset=?
415         """
416         parameters = (dataset,)
417         self.cursor.execute(sql, parameters)
418         for saved_model_name in self.cursor.fetchall():
419             if saved_model_name[0] == model_name:
420                 self.test_frame.model_status_label.configure(
421                     text="Model name taken",
422                     fg='red'
423                 )
424                 return
425
426         # Save model to random hex file name
427         file_location = f"school_project/saved-models/{uuid.uuid4().hex}.npz"
428         self.model.save_model_values(file_location=file_location)
429
430         # Save the model information to the database
431         sql = """
432         INSERT INTO Models
433         (Dataset, File_Location, Hidden_Layers_Shape, Learning_Rate, Name,
↪ Train_Dataset_Size, Use_ReLu)
434         VALUES (?, ?, ?, ?, ?, ?, ?)
435         """
436         parameters = (
437             dataset,
438             file_location,
439             self.hyper_parameter_frame.hidden_layers_shape_entry.get(),
440             self.hyper_parameter_frame.learning_rate_scale.get(),
441             model_name,
442             self.hyper_parameter_frame.train_dataset_size_scale.get(),
443             self.hyper_parameter_frame.use_relu_check_button_var.get()
444         )
445         self.cursor.execute(sql, parameters)
446         self.connection.commit()
447
448         self.enter_home_frame()
449
450     def delete_loaded_model(self) -> None:
451         """Delete saved model file and model data from the database."""
452         dataset = self.dataset_option_menu_var.get().replace(" ", "_")
453         model_name = self.load_model_frame.model_option_menu_var.get()
454
455         # Delete saved model
456         sql = f"""SELECT File_Location FROM Models WHERE Dataset=? AND Name=?"""
457         parameters = (dataset, model_name)
458         self.cursor.execute(sql, parameters)
459         os.remove(self.cursor.fetchone()[0])
460
461         # Remove model data from database
462         sql = """DELETE FROM Models WHERE Dataset=? AND Name=?"""
463         parameters = (dataset, model_name)
464         self.cursor.execute(sql, parameters)
465         self.connection.commit()
466
467         # Reload load model frame with new options
468         self.exit_load_model_frame()
469         self.enter_load_model_frame()
470
471     def enter_home_frame(self) -> None:
472         """Unpack test frame and pack home frame."""
473         self.model = None # Free up trained Model from memory
474         self.test_frame.pack_forget()

```

```

475         if self.saving_model:
476             self.save_model_label.pack_forget()
477             self.save_model_name_entry.delete(0, tk.END) # Clear entry's text
478             self.save_model_name_entry.pack_forget()
479             self.save_model_button.pack_forget()
480         self.exit_button.pack_forget()
481         self.home_frame.pack()
482         summary_tracker.create_summary() # BUG: Object summary seems to reduce
483                                         # memory leak greatly
484
485     def main() -> None:
486         """Entrypoint of project."""
487         root = tk.Tk()
488         school_project = SchoolProjectFrame(root=root, width=1280,
489                                             height=835, bg='white')
490         school_project.pack(side='top', fill='both', expand=True)
491         root.mainloop()
492
493         # Stop model training when GUI closes
494         if school_project.model != None:
495             school_project.model.set_running(value=False)
496
497 if __name__ == "__main__":
498     summary_tracker = tracker.SummaryTracker() # Setup object tracker
499     main()

```

---