

Computer Science NEA Report

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

Max Cotton

Contents

1	Analysis	2
1.1	About	2
1.2	Interview	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
2	Design	11
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

3	Technical Solution TODO	16
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

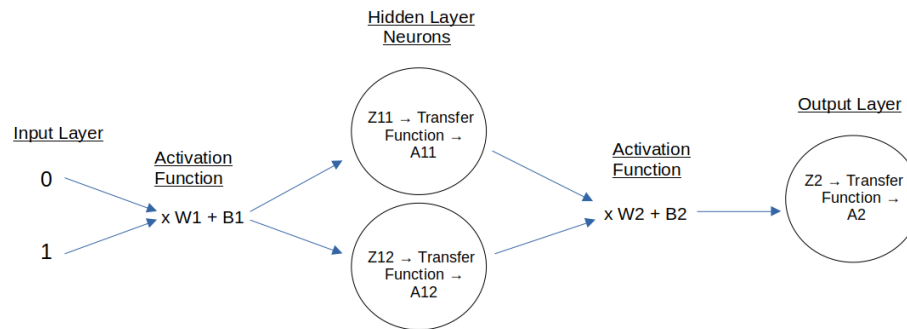


Figure 1: This shows an Artificial Neural Network with one single hidden layer and is known as a Shallow Neural Network.

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 useful for logistic regression where the output value can be considered as closer to zero or one allowing for a binary classification of predicting zero or one.

1.4.2 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

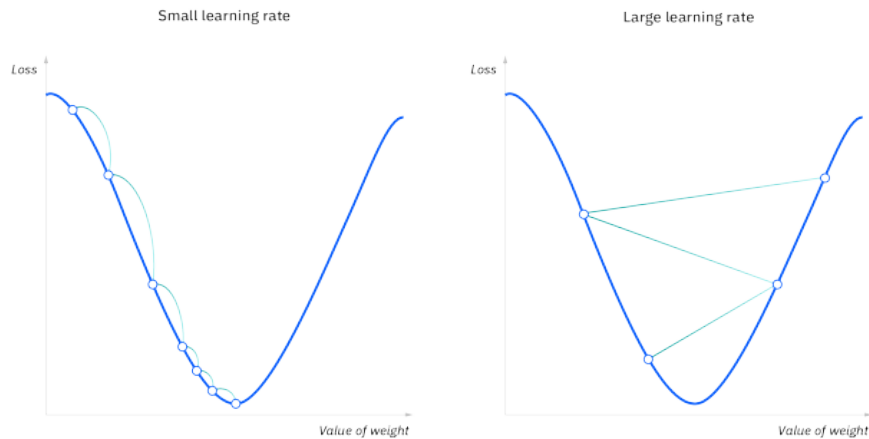


Figure 2: Gradient Descent
sourced from <https://www.ibm.com/topics/gradient-descent>

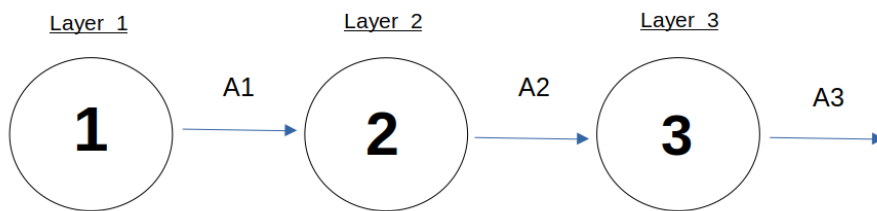


Figure 3: This shows an abstracted view of an Artificial Neural Network with multiple hidden layers and is known as a Deep Neural Network.

- 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 layer's 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

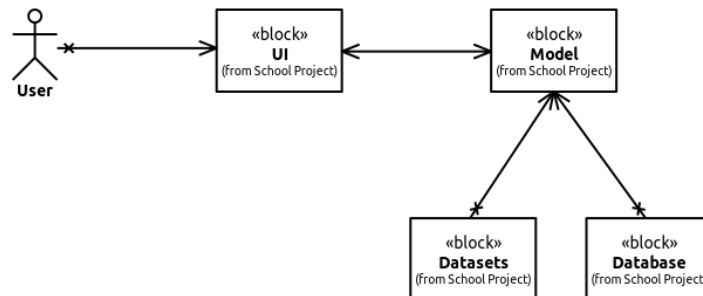
The following design focuses have been made for the project:

- 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

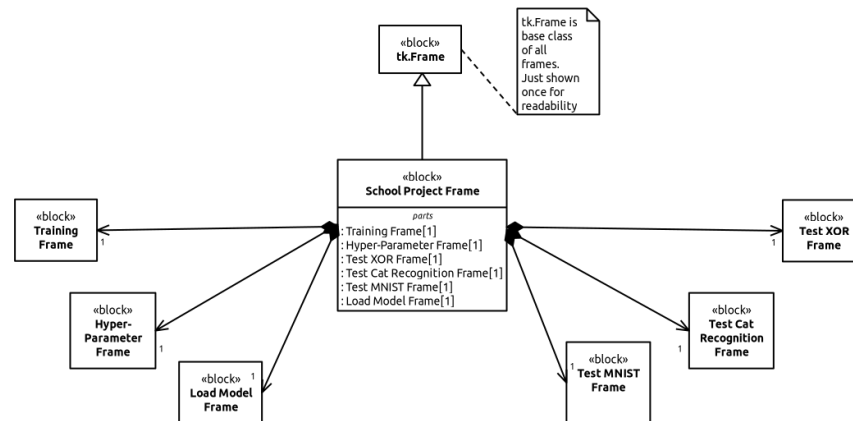
bdd [block] School Project Frame [System Architecture Diagram]



2.3 Class Diagrams

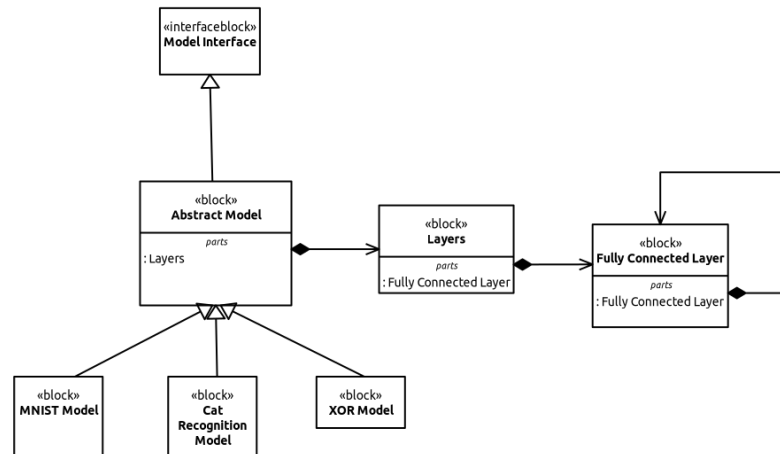
2.3.1 UI Class Diagram

bdd [package] School Project [UI Class Diagram]



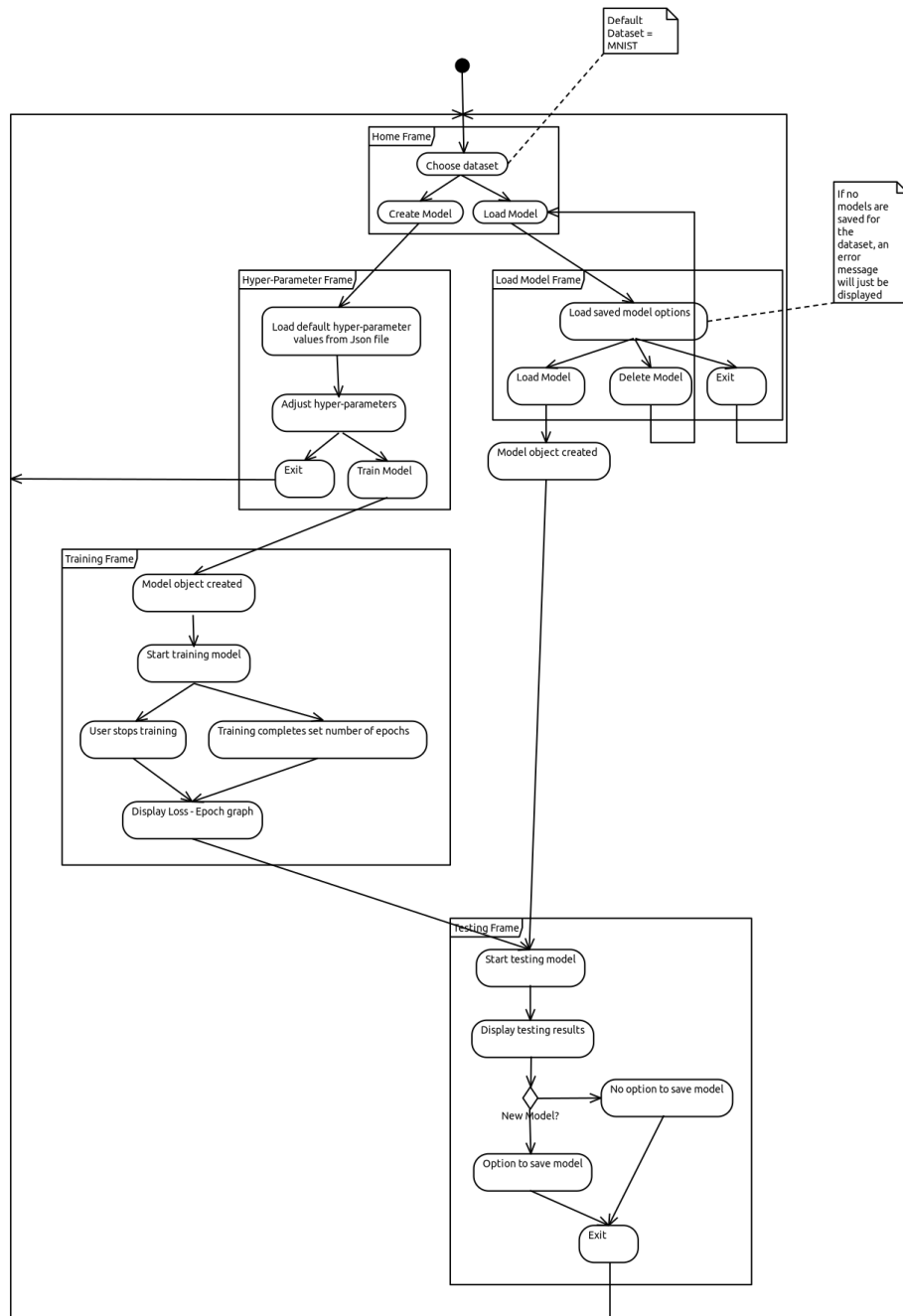
2.3.2 Model Class Diagram

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

```



```

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

```

```

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

```

```

162     Args:
163         output (numpy.ndarray | int | float):
164             the numpy.ndarray | int | float output of the sigmoid transfer
↪ 165     function.
166     Returns:
167         numpy.ndarray | float,
168         derivative of the sigmoid transfer function with respect to z.
169     Raises:
170         TypeError: if output is not of type numpy.ndarray | int | float.
171
172     """
173     return output * (1 - output)
174
175 def calculate_loss(input_count: int,
176                  outputs: np.ndarray,
177                  prediction: np.ndarray) -> float:
178     """Calculate average loss/error of the prediction to the outputs.
179
180     Args:
181         input_count (int): the number of inputs.
182         outputs (np.ndarray):
183             the train/test outputs array to compare with the prediction.
184         prediction (np.ndarray): the array of prediction values.
185     Returns:
186         float loss.
187     Raises:
188         ValueError:
189             if outputs is not a suitable multiplier with the prediction
190             (incorrect shapes)
191
192     """
193     return np.squeeze(- (1/input_count) * np.sum(outputs *
↪ 194     np.log(prediction) + (1 - outputs) * np.log(1 - prediction)))
195
196 def calculate_prediction_accuracy(prediction: np.ndarray,
197                                 outputs: np.ndarray) -> float:
198     """Calculate the percentage accuracy of the predictions.
199
200     Args:
201         prediction (np.ndarray): the array of prediction values.
202         outputs (np.ndarray):
203             the train/test outputs array to compare with the prediction.
204     Returns:
205         float prediction accuracy
206
207     """
208     return 100 - np.mean(np.abs(prediction - outputs)) * 100

```

- model.py module:

```

1  """Provides an abstract class for Artificial Neural Network models."""
2
3  import time
4
5  import numpy as np
6
7  from .tools import (
8      ModelInterface,
9      relu,
10     relu_derivative,
11     sigmoid,

```

```

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

```

```

74
75 def __repr__(self) -> str:
76     """Read values of the layer.
77
78     Returns:
79         a string description of the layers's
80         weights, bias and learning rate values.
81
82     """
83     return (f"Weights: {self.weights.tolist()}\n" +
84             f"Biases: {self.biases.tolist()}\n")
85
86 def init_layer_values_random(self) -> None:
87     """Initialise weights to random values and biases to 0s"""
88     np.random.seed(1) # Sets up pseudo random values for layer weight
89     ↪ arrays
90     self.weights = np.random.rand(self.output_neuron_count,
91     ↪ self.input_neuron_count) - 0.5
92     self.biases = np.zeros(shape=(self.output_neuron_count, 1))
93
94 def init_layer_values_zeros(self) -> None:
95     """Initialise weights to 0s and biases to 0s"""
96     self.weights = np.zeros(shape=(self.output_neuron_count,
97     ↪ self.input_neuron_count))
98     self.biases = np.zeros(shape=(self.output_neuron_count, 1))
99
100 def back_propagation(self, dloss_doutput) -> np.ndarray:
101     """Adjust the weights and biases via gradient descent.
102
103     Args:
104     ↪ dloss_doutput (numpy.ndarray): the derivative of the loss of the
105     ↪ layer's output, with respect to the layer's output.
106     Returns:
107         a numpy.ndarray derivative of the loss of the layer's input,
108         with respect to the layer's input.
109     Raises:
110         ValueError:
111         if dloss_doutput
112         is not a suitable multiplier with the weights
113         (incorrect shape)
114
115     """
116     match self.transfer_type:
117         case 'sigmoid':
118             dloss_dz = dloss_doutput *
119             ↪ sigmoid_derivative(output=self.output)
120         case 'relu':
121             dloss_dz = dloss_doutput *
122             ↪ relu_derivative(output=self.output)
123
124     dloss_dweights = np.dot(dloss_dz, self.input.T)
125     dloss_dbias = np.sum(dloss_dz)
126
127     assert dloss_dweights.shape == self.weights.shape
128
129     dloss_dinput = np.dot(self.weights.T, dloss_dz)
130
131     # Update weights and biases
132     self.weights -= self.learning_rate * dloss_dweights
133     self.biases -= self.learning_rate * dloss_dbias
134
135

```

```

130         return dloss_dinput
131
132     def forward_propagation(self, inputs) -> np.ndarray:
133         """Generate a layer output with the weights and biases.
134
135         Args:
136             inputs (np.ndarray): the input values to the layer.
137         Returns:
138             a numpy.ndarray of the output values.
139
140         """
141         self.input = inputs
142         z = np.dot(self.weights, self.input) + self.biases
143         if self.transfer_type == 'sigmoid':
144             self.output = sigmoid(z)
145         elif self.transfer_type == 'relu':
146             self.output = relu(z)
147         return self.output
148
149     class AbstractModel(ModelInterface):
150         """ANN model with variable number of hidden layers"""
151         def __init__(self,
152                     hidden_layers_shape: list[int],
153                     train_dataset_size: int,
154                     learning_rate: float,
155                     use_relu: bool) -> None:
156             """Initialise model values.
157
158             Args:
159                 hidden_layers_shape (list[int]):
160                     list of the number of neurons in each hidden layer.
161                 train_dataset_size (int): the number of train dataset inputs to
162                 ↪ use.
163                     learning_rate (float): the learning rate of the model.
164                 ↪ use_relu (bool): True or False whether the ReLu Transfer
165                     function
166                     should be used.
167
168             """
169             # Setup model data
170             self.train_inputs, self.train_outputs, \
171             self.test_inputs, self.test_outputs = self.load_datasets(
172                 ↪ train_dataset_size=train_dataset_size
173             )
174             self.train_losses: list[float]
175             self.test_prediction: np.ndarray
176             self.test_prediction_accuracy: float
177             self.training_progress = ""
178             self.training_time: float
179
180             # Setup model attributes
181             self._running = True
182             self.input_neuron_count: int = self.train_inputs.shape[0]
183             self.input_count = self.train_inputs.shape[1]
184             self.hidden_layers_shape = hidden_layers_shape
185             self.output_neuron_count = self.train_outputs.shape[0]
186             self.layers_shape = [f'{layer}' for layer in (
187                 [self.input_neuron_count] +
188                 self.hidden_layers_shape +
189                 [self.output_neuron_count]
190             )]

```

```

189         self.use_relu = use_relu
190
191         # Setup model values
192         self.layers = _Layers()
193         self.learning_rate = learning_rate
194
195     def __repr__(self) -> str:
196         """Read current state of model.
197
198         Returns:
199             a string description of the model's shape,
200             weights, bias and learning rate values.
201
202         """
203         return (f"Layers Shape: {'.'.join(self.layers_shape)}\n" +
204                 f"Learning Rate: {self.learning_rate}")
205
206     def set_running(self, value:bool):
207         self.__running = value
208
209     def _setup_layers(setup_values: callable) -> None:
210         """Setup model layers"""
211         def decorator(self, *args, **kwargs):
212             # Check if setting up Deep Network
213             if len(self.hidden_layers_shape) > 0:
214                 if self.use_relu:
215
216                     # Add input layer
217                     self.layers.head = _FullyConnectedLayer(
218
219                         ↪ learning_rate=self.learning_rate,
220
221                         ↪ input_neuron_count=self.input_neuron_count,
222
223                         ↪ output_neuron_count=self.hidden_layers_shape[0],
224                         transfer_type='relu'
225                     )
226                     current_layer = self.layers.head
227
228                     # Add hidden layers
229                     for layer in range(len(self.hidden_layers_shape) - 1):
230                         current_layer.next_layer = _FullyConnectedLayer(
231                             learning_rate=self.learning_rate,
232
233                             ↪ input_neuron_count=self.hidden_layers_shape[layer],
234
235                             ↪ output_neuron_count=self.hidden_layers_shape[layer
236                             ↪ + 1],
237                             transfer_type='relu'
238                         )
239                         current_layer.next_layer.previous_layer =
240                             ↪ current_layer
241                         current_layer = current_layer.next_layer
242
243             else:
244
245                 # Add input layer
246                 self.layers.head = _FullyConnectedLayer(
247
248                     ↪ learning_rate=self.learning_rate,
249
250                     ↪ input_neuron_count=self.input_neuron_count,
251
252                     ↪ output_neuron_count=self.hidden_layers_shape[0],

```

```

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

```



```

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

```

```

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

```

3.2.2 Artificial Neural Network implementations

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

- cat_recognition.py module:

```

1  """Implementation of Artificial Neural Network model on Cat Recognition
2  ↪ dataset."""
3
4  import h5py
5  import numpy as np
6
7  from .utils.model import AbstractModel
8
9  class CatRecognitionModel(AbstractModel):

```

```

9      """ANN model that trains to predict if an image is a cat or not a
↪      cat."""
10     def __init__(self,
11                   hidden_layers_shape: list[int],
12                   train_dataset_size: int,
13                   learning_rate: float,
14                   use_relu: bool) -> None:
15         """Initialise Model's Base class.
16
17         Args:
18             hidden_layers_shape (list[int]):
19                 list of the number of neurons in each hidden layer.
20             train_dataset_size (int): the number of train dataset inputs to
↪     use.
21             learning_rate (float): the learning rate of the model.
22             use_relu (bool): True or False whether the ReLu Transfer
↪     function
23                 should be used.
24
25         """
26         super().__init__(hidden_layers_shape=hidden_layers_shape,
27                           train_dataset_size=train_dataset_size,
28                           learning_rate=learning_rate,
29                           use_relu=use_relu)
30
31     def load_datasets(self, train_dataset_size: int) -> tuple[np.ndarray,
↪     np.ndarray,
32                                                                np.ndarray,
33                                                                ↪     np.ndarray]:
34
35         """Load image input and output datasets.
36
37         Args:
38             train_dataset_size (int): the number of train dataset inputs to
↪     use.
39         Returns:
40             tuple of image train_inputs, train_outputs,
41             test_inputs and test_outputs numpy.ndarrays.
42
43         Raises:
44             FileNotFoundError: if file does not exist.
45
46         """
47         # Load datasets from h5 files
48         # (h5 files stores large amount of data with quick access)
49         train_dataset: h5py.File = h5py.File(
50             r'school_project/models/datasets/train-cat.h5',
51             'r'
52         )
53         test_dataset: h5py.File = h5py.File(
54             r'school_project/models/datasets/test-cat.h5',
55             'r'
56         )
57
58         # Load input arrays,
59         # containing the RGB values for each pixel in each 64x64 pixel
60         ↪     image,
61         # for 209 images
62         train_inputs: np.ndarray =
        ↪     np.array(train_dataset['train_set_x'][:])
63         test_inputs: np.ndarray = np.array(test_dataset['test_set_x'][:])
64
65         # Load output arrays of 1s for cat and 0s for not cat

```

```

63     train_outputs: np.ndarray =
        ↪ np.array(train_dataset['train_set_y'][:])
64     test_outputs: np.ndarray = np.array(test_dataset['test_set_y'][:])
65
66     # Reshape input arrays into 1 dimension (flatten),
67     # then divide by 255 (RGB)
68     # to standardize them to a number between 0 and 1
69     train_inputs = train_inputs.reshape((train_inputs.shape[0],
70                                         -1)).T / 255
71     test_inputs = test_inputs.reshape((test_inputs.shape[0], -1)).T /
        ↪ 255
72
73     # Reshape output arrays into a 1 dimensional list of outputs
74     train_outputs = train_outputs.reshape((1, train_outputs.shape[0]))
75     test_outputs = test_outputs.reshape((1, test_outputs.shape[0]))
76
77     # Reduce train datasets' sizes to train_dataset_size
78     train_inputs = (train_inputs.T[:train_dataset_size]).T
79     train_outputs = (train_outputs.T[:train_dataset_size]).T
80
81     return train_inputs, train_outputs, test_inputs, test_outputs

```

- mnist.py module:

```

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

```

```

34     """Load image input and output datasets.
35     Args:
36         train_dataset_size (int): the number of dataset inputs to use.
37     Returns:
38         tuple of image train_inputs, train_outputs,
39         test_inputs and test_outputs numpy.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     """Implementation of Artificial Neural Network model on XOR dataset."""
2
3     import numpy as np
4
5     from .utils.model import AbstractModel
6
7     class XORModel(AbstractModel):
8         """ANN model that trains to predict the output of a XOR gate with two
9         inputs."""
10        def __init__(self,
11                    hidden_layers_shape: list[int],
12                    train_dataset_size: int,
13                    learning_rate: float,
14                    use_relu: bool) -> None:
15            """Initialise Model's Base class.
16
17            Args:
18                hidden_layers_shape (list[int]):
19                list of the number of neurons in each hidden layer.

```

```

20         train_dataset_size (int): the number of train dataset inputs to
↪ use.
21         learning_rate (float): the learning rate of the model.
22         use_relu (bool): True or False whether the ReLu Transfer
↪ function
23         should be used.
24
25         """
26         super().__init__(hidden_layers_shape=hidden_layers_shape,
27                           train_dataset_size=train_dataset_size,
28                           learning_rate=learning_rate,
29                           use_relu=use_relu)
30
31     def load_datasets(self, train_dataset_size: int) -> tuple[np.ndarray,
↪ np.ndarray,
32                                                                np.ndarray,
↪ np.ndarray]:
33
34         """Load XOR input and output datasets.
35
36         Args:
37             train_dataset_size (int): the number of dataset inputs to use.
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
↪ images.",
4          "epochCount": 150,
5          "hiddenLayersShape": [1000, 1000],
6          "minTrainDatasetSize": 1,
7          "maxTrainDatasetSize": 60000,
8          "maxLearningRate": 1
9      },
10     "Cat Recognition": {
11         "description": "An Image model trained on recognising if an image
↪ is a cat or not.",
12         "epochCount": 3500,

```

```

13         "hiddenLayersShape": [100, 100],
14         "minTrainDatasetSize": 1,
15         "maxTrainDatasetSize": 209,
16         "maxLearningRate": 0.3
17     },
18     "XOR": {
19         "description": "For experimenting with Artificial Neural Networks,
↪ a XOR gate model has been used for its lesser computation time.",
20         "epochCount": 4700,
21         "hiddenLayersShape": [100, 100],
22         "minTrainDatasetSize": 2,
23         "maxTrainDatasetSize": 4,
24         "maxLearningRate": 1
25     }
26 }

```

- create_model.py module:

```

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

```

```

41
42 # Setup widgets
43 self.title_label = tk.Label(master=self,
44                             bg=self.BG,
45                             font=('Arial', 20),
46                             text=dataset)
47 self.about_label = tk.Label(
48     master=self,
49     bg=self.BG,
50     font=('Arial', 14),
51
52     ↪ text=self.default_hyper_parameters['description']
53 )
54 self.learning_rate_scale = tk.Scale(
55     master=self,
56     bg=self.BG,
57     orient='horizontal',
58     label="Learning Rate",
59     length=185,
60     from_=0,
61
62     ↪ to=self.default_hyper_parameters['maxLearningRate'],
63     resolution=0.01
64 )
65 self.learning_rate_scale.set(value=0.1)
66 self.epoch_count_scale = tk.Scale(master=self,
67     bg=self.BG,
68     orient='horizontal',
69     label="Epoch Count",
70     length=185,
71     from_=0,
72     to=10_000,
73     resolution=100)
74 self.epoch_count_scale.set(
75
76     ↪ value=self.default_hyper_parameters['epochCount']
77 )
78 self.train_dataset_size_scale = tk.Scale(
79     master=self,
80     bg=self.BG,
81     orient='horizontal',
82     label="Train Dataset Size",
83     length=185,
84
85     ↪ from_=self.default_hyper_parameters['minTrainDatasetSize'],
86     to=self.default_hyper_parameters['maxTrainDatasetSize'],
87     resolution=1
88 )
89 self.train_dataset_size_scale.set(
90
91     ↪ value=self.default_hyper_parameters['maxTrainDatasetSize']
92 )
93 self.hidden_layers_shape_label = tk.Label(
94     master=self,
95     bg=self.BG,
96     font=('Arial', 12),
97     text="Enter the number of neurons in
98     ↪ each\n" +
99         "hidden layer, separated by
100     ↪ commas:"
101 )
102 self.hidden_layers_shape_entry = tk.Entry(master=self)

```



```

96     self.hidden_layers_shape_entry.insert(0, ".".join(
97         f"{neuron_count}" for neuron_count in
98         ↪ self.default_hyper_parameters['hiddenLayersShape']
99     ))
100     self.use_relu_check_button_var = tk.BooleanVar(value=True)
101     self.use_relu_check_button = tk.Checkbutton(
102         master=self,
103         width=13, height=1,
104         font=tkf.Font(size=12),
105         text="Use ReLu",
106         ↪ variable=self.use_relu_check_button_var
107     )
108     self.use_gpu_check_button_var = tk.BooleanVar()
109     self.use_gpu_check_button = tk.Checkbutton(
110         master=self,
111         width=13, height=1,
112         font=tkf.Font(size=12),
113         text="Use GPU",
114         ↪ variable=self.use_gpu_check_button_var
115     )
116     self.model_status_label = tk.Label(master=self,
117         bg=self.BG,
118         font=('Arial', 15))
119
120     # Pack widgets
121     self.title_label.grid(row=0, column=0, columnspan=3)
122     self.about_label.grid(row=1, column=0, columnspan=3)
123     self.learning_rate_scale.grid(row=2, column=0, pady=(50,0))
124     self.epoch_count_scale.grid(row=3, column=0, pady=(30,0))
125     self.train_dataset_size_scale.grid(row=4, column=0, pady=(30,0))
126     self.hidden_layers_shape_label.grid(row=2, column=1,
127         padx=30, pady=(50,0))
128     self.hidden_layers_shape_entry.grid(row=3, column=1, padx=30)
129     self.use_relu_check_button.grid(row=2, column=2, pady=(30, 0))
130     self.use_gpu_check_button.grid(row=3, column=2, pady=(30, 0))
131     self.model_status_label.grid(row=5, column=0,
132         columnspan=3, pady=50)
133
134     def load_default_hyper_parameters(self, dataset: str) -> dict[
135         str,
136         str | int | list[int] |
137         ↪ float
138     ]:
139         """Load the dataset's default hyper-parameters from the json file.
140
141         Args:
142             dataset (str): the name of the dataset to load
143             ↪ hyper-parameters
144             for. ('MNIST', 'Cat Recognition' or 'XOR')
145         Returns:
146             a dictionary of default hyper-parameter values.
147         """
148         with open('school_project/frames/hyper-parameter-defaults.json') as
149             ↪ f:
150             return json.load(f)[dataset]
151
152     def create_model(self) -> object:
153         """Create and return a Model using the hyper-parameters set.
154
155         Returns:

```

```

152         a Model object.
153         """
154         self.use_gpu = self.use_gpu_check_button_var.get()
155
156         # Validate hidden layers shape input
157         hidden_layers_shape_input = [layer for layer in
158             ↪ self.hidden_layers_shape_entry.get().replace(' ',
159             ↪ ' ').split(',') if layer != '']
160         for layer in hidden_layers_shape_input:
161             if not layer.isdigit():
162                 self.model_status_label.configure(
163                     text="Invalid hidden layers shape",
164                     fg='red'
165                 )
166                 raise ValueError
167
168         # Create Model
169         if not self.use_gpu:
170             if self.dataset == "MNIST":
171                 from school_project.models.cpu.mnist import MNISTModel as
172                 ↪ Model
173             elif self.dataset == "Cat Recognition":
174                 from school_project.models.cpu.cat_recognition import
175                 ↪ CatRecognitionModel as Model
176             elif self.dataset == "XOR":
177                 from school_project.models.cpu.xor import XORModel as Model
178             model = Model(hidden_layers_shape = [int(neuron_count) for
179                 ↪ neuron_count in hidden_layers_shape_input],
180                 train_dataset_size =
181                 ↪ self.train_dataset_size_scale.get(),
182                 learning_rate = self.learning_rate_scale.get(),
183                 use_relu = self.use_relu_check_button_var.get())
184             model.create_model_values()
185         else:
186             try:
187                 if self.dataset == "MNIST":
188                     from school_project.models.gpu.mnist import MNISTModel
189                     ↪ as Model
190                 elif self.dataset == "Cat Recognition":
191                     from school_project.models.gpu.cat_recognition import
192                     ↪ CatRecognitionModel as Model
193                 elif self.dataset == "XOR":
194                     from school_project.models.gpu.xor import XORModel as
195                     ↪ Model
196                 model = Model(hidden_layers_shape = [int(neuron_count) for
197                     ↪ neuron_count in hidden_layers_shape_input],
198                     train_dataset_size =
199                     ↪ self.train_dataset_size_scale.get(),
200                     learning_rate =
201                     ↪ self.learning_rate_scale.get(),
202                     use_relu =
203                     ↪ self.use_relu_check_button_var.get())
204                 model.create_model_values()
205             except ImportError as ie:
206                 self.model_status_label.configure(
207                     text="Failed to initialise GPU",
208                     fg='red'
209                 )
210                 raise ImportError
211         return model

```

```

201 class TrainingFrame(tk.Frame):
202     """Frame for training page."""
203     def __init__(self, root: tk.Tk, width: int,
204                 height: int, bg: str,
205                 model: object, epoch_count: int) -> None:
206         """Initialise training frame widgets.
207
208         Args:
209             root (tk.Tk): the widget object that contains this widget.
210             width (int): the pixel width of the frame.
211             height (int): the pixel height of the frame.
212             bg (str): the hex value or name of the frame's background
213     ↪ colour.
214             model (object): the Model object to be trained.
215             epoch_count (int): the number of training epochs.
216         Raises:
217             TypeError: if root, width or height are not of the correct
218     ↪ type.
219
220         """
221         super().__init__(master=root, width=width, height=height, bg=bg)
222         self.root = root
223         self.WIDTH = width
224         self.HEIGHT = height
225         self.BG = bg
226
227         # Setup widgets
228         self.model_status_label = tk.Label(master=self,
229                                           bg=self.BG,
230                                           font=('Arial', 15))
231         self.training_progress_label = tk.Label(master=self,
232                                                bg=self.BG,
233                                                font=('Arial', 15))
234
235         self.loss_figure: Figure = Figure()
236         self.loss_canvas: FigureCanvasTkAgg = FigureCanvasTkAgg(
237             ↪ figure=self.loss_figure,
238             master=self
239         )
240
241         # Pack widgets
242         self.model_status_label.pack(pady=(30,0))
243         self.training_progress_label.pack(pady=30)
244
245         # Start training thread
246         self.model_status_label.configure(
247             text="Training weights and
248             ↪ biases...",
249             fg='red'
250         )
251         self.train_thread: threading.Thread = threading.Thread(
252             ↪ target=model.train,
253             ↪ args=(epoch_count,)
254         )
255         self.train_thread.start()
256
257     def plot_losses(self, model: object) -> None:
258         """Plot losses of Model training.
259
260         Args:

```

```

257         model (object): the Model object thats been trained.
258
259         """
260         self.model_status_label.configure(
261             text=f"Weights and biases trained in
↪      {model.training_time}s",
262             fg='green'
263         )
264         graph: Figure.axes = self.loss_figure.add_subplot(111)
265         graph.set_title("Learning rate: " +
266                        f"{model.learning_rate}")
267         graph.set_xlabel("Epochs")
268         graph.set_ylabel("Loss Value")
269         graph.plot(np.squeeze(model.train_losses))
270         self.loss_canvas.get_tk_widget().pack()

```

- load_model.py module:

```

1  """Tkinter frames for loading a saved Artificial Neural Network Model."""
2
3  import sqlite3
4  import tkinter as tk
5  import tkinter.font as tkf
6
7  class LoadModelFrame(tk.Frame):
8      """Frame for load model page."""
9      def __init__(self, root: tk.Tk,
10                  width: int, height: int,
11                  bg: str, connection: sqlite3.Connection,
12                  cursor: sqlite3.Cursor, dataset: str) -> None:
13          """Initialise load model frame widgets.
14
15          Args:
16              root (tk.Tk): the widget object that contains this widget.
17              width (int): the pixel width of the frame.
18              height (int): the pixel height of the frame.
19              bg (str): the hex value or name of the frame's background
↪      colour.
20              connection (sqlite3.Connection): the database connection
↪      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
↪      type.
26
27          """
28          super().__init__(master=root, width=width, height=height, bg=bg)
29          self.root = root
30          self.WIDTH = width
31          self.HEIGHT = height
32          self.BG = bg
33
34          # Setup load model frame variables
35          self.connection = connection
36          self.cursor = cursor
37          self.dataset = dataset
38          self.use_gpu: bool
39          self.model_options = self.load_model_options()
40

```

```

41     # Setup widgets
42     self.title_label = tk.Label(master=self,
43                                bg=self.BG,
44                                font=('Arial', 20),
45                                text=dataset)
46     self.about_label = tk.Label(
47         master=self,
48         bg=self.BG,
49         font=('Arial', 14),
50         text=f"Load a pretrained model for the {dataset}"
51         ↪ dataset."
52     )
53     self.model_status_label = tk.Label(master=self,
54                                        bg=self.BG,
55                                        font=('Arial', 15))
56     # Don't give loaded model options if no models have been saved for
57     ↪ the
58     # dataset.
59     if len(self.model_options) > 0:
60         self.model_option_menu_label = tk.Label(
61             master=self,
62             bg=self.BG,
63             font=('Arial', 14),
64             text="Select a model to
65             ↪ load or delete:"
66         )
67         self.model_option_menu_var = tk.StringVar(
68             master=self,
69             ↪ value=self.model_options[0]
70         )
71         self.model_option_menu = tk.OptionMenu(
72             self,
73             ↪ self.model_option_menu_var,
74             *self.model_options
75         )
76         self.use_gpu_check_button_var = tk.BooleanVar()
77         self.use_gpu_check_button = tk.Checkbutton(
78             master=self,
79             width=7, height=1,
80             font=tkf.Font(size=12),
81             text="Use GPU",
82             ↪ variable=self.use_gpu_check_button_var
83         )
84     else:
85         self.model_status_label.configure(
86             text='No saved models for this
87             ↪ dataset.',
88             fg='red'
89         )
90     # Pack widgets
91     self.title_label.grid(row=0, column=0, columnspan=3)
92     self.about_label.grid(row=1, column=0, columnspan=3)
93     if len(self.model_options) > 0: # Check if options should be given
94         self.model_option_menu_label.grid(row=2, column=0, padx=(0,30),
95             ↪ pady=(30,0))
96         self.use_gpu_check_button.grid(row=2, column=2, rowspan=2,
97             ↪ pady=(30,0))

```

```

94         self.model_option_menu.grid(row=3, column=0, padx=(0,30),
95         ↪ pady=(10,0))
96     self.model_status_label.grid(row=4, column=0,
97         ↪ colspan=3, pady=50)
98
99     def load_model_options(self) -> list[str]:
100         """Load the model options from the database.
101
102         Returns:
103             a list of the model options.
104         """
105         sql = f"""
106         SELECT Name FROM Models WHERE Dataset=?
107         """
108         parameters = (self.dataset.replace(" ", "_"),)
109         self.cursor.execute(sql, parameters)
110
111         # Save the string value contained within the tuple of each row
112         model_options = []
113         for model_option in self.cursor.fetchall():
114             model_options.append(model_option[0])
115
116         return model_options
117
118     def load_model(self) -> object:
119         """Create model using saved weights and biases.
120
121         Returns:
122             a Model object.
123         """
124         self.use_gpu = self.use_gpu_check_button_var.get()
125
126         # Query data of selected saved model from database
127         sql = """
128         SELECT * FROM Models WHERE Dataset=? AND Name=?
129         """
130         parameters = (self.dataset.replace(" ", "_"),
131         ↪ self.model_option_menu_var.get())
132         self.cursor.execute(sql, parameters)
133         data = self.cursor.fetchone()
134         hidden_layers_shape_input = [layer for layer in data[3].replace('
135         ↪ ', '').split(',') if layer != '']
136
137         # Create Model
138         if not self.use_gpu:
139             if self.dataset == "MNIST":
140                 from school_project.models.cpu.mnist import MNISTModel as
141                 ↪ Model
142             elif self.dataset == "Cat Recognition":
143                 from school_project.models.cpu.cat_recognition import
144                 ↪ CatRecognitionModel as Model
145             elif self.dataset == "XOR":
146                 from school_project.models.cpu.xor import XORModel as Model
147
148         model = Model(
149             hidden_layers_shape=[int(neuron_count) for neuron_count in
150             ↪ hidden_layers_shape_input],
151             train_dataset_size=data[6],
152             learning_rate=data[4],
153             use_relu=data[7]
154         )
155         model.load_model_values(file_location=data[2])

```

```

150
151         else:
152             try:
153                 if self.dataset == "MNIST":
154                     from school_project.models.gpu.mnist import MNISTModel
155                     ↪ as Model
156                 elif self.dataset == "Cat Recognition":
157                     from school_project.models.gpu.cat_recognition import
158                     ↪ CatRecognitionModel as Model
159                 elif self.dataset == "XOR":
160                     from school_project.models.gpu.xor import XORModel as
161                     ↪ Model
162                 model = Model(
163                     hidden_layers_shape=[int(neuron_count) for neuron_count
164                     ↪ in hidden_layers_shape_input],
165                     train_dataset_size=data[6],
166                     learning_rate=data[4],
167                     use_relu=data[7]
168                 )
169                 model.load_model_values(file_location=data[2])
170             except ImportError as ie:
171                 self.model_status_label.configure(
172                     text="Failed to initialise
173                     ↪ GPU",
174                     fg='red'
175                 )
176                 raise ImportError
177         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  """The entrypoint of A-level Computer Science NEA Programming Project."""
2
3  import os
4  import sqlite3
5  import threading
6  import tkinter as tk
7  import tkinter.font as tkf
8  import uuid
9
10 import pympiler.tracker as tracker
11
12 from school_project.frames import (HyperParameterFrame, TrainingFrame,
13                                  LoadModelFrame, TestMNISTFrame,
14                                  TestCatRecognitionFrame, TestXORFrame)
15
16 class SchoolProjectFrame(tk.Frame):
17     """Main frame of school project."""
18     def __init__(self, root: tk.Tk, width: int, height: int, bg: str) -> None:
19         """Initialise school project pages.
20
21         Args:
22             root (tk.Tk): the widget object that contains this widget.
23             width (int): the pixel width of the frame.
24             height (int): the pixel height of the frame.
25             bg (str): the hex value or name of the frame's background colour.
26         Raises:

```

```

27         TypeError: if root, width or height are not of the correct type.
28
29     """
30     super().__init__(master=root, width=width, height=height, bg=bg)
31     self.root = root.title("School Project")
32     self.WIDTH = width
33     self.HEIGHT = height
34     self.BG = bg
35
36     # Setup school project frame variables
37     self.hyper_parameter_frame: HyperParameterFrame
38     self.training_frame: TrainingFrame
39     self.load_model_frame: LoadModelFrame
40     self.test_frame: TestMNISTFrame | TestCatRecognitionFrame | TestXORFrame
41     self.connection, self.cursor = self.setup_database()
42     self.model = None
43
44     # Record if the model should be saved after testing,
45     # as only newly created models should be given the option to be saved.
46     self.saving_model: bool
47
48     # Setup school project frame widgets
49     self.exit_hyper_parameter_frame_button = tk.Button(
50         master=self,
51         width=13,
52         height=1,
53         font=tkf.Font(size=12),
54         text="Exit",
55         command=self.exit_hyper_parameter_frame
56     )
57     self.exit_load_model_frame_button = tk.Button(
58         master=self,
59         width=13,
60         height=1,
61         font=tkf.Font(size=12),
62         text="Exit",
63         command=self.exit_load_model_frame
64     )
65     self.train_button = tk.Button(master=self,
66         width=13,
67         height=1,
68         font=tkf.Font(size=12),
69         text="Train Model",
70         command=self.enter_training_frame)
71     self.stop_training_button = tk.Button(
72         master=self,
73         width=15, height=1,
74         font=tkf.Font(size=12),
75         text="Stop Training Model",
76         command=lambda: self.model.set_running(
77             value=False
78         )
79     )
80     self.test_created_model_button = tk.Button(
81         master=self,
82         width=13, height=1,
83         font=tkf.Font(size=12),
84         text="Test Model",
85         command=self.test_created_model
86     )
87     self.test_loaded_model_button = tk.Button(
88         master=self,

```



```

89         width=13, height=1,
90         font=tkf.Font(size=12),
91         text="Test Model",
92         command=self.test_loaded_model
93     )
94     self.delete_loaded_model_button = tk.Button(
95         master=self,
96         width=13, height=1,
97         font=tkf.Font(size=12),
98         text="Delete Model",
99         command=self.delete_loaded_model
100    )
101    self.save_model_label = tk.Label(
102        master=self,
103        text="Enter a name for your trained model:",
104        bg=self.BG,
105        font=('Arial', 15)
106    )
107    self.save_model_name_entry = tk.Entry(master=self, width=13)
108    self.save_model_button = tk.Button(master=self,
109        width=13,
110        height=1,
111        font=tkf.Font(size=12),
112        text="Save Model",
113        command=self.save_model)
114    self.exit_button = tk.Button(master=self,
115        width=13, height=1,
116        font=tkf.Font(size=12),
117        text="Exit",
118        command=self.enter_home_frame)
119
120    # Setup home frame
121    self.home_frame = tk.Frame(master=self,
122        width=self.WIDTH,
123        height=self.HEIGHT,
124        bg=self.BG)
125    self.title_label = tk.Label(
126        master=self.home_frame,
127        bg=self.BG,
128        font=('Arial', 20),
129        text="A-level Computer Science NEA Programming Project"
130    )
131    self.about_label = tk.Label(
132        master=self.home_frame,
133        bg=self.BG,
134        font=('Arial', 14),
135        text="An investigation into how Artificial Neural Networks work, " +
136        "the effects of their hyper-parameters and their applications " +
137        "in Image Recognition.\n\n" +
138        " - Max Cotton"
139    )
140    self.model_menu_label = tk.Label(master=self.home_frame,
141        bg=self.BG,
142        font=('Arial', 14),
143        text="Create a new model " +
144        "or load a pre-trained model "
145        "for one of the following datasets:")
146    self.dataset_option_menu_var = tk.StringVar(master=self.home_frame,
147        value="MNIST")
148    self.dataset_option_menu = tk.OptionMenu(self.home_frame,
149        self.dataset_option_menu_var,
150        "MNIST",

```

```

151                                     "Cat Recognition",
152                                     "XOR")
153 self.create_model_button = tk.Button(
154     master=self.home_frame,
155     width=13, height=1,
156     font=tkf.Font(size=12),
157     text="Create Model",
158     command=self.enter_hyper_parameter_frame
159 )
160 self.load_model_button = tk.Button(master=self.home_frame,
161     width=13, height=1,
162     font=tkf.Font(size=12),
163     text="Load Model",
164     command=self.enter_load_model_frame)
165
166 # Grid home frame widgets
167 self.title_label.grid(row=0, column=0, columnspan=4, pady=(10,0))
168 self.about_label.grid(row=1, column=0, columnspan=4, pady=(10,50))
169 self.model_menu_label.grid(row=2, column=0, columnspan=4)
170 self.dataset_option_menu.grid(row=3, column=0, columnspan=4, pady=30)
171 self.create_model_button.grid(row=4, column=1)
172 self.load_model_button.grid(row=4, column=2)
173
174 self.home_frame.pack()
175
176 # Setup frame attributes
177 self.grid_propagate(flag=False)
178 self.pack_propagate(flag=False)
179
180 @staticmethod
181 def setup_database() -> tuple[sqlite3.Connection, sqlite3.Cursor]:
182     """Create a connection to the pretrained_models database file and
183     setup base table if needed.
184
185     Returns:
186         a tuple of the database connection and the cursor for it.
187
188     """
189     connection = sqlite3.connect(
190         database='school_project/saved_models.db'
191     )
192     cursor = connection.cursor()
193     cursor.execute("""
194     CREATE TABLE IF NOT EXISTS Models
195     (Model_ID INTEGER PRIMARY KEY,
196     Dataset TEXT,
197     File_Location TEXT,
198     Hidden_Layers_Shape TEXT,
199     Learning_Rate FLOAT,
200     Name TEXT,
201     Train_Dataset_Size INTEGER,
202     Use_ReLu INTEGER,
203     UNIQUE (Dataset, Name))
204     """)
205     return (connection, cursor)
206
207 def enter_hyper_parameter_frame(self) -> None:
208     """Unpack home frame and pack hyper-parameter frame."""
209     self.home_frame.pack_forget()
210     self.hyper_parameter_frame = HyperParameterFrame(
211         root=self,
212         width=self.WIDTH,

```

```

213             height=self.HEIGHT,
214             bg=self.BG,
215             dataset=self.dataset_option_menu_var.get()
216         )
217     self.hyper_parameter_frame.pack()
218     self.train_button.pack()
219     self.exit_hyper_parameter_frame_button.pack(pady=(10,0))
220
221     def enter_load_model_frame(self) -> None:
222         """Unpack home frame and pack load model frame."""
223         self.home_frame.pack_forget()
224         self.load_model_frame = LoadModelFrame(
225             root=self,
226             width=self.WIDTH,
227             height=self.HEIGHT,
228             bg=self.BG,
229             connection=self.connection,
230             cursor=self.cursor,
231             dataset=self.dataset_option_menu_var.get()
232         )
233         self.load_model_frame.pack()
234
235         # Don't give option to test loaded model if no models have been saved
236         # for the dataset.
237         if len(self.load_model_frame.model_options) > 0:
238             self.test_loaded_model_button.pack()
239             self.delete_loaded_model_button.pack(pady=(5,0))
240
241         self.exit_load_model_frame_button.pack(pady=(5,0))
242
243     def exit_hyper_parameter_frame(self) -> None:
244         """Unpack hyper-parameter frame and pack home frame."""
245         self.hyper_parameter_frame.pack_forget()
246         self.train_button.pack_forget()
247         self.exit_hyper_parameter_frame_button.pack_forget()
248         self.home_frame.pack()
249
250     def exit_load_model_frame(self) -> None:
251         """Unpack load model frame and pack home frame."""
252         self.load_model_frame.pack_forget()
253         self.test_loaded_model_button.pack_forget()
254         self.delete_loaded_model_button.pack_forget()
255         self.exit_load_model_frame_button.pack_forget()
256         self.home_frame.pack()
257
258     def enter_training_frame(self) -> None:
259         """Load untrained model from hyper parameter frame,
260         unpack hyper-parameter frame, pack training frame
261         and begin managing the training thread.
262         """
263         try:
264             self.model = self.hyper_parameter_frame.create_model()
265         except (ValueError, ImportError) as e:
266             return
267         self.hyper_parameter_frame.pack_forget()
268         self.train_button.pack_forget()
269         self.exit_hyper_parameter_frame_button.pack_forget()
270         self.training_frame = TrainingFrame(
271             root=self,
272             width=self.WIDTH,
273             height=self.HEIGHT,
274             bg=self.BG,

```

```

275         model=self.model,
276         epoch_count=self.hyper_parameter_frame.epoch_count_scale.get()
277     )
278     self.training_frame.pack()
279     self.stop_training_button.pack()
280     self.manage_training(train_thread=self.training_frame.train_thread)
281
282 def manage_training(self, train_thread: threading.Thread) -> None:
283     """Wait for model training thread to finish,
284     then plot training losses on training frame.
285
286     Args:
287         train_thread (threading.Thread):
288             the thread running the model's train() method.
289
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
336     self.test_frame.pack()
337     self.manage_testing(test_thread=self.test_frame.test_thread)

```

```

337 def test_loaded_model(self) -> None:
338     """Load saved model from load model frame, unpack load model frame,
339     pack test frame for the dataset and begin managing the test thread."""
340     self.saving_model = False
341     try:
342         self.model = self.load_model_frame.load_model()
343     except (ValueError, ImportError) as e:
344         return
345     self.load_model_frame.pack_forget()
346     self.test_loaded_model_button.pack_forget()
347     self.delete_loaded_model_button.pack_forget()
348     self.exit_load_model_frame_button.pack_forget()
349     if self.load_model_frame.dataset == "MNIST":
350         self.test_frame = TestMNISTFrame(
351             root=self,
352             width=self.WIDTH,
353             height=self.HEIGHT,
354             bg=self.BG,
355             use_gpu=self.load_model_frame.use_gpu,
356             model=self.model
357         )
358     elif self.load_model_frame.dataset == "Cat Recognition":
359         self.test_frame = TestCatRecognitionFrame(
360             root=self,
361             width=self.WIDTH,
362             height=self.HEIGHT,
363             bg=self.BG,
364             use_gpu=self.load_model_frame.use_gpu,
365             model=self.model
366         )
367     elif self.load_model_frame.dataset == "XOR":
368         self.test_frame = TestXORFrame(root=self,
369             width=self.WIDTH,
370             height=self.HEIGHT,
371             bg=self.BG,
372             model=self.model)
373     self.test_frame.pack()
374     self.manage_testing(test_thread=self.test_frame.test_thread)
375
376 def manage_testing(self, test_thread: threading.Thread) -> None:
377     """Wait for model test thread to finish,
378     then plot results on test frame.
379
380     Args:
381         test_thread (threading.Thread):
382             the thread running the model's predict() method.
383     Raises:
384         TypeError: if test_thread is not of type threading.Thread.
385
386     """
387     if not test_thread.is_alive():
388         self.test_frame.plot_results(model=self.model)
389         if self.saving_model:
390             self.save_model_label.pack(pady=(30,0))
391             self.save_model_name_entry.pack(pady=10)
392             self.save_model_button.pack()
393             self.exit_button.pack(pady=(20,0))
394         else:
395             self.after(1_000, self.manage_testing, test_thread)
396
397 def save_model(self) -> None:
398     """Save the model, save the model information to the database, then

```

```

399         enter the home frame."""
400     model_name = self.save_model_name_entry.get()
401
402     # Check if model name is empty
403     if model_name == '':
404         self.test_frame.model_status_label.configure(
405             text="Model name can not be blank",
406             fg='red'
407         )
408         return
409
410     # Check if model name has already been taken
411     dataset = self.dataset_option_menu_var.get().replace(" ", "_")
412     sql = """
413     SELECT Name FROM Models WHERE Dataset=?
414     """
415     parameters = (dataset,)
416     self.cursor.execute(sql, parameters)
417     for saved_model_name in self.cursor.fetchall():
418         if saved_model_name[0] == model_name:
419             self.test_frame.model_status_label.configure(
420                 text="Model name taken",
421                 fg='red'
422             )
423             return
424
425     # Save model to random hex file name
426     file_location = f"school_project/saved-models/{uuid.uuid4().hex}.npz"
427     self.model.save_model_values(file_location=file_location)
428
429     # Save the model information to the database
430     sql = """
431     INSERT INTO Models
432     (Dataset, File_Location, Hidden_Layers_Shape, Learning_Rate, Name,
↪ Train_Dataset_Size, Use_ReLu)
433     VALUES (?, ?, ?, ?, ?, ?, ?)
434     """
435     parameters = (
436         dataset,
437         file_location,
438         self.hyper_parameter_frame.hidden_layers_shape_entry.get(),
439         self.hyper_parameter_frame.learning_rate_scale.get(),
440         model_name,
441         self.hyper_parameter_frame.train_dataset_size_scale.get(),
442         self.hyper_parameter_frame.use_relu_check_button_var.get()
443     )
444     self.cursor.execute(sql, parameters)
445     self.connection.commit()
446
447     self.enter_home_frame()
448
449     def delete_loaded_model(self) -> None:
450         """Delete saved model file and model data from the database."""
451         dataset = self.dataset_option_menu_var.get().replace(" ", "_")
452         model_name = self.load_model_frame.model_option_menu_var.get()
453
454         # Delete saved model
455         sql = f"""SELECT File_Location FROM Models WHERE Dataset=? AND Name=?"""
456         parameters = (dataset, model_name)
457         self.cursor.execute(sql, parameters)
458         os.remove(self.cursor.fetchone()[0])
459

```

```

460         # Remove model data from database
461         sql = """DELETE FROM Models WHERE Dataset=? AND Name=?"""
462         parameters = (dataset, model_name)
463         self.cursor.execute(sql, parameters)
464         self.connection.commit()
465
466         # Reload load model frame with new options
467         self.exit_load_model_frame()
468         self.enter_load_model_frame()
469
470     def enter_home_frame(self) -> None:
471         """Unpack test frame and pack home frame."""
472         self.model = None # Free up trained Model from memory
473         self.test_frame.pack_forget()
474         if self.saving_model:
475             self.save_model_label.pack_forget()
476             self.save_model_name_entry.delete(0, tk.END) # Clear entry's text
477             self.save_model_name_entry.pack_forget()
478             self.save_model_button.pack_forget()
479         self.exit_button.pack_forget()
480         self.home_frame.pack()
481         summary_tracker.create_summary() # BUG: Object summary seems to reduce
482                                         # memory leak greatly
483
484     def main() -> None:
485         """Entrypoint of project."""
486         root = tk.Tk()
487         school_project_frame = SchoolProjectFrame(root=root, width=1280,
488                                                    height=835, bg='white')
489         school_project_frame.pack(side='top', fill='both', expand=True)
490         root.mainloop()
491
492         # Stop model training when GUI closes
493         if school_project_frame.model != None:
494             school_project_frame.model.set_running(value=False)
495
496     if __name__ == "__main__":
497         summary_tracker = tracker.SummaryTracker() # Setup object tracker
498         main()

```
