# Computer Science NEA Report

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

Max Cotton

## Contents

# 1 Analysis

## 1.1 About

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

## 1.2 Interview TODO

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:""
  A:""

## 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
  - Store trained weights with sqlite
- Develop a Graphical User Interface
  - Provide controls for hyper-parameters of models
  - Display and compare the results each model's predictions

## 1.4 Theory behind Artificial Neural Networks

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

### 1.4.1 Structure

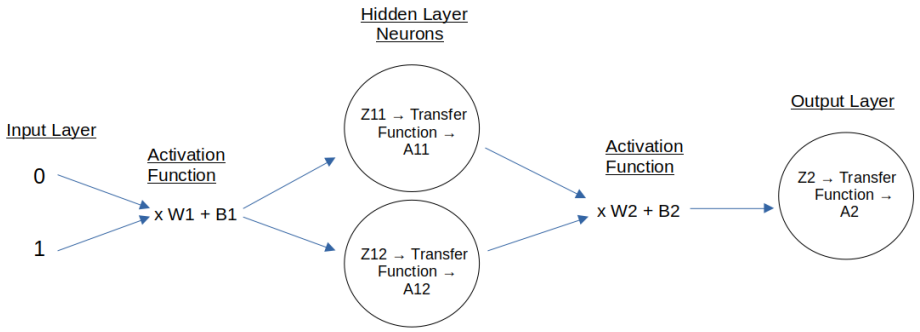



Figure 1: This shows an Artificial Neural Network with one single hidden layer and is known as a Shallow Neural Netwok.

I have focused on Feed-Forward Artificial Neural Networks, where values are entered to the input layer and passed forwards repetitively to the next layer until reaching the output layer. Within this, I have learnt two types of Feed-Forward Artificial Neural Networks: Perceptron Artificial Neural Networks, that contain no hidden layers and are best at learning more linear patterns and Multi-Layer Perceptron Artificial Neural Networks, that contain at least one hidden layer, as a result increasing the non-linearity in the Artificial Neural Network and allowing it to learn more complex / non-linear problems.

Multi-Layer Perceptron Artificial Neural Networks consist of:

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

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

- When Adding two matrices, both matrices must have the same number of rows and columns.

- 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 matrice with the column of the other, by multiplying matching members and then summing up.
- Transposing a matrix will turn all rows of the matrix into columns and all columns into rows.
- A matrix of values can be classified as a rank of Tensors, depending on the number of dimensions of the matrix. (Eg: A 2-dimensional matrix is a Tensor of rank 2)

I have focused on just using Fully-Connected layers, that will take in input values and apply the following calculations to produce an output of the layer:

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

### 1.4.2 How Artificial Neural Networks learn

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

- Forward Propagation:
  - The process of feeding inputs in and getting a prediction (moving forward through the network)
- Back Propagation:
  - The process of calculating the Loss in the prediction and then adjusting the weights and biases accordingly

- I have used Supervised Learning to train the Artificial Neural Networks, where the output prediction of the Artificial Neural Network is compared to the values it should have predicted. With this, I can calculate the Loss value of the prediction (how wrong the prediction is from the actual value).
- I then move back through the network and update the weights and biases via Gradient Descent:
  * Gradient Descent aims to reduce the Loss value of the prediction to a minimum, by subtracting the rate of change of Loss with respect to the weights/ biases, multiplied with a learning rate, from the weights/biases.
  * To calculate the rate of change of Loss with respect to the weights/biases, you must use the following calculus methods:
    · Partial Differentiation, in order to differentiate the multivariable functions, by taking respect to one variable and treating the rest as constants.
    · The Chain Rule, where for $y = f(u)$ and $u = g(x)$, $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} * \frac{\partial u}{\partial x}$
    · For a matrice of $f(x)$ values, the matrice of $\frac{\partial f(x)}{\partial x}$ values is known as the Jacobian matrix
  * This repetitive process will continue to reduce the Loss to a minimum, if the learning rate is set to an appropriate value

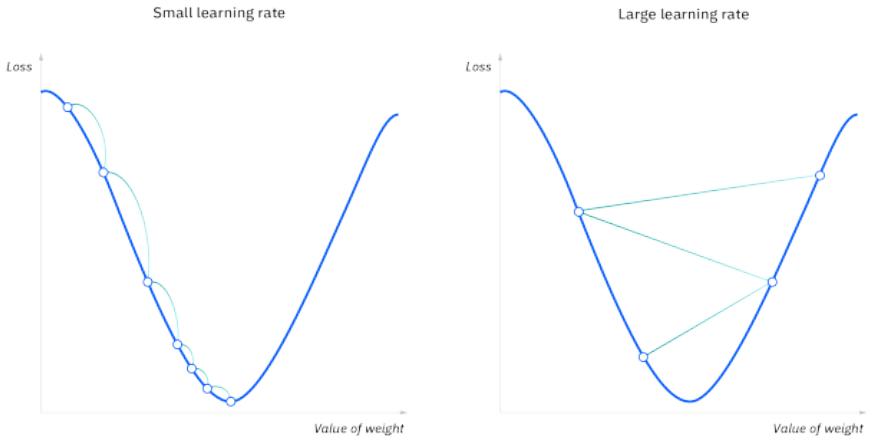



Figure 2: Gradient Descent
sourced from https://www.ibm.com/topics/gradient-descent

  * However, during backpropagation some issues can occur, such as the following:
    · Finding a false local minimum rather than the global minimum of the function

· Having an 'Exploding Gradient', where the gradient value grows exponentially to the point of overflow errors

## 1.5 Theory Behind Deep Artificial Neural Networks

### 1.5.1 Setup

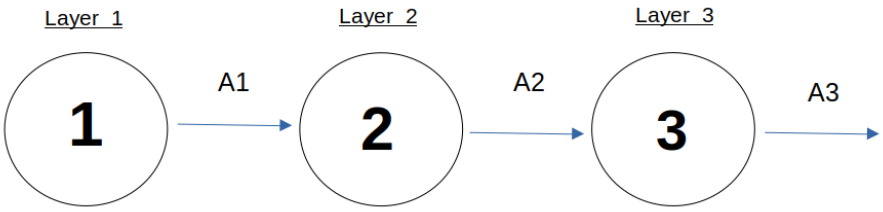



Figure 3: This shows an abstracted view of an Artificial Neural Network with multiple hidden layers and is known as a Deep Neural Netwok.

- Where a layer takes the previous layer's output as its input X
- Then it applies an Activation function to X to obtain Z, by taking the dot product of X with a weight matrix W, then sums the result with a bias matrix B. At first the weights are intialised to random values and the biases are set to zeros.
  - $Z = W * X + B$
- Then it applies a Transfer function to Z to obtain the layer's output
  - For the output layer, the sigmoid function (explained previously) must be used for either for binary classification via logistic regression, or for multi- class classification where it predicts the output neuron, and the associated class, that has the highest value between zero and one.
    - Where $sigmoid(Z) = \frac{1}{1+e^{-Z}}$
  - However, for the input layer and the hidden layers, another transfer function known as ReLu (Rectified Linear Unit) can be better suited as it produces largers values of $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial B}$ for Gradient Descent than Sigmoid, so updates at a quicker rate.
    - Where $relu(Z) = max(0, Z)$

### 1.5.2 Forward Propagation:

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

### 1.5.3 Back Propagation:

- First the Loss value L is calculated using the following Log-Loss function, which calculates the average difference between A and the value it should have predicted Y. Then the average is found by summing the result of the Loss function for each value in the matrix A, then dividing by the number of predictions m, resulting in a Loss value to show how well the network is performing.
  - Where $L = -(\frac{1}{m}) * \sum(Y * log(A) + (1 - Y) * log(1 - A))$ and "log()" is the natural logarithm
- I then move back through the network, adjusting the weights and biases via Gradient Descent. For each layer, the weights and biases are updated with the following formulae:
  - $W = W - learningRate * \frac{\partial L}{\partial W}$
  - $B = B - learningRate * \frac{\partial L}{\partial B}$
- The derivation for Layer 2's $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial B}$ can be seen below:
  - Functions used so far:
    1. $Z = W * X + B$
    2. $A_{relu} = max(0, Z)$
    3. $A_{sigmoid} = \frac{1}{1+e^{-Z}}$
    4. $L = -(\frac{1}{m}) * \sum(Y * log(A) + (1 - Y) * log(1 - A))$
  - $\frac{\partial L}{\partial A2} = \frac{\partial L}{\partial A3} * \frac{\partial A3}{\partial Z3} * \frac{\partial Z3}{\partial A2}$
    By using function 1, where A2 is X for the 3rd layer, $\frac{\partial Z3}{\partial A2} = W3$
    $=> \frac{\partial L}{\partial A2} = \frac{\partial L}{\partial A3} * \frac{\partial A3}{\partial Z3} * W3$
  - $\frac{\partial L}{\partial W2} = \frac{\partial L}{\partial A2} * \frac{\partial A2}{\partial Z2} * \frac{\partial Z2}{\partial W2}$
    By using function 1, where A1 is X for the 2nd layer, $\frac{\partial Z2}{\partial W2} = A1$
    $=> \frac{\partial L}{\partial W2} = \frac{\partial L}{\partial A2} * \frac{\partial A2}{\partial Z2} * A1$
  - $\frac{\partial L}{\partial B2} = \frac{\partial L}{\partial A2} * \frac{\partial A2}{\partial Z2} * \frac{\partial Z2}{\partial B2}$
    By using function 1, $\frac{\partial Z2}{\partial B2} = 1$
    $=> \frac{\partial L}{\partial W2} = \frac{\partial L}{\partial A2} * \frac{\partial A2}{\partial Z2} * 1$
- As you can see, when moving back through the network, the $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial B}$ of the layer can be calculated with the rate of change of loss with respect to its output, which is calculated by the previous layer using the above formula; the derviative of the layer's transfer function, and the layers input (which in this case is A1)
  - Where by using function 2, $\frac{\partial A_{relu}}{\partial Z} = 1$ when $Z >= 0$ otherwise $\frac{\partial A_{relu}}{\partial Z} = 0$
  - Where by using function 3, $\frac{\partial A_{sigmoid}}{\partial Z} = A * (1 - A)$

- At the start of backpropagation, the rate of change of loss with respect to the output layer's output has no previous layer's caluculations, so instead it can be found with the derivative of the Log-Loss function, as shown in the following:
  - Using function 4, $\frac{\partial L}{\partial A} = (-\frac{1}{m})(\frac{Y-A}{A*(1-A)})$

## 1.6 Theory behind training the Artificial Neural Networks

Training an Artificial Neural Network's weights and biases to predict on a dataset, will create a trained model for that dataset, so that it can predict on future images inputted. However, training Artificial Neural Networks can involve some problems such as Overfitting, where the trained model learns the patterns of the training dataset too well, causing worse prediction on a different test dataset. This can occur when the training dataset does not cover enough situations of inputs and the desired outputs (by being too small for example), if the model is trained for too many epochs on the poor dataset and having too many layers in the Neural Network. Another problem is Underfitting, where the model has not learnt the patterns of the training dataset well enough, often when it has been trained for too few epochs, or when the Neural Network is too simple (too linear).

### 1.6.1 Datasets

- MNIST dataset
  - The MNIST dataset is a famouse dataset of images of handwritten digits from zero to ten and is commonly used to test the performance of an Artificial Neural Network.
  - The dataset consists of 60,000 input images, made up from 28x28 pixels and each pixel has an RGB value from 0 to 255
  - To format the images into a suitable format to be inputted into the Artificial Neural Networks, each image's matrice of RGB values are 'flattened' into a 1 dimensional matrix of values, where each element is also divided by 255 (the max RGB value) to a number between 0 and 1, to standardize the dataset.
  - The output dataset is also loaded, where each output for each image is an array, where the index represents the number of the image, by having a 1 in the index that matches the number represented and zeros for all other indexes.
  - To create a trained Artificial Neural Network model on this dataset, the model will require 10 output neurons (one for each digit), then by using the Sigmoid Transfer function to output a number between one and zero to each neuron, whichever neuron has the highest value is predicted. This is multi-class classification, where the model must predict one of 10 classes (in this case, each class is one of the digits from zero to ten).
- Cat dataset

- I will also use a dataset of images sourced from https://github.com/marcopeix, where each image is either a cat or not a cat.
- The dataset consists of 209 input images, made up from 64x64 pixels and each pixel has an RGB value from 0 to 255
- To format the images into a suitable format to be inputted into the Artificial Neural Networks, each image's matrice of RGB values are 'flattened' into a 1 dimensional array of values, where each element is also divided by 255 (the max RGB value) to a number between 0 and 1, to standardize the dataset.
- The output dataset is also loaded, and is reshaped into a 1 dimensional array of 1s and 0s, to store the output of each image (1 for cat, 0 for non cat)
- To create a trained Artificial Neural Network model on this dataset, the model will require only 1 output neuron, then by using the Sigmoid Transfer function to output a number between one and zero for the neuron, if the neuron's value is closer to 1 it predicts cat, otherwise it predicts not a cat. This is binary classification, where the model must use logistic regression to predict whether it is a cat or not a cat.

- XOR dataset
  - For experimenting with Artificial Neural Networks, I solve the XOR gate problem, where the Neural Network is fed input pairs of zeros and ones and learns to predict the output of a XOR gate used in circuits.
  - This takes much less computation time than image datasets, so is usefull for quickly comparing different hyper-parameters of a Network.

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

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

# 2 Design

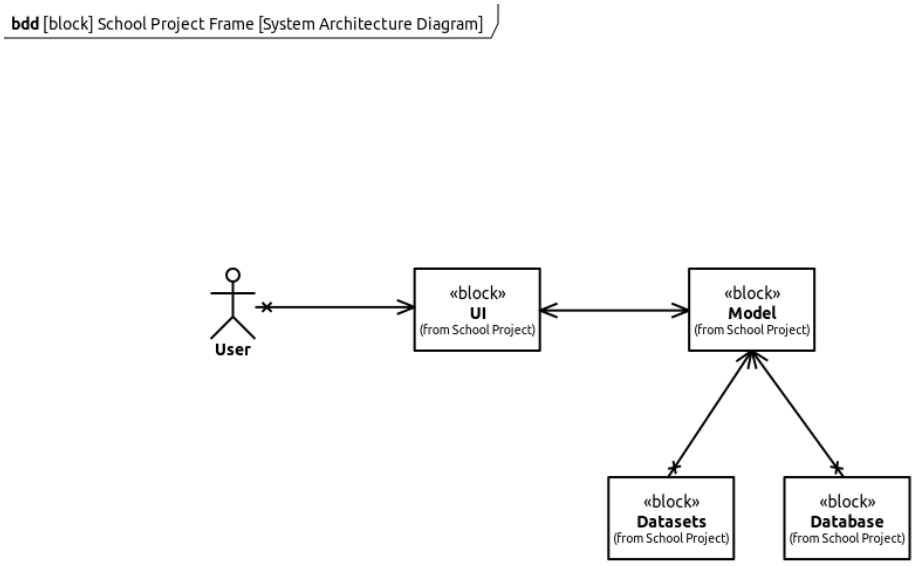
## 2.1 Introduction

The following design focuses have been made for the project:

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

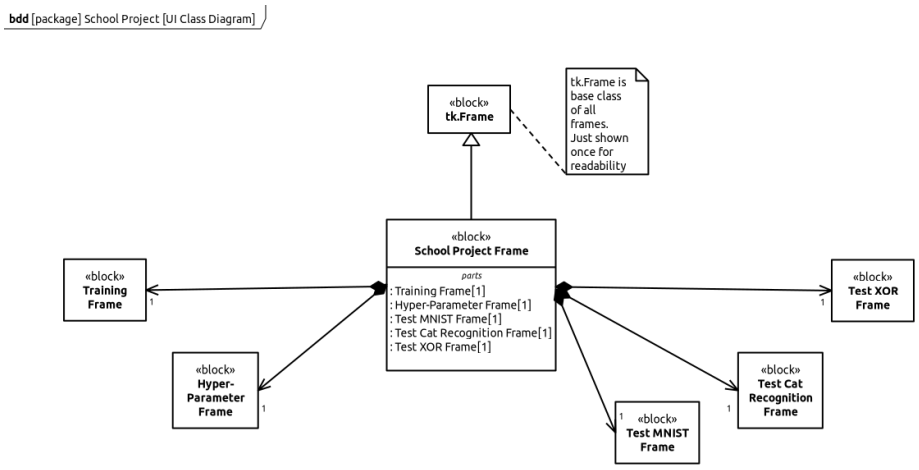
I will also be using SysML for designing the following diagrams.

## 2.2 System Architecture

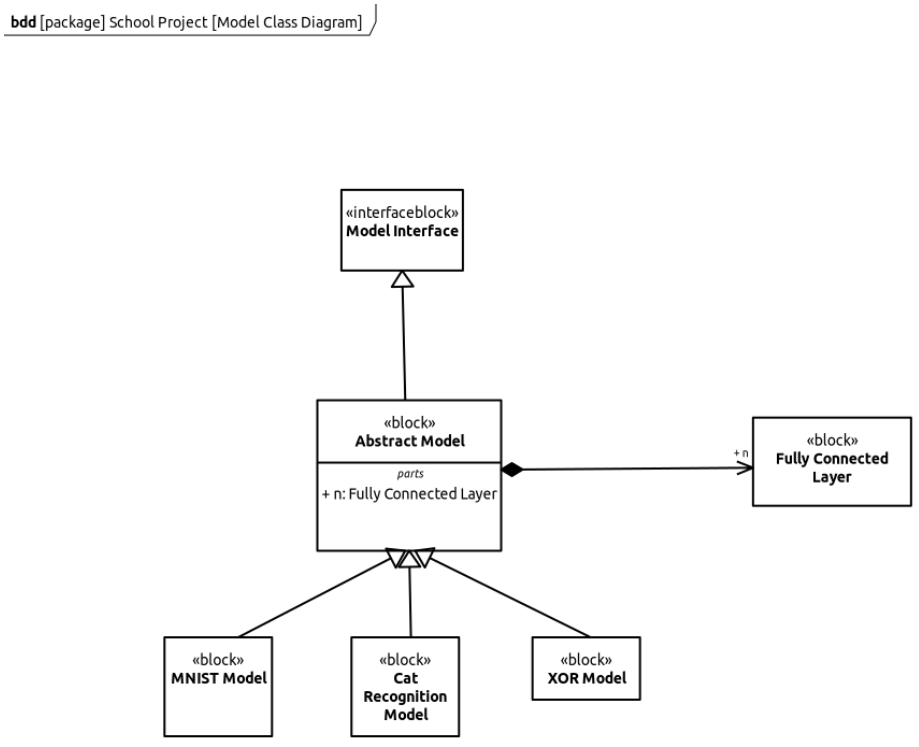


## 2.3 Class Diagrams

### 2.3.1 UI Class Diagram




### 2.3.2 Model Class Diagram



## 2.4 System Flow chart

- The general flow of the program TODO

## 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:

- 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.
- Dictionaries for loading the default hyper-parameter values from a JSON file.

## 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 a pretrained model TODO

## 2.8 Database Design

- TODO

## 2.9 Queries

- TODO

## 2.10 HCI

- Labeled screenshots of UI TODO

## 2.11 Hardware Design

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

# 3 Technical Solution

## 3.1 Test

```
    import x
    class Y():
        """"""
        @_decorator
        def output(self):  # comment
            self.attribute = "lkdsfjldskfj"
            print("Hello World")
```

```
import os
import sqlite3
import threading
import tkinter as tk
import tkinter.font as tkf
import uuid

import pympler.tracker as tracker

from school_project.frames.create_model import (HyperParameterFrame,
                                                TrainingFrame)
from school_project.frames.load_model import LoadModelFrame
from school_project.frames.test_model import (TestMNISTFrame,
                                              TestCatRecognitionFrame,
                                              TestXORFrame)

class SchoolProjectFrame(tk.Frame):
    """Main frame of school project."""
    def __init__(self, root: tk.Tk, width: int, height: int, bg: str) -> None:
        """Initialise school project pages.

        Args:
            root (tk.Tk): the widget object that contains this widget.
            width (int): the pixel width of the frame.
            height (int): the pixel height of the frame.
            bg (str): the hex value or name of the frame's background colour.
        Raises:
            TypeError: if root, width or height are not of the correct type.

        """
        super().__init__(master=root, width=width, height=height, bg=bg)
        self.root = root.title("School Project")
        self.WIDTH = width
        self.HEIGHT = height
        self.BG = bg

        # Setup school project frame variables
        self.hyper_parameter_frame: HyperParameterFrame
        self.training_frame: TrainingFrame
        self.load_model_frame: LoadModelFrame
        self.test_frame: TestMNISTFrame | TestCatRecognitionFrame | TestXORFrame
        self.connection, self.cursor = self.setup_database()
        self.model = None

        # Record if the model should be saved after testing,
        # as only newly created models should be given the option to be saved.
        self.saving_model: bool

```

```
49          # Setup school project frame widgets
50          self.exit_hyper_parameter_frame_button = tk.Button(
51                                          master=self,
52                                          width=13,
53                                          height=1,
54                                          font=tkf.Font(size=12),
55                                          text="Exit",
56                                          command=self.exit_hyper_parameter_frame
57                                          )
58          self.exit_load_model_frame_button = tk.Button(
59                                               master=self,
60                                               width=13,
61                                               height=1,
62                                               font=tkf.Font(size=12),
63                                               text="Exit",
64                                               command=self.exit_load_model_frame
65                                               )
66          self.train_button = tk.Button(master=self,
67                                        width=13,
68                                        height=1,
69                                        font=tkf.Font(size=12),
70                                        text="Train Model",
71                                        command=self.enter_training_frame)
72          self.stop_training_button = tk.Button(
73                                           master=self,
74                                           width=15, height=1,
75                                           font=tkf.Font(size=12),
76                                           text="Stop Training Model",
77                                           command=lambda: self.model.set_running(
78                                                                          value=False
79                                                                          )
80                                           )
81          self.test_created_model_button = tk.Button(
82                                                 master=self,
83                                                 width=13, height=1,
84                                                 font=tkf.Font(size=12),
85                                                 text="Test Model",
86                                                 command=self.test_created_model
87                                                 )
88          self.test_loaded_model_button = tk.Button(
89                                                  master=self,
90                                                  width=13, height=1,
91                                                  font=tkf.Font(size=12),
92                                                  text="Test Model",
93                                                  command=self.test_loaded_model
94                                                  )
95          self.delete_loaded_model_button = tk.Button(
96                                                  master=self,
97                                                  width=13, height=1,
98                                                  font=tkf.Font(size=12),
99                                                  text="Delete Model",
100                                                 command=self.delete_loaded_model
101                                                 )
102         self.save_model_label = tk.Label(
103                                     master=self,
104                                     text="Enter a name for your trained model:",
105                                     bg=self.BG,
106                                     font=('Arial', 15)
107                                     )
108         self.save_model_name_entry = tk.Entry(master=self, width=13)
109         self.save_model_button = tk.Button(master=self,
110                                            width=13,
```

```
111                                         height=1,
112                                         font=tkf.Font(size=12),
113                                         text="Save Model",
114                                         command=self.save_model)
115         self.exit_button = tk.Button(master=self,
116                                      width=13, height=1,
117                                      font=tkf.Font(size=12),
118                                      text="Exit",
119                                      command=self.enter_home_frame)
120
121         # Setup home frame
122         self.home_frame = tk.Frame(master=self,
123                                    width=self.WIDTH,
124                                    height=self.HEIGHT,
125                                    bg=self.BG)
126         self.title_label = tk.Label(
127                         master=self.home_frame,
128                         bg=self.BG,
129                         font=('Arial', 20),
130                         text="A-level Computer Science NEA Programming Project"
131                         )
132         self.about_label = tk.Label(
133            master=self.home_frame,
134            bg=self.BG,
135            font=('Arial', 14),
136            text="An investigation into how Artificial Neural Networks work, " +
137            "the effects of their hyper-parameters and their applications " +
138            "in Image Recognition.\n\n" +
139            " - Max Cotton"
140            )
141         self.model_menu_label = tk.Label(master=self.home_frame,
142                                          bg=self.BG,
143                                          font=('Arial', 14),
144                                          text="Create a new model " +
145                                          "or load a pre-trained model "
146                                          "for one of the following datasets:")
147         self.dataset_option_menu_var = tk.StringVar(master=self.home_frame,
148                                                     value="MNIST")
149         self.dataset_option_menu = tk.OptionMenu(self.home_frame,
150                                                  self.dataset_option_menu_var,
151                                                  "MNIST",
152                                                  "Cat Recognition",
153                                                  "XOR")
154         self.create_model_button = tk.Button(
155                                      master=self.home_frame,
156                                      width=13, height=1,
157                                      font=tkf.Font(size=12),
158                                      text="Create Model",
159                                      command=self.enter_hyper_parameter_frame
160                                      )
161         self.load_model_button = tk.Button(master=self.home_frame,
162                                            width=13, height=1,
163                                            font=tkf.Font(size=12),
164                                            text="Load Model",
165                                            command=self.enter_load_model_frame)
166
167         # Grid home frame widgets
168         self.title_label.grid(row=0, column=0, columnspan=4, pady=(10,0))
169         self.about_label.grid(row=1, column=0, columnspan=4, pady=(10,50))
170         self.model_menu_label.grid(row=2, column=0, columnspan=4)
171         self.dataset_option_menu.grid(row=3, column=0, columnspan=4, pady=30)
172         self.create_model_button.grid(row=4, column=1)
```

```
173             self.load_model_button.grid(row=4, column=2)
174
175             self.home_frame.pack()
176
177             # Setup frame attributes
178             self.grid_propagate(flag=False)
179             self.pack_propagate(flag=False)
180
181         @staticmethod
182         def setup_database() -> tuple[sqlite3.Connection, sqlite3.Cursor]:
183             """Create a connection to the pretrained_models database file and
184                setup base table for each dataset if needed.
185
186                Returns:
187                    a tuple of the database connection and the cursor for it.
188
189             """
190             connection = sqlite3.connect(
191                                     database='school_project/saved_models.db'
192                                     )
193             cursor = connection.cursor()
194             cursor.execute("""
195             CREATE TABLE IF NOT EXISTS MNIST
196             (Model_Name TEXT PRIMARY KEY,
197             File_Location TEXT,
198             Hidden_Layers_Shape TEXT,
199             Train_Dataset_Size INTEGER,
200             Learning_Rate FLOAT,
201             Use_ReLu INTEGER)
202             """)
203             cursor.execute("""
204             CREATE TABLE IF NOT EXISTS Cat_Recognition
205             (Model_Name TEXT PRIMARY KEY,
206             File_Location TEXT,
207             Hidden_Layers_Shape TEXT,
208             Train_Dataset_Size INTEGER,
209             Learning_Rate FLOAT,
210             Use_ReLu INTEGER)
211             """)
212             cursor.execute("""
213             CREATE TABLE IF NOT EXISTS XOR
214             (Model_Name TEXT PRIMARY KEY,
215             File_Location TEXT,
216             Hidden_Layers_Shape TEXT,
217             Train_Dataset_Size INTEGER,
218             Learning_Rate FLOAT,
219             Use_ReLu INTEGER)
220             """)
221             return (connection, cursor)
222
223         def enter_hyper_parameter_frame(self) -> None:
224             """Unpack home frame and pack hyper-parameter frame."""
225             self.home_frame.pack_forget()
226             self.hyper_parameter_frame = HyperParameterFrame(
227                                          root=self,
228                                          width=self.WIDTH,
229                                          height=self.HEIGHT,
230                                          bg=self.BG,
231                                          dataset=self.dataset_option_menu_var.get()
232                                          )
233             self.hyper_parameter_frame.pack()
234             self.train_button.pack()
```

```
235             self.exit_hyper_parameter_frame_button.pack(pady=(10,0))
236
237         def enter_load_model_frame(self) -> None:
238             """Unpack home frame and pack load model frame."""
239             self.home_frame.pack_forget()
240             self.load_model_frame = LoadModelFrame(
241                                         root=self,
242                                         width=self.WIDTH,
243                                         height=self.HEIGHT,
244                                         bg=self.BG,
245                                         connection=self.connection,
246                                         cursor=self.cursor,
247                                         dataset=self.dataset_option_menu_var.get()
248                                         )
249             self.load_model_frame.pack()
250
251             # Don't give option to test loaded model if no models have been saved
252             # for the dataset.
253             if len(self.load_model_frame.model_options) > 0:
254                 self.test_loaded_model_button.pack()
255                 self.delete_loaded_model_button.pack(pady=(5,0))
256
257             self.exit_load_model_frame_button.pack(pady=(5,0))
258
259         def exit_hyper_parameter_frame(self) -> None:
260             """Unpack hyper-parameter frame and pack home frame."""
261             self.hyper_parameter_frame.pack_forget()
262             self.train_button.pack_forget()
263             self.exit_hyper_parameter_frame_button.pack_forget()
264             self.home_frame.pack()
265
266         def exit_load_model_frame(self) -> None:
267             """Unpack load model frame and pack home frame."""
268             self.load_model_frame.pack_forget()
269             self.test_loaded_model_button.pack_forget()
270             self.delete_loaded_model_button.pack_forget()
271             self.exit_load_model_frame_button.pack_forget()
272             self.home_frame.pack()
273
274         def enter_training_frame(self) -> None:
275             """Load untrained model from hyper parameter frame,
276                unpack hyper-parameter frame, pack training frame
277                and begin managing the training thread.
278             """
279             try:
280                 self.model = self.hyper_parameter_frame.create_model()
281             except (ValueError, ImportError) as e:
282                 return
283             self.hyper_parameter_frame.pack_forget()
284             self.train_button.pack_forget()
285             self.exit_hyper_parameter_frame_button.pack_forget()
286             self.training_frame = TrainingFrame(
287                     root=self,
288                     width=self.WIDTH,
289                     height=self.HEIGHT,
290                     bg=self.BG,
291                     model=self.model,
292                     epoch_count=self.hyper_parameter_frame.epoch_count_scale.get()
293                     )
294             self.training_frame.pack()
295             self.stop_training_button.pack()
296             self.manage_training(train_thread=self.training_frame.train_thread)
```

```
297
298         def manage_training(self, train_thread: threading.Thread) -> None:
299             """Wait for model training thread to finish,
300                then plot training losses on training frame.
301
302             Args:
303                 train_thread (threading.Thread):
304                 the thread running the model's train() method.
305             Raises:
306                 TypeError: if train_thread is not of type threading.Thread.
307
308             """
309             if not train_thread.is_alive():
310                 self.training_frame.training_progress_label.pack_forget()
311                 self.training_frame.plot_losses(model=self.model)
312                 self.stop_training_button.pack_forget()
313                 self.test_created_model_button.pack(pady=(30,0))
314             else:
315                 self.training_frame.training_progress_label.configure(
316                                                     text=self.model.training_progress
317                                                     )
318                 self.after(100, self.manage_training, train_thread)
319
320         def test_created_model(self) -> None:
321             """Unpack training frame, pack test frame for the dataset
322                and begin managing the test thread."""
323             self.saving_model = True
324             self.training_frame.pack_forget()
325             self.test_created_model_button.pack_forget()
326             if self.hyper_parameter_frame.dataset == "MNIST":
327                 self.test_frame = TestMNISTFrame(
328                                         root=self,
329                                         width=self.WIDTH,
330                                         height=self.HEIGHT,
331                                         bg=self.BG,
332                                         use_gpu=self.hyper_parameter_frame.use_gpu,
333                                         model=self.model
334                                         )
335             elif self.hyper_parameter_frame.dataset == "Cat Recognition":
336                 self.test_frame = TestCatRecognitionFrame(
337                                         root=self,
338                                         width=self.WIDTH,
339                                         height=self.HEIGHT,
340                                         bg=self.BG,
341                                         use_gpu=self.hyper_parameter_frame.use_gpu,
342                                         model=self.model
343                                         )
344             elif self.hyper_parameter_frame.dataset == "XOR":
345                 self.test_frame = TestXORFrame(root=self,
346                                                width=self.WIDTH,
347                                                height=self.HEIGHT,
348                                                bg=self.BG,
349                                                model=self.model)
350             self.test_frame.pack()
351             self.manage_testing(test_thread=self.test_frame.test_thread)
352
353         def test_loaded_model(self) -> None:
354             """Load saved model from load model frame, unpack load model frame,
355                pack test frame for the dataset and begin managing the test thread."""
356             self.saving_model = False
357             try:
358                 self.model = self.load_model_frame.load_model()
```

```
359             except (ValueError, ImportError) as e:
360                 return
361             self.load_model_frame.pack_forget()
362             self.test_loaded_model_button.pack_forget()
363             self.delete_loaded_model_button.pack_forget()
364             self.exit_load_model_frame_button.pack_forget()
365             if self.load_model_frame.dataset == "MNIST":
366                 self.test_frame = TestMNISTFrame(
367                                             root=self,
368                                             width=self.WIDTH,
369                                             height=self.HEIGHT,
370                                             bg=self.BG,
371                                             use_gpu=self.load_model_frame.use_gpu,
372                                             model=self.model
373                                             )
374             elif self.load_model_frame.dataset == "Cat Recognition":
375                 self.test_frame = TestCatRecognitionFrame(
376                                             root=self,
377                                             width=self.WIDTH,
378                                             height=self.HEIGHT,
379                                             bg=self.BG,
380                                             use_gpu=self.load_model_frame.use_gpu,
381                                             model=self.model
382                                             )
383             elif self.load_model_frame.dataset == "XOR":
384                 self.test_frame = TestXORFrame(root=self,
385                                                width=self.WIDTH,
386                                                height=self.HEIGHT,
387                                                bg=self.BG,
388                                                model=self.model)
389             self.test_frame.pack()
390             self.manage_testing(test_thread=self.test_frame.test_thread)
391 
392         def manage_testing(self, test_thread: threading.Thread) -> None:
393             """Wait for model test thread to finish,
394                then plot results on test frame.
395 
396             Args:
397                 test_thread (threading.Thread):
398                 the thread running the model's predict() method.
399             Raises:
400                 TypeError: if test_thread is not of type threading.Thread.
401 
402             """
403             if not test_thread.is_alive():
404                 self.test_frame.plot_results(model=self.model)
405                 if self.saving_model:
406                     self.save_model_label.pack(pady=(30,0))
407                     self.save_model_name_entry.pack(pady=10)
408                     self.save_model_button.pack()
409                 self.exit_button.pack(pady=(20,0))
410             else:
411                 self.after(1_000, self.manage_testing, test_thread)
412 
413         def save_model(self) -> None:
414             """Save the model, save the model information to the database, then
415                enter the home frame."""
416             # Save model to random hex file name
417             file_location = f"school_project/saved-models/{uuid.uuid4().hex}.npz"
418             self.model.save_model_values(file_location=file_location)
419 
420             # Check if model name has already been taken
```

```
421             dataset = self.dataset_option_menu_var.get().replace(" ", "_")
422             model_name = self.save_model_name_entry.get()
423             self.cursor.execute(f"""
424             SELECT Model_Name FROM {dataset}
425             """)
426             for saved_model_name in self.cursor.fetchall():
427                 if saved_model_name[0] == model_name:
428                     self.test_frame.model_status_label.configure(
429                                                         text="Model name taken",
430                                                         fg='red'
431                                                         )
432                     return
433 
434             # Save the model information to the database
435             sql = f"""
436             INSERT INTO {dataset}
437             (Model_Name, File_Location, Hidden_Layers_Shape, Train_Dataset_Size, Learning_Rate, Use_ReLu)
438             VALUES (?, ?, ?, ?, ?, ?)
439             """
440             parameters = (
441                         model_name,
442                         file_location,
443                         self.hyper_parameter_frame.hidden_layers_shape_entry.get(),
444                         self.hyper_parameter_frame.train_dataset_size_scale.get(),
445                         self.hyper_parameter_frame.learning_rate_scale.get(),
446                         self.hyper_parameter_frame.use_relu_check_button_var.get()
447                         )
448             self.cursor.execute(sql, parameters)
449             self.connection.commit()
450 
451             self.enter_home_frame()
452 
453         def delete_loaded_model(self) -> None:
454             """Delete saved model file and model data from the database."""
455             dataset = self.dataset_option_menu_var.get().replace(" ", "_")
456             model_name = self.load_model_frame.model_option_menu_var.get()
457 
458             # Delete saved model
459             sql = f"""SELECT * FROM {dataset} WHERE Model_Name = ?"""
460             parameters = (model_name,)
461             self.cursor.execute(sql, parameters)
462             os.remove(self.cursor.fetchall()[0][1])
463 
464             # Remove model data from database
465             sql = f"""DELETE FROM {dataset} WHERE Model_Name = ?"""
466             parameters = (model_name,)
467             self.cursor.execute(sql, parameters)
468             self.connection.commit()
469 
470             # Reload load model frame with new options
471             self.exit_load_model_frame()
472             self.enter_load_model_frame()
473 
474         def enter_home_frame(self) -> None:
475             """Unpack test frame and pack home frame."""
476             self.model = None  # Free up trained Model from memory
477             self.test_frame.pack_forget()
478             if self.saving_model:
479                 self.save_model_label.pack_forget()
480                 self.save_model_name_entry.delete(0, tk.END)  # Clear entry's text
481                 self.save_model_name_entry.pack_forget()
482                 self.save_model_button.pack_forget()
```

```
483             self.exit_button.pack_forget()
484             self.home_frame.pack()
485             summary_tracker.create_summary()  # BUG: Object summary seems to reduce
486                                               # memory leak greatly
487
488     def main() -> None:
489         """Entrypoint of project."""
490         root = tk.Tk()
491         school_project = SchoolProjectFrame(root=root, width=1280,
492                                             height=835, bg='white')
493         school_project.pack(side='top', fill='both', expand=True)
494         root.mainloop()
495
496         # Stop model training when GUI closes
497         if school_project.model != None:
498             school_project.model.set_running(value=False)
499
500     if __name__ == "__main__":
501         summary_tracker = tracker.SummaryTracker()  # Setup object tracker
502         main()
```