

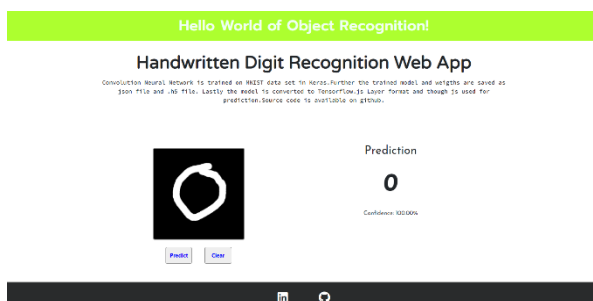
Analysis I/II: DigiTrace Project Introduction

The premise of my project – called “DigiTrace” - is to exhibit the process of machine learning in the context of recognising handwritten digits. This would include a GUI interface to display the samples used to train the model, take handwritten samples from the user, show the samples being used, visualise the neural network and output digit predictions for a given input. It would have a learning mode, where samples can be input to train the model, and a prediction mode, where a sample given via user input can lead to a digit prediction. I have found a large dataset of handwritten digits on Kaggle which will be the primary source of samples. I will be using Java since an object-oriented approach will benefit me greatly, and will keep the project structure and the functions I create organised. I won't be using any AI/ML libraries, as these libraries do most of the work for you, so I probably would not be able to produce a substantial amount of code and evidence this way. I will need to make a linear algebra library, and classes for samples.

I was inspired to make a project like this after learning the maths behind machine learning, and how it uses linear algebra for layers and weight matrices, but also calculus for backwards propagation (adjusting weights and biases based on the result). I found it very interesting and wanted to implement it myself. I need to find out more about graphics in Java Swing UI, considering I'll need to visualise the progress of the model's training. I will look at other AI apps online and recognise key features and components to inspire my app, perhaps with UI preferences or visualisation techniques.

Analysis III: Research

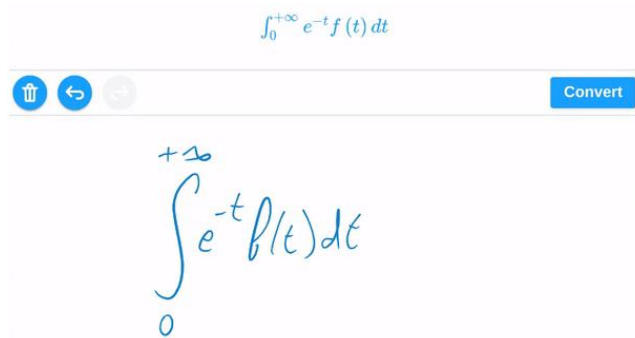
A project very similar to mine is a digit recognition web app at [Digit Recognition WebApp \(maneprajakta.github.io\)](https://maneprajakta.github.io/Digit-Recognition-WebApp/) .



This project uses JSON/H5 files to store weights and biases for the AI model, which I would like to include in my project. This would make the AI models persistent, and eliminates the need to retrain a model (a long and gruelling process) every time the app is reloaded. This would be included in the prediction mode. I don't, however, like

the colour scheme, font, or UI style of this app, but am in favour of the UI layout, and will adapt something similar for the prediction mode interface.

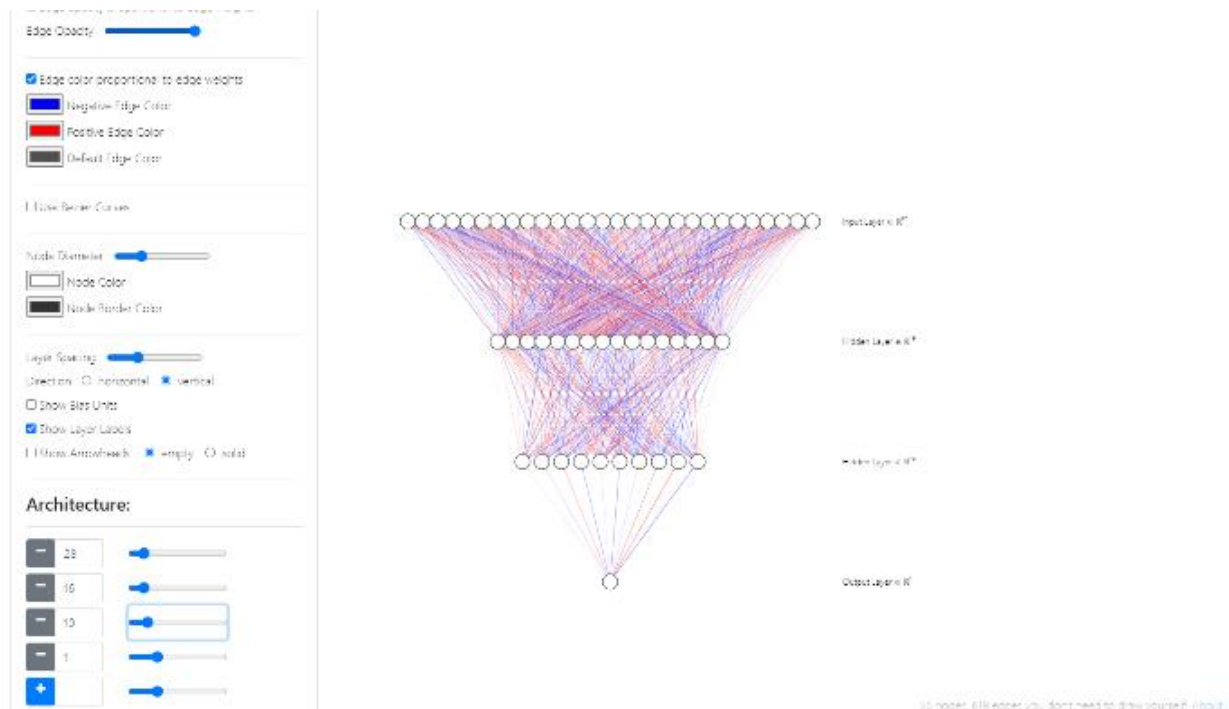
[myscript-math-web](#) is a handwritten maths recogniser that can be put in web UI components, so works in a very similar way to my project.



I like the UI style of this project a lot. The colour scheme is consistent, and the buttons contain icons as opposed to words, which I believe looks clearer, more professional, and more aesthetically pleasing; it also removes a lot of language dependency. Circular buttons are also a nice addition; I may incorporate that also. Although the colour scheme is consistent, blue and

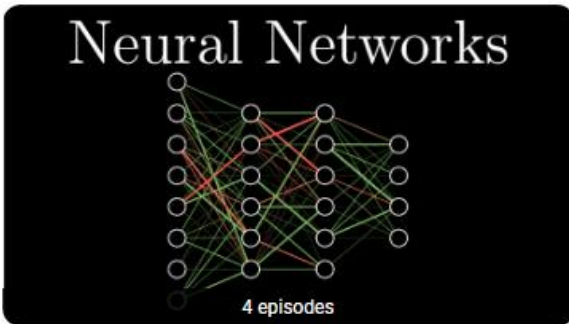
white seems very primitive, so I plan to add a more intricate colour scheme to my app.

[NN-SVG](#) generates SVG files to visualise neural networks.



Although I won't use this tool in my project, I love the style of visualisation, specifically the look of the nodes and the coloured weights to show the strength and positivity/negativity. I also like the mathematical accuracy in the layer labels (details that

the layers are n-dimensional vectors, i.e. elements of \mathbb{R}^n). I will try to add a similar mathematical notation, though I'm not sure if I can use LaTeX in Java GUI



Neural networks
3Blue1Brown

I learned about how basic AI/ML works in a mathematical context from this series from 3Blue1Brown. It taught me all about the technical aspects like layers, prediction, cost and backwards-propagation, and the maths behind it. I will implement these concepts in java through classes for the elementary objects like matrices and vectors, which can be used to make classes for neural networks and their parts. The prediction process will be demonstrated in the prediction mode of the app, and the backwards-propagation algorithm will be demonstrated in the learning mode.

Analysis IV: Stakeholders

The stakeholders for my project may include computer science teachers and students, aiming to demonstrate or visualise the machine learning process. This would typically be interesting to A-Level students, as they would have learnt the appropriate maths and computer science theory to get the most out of this program. It would also be a comprehensible example for students as it is the simplest form of artificial intelligence.

Analysis V/VI: User Requirements

Key	
Essential	Green
Desirable	Yellow
Optional	Red

<i>Feature</i>	<i>Component</i>	<i>Description</i>	<i>Justification</i>	<i>Necessity</i>
<i>Startup Menu</i>	Title	An image of the title "DigiTrace" in a bold font, with a small graphic/logo before the word.	Having a clear title is instrumental to clear presentation of the project. An attractive title with a logo will make the project seem well structured and well-	Green

<i>Prediction Mode</i>			designed as soon as it is opened.	
	Background	An abstract, impasto style background with very low opacity all around, yet almost no opacity around the title and logo.	Having a bit of colour in the background at the start will add a more vibrant, elegant feel to the UI, and will give a nice contrast to the feel of the rest of the program, which is very logical and mathematical.	
	Load Model Button	A button containing the text "Load Model".	This button is necessary to take the user to the prediction mode of the program, where they can load a previously trained model, draw a digit and get a prediction.	
	Train New Model	A button containing the text "Train New Model".	This button will take the user to the learning mode of the program, where they can train a new model on samples (both their own and pre-installed).	
	Exit Button	A button containing the text "Exit"	Closes the app	
	Model File Selection	A bar in the top of the window, which opens the file explorer once clicked, and displays the file path once selected.	This will allow the user to select the JSON files for their models, and will load the weights, biases and name into the program.	
	Digit Drawing Grid	A 28x28 square grid (each square representing a pixel of an image). Black and white colouring.	This grid is necessary for the user to input a handwritten digit. It will be inputted into the model, which will form a prediction. The user can draw with the mouse.	

<i>Learning Mode</i>	Clear Button	A button under the digit drawing grid containing a bin icon.	This button clears the digit drawing grid.	
	Predict Button	A button underneath the digit drawing grid containing the text "Predict".	This button confirms the user's digit on the grid and passes it as input to the model in the form of an array.	
	Eraser Button	A button underneath the digit drawing grid containing a rubber icon.	Allows the user to erase parts of the drawing on the grid.	
	Visualisation / Calculation window	A window containing model visualisation, i.e. a neural network diagram.	This window will show some display of progress in the prediction calculation process, and the state of the neural network.	
	Prediction box	A box containing a digit.	The digit prediction of the model will be shown here.	
	Digit Drawing Grid	A 28x28 square grid (each square representing a pixel of an image). Black and white colouring.	A grid which will display either a loaded sample, or a user-created sample.	
	Label Box	A box containing a digit.	The sample's digit label will be displayed in this box, or inputted by the user if it is user-created.	
	Sample Number	A text display showing two numbers in form "Sample No. / Total Samples".	Will display which sample is being looked at, out of how many samples have been loaded.	
	Sample Scrolling Buttons	A left and right arrow button either side of the sample number display.	Will allow the user to scroll through all the loaded samples.	
	Load Preinstalled Samples Button	A one-use button containing the text "Load Preinstalled Samples".	Will allow the user to load in the thousands of pre-installed samples.	

All Modes	New Sample Button	A button containing a plus icon.	Will allow the user to create a new sample.	
	Train Button	A button containing the word "Train".	Will take all the samples in and begin the machine learning process.	
	Cost Graph	A small line graph/table of cost to iteration number.	Will display the cost of the model gradually decreasing as it learns.	
	Save Model Button	A button containing a file icon.	When the model has trained with all samples, it can be saved and used in the app's prediction mode.	
	Training Progress Bar	A bar showing how many samples have been used out of the total samples.	Will give a sense of progress to the user. Without this, the UI may be confusing, and wouldn't give any intuition into what is happening.	
	Back Button	A button containing a < symbol	Necessary to take the user back to the Startup page. Will warn the user if going back would stop a process like training or predicting, or if a model is left unsaved in the learning mode.	
AI/ML Processes	Linear Algebra Library	A library containing definitions, functions and operations involving vectors and matrices.	Necessary for AI/ML, as models are neural networks made from layers (vectors), weight matrices and bias vectors.	
	Neural Network Class	A class holding number and sizes of layers, functions to get predictions, calculate cost, back-propagate and save all weights and	Would have algorithms implemented for	

Limitations

I have decided not to add **live visualisation** of the network as it trains. Although this would be a great feature, it would be very hard to implement and would slow down training by a great deal.

I have also decided not to add an **animated background** as a static image would render better in the window, and making an animation would be a difficult task, considering that I am inexperienced in making animated artwork.

Analysis VII: Computational Methods

Abstraction and Visualisation

The use of abstraction in my project will be instrumental, especially with information hiding and representational abstraction. At its core, all that is happening is calculations, specifically matrix-vector operations and layer activation, which just boils down to *lots* of simple multiplication, addition, and functions. These operations are not shown to the user but are instead hidden behind interfaces which simply show an input and a prediction, or a gradual decrease in network cost. All the numerical information which is essential to the model can be hidden entirely from the user. Furthermore, considering the layers, weights and biases as objects of linear algebra is an example of abstraction by generalisation; this way, layers and biases can share a generalised vector structure, which in turn can be generalised to a matrix (in this case a vector can be deemed a matrix with one column). When the model is visualised also, only its general structure need be shown, i.e., its layers and weights – another example of representational abstraction.

Pipelining

Neural networks, being made from layers, use pipelining to obtain a desired result. In forward propagation, the input layer is collected, which is then activated and weighted, which gives the next layer. This new layer is now activated and weighted, which gives the next layer, and so on. By the last layer, which consists of 10 nodes, each holding the probability of the input being each digit 0 through 9, a prediction can be made (the digit with the highest probability). It is also used in backward propagation, where the current state of the model is used to make a prediction from an input sample, and is then updated based on the cost (representative of how 'off' the prediction was). This state will then be used for the next sample, and so on.

Decomposition

Breaking down the prediction and training processes into their constituent parts will usually result in taking input, computation and presenting/using some kind of output. In the case of prediction, the output will be shown. For training, however, the output will be used to calculate cost, then weights and biases will be changed based on that.

Performance Modelling

In AI and ML, performance modelling is of course vital. It is the basis on which a machine can learn – by knowing how well it performed and improving itself based on that (changing weights and biases based on cost). It can also be used by the user to test the validity of the model as a final product. If the model consistently predicts wrongly, or is not confident in its predictions, it clearly needs to be retrained, perhaps with more samples of higher clarity.

Heuristics

It is highly unlikely, if not impossible, that any model trained would be optimal. In the machine learning process, the cost function is being minimised, yet after every tiny change made to the model as it learns, the cost function itself, of course, changes. The model can home in on an ideal solution, albeit not a perfect one, through a heuristic process. This is still good enough to make accurate predictions.

Overview I: GUI Designs

Below is the basis of the design for the program's three different pages. Being a rough outline, it doesn't include some of the more detailed features I stated previously, like background art for the title page and buttons with icons being circular, but the general layout should be similar.

The primary colour I have chosen is #FFE599, and I will keep the palette simplistic; I will probably only need that colour, black, white and grey. The progress bar will be green, however.

Main Menu

The program was made with the intention of simplifying the ideas behind machine learning and AI prediction in practice, and exhibiting them in a convenient manner, hence why on the title screen (top), I have deliberately kept it as minimalist as possible. There is no need for it to have many elements, and this way it will not overwhelm the user, maintaining user-friendliness. The buttons on the page are centred to make the program's options clear, and the title and logo (which will not stay as just a white hexagon) are enlarged and highlighted at the top, as to mark the program title and give a direct introduction to the program. The font I have selected for the whole program is

`Courier New`, the de-facto standard font for code-related text. I felt that it would give the program a more technical feel to go with the highly computational matters at hand. I haven't used any particularly ornate shapes, patterns or colour schemes, as it is unnecessary – especially considering the function of the program. Keeping it as straightforward as possible is key to not overwhelming the user.

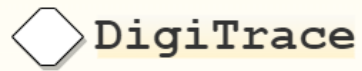
Prediction Mode

After selecting “Load Model”, The user is taken to another uncluttered UI (middle). I relied on mostly icons to convey functionality in a more accessible way; this way language barriers are bypassed, and words aren't littering the page. This can be seen in the digit drawing grid on the left, where a bin, rubber and tick are on the clear, eraser and confirm buttons respectively. The model's prediction is then labelled and displayed on the right. Separating input on the left from output on the right adds a new level of clarity to the program. The whole UI has only 3 main components: File selection, digit input and prediction output. Also, the digit input and prediction output will be disabled (i.e., greyed out and unclickable) until a file is selected. This makes for a more user-friendly experience, as it is clear in what order the program's tasks must be executed.

Learning Mode

After selecting “Train New Model”, the user is taken to a slightly more detailed page (bottom). Even though there are more things to consider in the learning mode compared to prediction mode, I've maintained the same stylistic choices and simple designs consistent with the rest of the program; it is more important to keep an undemanding UI here, as the ideas presented in learning mode are arguably the hardest ones to grasp conceptually, therefore leading to the most daunting UI – a user-friendly layout in this case is especially important. Like in prediction mode, the input and output have been split to the left and right respectively – this is even more important in this case, as there are so many more controls. The sample panel consists of a sample selector (top left) so the samples can be cycled through and edited. The label display shows what digit the sample is labelled as. In both components, I used smaller text, as to not draw attention to these lesser components over the larger, more important buttons or use up extra space. The icon buttons on the digit drawing grid remain the same as prediction mode for accessibility. Pre-installed samples can also be loaded, but only once as to not allow sample repeats – to achieve one-time use allowance, I will disable the button as soon as it is used. All the training components are on the right side of the page. The cost table (1), train button (2) and progress bar (3) are the first of them that are usable. When the train button is selected, the cost table will fill, showing the most recent cost values with their iterations, and the progress bar will show the fraction of samples processed. The

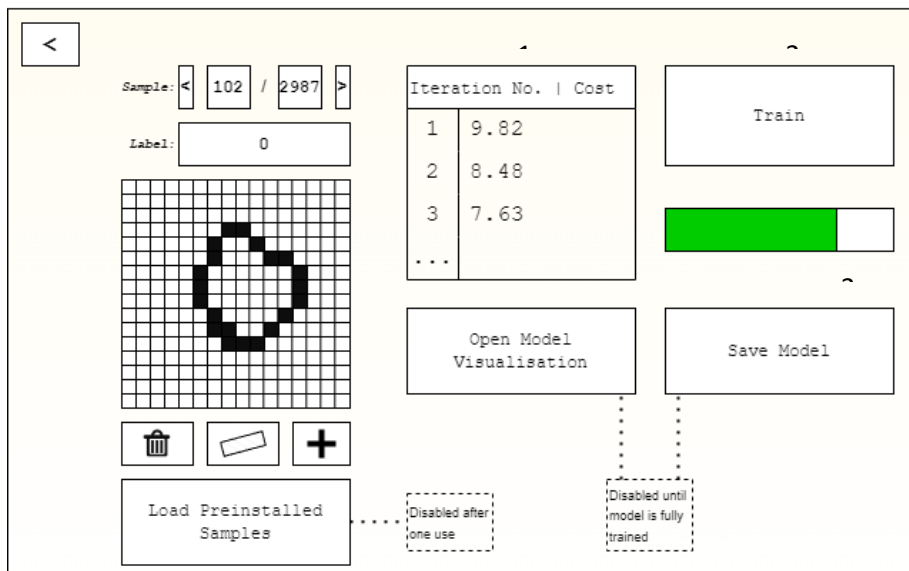
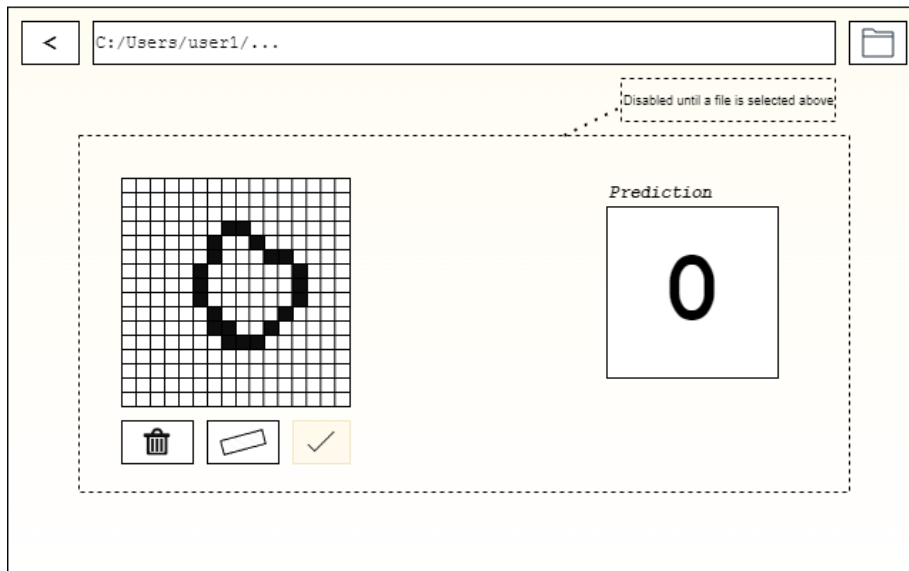
model can be visualised and saved after training is complete (their respective buttons will become active). Visualisation will most likely include generation of a PNG of the network, which can be saved. An organised layout like this one is vital for user-friendliness, especially considering there are so many more controls than the rest of the layout. Making sure components are aligned and not appearing to be random is one of the things I did to ensure this.



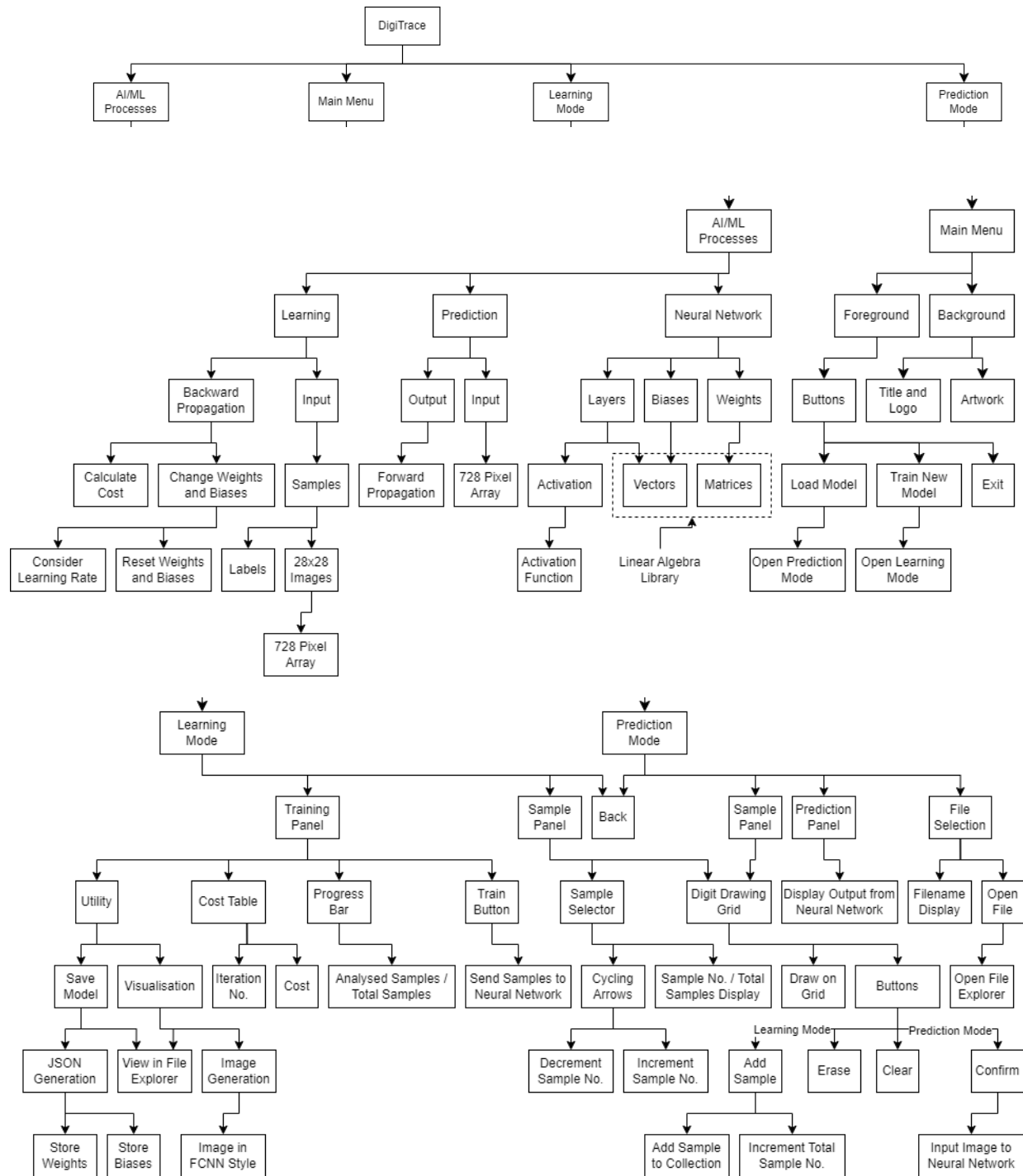
Load Model

Train New Model

Exit



Overview II: Structure Diagram



Overview III: Development Plan

Development cycles:

❖ **Stage I: Core Mathematical Classes**

- Matrix Class
 - Polymorphic Constructor (Takes either a list or sets all elements to a given value)
 - Multiplication method
 - Addition method
- Vector Class
 - Subclass of matrix class with 1 column
- Static Utils Class
 - Polymorphic activation methods (takes either a matrix or a value)
 - Print methods for debugging

❖ **Stage II: Basic Input**

- Specialised Image Class
 - Constructor takes file path
 - Map image to binary array method
- Sample Class
 - Label
 - Pixel array

❖ **Stage III: Neural Network Classes**

- Neural Network Class
 - List of Layers
 - List of Samples
 - Take input sample (not blank), forward propagation to output
 - Cost calculation
- Layer Class
 - Values set from previous layer in a vector
 - Activation
 - Weight matrix and bias vector
 - Send values to next layer
 - Different layer types (input, hidden, output)
- Storing Networks as JSON
 - `{layer: {weights: [...], biases: [...]}}, ...}`

❖ **Stage IV: Machine Learning**

- Backwards Propagation
 - Change each weight and bias depending on cost and learning rate
 - Going beyond the previous layer
 - Running many iterations
- Finding and utilising a formula for weight/bias adjustment
 - Using chain rule to find a formula based on cost and weight/bias index for weight/bias adjustment

❖ **Stage V: GUI**

- Set up GUI for title screen
 - Each menu option takes you to a new page (unless “Exit” option)
 - Make background
 - Make title and logo
- Set up GUI for prediction and learning mode
 - Does not need to be functional at this stage
 - Make digit drawing display component
 - Can turn grid tiles on and off

❖ **Stage VI: Learning Mode Core**

- Create Neural Network Instance
 - Initialise all values randomly
- Draw Samples
 - Take drawn samples from grid and add them to the neural network’s collection.
- Load Preinstalled Samples
 - Iterate through image files
 - Turn them into samples and add them to the neural network’s collection
- Train
 - Iterate over all samples
 - Get prediction, calculate cost, and change weights and biases accordingly (back-propagate)
- Save Model
 - Save weights and biases in a JSON file

❖ **Stage VII: Learning Mode UX**

- Progress Bar
 - Display fraction of bar that is green as fraction of samples analysed out of total samples
- Cost Graph
 - For every sample iteration, display the new network cost in the cost table
- Visualise Model (if possible)

❖ **Stage VIII: Prediction Mode**

- Load Model
 - Open File Explorer so a model JSON can be chosen
 - Enable the rest of the UI once a model is chosen
- Take Input from Grid
 - Take grid input as an array and input it into the model
- Display Output
 - Show digit output in prediction box

<i>Stage</i>	<i>Part</i>	<i>Test</i>	<i>Type</i>	<i>Test Data</i>	<i>Expected Result</i>	<i>Actual Result</i>
<i>I</i>	<i>a</i>	Matrix multiplication and addition works	Valid	Multiply randomly generated matrices	The correct resulting matrix (check with calculator)	
	<i>b</i>	Activation functions take floats or matrices	Valid	Apply the function to randomly generated floats and matrices	No errors in calculation	
	<i>c</i>	Multiplication of two matrices where the number of columns of the first does not equal the number of rows of the second doesn't work	Invalid	Multiply two different matrices with wrong dimensions	-1 matrix	
	<i>d</i>	Print methods work	Valid	Attempt to print a matrix	The correct output matrix printed	
<i>II</i>	<i>a</i>	Correct binary array generated from image	Valid	Process 3 images from the pre-installed image library	A binary array, with 1s for black pixels and 0s for white pixels, in row-priority order	
	<i>b</i>	Sample class instance generates without errors from images alone	Valid	Process 3 images from the pre-installed image library	Sample classes with the correct label and pixel array	
<i>III</i>	<i>a</i>	Neural network takes sample as input without error, and gives a prediction	Valid	Process 3 images from the pre-installed	A prediction (not necessarily correct)	

IV	b	Neural network does not take a blank sample (all 0 binary array)	Invalid	Input a blank sample	An error
	c	Neural network stores as JSON correctly	Valid	Attempt to save NN as JSON	JSON with weights and biases stored in row-priority order
	a	Calculate cost	Valid	Output cost from a given prediction	A reasonable value for cost (not too large or small)
	b	Weights and biases are adjusted	Valid	Output weight matrices and bias vectors before and after a sample is processed	A difference in weights and biases before and after
V	a	Title screen GUI takes you to separate pages	Valid	Click GUI buttons to open pages	New pages open and main menu closes
VI	a	Samples are added to neural network	Valid	Draw samples and click confirm	Samples added with correct arrays and labels
	b	Preinstalled samples can be loaded to neural network	Valid	Click "load pre-installed samples" button	All pre-installed samples added
	c	Empty sample cannot be added	Boundary	Click confirm while grid is empty	No sample added
	d	Labelless sample cannot be added	Boundary	Click confirm while label is empty	No sample added
	e	Model trains	Valid	Click "Train" button	Weights and biases repeatedly change

VII	f	Model saves	Valid	Click "Save Model" and choose a file location	A new JSON file created in the system with the correct parameters
	g	Model cannot be trained without samples	Boundary	Try to click train when there are no added samples	The model does not attempt to train
	a	Progress bar displays fraction of samples analysed over total samples	Valid	Click "Train" button	Progress bar gradually fills
	b	Cost table updates	Valid	Click "Train" button	Cost table is being constantly filled with new values and auto scrolled
VIII	a	Model loads from file	Valid	Choose a file in the file selector	Model loads in without error
	b	Prediction given from input	Valid	Draw a digit and confirm	A prediction is given without error

Post development stage:

When the project is complete, alpha testing will start, and I will test all the different modes myself, starting with learning mode to train and save a model, then prediction mode to load the model and test it. I will test general user experience and usability, and I will also test how accurate the models turn out. I will make multiple models to check the consistency of the training. This final testing will mainly aim to check that the whole program works in unison, such that you can create a model, train it, and use it. I will go through the following list to check the program's general function.

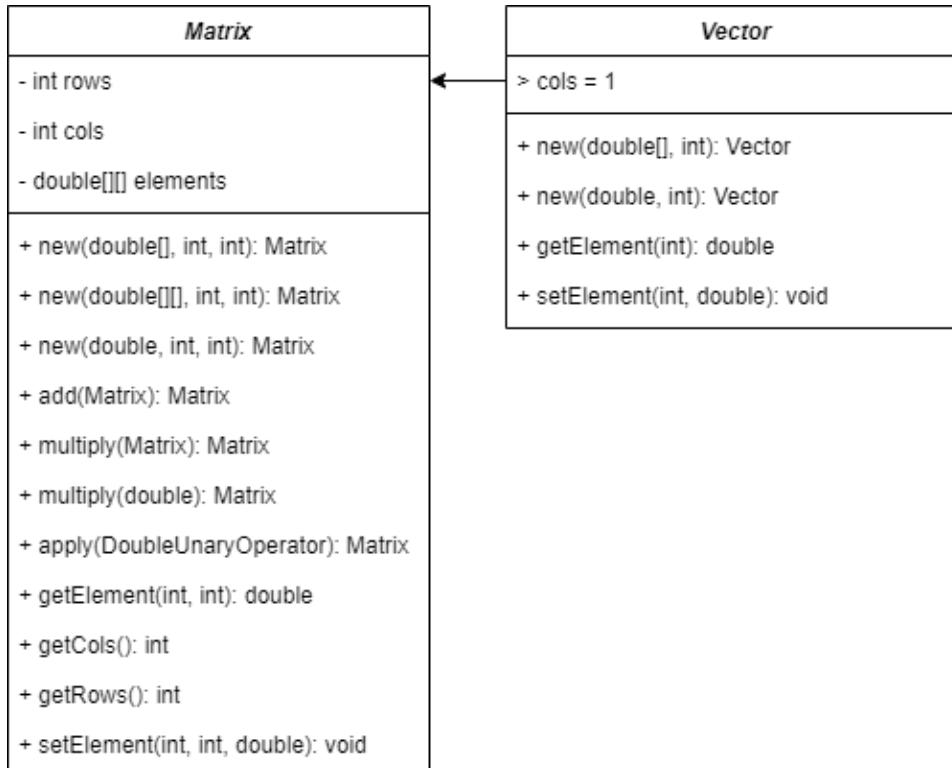
- Can the program be navigated in its entirety?
- Can a model be trained?
- Can the trained model be saved?
- Can a user-created model be loaded?
- Can a prediction be made from user input?
- How accurate are the models?

I will then give a group of others a copy of the program with the following survey for beta testing:

Feature	Question	Strongly Disagree	Disagree	=	Agree	Strongly agree	Describe issues
Usability	Are the buttons functional?						
	Is the layout well-structured and neat?						
	Is the UI cluttered?						
User-friendliness	Does the UI feel overwhelming?						
	Did you understand what to do?						
Aesthetic	Does the colour scheme look good?						
	Is the font appropriate?						
Accessibility	Are the meanings of the icons clear?						
	Is the language too technical?						

Stage I: Core Mathematical Classes

For the linear algebra library, there will be a matrix class which the vector class will inherit from. It can be represented with the below class diagram. The matrix class has a polymorphic constructor that can either take a 1D array of size `rows * cols`, a 2D array of dimensions `[rows][cols]`, or a double which the entire matrix will be initialised to. It always takes the dimensions. There are methods for multiplication, addition, and for applying a function to the whole matrix. Vectors extend matrices, but sets `cols` to 1.



First, I added all constructors to the Matrix class.

```
Matrix.java
1 package linearAlgebra;
2
3 import java.util.Arrays;
4
5 no usages
6 public class Matrix {
7     3 usages
8     private int rows, cols;
9     5 usages
10    private double[][] elements;
11
12    9 usages
13    public Matrix(double[] e, int rows, int cols) {
14        this.rows = rows;
15        this.cols = cols;
16        this.elements = new double[rows][cols];
17
18        for (int i = 0; i < rows; ++i) {
19            for (int j = 0; j < cols; ++j) {
20                this.elements[i][j] = e[cols + rows * i];
21            }
22        }
23    }
24
25    9 usages
26    public Matrix(double[][], e, int rows, int cols) {
27        this.rows = rows;
28        this.cols = cols;
29        this.elements = e;
30    }
31
32    9 usages
33    public Matrix(double e, int rows, int cols) {
34        this.rows = rows;
35        this.cols = cols;
36        this.elements = new double[rows][cols];
37
38        for (int i = 0; i < rows; ++i) {
39            Arrays.fill(this.elements[i], e);
40        }
41    }
42}
```

Error: `ArrayIndexOutOfBoundsException` caused by 1D array constructor.

Fix: 1D input array was being accessed in the wrong way (using `cols` instead of iterator `j`).

```
this.elements[i][j] = e[j + rows * i];
```

Error: 1D array constructor works does not instantiate the correct matrix.

Fix: use `cols` instead of `rows` when going through the input array.

```
this.elements[i][j] = e[j + cols * i];
```

If an error occurs in any of the methods of this class, perhaps when there are mismatching inputs, I have decided it will return a matrix full of negative 1's of the same dimensions as the matrix instance. All attributes are encapsulated with only specific access and mutation limits as to minimise logic errors in the future. The matrix multiplication method can take either a matrix or a double; for doubles, the multiply method just uses the apply method, which takes a lambda function and applies it to all elements in the matrix. The add function simply adds all corresponding elements in each matrix to form a new one. The vector class extends the matrix class, as it is just a 1-column case.

Matrix multiplication pseudocode:

```
FUNCTION multiply(m, n)
    IF m.cols != n.rows
        RETURN Matrix(-1, m.rows, m.cols)
    ELSE
        DOUBLE[][] res = DOUBLE[m.rows][n.cols]
        FOR i = 0 TO m.rows - 1
            FOR j = 0 TO n.cols - 1
                DOUBLE productSum = 0
                FOR k = 0 TO m.cols - 1
                    productSum += m[i][k] * n[k][j]
                ENDFOR
                res[i][j] = productSum
            ENDFOR
        ENDFOR
        RETURN Matrix(res, m.rows, n.cols)
    ENDIF
```

ENDFUNCTION

```
no usages
37 public int getRows() {
38     return rows;
39 }
40
no usages
41 public int getCols() {
42     return cols;
43 }
44
no usages
45 public double getElement(int row, int col) {
46     return elements[row][col];
47 }
48
no usages
49 public void setElement(int row, int col, double element) {
50     elements[row][col] = element;
51 }

no usages
54 @
55 public Matrix add(Matrix m) {
56     if (!m.getRows() == rows && m.getCols() == cols) {
57         return new Matrix(-1, rows, cols);
58     } else {
59         double[][] res = new double[rows][cols];
60
61         for (int i = 0; i < rows; ++i) {
62             for (int j = 0; j < cols; ++j) {
63                 res[i][j] = elements[i][j] + m.getElement(i, j);
64             }
65         }
66
67         return new Matrix(res, rows, cols);
68     }
69 }

1 usage
70 public Matrix apply(DoubleUnaryOperator f) {
71     double[][] res = new double[rows][cols];
72
73     for (int i = 0; i < rows; ++i) {
74         for (int j = 0; j < cols; ++j) {
75             res[i][j] = f.applyAsDouble(elements[i][j]);
76         }
77     }
78
79     return new Matrix(res, rows, cols);
80 }
81
```

```

82 // A (m x n) . B (p x q) = C (m x q), given n == p
no usages
83 @ public Matrix multiply(Matrix m) {
84     if (cols != m.getRows()) return new Matrix(e, -1, rows, cols);
85     else {
86         double[][] res = new double[rows][m.getCols()];
87
88         for (int i = 0; i < rows; ++i) {
89             for (int j = 0; j < m.getCols(); ++j) {
90                 double productSum = 0;
91                 for (int k = 0; k < cols; ++k) {
92                     productSum += elements[i][k] * m.getElement(k, j);
93                 }
94
95                 res[i][j] = productSum;
96             }
97         }
98
99         return new Matrix(res, rows, m.getCols());
100     }
101 }
no usages
103 public Matrix multiply(double n) {
104     return apply(x -> (n * x));
105 }

```

```

3 public class Vector extends Matrix {
4     3 usages
5     public Vector(double[] e, int rows) {
6         super(e, rows, cols: 1);
7     }
8
9     3 usages
10    public Vector(double e, int rows) {
11        super(e, rows, cols: 1);
12    }
13
14    no usages
15    public double getElement(int row) {
16        return super.getElement(row, col: 0);
17    }
18
19    no usages
20    public void setElement(int row, double element) {
21        super.setElement(row, col: 0, element);
22    }
23 }

```

No errors: Everything works correctly.

I will also add some static utility methods for printing, or commonly used mathematical functions. The only ones I think I will need are a print matrix procedure and a sigmoid activation function (commonly used in ML):

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

```

no usages
5 public class Utils {
    no usages
6 @ public static void printMatrix(Matrix m) {
7     for (int i = 0; i < m.getRows(); ++i) {
8         System.out.print("[ ");
9         for (int j = 0; j < m.getCols(); ++j) {
10            System.out.print(m.getElement(i, j) + " ");
11        }
12        System.out.print("\n");
13    }
14 }
15 }

no usages
3 public class Functions {
    no usages
4 public static double sigmoid(double z) {
5     return 1 / (1 + Math.exp(-z));
6 }
7 }
8

```

No errors: All functions work correctly.

Stage	Part	Test	Type	Test Data	Expected Result	Actual Result
/	a	Matrix multiplication and addition works	Valid	Multiply randomly generated matrices	The correct resulting matrix (check with calculator)	<pre> M = [1.0 2.0] [3.0 4.0] N = [5.0 6.0] [7.0 8.0] M.N = [19.0 22.0] [43.0 50.0] M+N = [6.0 8.0] [10.0 12.0] </pre>
	b	Activation functions take floats or matrices	Valid	Apply the function to randomly generated floats and matrices	No errors in calculation	<pre> σ(2) = 0.8807970779778823 Let M = [1.0 2.0] [3.0 4.0] σ(M) = [0.7310585786300049 0.8807970779778823] [0.9525741268224334 0.9820137900379085] </pre>

c	Multiplication of two matrices where the number of columns of the first does not equal the number of rows of the second does not work	Invalid	Multiply two different matrices with wrong dimensions	-1 matrix	<pre>[1.0 2.0 3.0] [4.0 5.0 6.0] [7.0 8.0 9.0] * [1.0 2.0 3.0] [4.0 5.0 6.0] = [-1.0 -1.0 -1.0] [-1.0 -1.0 -1.0] [-1.0 -1.0 -1.0]</pre>
d	Print methods work	Valid	Attempt to print a matrix	The correct output matrix printed	<pre>[1.0 2.0 3.0] [4.0 5.0 6.0] [7.0 8.0 9.0]</pre>

I also added some comments explaining each method, to aid future development.

```
// Instantiate with 1D Array
public Matrix(double[] e, int rows, int cols) {

// Instantiate with 2D array
public Matrix(double[][] e, int rows, int cols) {

// Instantiate entire matrix with 1 double
public Matrix(double e, int rows, int cols) {

// Adds two matrices of the same dimension
public Matrix add(Matrix m) {

// Adds corresponding elements
res[i][j] = elements[i][j] + m.getElement(i, j);

// Applies a function to each element of the array
public Matrix apply(DoubleUnaryOperator f) {

// A (m x n) . B (p x q) = C (m x q), given n == p
public Matrix multiply(Matrix m) {
```



```
// Dot product of each of A's rows with each of B's columns
for (int i = 0; i < rows; ++i) {
    for (int j = 0; j < m.getCols(); ++j) {
        double productSum = 0;
        for (int k = 0; k < cols; ++k) {
            productSum += elements[i][k] * m.getElement(k, j);
        }

        res[i][j] = productSum;
    }
}

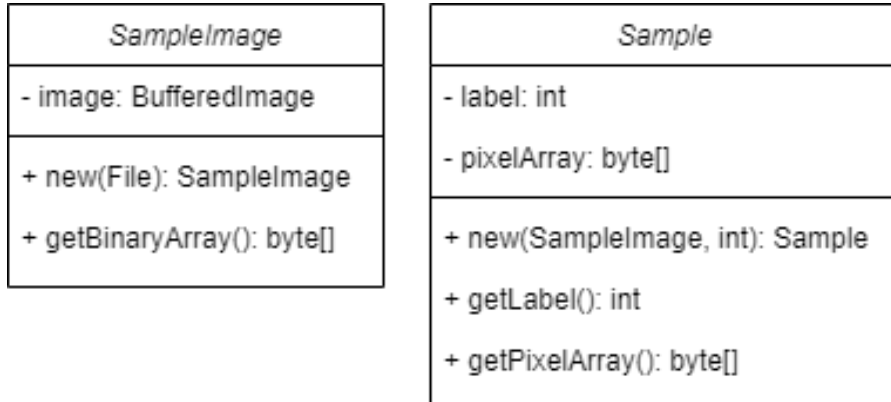
// Multiply by scalar
public Matrix multiply(double n) { return apply(x -> (n * x)); }
```

That concludes stage I. I am happy with this stage as I completed everything I needed and it is all functional. All errors had easy fixes as most of them were just logic errors.

Feature	Component	Description	Justification	Completion
AI/ML Processes	Linear Algebra Library	A library containing definitions, functions and operations involving vectors and matrices.	Necessary for AI/ML, as models are neural networks made from layers (vectors), weight matrices and bias vectors.	

Stage II: Basic Input

The focus of this stage is to be able to take images as input, break them down into 1D binary arrays and make samples from them. The class diagrams are as follows:



The function `getBinaryArray()` will follow a simple algorithm to gather a 1D binary array from a 28x28 image:

```
GLOBAL image

FUNCTION getBinaryArray()
    BYTE[784] binaryArray
    FOR i = 0 TO 27
        FOR j = 0 TO 27
            binaryArray[j + 28 * i] = image.RGB(i, j) == BLACK
        ENDFOR
    ENDFOR
    RETURN binaryArray
ENDFUNCTION
```

I programmed the sample image class constructor first.

```

public SampleImage(File file) {
    try {
        image = ImageIO.read(file);
    } catch (IOException ioE) {
        System.err.println(Arrays.toString(ioE.getStackTrace()));
    }
}

```

It became apparent that reading the image file could throw an `IOException`, so I added a try-catch case to print the error stack trace in that event.

```

[java.desktop/javafx.imageio.ImageIO.read(ImageIO.java:1310), Sampling.SampleImage.<init>(SampleImage.java:14), Main.main(Main.java:9)]
[java.desktop/javafx.imageio.ImageIO.read(ImageIO.java:1310), Sampling.SampleImage.<init>(SampleImage.java:14), Main.main(Main.java:10)]
[java.desktop/javafx.imageio.ImageIO.read(ImageIO.java:1310), Sampling.SampleImage.<init>(SampleImage.java:14), Main.main(Main.java:11)]
Successfully created objects:
Sampling.SampleImage@5b37e0d2
Sampling.SampleImage@4459eb14
Sampling.SampleImage@5a2e4553

```

Error: The objects were created, but the images could not be read.

Fix: Used absolute file paths to select images. Not ideal, need to find a way to use relative file paths so it works on all machines.

Relative file paths did not seem to work at first, but I had just considered the current directory as the one containing the main class; it was actually the project directory.

I then implemented the `getBinaryArray()` method.

```

public byte[] getBinaryArray() {
    byte[] binaryArray = new byte[784];
    for (int i = 0; i < 28; ++i) {
        for (int j = 0; j < 28; ++j) {
            binaryArray[j + 28 * i] = (byte) (image.getRGB(i, j) == 0x00000000 ? 1 : 0);
        }
    }
    return binaryArray;
}

```

Error: Array of only 0's is always generated.

Fix: RGB value will equal `0xFF000000` if black, not `0x00000000`, as the alpha value byte is equal to 255.

```

binaryArray[j + 28 * i] = (byte) (image.getRGB(i, j) == 0xFF000000 ? 1 : 0);

```

Error: Generates a transposed version of the image (rows and columns seemingly switched)

Fix: `i` and `j` iterators were the wrong way around when accessing RGB values.

```

image.getRGB(j, i)

```

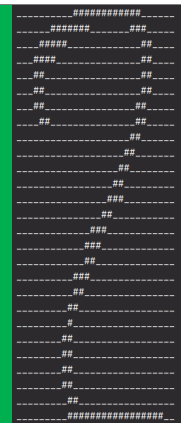
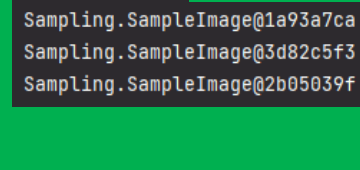
Moving on to the sample class, all it needed were the private attributes, a basic constructor, and access functions.


```
public class Sample {
    private int label;
    private byte[] pixelArray;

    public Sample(SampleImage image, int label) {
        this.label = label;
        this.pixelArray = image.getBinaryArray();
    }

    public int getLabel() {
        return label;
    }

    public byte[] getPixelArray() {
        return pixelArray;
    }
}
```

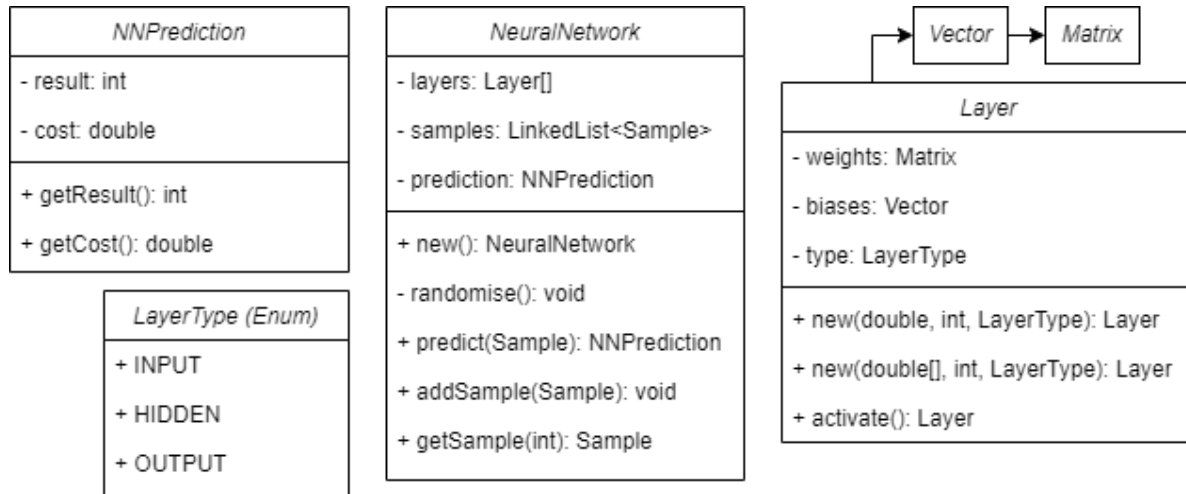
Stage	Part	Test	Type	Test Data	Expected Result	Actual Result
//	a	Correct binary array generated from image	Valid	Process 3 images from the pre-installed image library	A binary array, with 1s for black pixels and 0s for white pixels, in row-priority order	
	b	Sample class instance generates without	Valid	Process 3 images from the pre-	Sample classes with the correct	

	errors from images alone	installed image library	label and pixel array	
--	-----------------------------	-------------------------------	--------------------------	---

Concluding stage II, I am also happy with this stage as samples can now be created from images. It wasn't a large stage, so it wasn't a difficult one.

Stage III: Neural Network Classes

The objective in this stage is to create the classes that will be central to the neural networks. I have decided that all networks generated by the application will consist of a 784-node input layer (which will take the sample binary arrays), two 16-node hidden layers, and a 10-node output layer, which will contain the probability of each digit. The class diagrams are displayed below, but are subject to change, as I'm not sure what extra classes I may need in the future, and I have yet to implement backwards propagation (stage IV will include this).



It occurred to me that I need to change the modifiers of all the Matrix class's attributes to protected, so all its subclasses (Vector and now Layer as a 2nd order subclass), can access their own information without having to use methods of the superclass.

```
protected int rows, cols;
protected double[][] elements;
```

I also realised I needed to add access and mutation functions to the layer class, that allow reading and writing to the weights and biases. They only need be changed through altering; that is to say, you will only ever need to increment or decrement their value by a certain amount, never reset it entirely. The layer's type cannot be changed once it is created.

```

public Matrix getWeights() {
    return weights;
}

public void alterWeight(int row, int col, double value) {
    weights.setElement(row, col, element: weights.getElement(row, col) + value);
}

public Vector getBiases() {
    return biases;
}

public void alterBias(int row, double value) {
    biases.setElement(row, element: biases.getElement(row) + value);
}

public LayerType getType() {
    return type;
}

```

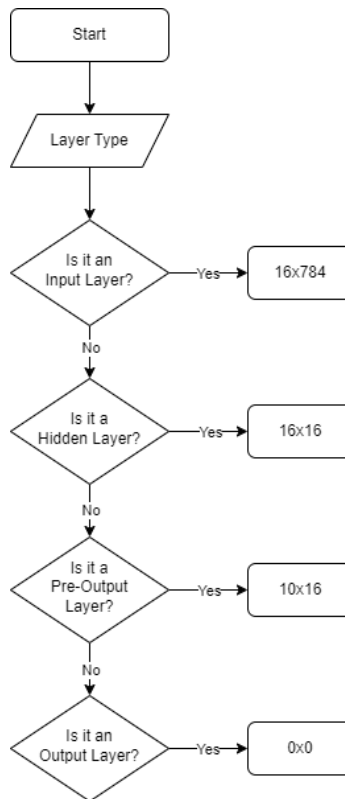
When creating a layer instance, you can set the dimensions the layer itself, and its weight and bias objects purely from its type. This eliminates the need for any input to the constructor other than the layer type. This does however create the need for an extra layer type – pre-output, since if a layer is hidden, it is assumed to next layer is also hidden, which is not true for the pre-output layer.

```

public enum LayerType {
    INPUT,
    HIDDEN,
    PRE_OUTPUT,
    OUTPUT;
}

```

The algorithm for deciding the dimensions of the weight matrices can be shown in the following flowchart. The dimensions of the bias vector are the same as the layer itself. I also decided to add all of the dimensions as constants in the utils class, just so I don't have to memorise anything or change lots of uses of the constants if I decided to refactor anything.



```

public static final int INPUT_LAYER_SIZE = 784;
public static final int HIDDEN_LAYER_SIZE = 16;
public static final int OUTPUT_LAYER_SIZE = 10;

```

The constructor the layer class ultimately used a lot of conditional statements, so I added comments listing all of the dimensions to help with any future confusion.

```

weights =
    // Weight matrix dimensions in form LayerType x LayerType
    // (Where the type indicates the size of that layer type)
    type == LayerType.INPUT
    // INPUT : hidden x input
    ? new Matrix( e: 0, Utils.HIDDEN_LAYER_SIZE, Utils.INPUT_LAYER_SIZE) :
    type == LayerType.HIDDEN
    // HIDDEN : hidden x hidden
    ? new Matrix( e: 0, Utils.HIDDEN_LAYER_SIZE, Utils.HIDDEN_LAYER_SIZE) :
    type == LayerType.PRE_OUTPUT
    // PRE-OUTPUT : output x hidden
    ? new Matrix( e: 0, Utils.OUTPUT_LAYER_SIZE, Utils.HIDDEN_LAYER_SIZE)
    // OUTPUT : 0 x 0 (needs no weights)
    : new Matrix( e: 0, rows: 0, cols: 0);

```

Next, I started on the neural network class. The constructor takes no input, and simply just creates an empty network. I chose to use a linked list structure for storing the samples because it is quicker than an array list to add items, whereas array list is quicker to get items; Adding

samples will be done far more than fetching them. The following is the start of the NN class and the simple NN prediction class.

```
public class NeuralNetwork {
    private Layer[] layers;
    private LinkedList<Sample> samples;
    private NNPrediction prediction;

    public NeuralNetwork() {
        layers = new Layer[5];
        layers[0] = new Layer(LayerType.INPUT);
        layers[1] = new Layer(LayerType.HIDDEN);
        layers[2] = new Layer(LayerType.HIDDEN);
        layers[3] = new Layer(LayerType.PRE_OUTPUT);
        layers[4] = new Layer(LayerType.OUTPUT);

        samples = new LinkedList<>();
    }
}

public class NNPrediction {
    private int result;
    private double cost;

    public NNPrediction(int result, double cost) {
        this.result = result;
        this.cost = cost;
    }

    public int getResult() { return result; }

    public double getCost() { return cost; }
}
```

For the NN class methods, the access and mutation ones are quite self-explanatory. I also added a `removeSample(int): void` method, since I need that functionality also.

```
public void addSample(Sample sample) {
    samples.add(sample);
}

public void removeSample(int idx) {
    samples.remove(idx);
}

public Sample getSample(int idx) {
    return samples.get(idx);
}
```

The other methods are a bit more complicated. The `randomise` method goes through every weight and bias in every layer and assigns it a small random value between -1 and 1 .

GLOBAL layers

PRIVATE PROCEDURE `randomise()`

```

FOR layer IN layers
    FOR i = 0 TO layer.getWeights().getRows()
        DOUBLE random
        FOR j = 0 TO layer.getWeights().getCols()
            random = RANDOM(-1, 1)
            layer.alterWeight(i, j, random)
        ENDFOR
        random = RANDOM(-1, 1)
        layer.alterBias(i, random)
    ENDFOR
ENDFOR
ENDPROCEDURE

```

```

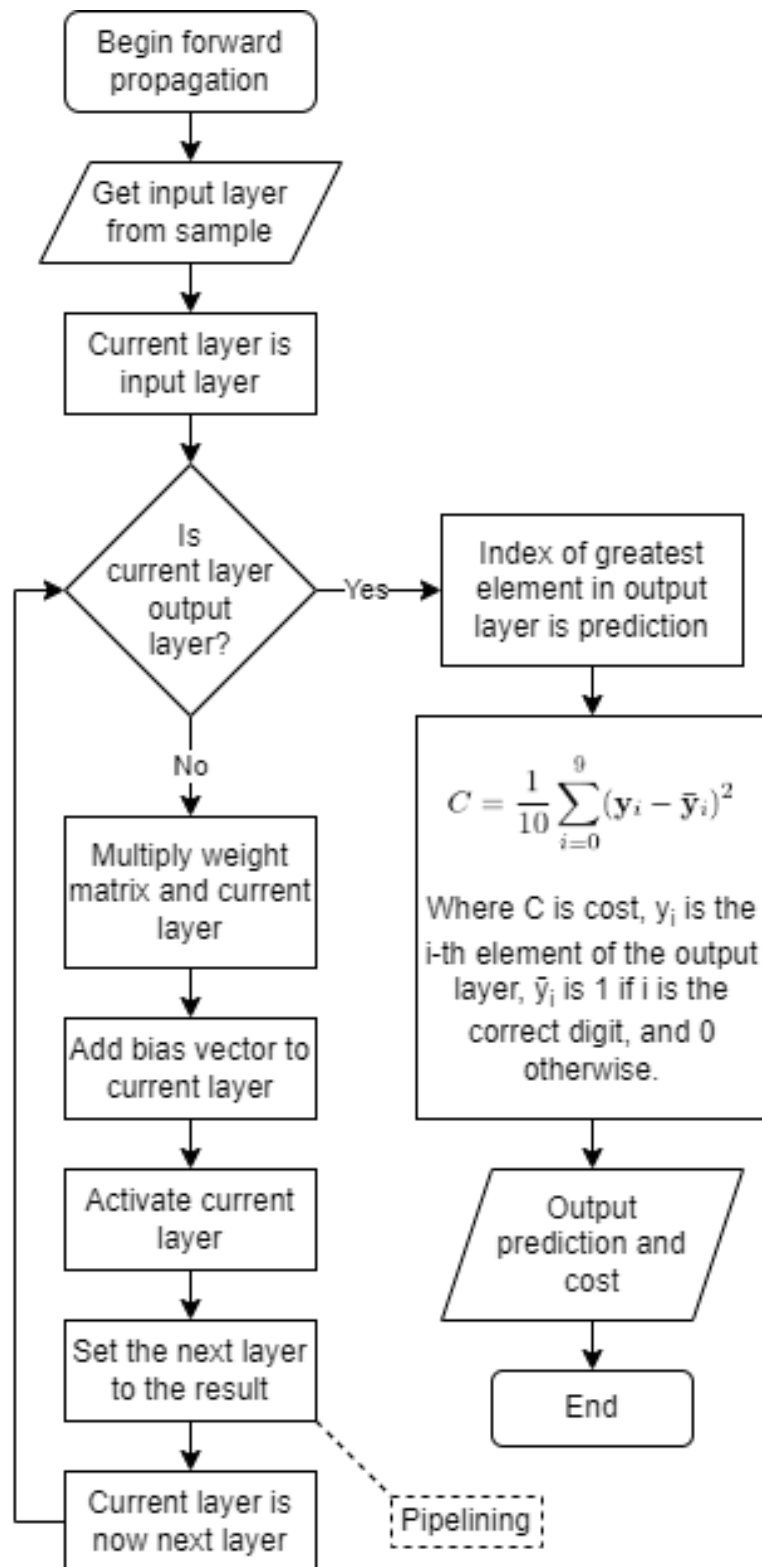
private void randomise() {
    for (Layer layer : layers) {
        for (int i = 0; i < layer.getWeights().getRows(); ++i) {
            double rand;
            for (int j = 0; j < layer.getWeights().getCols(); ++j) {
                // Random value between -1 and 1
                rand = Math.random() * 2 - 1;
                layer.alterWeight(i, j, rand);
            }
            rand = Math.random() * 2 - 1;
            layer.alterBias(i, rand);
        }
    }
}

```

All Weights and biases randomise.

The neural network is randomised upon instantiation.

The algorithm for forward propagation through the network makes use of pipelining; each layer is weighted and biased, then is sent to the next layer, where the same thing happens again. It can be visualised in the following flowchart.



The cost is calculated as the mean squared error (MSE) of the network. It can also be formulated as: $C = \frac{1}{10} |y - \bar{y}|^2$. It became appropriate to add a method to the layer class that will allow me to set the layer as an array, not just one by one as elements, since this would be far more efficient. I also needed to change the type of the pixel array of the samples from `byte[]` to `int[]`, since bytes cannot be cast to doubles easily (it seems that bytes are more often put together to make up a double, rather than converted to a double). This is less memory efficient but saves a lot of trouble.

```

public void setLayer(double[] array) {
    if (array.length == getRows()) {
        elements[0] = array;
    } else {
        throw new IndexOutOfBoundsException("Attempted to set layer to mismatched array");
    }
}

```

Error: only sets first element in layer

Fix: Set the element in each row to the corresponding element in the input array

```

public void setLayer(double[] array) {
    if (array.length == rows) {
        for (int i = 0; i < rows; ++i) {
            setElement(i, col: 0, array[i]);
        }
    } else {
        throw new IndexOutOfBoundsException("Attempted to set layer to mismatched array");
    }
}

```

I'm also going to add a magnitude function to the vector class, as this will simplify the code for the prediction method i.e., instead of writing all of the loops necessary for the cost calculation, I could just utilise the method. I then noticed that only the squared magnitude will be used in calculations, so it is computationally inefficient having the magnitude function use a square root, only to square it again immediately after. For this reason, I rewrote the function such that the squared magnitude is calculated, then the actual magnitude can be calculated by square rooting that result if needs be.

<pre> public double magnitude() { return Math.sqrt(Arrays.stream(asArray()) .map(e -> Math.pow(e, 2)) .sum()); } protected double[] asArray() { double[] elems = new double[rows]; for (int i = 0; i < rows; ++i) { elems[i] = elements[i][0]; } return elems; } </pre>	<pre> public double square_magnitude() { return Arrays.stream(asArray()) .map(e -> Math.pow(e, 2)) .sum(); } public double magnitude() { return Math.sqrt(square_magnitude()); } </pre>
---	---

Works as expected. Also wrote method `asArray()` which returns the vector in the form of a 1D array, which may be useful later.

It also became apparent to me that there is no need for a prediction attribute on the neural network class – my initial idea was to cache predictions for given samples so it wouldn't have to recalculate them, but I now don't think it's necessary. The prediction method is shown below, along with a short utility function to create one-hot vectors for given digits (\bar{y} as seen in the flowchart):

```
public NNPrediction predict(Sample sample) {
    Layer currentLayer;
    int currentLayerIdx;

    // Set input layer to input pixel array
    layers[0].setLayer(
        Arrays.stream(sample.getPixelArray())
            .asDoubleStream()
            .toArray()
    );

    // Current layer is input layer
    currentLayerIdx = 0;
    currentLayer = layers[currentLayerIdx];
    // Loop if current layer is not output layer
    Layer result;
    while (currentLayerIdx != 5) {
        result = (Layer) currentLayer
            // Multiply weight matrix
            .getWeights()
            .multiply(currentLayer)
            // Add biases
            .add(currentLayer.getBiases());
        // Activate layer
        result = result.activate();
        // Set next layer to result
        // and increment current layer
        layers[++currentLayerIdx] = result;
        currentLayer = layers[currentLayerIdx];
    }

    // Current layer is now output layer
    int prediction_digit = 0;
    double cost;

    for (int i = 0; i < 10; ++i) {
        double element = currentLayer.getElement(i);
        // Finds the greatest element,
        // the index of which is the prediction
        prediction_digit =
            element > prediction_digit
            ? i : prediction_digit;
    }

    //  $y - \bar{y}$ 
    Vector error_vector =
        (Vector) currentLayer
            .add(Utils.oneHotVector(sample.getLabel()).multiply(-1));
    //  $C = |y - \bar{y}|^2 / 10$ 
    cost = error_vector.square_magnitude() / 10;

    return new NNPrediction(prediction_digit, cost);
}
```

Error: Having trouble casting Matrix type to Layer type

Fix: Manually move every element from Matrix to Layer

```
public void setLayer(double[] array) {  
    if (array.length == rows) {  
        for (int i = 0; i < rows; ++i) {  
            setElement(i, col: 0, array[i]);  
        }  
    } else {  
        throw new IndexOutOfBoundsException("Attempted to set layer to mismatched array");  
    }  
}
```

Error: Now all Matrix to Layer casts don't work. May have to initialise everything to a matrix then directly cast it element by element to a new layer instance.

Fix: Remove all explicit Matrix to Layer casts, and manually transfer all data to a new layer instance instead

```
public Layer activate() {  
    Matrix result = apply(Functions::sigmoid);  
    double[] elems = new double[rows];  
    // Resets the layer directly from matrix  
    // (Explicit cast didn't work)  
    for (int i = 0; i < rows; ++i) {  
        elems[i] = result.getElement(i, col: 0);  
    }  
    Layer l = new Layer(type);  
    l.setLayer(elems);  
    return l;  
}
```

Error: Array out-of-bounds exception for accessing the layers list

Fix: Last index of a 5-element array is 4

```
while (currentLayerIdx < 4) {
```

Error: Matrix multiplication/addition results giving -1 matrices, implying the dimensions are mismatched.

Fix: Was attempting to add mismatched bias vectors. This is due to bias vectors being given the same dimensions as their layer upon initialisation, when they need to be given the same dimensions as the next layer.

```

biases =
    // Bias vector has the dimensions of the NEXT layer
    type == LayerType.INPUT ? new Vector(0, Utils.HIDDEN_LAYER_SIZE) :
    type == LayerType.HIDDEN ? new Vector(0, Utils.HIDDEN_LAYER_SIZE) :
    type == LayerType.PRE_OUTPUT ? new Vector(0, Utils.OUTPUT_LAYER_SIZE) : null;

```

Error: Forgot to account for nullified weights/biases on the output layer

Fix: Don't try and access weights or biases for the output layer

```

if (layer.getWeights() == null) return;

```

Error: Apparently layers are being set incorrectly now. Realise I wasn't resetting the result after every process, as the functions return new objects, as opposed to being void procedures that just reset it.

After debugging, it seems the weight matrices for hidden layers are not initialised to random values, but stay as 0's.

Update: Weight matrices initialise fine but get changed to 0's later for some reason.

Fix: Activation function was returning a new layer without the weights of its original instantiation. Changed it to a procedure that just activates the layer (makes more sense this way anyway).

```

public void activate() {
    Matrix result = apply(Functions::sigmoid);
    double[] elems = new double[rows];
    // Resets the layer directly from matrix
    // (Explicit cast didn't work)
    for (int i = 0; i < rows; ++i) {
        elems[i] = result.getElement(i, col: 0);
    }
    setLayer(elems);
}

```

Error: Prediction is always 1

Fix: Was not assessing the greatest element in the output layer correctly. The whole forward propagation process now works.

```

double greatestElement = 0;
for (int i = 0; i < 10; ++i) {
    double element = currentLayer.getElement(i);
    // Finds the greatest element,
    // the index of which is the prediction
    if (element > greatestElement) {
        greatestElement = element;
        prediction_digit = i;
    }
}

```

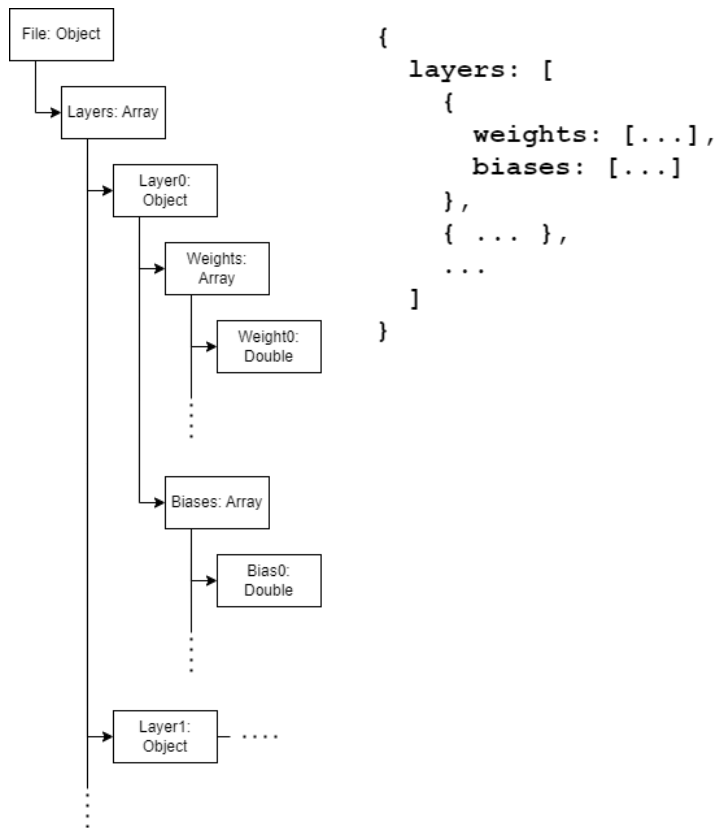
Below is the one-hot vector utility function.

```

public static Vector oneHotVector(int digit) {
    double[] elements = new double[10];
    // All elements 0,
    // except the one with the correct digit, which is 1
    elements[digit] = 1;
    return new Vector(elements, rows: 10);
}

```

The final part will be to store networks in JSON format. It will have a structure as follows:



It will be necessary to add a method to the neural network class to store the neural network as a JSON file, in a special 'networks' directory. I will add a function to load it in stage VII. The method and object pseudocode are shown below.

```
PRIVATE CLASS NetworkObject
```

```
    PUBLIC LayerObject[] layers
```

```
    PUBLIC PROCEDURE NetworkObject(LayerObject[] layers)
```

```
        this.layers = layers
```

```
    ENDPROCEDURE
```

```
ENDCLASS
```

```
PRIVATE CLASS LayerObject
```

```
    PUBLIC DOUBLE[][] weights
```

```
    PUBLIC DOUBLE[] biases
```

```
    PUBLIC PROCEDURE LayerObject(DOUBLE[][] weights,DOUBLE[] biases)
```

```
        this.weights = weights
```

```

        this.biases = biases
    ENDPROCEDURE
ENDCLASS

```

I will make use of Java's record classes to reduce boilerplate in the private classes here.

```

PROCEDURE storeAsJSON (STRING fileName)

    FILE file

    NetworkObject NNO

    LayerObject[] layerObjs = new LayerObject[5]

    FOR i FROM 1 TO 4

        // 'this' is the NN class

        LayerObject[i] = new
            LayerObject(this.layers[i].getWeights(),
                this.layers[i].getBiases())

    ENDFOR

    NNO = new NetworkObject(layerObjs)

    IF NOT fileName ENDS WITH ".json"
        fileName = fileName + ".json"
    ENDIF

    TRY file = FILE("/Networks/" + fileName)
    CATCH fileIOException
        OUTPUT(fileIOException)
    ENDTRY

    MAP(NNO TO FILE)

ENDPROCEDURE

```

I will be using the Jackson API for JSON serialisation. I've also decided to add an access function for the elements to the Matrix class, which will return them as a 2D array. I also needed then to refactor the name of the method with the same purpose in the Vector class.

<pre>public double[][] asArray() { return elements; }</pre>	<pre>public double[] as1DArray() { double[] elems = new double[rows]; for (int i = 0; i < rows; ++i) { elems[i] = elements[i][0]; } return elems; }</pre>	<p>REFACTORED:</p> <p>asArray -> as1DArray</p>
---	--	---

The whole store as JSON function and classes are shown below.

```

/*
Abstractions of Layer / NeuralNetwork classes
containing only weight and bias values.
*/
private record LayerObject(double[][] weights, double[] biases) {}
private record NetworkObject(LayerObject[] layers) {}

public void storeAsJSON(String fileName) {
    File file;
    NetworkObject NNO;

    // Copy all weights and biases to layer objects
    LayerObject[] layerObjs = new LayerObject[5];
    for (int i = 0; i < 5; ++i) {
        layerObjs[i] = new LayerObject(
            this.layers[i].getWeights().toArray(),
            this.layers[i].getBiases().as1DArray()
        );
    }
    NNO = new NetworkObject(layerObjs);

    if (!fileName.endsWith(".json")) {
        fileName = fileName.concat( str: ".json");
    }
    // Create JSON file
    file = new File( pathname: "./Networks/" + fileName);

    // ObjectMapper can write Objs as JSON
    ObjectMapper mapper = new ObjectMapper();
    // String is an error message by default, which is reset if rewritten
    String json = "ERROR: COULD NOT WRITE JSON";
    try {
        // Neural Network Object as JSON
        json = mapper.writeValueAsString(NNO);
    } catch (JsonProcessingException e) {
        e.printStackTrace();
    }

    try {
        // Write JSON to file
        FileWriter writer = new FileWriter(file);
        writer.write(json);
        writer.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

Error: Weights returning null for final layer, which is correct; I just forgot to account for it.

Fix: Only try to access weights and biases up to the 4th layer

Error: Jackson API classes are undefined since some of their dependencies are not installed.

Fix: Install dependencies

Function works as expected.

Also added this condition to stop empty samples getting in.

```
public void addSample(Sample sample) {  
    if (Arrays.stream(sample.getPixelArray()).allMatch(x -> x == 0)) {  
        System.out.println("Empty Sample, not adding");  
        return;  
    }  
    samples.add(sample);  
}
```

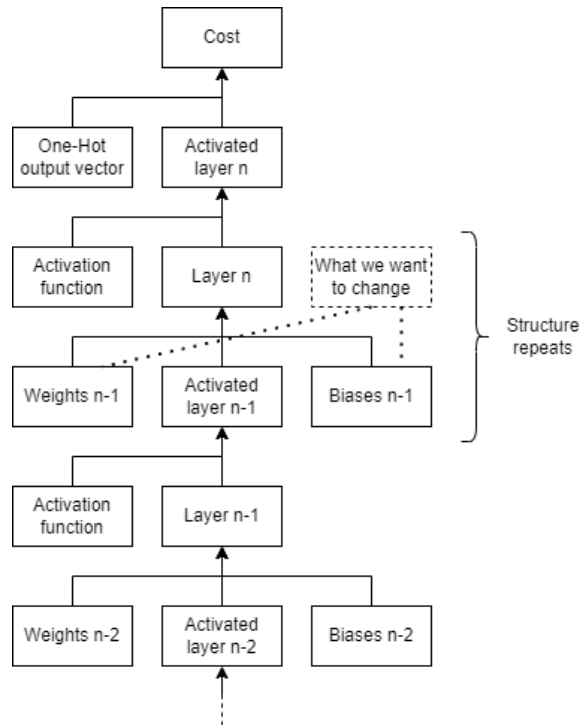
Stage	Part	Test	Type	Test Data	Expected Result	Actual Result
III	a	Neural network takes sample as input without error, and gives a prediction	Valid	Process 3 images from the pre-installed	A prediction (not necessarily correct)	result: 8 cost: 0.172053
	b	Neural network does not take a blank sample (all 0 binary array)	Invalid	Input a blank sample	An error	Empty Sample, not adding
	c	Neural network stores as JSON correctly	Valid	Attempt to save NN as JSON	JSON with weights and biases stored in row-priority order	["Layers": [{"weights": [{"-0.9652819979261971, -0.2887245871427, 0.24898298917325823, -0.1588230793730312, 0.5258732859525785, -0.968583705345868, -0.1776197132986181, 0.31244980431084477, 0.9276145844398693, -0.05671062145994288, -0.9030964771860976, 0.6018976532210324, -0.921750134817708, 0.37011567464608786, -0.5974143813874959, -0.026596483354137446, 0.5140792343584353, 0.4415346781303804, 0.44513726667989006, 0.5747798906921788, 0.34895021048921754, -0.0649268676611328, 0.3072767409221042, 0.8761627807853281, -0.6844955371946781, 0.5288242574198725, -0.597164004586271, 0.5826572091228694, -0.003915163628989804, 0.7059724232960885, 0.14072521486515543, 0.7797150881331965, -0.9824243654533298, 0.48257653407326595, 0.7024267101565793, 0.6035755296060262, -0.6184650999064855, 0.9310096746972991, 0.1848335144571971, 0.48264030051979856, -0.04385314840635113, 0.3649852164215919, -0.9987306500325124, -0.6864193000350076, 0.42693161455793893, 0.44763994349939207, -0.708580040343677, 0.1824771039633224, -0.21283295003703606, 0.7523445369191455, 0.5639918097214132, -0.8138816385277579, -0.3879025290311999, 0.39703007818580227, -0.6761660809480499, -0.5129595121307264}

<i>Feature</i>	Component	Description	Justification	Completion
<i>AI/ML Processes</i>	Neural Network Class	A class holding number and sizes of layers, functions to get predictions, calculate cost, back-propagate and save all weights and biases in a JSON file.	Would have algorithms implemented for	

That concludes stage III. Again, this stage was mainly class creation, so wasn't too difficult. Initially, I wasn't planning on having to install a library for JSON creation, but I think it was the right call – implementing that functionality myself would have used a lot of precious time.

Stage IV: Machine Learning

The way to work out how much a weight should alter by is by using the chain rule. A method should be created to adjust all weights and biases based on a given prediction, with the sample that was used to create the prediction. A learning rate attribute should also be given to the neural network. Mathematically, the weight and bias adjustment can be expressed by the following:



$$C = \frac{1}{10} \sum_{i=0}^9 (a_i^{(n)} - \bar{y}_i)^2$$

$$a_i^{(n)} = \sigma(z_i^{(n)})$$

$$z_i^{(n)} = \left[\sum_{k=0}^{\dim a^{(n-1)}} w_{i,k}^{(n-1)} \cdot a_k^{(n-1)} \right] + b_i^{(n-1)}$$

So then, we arrive at a final formula for weight and bias adjustment in the penultimate layer. In general:

$$\frac{\partial C}{\partial w^{(n-1)}} = \frac{\partial C}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w^{(n-1)}}$$

$$\text{or } \frac{\partial C}{\partial b^{(n-1)}} = \frac{\partial C}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial b^{(n-1)}}$$

Each term:

$$\frac{\partial C}{\partial a_i^{(n)}} = \frac{1}{5} (a_i^{(n)} - \bar{y}_i)$$

$$\frac{\partial a_i^{(n)}}{\partial z_i^{(n)}} = \sigma'(z_i^{(n)})$$

$$\frac{\partial z_i^{(n)}}{\partial w_{i,j}^{(n-1)}} = a_j^{(n-1)}$$

$$\frac{\partial z_i^{(n)}}{\partial b_i^{(n-1)}} = 1 \text{ (We can ignore this term as it is 1)}$$

Then, to propagate even further backwards, we can keep adding terms to the chain, which can be calculated in the same way.

$$\frac{\partial C}{\partial w^{(n-2)}} = \frac{\partial C}{\partial a^{(n)}} \cdot \frac{\partial a^{(n)}}{\partial z^{(n)}} \cdot \frac{\partial z^{(n)}}{\partial a^{(n-1)}} \cdot \frac{\partial a^{(n-1)}}{\partial z^{(n-1)}} \cdot \frac{\partial z^{(n-1)}}{\partial w^{(n-2)}}$$

$$\text{And } \frac{\partial z_i^{(n)}}{\partial a_i^{(n-1)}} = w_{i,i}^{(n-1)}$$

Or, in more general form:

$$\frac{\partial C}{\partial w^{(n-k)}} = \frac{\partial C}{\partial a^{(n)}} \cdot \left[\prod_{m=0}^{k-2} \frac{\partial a^{(n-m)}}{\partial z^{(n-m)}} \cdot \frac{\partial z^{(n-m)}}{\partial a^{(n-m-1)}} \right] \cdot \frac{\partial a^{(n-k+1)}}{\partial z^{(n-k+1)}} \cdot \frac{\partial z^{(n-k+1)}}{\partial w^{(n-k)}}$$

So then, using our results for all these terms:

$$\frac{\partial C}{\partial w_{i,j}^{(n-k)}} = \frac{1}{5} (a_i^{(n)} - \bar{y}_i) \left[\prod_{m=0}^{k-2} \sigma' (z_i^{(n-m)}) \cdot w_{i,i}^{(n-m-1)} \right] \cdot \sigma' (z_i^{(n-k+1)}) \cdot a_j^{(n-k)}$$

$$\frac{\partial C}{\partial b_i^{(n-k)}} = \frac{1}{5} (a_i^{(n)} - \bar{y}_i) \left[\prod_{m=0}^{k-2} \sigma' (z_i^{(n-m)}) \cdot w_{i,i}^{(n-m-1)} \right] \cdot \sigma' (z_i^{(n-k+1)})$$

After calculating these derivatives for each weight and bias, we can alter them simply by adding a scaled version of the derivative. Therefore, as the derivative gets closer to 0, i.e. the weights and biases get closer to optimisation, the weights and biases will be altered less and less. The derivatives will be scaled by some learning rate $\alpha \in (0, 1)$. For this project, I will let $\alpha = 0.2$ as it's not so high that the model will be rushed and inaccurate, but not so low that the model will take too long to optimise.

```
private final double ALPHA = 0.2d;
```

The pseudocode for the learning procedure can be seen below:


```

PROCEDURE learn(Sample sample) // Done straight after prediction
    INT n=5
    DOUBLE dCdw
    DOUBLE dCdb
    DOUBLE productChain
    FOR k=1 TO 4
        INT iBound = layers[n-k].size()
        INT jBound = layers[n-k-1].size()
        FOR i=0 TO iBound
            dCdb = (0.2)
                *(layers[n][i]-(i==sample.getLabel() ? 1 : 0))
            productChain=1
            FOR m=0 TO k-2 // only runs if k >= 2
                productChain *=
                    sigmoidDiff(sigmoidInv(layers[n-m][i]))
                    * layers[n-m-1].getWeight(i, i)
            ENDFOR
            dCdb *= productChain
                * sigmoidDiff(sigmoidInv(layers[n-k+1][i]))
            layers[n-k].alterBias(i, -ALPHA * dCdb)
            FOR j=0 TO jBound
                dCdw = dCdb * layers[n-k][j]
                layers[n-k].alterWeight(i, j, -ALPHA * dCdw)
            ENDFOR
        ENDFOR
    ENDFOR
ENDPROCEDURE

```

From this, we get:

```

public void learn(Sample sample) {
    final int n = 5;
    double dCdw, dCdb, productChain;
    int iBound, jBound;
    for (int k = 1; k <= 4; ++k) {
        iBound = layers[n-k].getRows();
        jBound = layers[n-k-1].getRows();
        for (int i = 0; i <= iBound; ++i) {
            dCdb = 0.2
                *(layers[n].getElement(i) - i == sample.getLabel() ? 1 : 0);
            productChain = 1;
            for (int m = 0; m <= k - 2; ++m) {
                productChain *=
                    sigmoidDiff(sigmoidInv(layers[n-m].getElement(i)))
                    * layers[n-m-1].getWeights().getElement(i, i);
            }
            dCdb *= productChain
                * sigmoidDiff(sigmoidInv(layers[n-k+1].getElement(i)));
            layers[n-k].alterBias(i, value: -ALPHA*dCdb);
            for (int j = 0; j <= jBound; ++j) {
                dCdw = dCdb * layers[n-k].getElement(j);
                layers[n-k].alterWeight(i, j, value: -ALPHA*dCdw);
            }
        }
    }
}

```

Error: Index out of bounds when accessing layers

Fix: n should be 4 because the first layer is layer 0.

Error: Index out of bounds when accessing weights and biases

Fix: Loops go UP TO iBound and jBound

Error: Index out of bounds accessing final layer.

Fix: don't allow index to go over 9 when accessing this layer, i.e. access with MIN(size, i)

Error: Always altering weights and biases by 0?

Fix: Ternary operator at the end was resetting the entire line to 0. Isolated the expression.

```

public void learn(Sample sample) {
    final int n = 4;
    double dCdw, dCdb, productChain;
    int iBound, jBound;
    for (int k = 1; k <= 4; ++k) {
        dCdw = 1.0d;
        dCdb = 1.0d;
        iBound = layers[n-k].getRows();
        if (n-k-1 >= 0) {
            jBound = layers[n - k - 1].getRows();
        } else {
            jBound = Utils.INPUT_LAYER_SIZE;
        }
        for (int i = 0; i < iBound; ++i) {
            // MIN functions used to not let indexes go out of bounds
            // Ternary expression at the end equivalent to one-hot vector
            dCdb = layers[n].getElement(Math.min(layers[n].getRows()-1, i)) - (i == sample.getLabel() ? 1 : 0);
            dCdb *= 0.2d;
            productChain = 1;
            for (int m = 0; m <= k - 2; ++m) {
                productChain *=
                    sigmoidDiff(sigmoidInv(layers[n-m].getElement(Math.min(layers[n-m].getRows()-1, i))))
                    * layers[n-m-1].getWeights().getElement(
                        Math.min(layers[n-m-1].getWeights().getRows()-1, i),
                        Math.min(layers[n-m-1].getWeights().getCols()-1, i)
                    );
            }
            dCdb *= productChain
                * sigmoidDiff(sigmoidInv(layers[n-k+1].getElement(Math.min(layers[n-k+1].getRows()-1, i))));
            layers[n-k].alterBias(Math.min(layers[n-k].getBiases().getRows()-1, i), value: -ALPHA*dCdb);
            for (int j = 0; j < jBound; ++j) {
                dCdw = dCdb * layers[n-k].getElement(Math.min(layers[n-k].getRows()-1, j));
                layers[n-k].alterWeight(
                    Math.min(layers[n-k].getWeights().getRows()-1, i),
                    Math.min(layers[n-k].getWeights().getCols()-1, j),
                    value: -ALPHA*dCdw);
            }
        }
    }
}

```

I tested it with the following loop, making it learn from one image over and over.

```

NeuralNetwork NN = new NeuralNetwork();
Sample sample1 = new Sample(new SampleImage(new File( pathname: "./HN/CompleteImages/All data (Compressed)/0/0_1_0.png")), label: 0);
NNPrediction p;
for (int i = 0; i < 100; ++i) {
    p = NN.predict(sample1);
    System.out.printf("result: %d cost: %f\n", p.getResult(), p.getCost());
    NN.learn(sample1);
}

```

The result I got shows it is fully functional.

```

result: 8 cost: 0.237264 result: 0 cost: 0.039249
result: 8 cost: 0.227635 result: 0 cost: 0.038679
result: 8 cost: 0.217861 result: 0 cost: 0.038124
result: 8 cost: 0.208387 result: 0 cost: 0.037582
result: 8 cost: 0.199625 result: 0 cost: 0.037054
result: 8 cost: 0.191827 result: 0 cost: 0.036539
result: 8 cost: 0.185036 result: 0 cost: 0.036036
result: 8 cost: 0.179154 result: 0 cost: 0.035545
result: 8 cost: 0.174023 result: 0 cost: 0.035066
result: 8 cost: 0.169484 result: 0 cost: 0.034598
Process finished with exit code 0

```

As you can see on the right, the cost gradually decreases with every iteration. After iterating enough times, it eventually predicts the correct number.

Stage	Part	Test	Type	Test Data	Expected Result	Actual Result
IV	a	Calculate cost	Valid	Output cost from a given prediction	A reasonable value for cost (not too large or small)	<pre> result: 8 cost: 0.237264 result: 8 cost: 0.227635 result: 8 cost: 0.217861 result: 8 cost: 0.208387 result: 8 cost: 0.199625 result: 8 cost: 0.191827 result: 8 cost: 0.185036 result: 8 cost: 0.179154 result: 8 cost: 0.174023 result: 8 cost: 0.169484 </pre>
	b	Weights and biases are adjusted	Valid	Output weight matrices and bias vectors before and after a sample is processed	A difference in weights and biases before and after	Cost decreasing shows weights and biases are adjusted

That concludes stage IV, easily the most difficult stage so far. Implementing the maths was troublesome to say the least, but I got it to work in the end. Most of the issues were with so different vectors and matrices being different sizes in the same iteration. The workaround I used – never letting indices go over array size using MIN functions - was probably a quick fix that isn't fully mathematically rigorous. It does work, but I'm honestly not sure if it makes the learning any less efficient.

Stage V: GUI

I will start with the title screen in IntelliJ Swing Editor

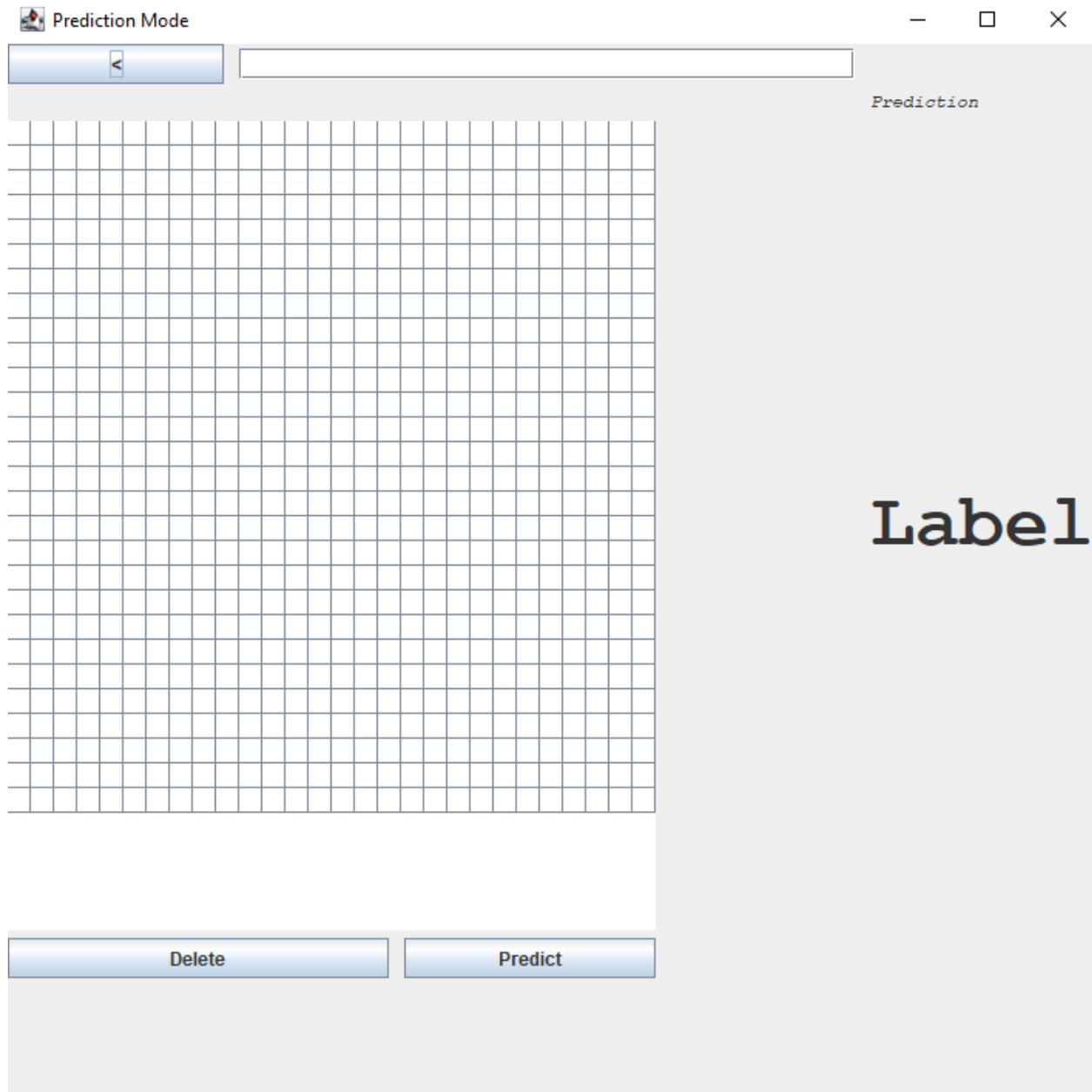


As it turns out, dealing with images in java applets is very difficult to get right, so for now I will be leaving out a logo or artistic background. These features are redundant anyway and would have been purely aesthetic. I will be avoiding icons for the same reason. The exit button should close the main page.

```
exitButton.addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        dispose();  
    }  
});
```

Works as it should

Now for the Prediction Mode page.



The “label” part will contain the prediction and the text bar at the top will contain a file path when it is functional. This page should close when the back button (top left) is pressed, and the grid should be alterable. The grid will be a Jtable which can be drawn on via the mouse.

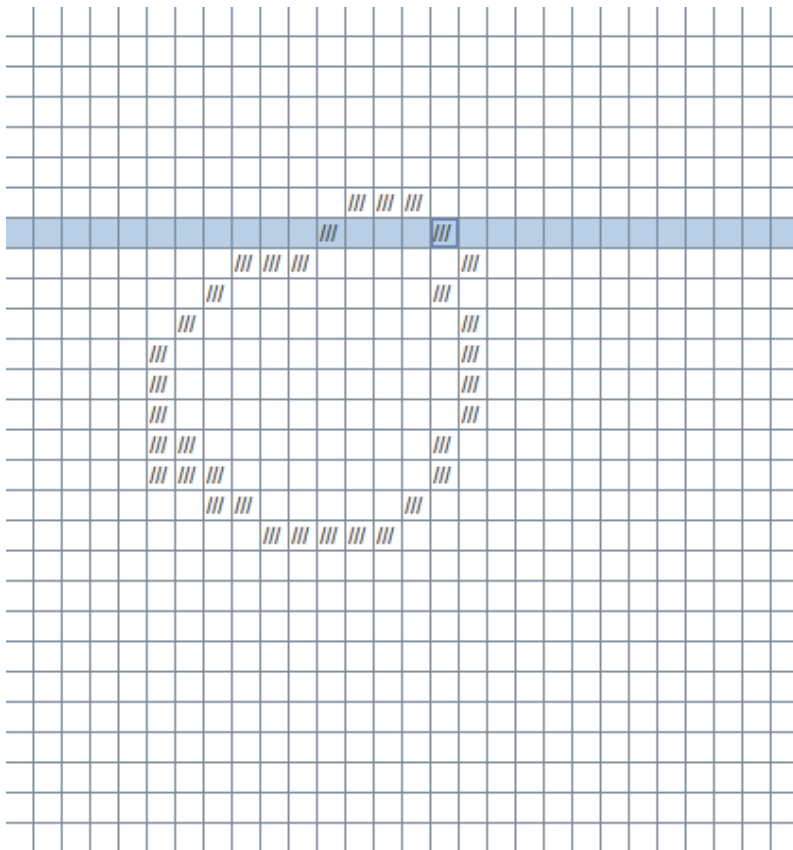
```

drawingGrid.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        rowSelected = drawingGrid.rowAtPoint(e.getPoint());
        colSelected = drawingGrid.columnAtPoint(e.getPoint());
        if (drawingGrid.getValueAt(rowSelected, colSelected) == null) {
            drawingGrid.setValueAt( aValue: "///", rowSelected, colSelected);
        } else {
            drawingGrid.setValueAt( aValue: null, rowSelected, colSelected);
        }
    }
});
backButton.addActionListener(e -> dispose());

```

Works as expected, however, the mouse cannot be held down to draw. I can't think of a way this could be done unfortunately.

Will leave to just clicking for now. If it functions, minute things like these are not massive deals.



Example of the drawn-on grid

The delete button should also clear the table.

```
for (int i = 1; i <= 28; ++i) {  
    for (int j = 1; j <= 28; ++j) {  
        drawingGrid.setValueAt( aValue: null, i, j);  
    }  
}
```

Error: array index out of bounds; I assumed the table was not 0-indexed.

Fix: Loop from 0 UP TO 28

```
for (int i = 0; i < 28; ++i) {  
    for (int j = 0; j < 28; ++j) {
```

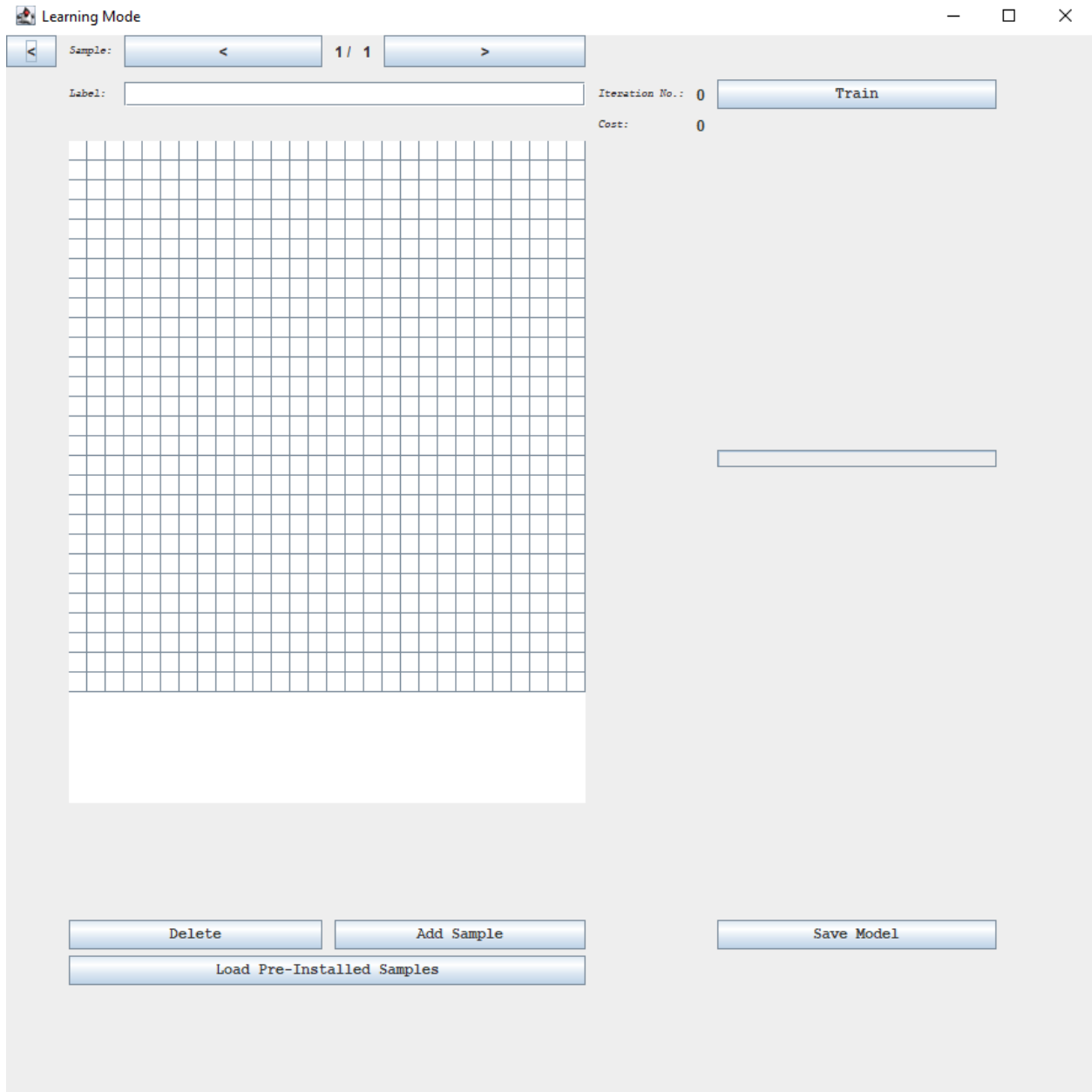
Works as expected.

The cells of the grid should not be manually editable, otherwise I may run into problems when it comes to converting the grid into an array for a neural net, and it makes drawing very finicky.

```
drawingGrid = new JTable( numRows: 28, numColumns: 28) {  
    @Override  
    public boolean isCellEditable(int row, int column) { return false; }  
};
```

Cells are no longer manually editable.

Now for the learning mode interface.



The drawing grid for learning mode works in the same way as that of prediction mode. The GUI is now all set up.

Stage	Part	Test	Type	Test Data	Expected Result	Actual Result
V	a	Title screen GUI takes you to separate pages	Valid	Click GUI buttons to open pages	New pages open and main menu closes	See evidence \GUI-test.mp4

Feature	Component	Description	Justification	Completion
---------	-----------	-------------	---------------	------------

Startup Menu	Title	An image of the title “DigiTrace” in a bold font, with a small graphic/logo before the word.	Having a clear title is instrumental to clear presentation of the project. An attractive title with a logo will make the project seem well structured and well-designed as soon as it is opened.	
	Background	An abstract, impasto style background with very low opacity all around, yet almost no opacity around the title and logo.	Having a bit of colour in the background at the start will add a more vibrant, elegant feel to the UI, and will give a nice contrast to the feel of the rest of the program, which is very logical and mathematical.	
	Load Model Button	A button containing the text “Load Model”.	This button is necessary to take the user to the prediction mode of the program, where they can load a previously trained model, draw a digit and get a prediction.	
	Train New Model	A button containing the text “Train New Model”.	This button will take the user to the learning mode of the program, where they can train a new model on samples (both their own and pre-installed).	
	Exit Button	A button containing the text “Exit”	Closes the app	

That concludes stage V. I had to leave a few GUI features I proposed in planning out of the project; they were marked as non-necessary anyway, but it’s a shame I couldn’t add them. It would have taken too much time to implement. All the necessary things I managed to add.

Stage VI/VII: Learning Mode Core/UX

I've combined these stages as the UX will be very easy to implement alongside the core process and GUI.

The objective of this stage is to make the learning mode functional, with all the criteria I have stated in the original stage plan. My first thought is to make utility functions, one that compiles the contents of a JTable into a binary array that can be passed into the network as a sample, and one that clears the JTable (this one is trivial). The pseudocode can be seen below:

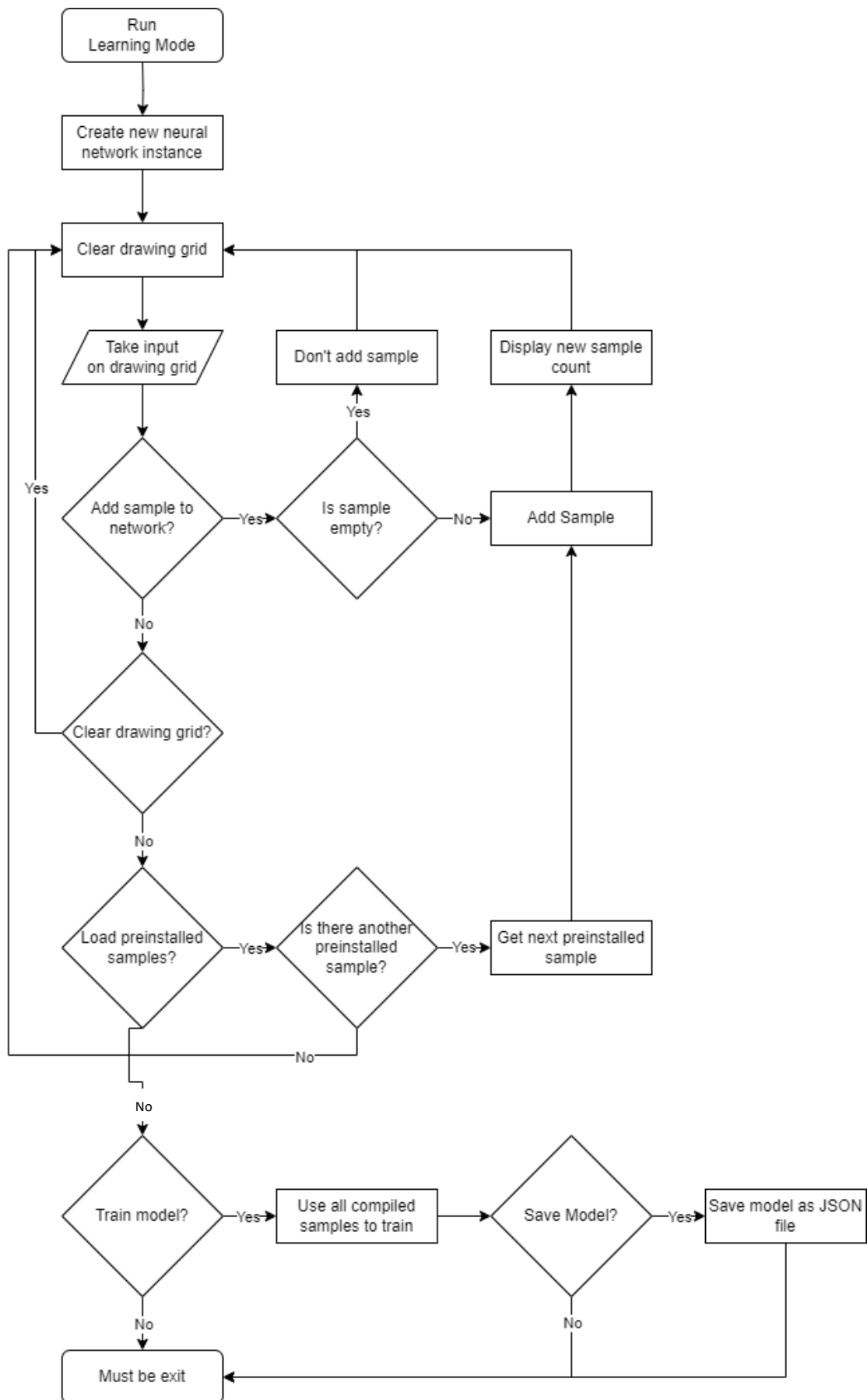
```
FUNCTION tableToArray(JTable table)
    INT[784] binaryArray
    FOR i=0 UP TO 28
        FOR j=0 UP TO 28
            binaryArray[28 * i + j]
                = table.getValue(i, j) != null
        ENDFOR
    ENDFOR
    RETURN binaryArray
ENDFUNCTION
```

From this we get:

```
public static int[] tableToBinaryArray(JTable table) {
    int[] binaryArray = new int[784];
    for (int i = 0; i < 28; ++i) {
        for (int j = 0; j < 28; ++j) {
            binaryArray[i * 28 + j] = table.getValueAt(i, j) == null ? 0 : 1;
        }
    }
    return binaryArray;
}
```

Works as expected.

The whole process for learning mode can be seen in the flowchart below:



When getting the label from the user input, I must make sure I'm only ever collecting digits. The entire process for adding a sample can be seen below.

```
addSampleButton.addActionListener(e -> {
    int[] binaryArray;
    Sample sample;
    int label;

    binaryArray = Utils.tableToBinaryArray(drawingGrid);

    try { // Check if there's actually an integer inputted
        label = Integer.parseInt(labelTextField.getText().substring(0, 1));
    } catch (NumberFormatException nfe) {
        // Don't add sample if invalid label
        return;
    }

    // Empty array check already implemented elsewhere
    sample = new Sample(new SampleImage(binaryArray), label);
    NN.addSample(sample);
    // If all successful clear grid
    clear();
    // Update sample count
    totalSamplesLabel.setText(String.format("%d", NN.getSampleAmount()));
});
```

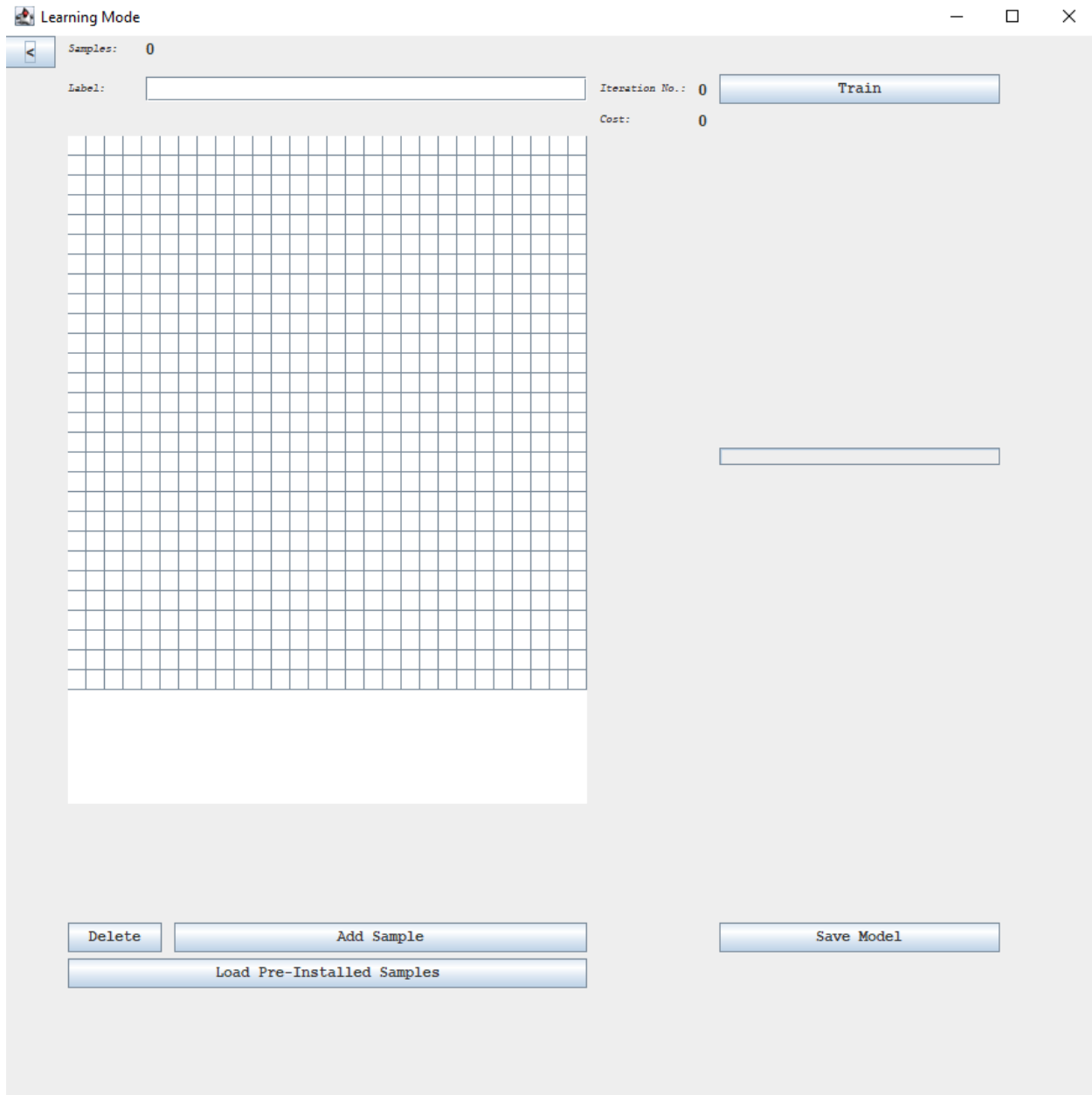
Error: GUI part works but should clear the label input upon adding. On the other hand, it is only ever reading empty samples (unless some of the first row has been selected?) apparently, so never adds them.

Fix: Was converting input into images wrong, was only reading off the first row.

```
for (int i = 0; i < 28; ++i) {
    for (int j = 0; j < 28; ++j) {
        image.setRGB(j, i, pixelArray[i * 28 + j] == 1 ? 0xFF000000 : 0xFFFFFFFF);
    }
}
```

Works as expected.

I have decided to remove the ability to flick through samples in the program. It would make the UI far too confusing, as the same interface parts would be used to edit and view samples. The new UI looks like this; it now has no sample scrolling options, only showing how many samples there are. (top-middle-left).



Now for loading pre-installed samples. It should loop through all files in the samples folder "HN/Completemages/All Data (Compressed)/" and create a sample for each one. Luckily, all file names have their respective digit at the beginning. The pseudocode for the procedure can be seen below:

```

PROCEDURE loadPreInstalledSamples()

    FILE[] filepaths =
        WALK("./HN/CompleteImages/All Data (Compressed)/")
    FOR file IN filepaths

        SampleImage sImg = new SampleImage(file)
        // file[0] will be the label
        Sample s = new Sample(sImg, INT(file[0]))
        NN.addSample(s)

        totalSamplesLabel.setText(NN.getSampleAmount())

    ENDFOR
ENDPROCEDURE

```

```

loadPreInstalledSamplesButton.addActionListener(e -> {
    // Disable the button (one time use)
    loadPreInstalledSamplesButton.setText("Loading...");
    loadPreInstalledSamplesButton.setEnabled(false);

    Stream<Path> paths;
    try {
        // Recursive walk around sample directory
        paths = Files.walk(Paths.get( first: "./HN/CompleteImages/All data (Compressed)"));
    } catch (IOException ioE) {
        ioE.printStackTrace();
        return;
    }
    List<File> files = paths
        // Gets rid of directories
        .filter(Files::isRegularFile) Stream<Path>
        // Converts files to absolute paths
        .map(Path::toAbsolutePath)
        // Path -> File
        .map(Path::toFile) Stream<File>
        .toList();

    paths.close();

    files.forEach(f -> {
        SampleImage sImg = new SampleImage(f);
        // file[0] (first char) will be the label
        Sample s = new Sample(sImg, Integer.parseInt(f.getName().substring(0, 1)));
        NN.addSample(s);
        totalSamplesLabel.setText(String.format("%d", NN.getSampleAmount()));
    });
});

```

Seems to work but takes stupidly long. Will try and find a way to optimise, or just reduce the number of samples. Concurrency perhaps?

Made the GUI and sample loading concurrent, now the GUI isn't held up by all the samples. Also optimised the sample reading more; for some reason, when making a sample from a binary array and not an image, I would first convert it to an image then back to a binary array to be put into the network. Suffice to say that implementation has been replaced.

```
loadPreInstalledSamplesButton.addActionListener(e -> {
    // Disable the button (one time use)
    Thread guiChanges = new Thread(() -> {
        loadPreInstalledSamplesButton.setText("Loading...");
        loadPreInstalledSamplesButton.setEnabled(false);
    });
    guiChanges.start();
    Thread loadingSamples = new Thread(() -> {
        loadPreInstalledSamples();
        loadPreInstalledSamplesButton.setText("Pre-Installed Samples Loaded");
    });
    loadingSamples.start();
});
```

```
private BufferedImage image;
private int[] binaryArray;

public SampleImage(File file) {
    binaryArray = null;
    try {
        image = ImageIO.read(file);
    } catch (IOException ioE) {
        System.err.println(Arrays.toString(ioE.getStackTrace()));
    }
}

public SampleImage(int[] pixelArray) {
    image = null;
    if (pixelArray.length != 784) {
        throw new IndexOutOfBoundsException("Attempted to make sample of wrong size");
    }
    binaryArray = pixelArray;
}

public int[] getBinaryArray() {
    if (image != null) {
        binaryArray = new int[784];
        for (int i = 0; i < 28; ++i) {
            for (int j = 0; j < 28; ++j) {
                binaryArray[j + 28 * i] = image.getRGB(j, i) == 0xFF000000 ? 1 : 0;
            }
        }
    }
    return binaryArray;
}
```


Now samples can be loaded, model training can be done. The training procedure will commence when the train button is pressed. Every time a sample is used to learn, all the GUI elements need to be updated too: the iteration number label, cost label and progress bar. The pseudocode for this procedure can be seen below:

```
PROCEDURE train()
    INT i
    INT sampleAmount = NN.getSampleAmount()
    NNPrediction currPred
    Sample currSample
    FOR i=0 UP TO sampleAmount
        currSample = NN.getSample(i)
        currPred = NN.Predict(currSample)
        NN.learn(currSample)
        // GUI updates
        costLabel.setText(currPred.getCost())
        iterationNumberLabel.setText(i+1)
        progressBarFaction = i / sampleAmount
    ENDFOR
ENDPROCEDURE
```

I realised I could set the progress bar's minimum and maximum values to 0 and the number of samples respectively, making the GUI update a bit more efficient every time.

```

trainButton.addActionListener(e -> {
    int i;
    int sampleAmount = NN.getSampleAmount();
    NNPrediction currPred;
    Sample currSample;

    learningProgressBar.setMaximum(sampleAmount);

    for (i = 0; i < sampleAmount; ++i)
    {
        currSample = NN.getSample(i);
        currPred = NN.predict(currSample);
        NN.learn(currSample);

        iterationNoValueLabel.setText(String.format("%d", i+1));
        costLabel.setText(String.format("%f", currPred.getCost()));
        // Cast to float otherwise will do integer division
        learningProgressBar.setValue(i+1);
    }
});

```

Tested with few samples and worked, though when training with a lot (which is the norm) the GUI doesn't update as the process happens, meaning I must run the GUI and process concurrently again.

```

trainButton.addActionListener(e -> {
    sampleAmount = NN.getSampleAmount();

    learningProgressBar.setMaximum(sampleAmount);

    Thread machineLearning = new Thread(this::train);
    machineLearning.start();

    Thread guiUpdates = new Thread(() -> {
        while (iterNumber < sampleAmount - 1) {
            iterationNoValueLabel.setText(String.format("%d", iterNumber + 1));
            costLabel.setText(String.format("%f", currPred.getCost()));
            learningProgressBar.setValue(iterNumber + 1);
        }
    });
    guiUpdates.start();
});
}

```

```

private void train() {
    Sample currSample;
    for (iterNumber = 0; iterNumber < sampleAmount; ++iterNumber) {
        currSample = NN.getSample(iterNumber);
        currPred = NN.predict(currSample);
        NN.learn(currSample);
    }
}

```

Works as expected, but takes a long time to get through all the samples; perhaps I should reduce the number of samples? I also made the train button disable once pressed.

Will reduce the number of samples from 206,750 to 75,000 (7,500 for each digit).

Now for saving the model. This can easily be done with the procedure I created in stage III. It will randomly generate a serial number to put at the end of the file to differentiate between other saved models. It will also replace the text in the save model button with the name of the file the network is stored in.

```

saveModelButton.addActionListener(e -> {
    // 10 digit random number sequence
    long serialID = Math.round(Math.random() * Math.pow(10, 10));
    String serialStringID = Long.toString(serialID);
    while (serialStringID.length() < 10) {
        // Add 0's to end if not 10 digits
        serialStringID = serialStringID.concat( str: "0");
    }
    String fileName = "NETWORK" + serialStringID;
    NN.storeAsJSON(fileName);
    saveModelButton.setText("Model stored in " + fileName + ".json");
    saveModelButton.setEnabled(false);
});

```

Works, all values stored in a named JSON file.

That concludes both stage VI and VII, as the UX was easy to implement alongside.

Stage	Part	Test	Type	Test Data	Expected Result	Actual Result
VI	a	Samples are added to neural network	Valid	Draw samples and click confirm	Samples added with correct arrays and labels	See evidence \AddSamples-test.mp4
	b	Preinstalled samples can be loaded to neural network	Valid	Click "load pre-installed samples" button	All pre-installed samples added	See evidence \AddSamples-test.mp4
	c	Empty sample cannot be added	Boundary	Click confirm while grid is empty	No sample added	See evidence \EmptySample-test.mp4
	d	Labelless sample cannot be added	Boundary	Click confirm while label is empty	No sample added	See evidence \EmptySample-test.mp4
	e	Model trains	Valid	Click "Train" button	Weights and biases repeatedly change	See evidence \Training

VII	f	Model saves	Valid	Click "Save Model" and choose a file location	A new JSON file created in the system with the correct parameters	: test.mp4 See evidence \SaveModel-test.mp4
	g	Model cannot be trained without samples	Boundary	Try to click train when there are no added samples	The model does not attempt to train	After change* See evidence \NoSamplesTrain-test.mp4
	a	Progress bar displays fraction of samples analysed over total samples	Valid	Click "Train" button	Progress bar gradually fills	See evidence \Training-test.mp4
	b	Cost table updates	Valid	Click "Train" button	Cost table is being constantly filled with new values and auto scrolled	See evidence \Training-test.mp4

*Did not implement this originally. Added this conditional when the train button is clicked:

```
if (sampleAmount == 0) return;
```

Now it works.

Feature	Component	Description	Justification	
Learning Mode	Digit Drawing Grid	A 28x28 square grid (each square representing a pixel of an image). Black and white colouring.	A grid which will display either a loaded sample, or a user-created sample.	
	Label Box	A box containing a digit.	The sample's digit label will be displayed in this box, or inputted by the user if it is user-created.	

Sample Number	A text display showing two numbers in form "Sample No. / Total Samples".	Will display which sample is being looked at, out of how many samples have been loaded.	
Sample Scrolling Buttons	A left and right arrow button either side of the sample number display.	Will allow the user to scroll through all the loaded samples.	Decided seeing all the samples was not necessary near the start of stage VI
Load Preinstalled Samples Button	A one-use button containing the text "Load Preinstalled Samples".	Will allow the user the load in the thousands of pre-installed samples.	
New Sample Button	A button containing a plus icon.	Will allow the user to create a new sample.	
Train Button	A button containing the word "Train".	Will take all the samples in and begin the machine learning process.	
Cost Graph	A small line graph/table of cost to iteration number.	Will display the cost of the model gradually decreasing as it learns.	Still Display cost, just not on a graph
Save Model Button	A button containing a file icon.	When the model has trained with all samples, it can be saved and used in the app's prediction mode.	
Training Progress Bar	A bar showing how many samples have been used out of the total samples.	Will give a sense of progress to the user. Without this, the UI may be confusing, and wouldn't give any intuition into what is happening.	

Stage VIII: Prediction Mode

The first step in this stage is being able to load a network back from a JSON file. This can be done in reversed way to storing the JSON file. I will also add an option to the neural network constructor that makes the network initialise all the values to zero. The pseudocode can be seen below:

```
FUNCTION loadFromJSON(FILE file)

    TRY

        String jsonString = READ(file)

    CATCH FileNotFound

        RETURN 1

    ENDMETHOD

    JSONObject jsonObj = PARSEJSON(jsonString)

    FOR layerIdx=0 UP TO 4

        FOR i=0 UP TO layers[layerIdx+1].length

            // alterWeight/alterBias equivalent to
            // setting when initial value is 0

            layers[layerIdx].alterBias(

                i, jsonObj.layers[layerIdx].biases[i]

            )

            FOR j=0 UP TO layers[layerIdx]

                layers[layerIdx].alterWeight(

                    i, j,

                    jsonObj.layers[layerIdx]

                        .weights[i*layers[layerIdx] + j]

                )

            ENDFOR

        ENDFOR

    ENDFOR

    RETURN 0

ENDPROCEDURE
```

The new NN option:

```
public NeuralNetwork(boolean initToZero) {  
    layers = new Layer[5];  
    layers[0] = new Layer(LayerType.INPUT);  
    layers[1] = new Layer(LayerType.HIDDEN);  
    layers[2] = new Layer(LayerType.HIDDEN);  
    layers[3] = new Layer(LayerType.PRE_OUTPUT);  
    layers[4] = new Layer(LayerType.OUTPUT);  
    samples = new LinkedList<>();  
    if (!initToZero) randomise();  
}
```

and the load NN procedure:

```
public int loadFromJSON(String fileName) {  
    File file;  
    Scanner scanner;  
    StringBuilder sb; // To piece file together  
    String jsonString;  
  
    // Open file  
    file = new File(fileName);  
    try {  
        scanner = new Scanner(file);  
    } catch (FileNotFoundException e) {  
        return 1;  
    }  
  
    // Read file  
    sb = new StringBuilder();  
    while (scanner.hasNextLine()) {  
        sb.append(scanner.nextLine());  
    }  
    scanner.close();  
  
    jsonString = sb.toString();  
    JSONObject parsedJSON = new JSONObject(jsonString);  
    JSONArray layersJSON = new JSONArray(parsedJSON.getJSONArray( key: "layers"));  
    for (int layerIdx = 0; layerIdx < 4; ++layerIdx) {  
        for (int i = 0; i < layers[layerIdx+1].getRows(); ++i) {  
            // alterWeight/alterBias equivalent to  
            // setting when initial value is 0  
            JSONObject layerJSON = new JSONObject(layersJSON.getJSONObject(i));  
            JSONArray weightsJSON = new JSONArray(layerJSON.getJSONArray( key: "weights"));  
            JSONArray biasesJSON = new JSONArray(layerJSON.getJSONArray( key: "biases"));  
            layers[layerIdx].alterBias(  
                i, biasesJSON.getDouble(i)  
            );  
            for (int j = 0; j < layers[layerIdx].getRows(); ++j) {  
                layers[layerIdx].alterWeight(  
                    i, j,  
                    weightsJSON.getDouble( index: i * layers[layerIdx].getRows() + j)  
                );  
            }  
        }  
    }  
    return 0;  
}
```


I tested it with the following:

```
NeuralNetwork NN = new NeuralNetwork( initToZero: true);
int load = NN.loadFromJSON( fileName: "C:\\Users\\ewan.hallatt2
if (load == 0) System.out.println("Load success");
else System.out.println("Could not load");
```

```
// For testing
for (int i = 0; i < 4; ++i) {
    System.out.println("Weights");
    Utils.printMatrix(layers[i].getWeights());
    System.out.println("Biases");
    Utils.printMatrix(layers[i].getBiases());
}
```

Error: could get layer objects from JSON but having trouble getting weights.

Fix: getJSONObject(i) not returning i-th layer but rather an object representing it. used the methods properly and don't create a new object every time. Also was accessing layers with i and not layerIdx.

```
JSONObject parsedJSON = new JSONObject(jsonString);
JSONArray layersJSON = new JSONArray(parsedJSON.getJSONObject( key: "layers"));
for (int layerIdx = 0; layerIdx < 4; ++layerIdx) {
    JSONObject layerJSON = layersJSON.getJSONObject(layerIdx);
    for (int i = 0; i < layers[layerIdx+1].getRows(); ++i) {
        // alterWeight/alterBias equivalent to
        // setting when initial value is 0
        JSONArray weightsJSON = layerJSON.getJSONArray( key: "weights");
        JSONArray biasesJSON = layerJSON.getJSONArray( key: "biases");
```

Error: accessing weights as if it were a 1D array.

Fix: use 2D indexing.

```
layers[layerIdx].alterWeight(
    i, i,
    weightsJSON.getJSONArray(i).getDouble(i)
);
```

Works as expected, neural network weights and biases match that of JSON file.

```
{ "layers": [ { "weights": [ [-0.01365309895557787, -0.5380898291048837, 0.568073044013855, -0.6036999327111843, 0.026528786025633577,
```

Weights

```
[ -0.01365309895557787 -0.5380898291048837 0.568073044013855 -0.6036999327111843 0.026528786025633577 0
```

Now I must use this in the prediction mode itself. I will add a button to confirm the file to attempt to load and will disable most of the interface until a file has been successfully loaded. The text in the file name text box will be red if the file could not load, and green if it could. I also must only allow JSON files to be taken.

```
if (!fileName.endsWith(".json")) return 1;
```

```
predictButton.setEnabled(false);
deleteButton.setEnabled(false);
NeuralNetwork NN = new NeuralNetwork( initToZero: true);

confirmFileButton.addActionListener(e -> {
    String fileName = fileNameTextField.getText();
    int loadStatus = NN.loadFromJSON(fileName);

    // File could not load
    if (loadStatus == 1) {
        fileNameTextField.setForeground(Color.RED);
        predictButton.setEnabled(false);
        deleteButton.setEnabled(false);
        return;
    }

    // Could load
    fileNameTextField.setForeground(Color.GREEN);
    predictButton.setEnabled(true);
    deleteButton.setEnabled(true);
});
```

Works as expected.

Now, finally, take input from the grid and give a prediction on it.

```

predictButton.addActionListener(e -> {
    int[] binaryArray;
    NNPrediction prediction;
    Sample sampleInput;

    binaryArray = Utils.tableToBinaryArray(drawingGrid);
    // We do not care about the label here
    sampleInput = new Sample(new SampleImage(binaryArray), label: 0);
    prediction = NN.predict(sampleInput);
    predictionLabelResult.setText(String.format("%d", prediction.getResult()));
});

```

Error: Predicts first time, then result does not change.

Calculations seem to work, models just tend to always turn out bad from pre-installed samples; they are especially more likely to only predict 9 for some reason. Shuffling the pre-installed samples – as opposed to showing the network all of the samples for the same digit one after the other - could potentially make a difference.

```

// So it can sample in a random order
Collections.shuffle(files);

```

Actually gives different predictions now.

Stage	Part	Test	Type	Test Data	Expected Result	Actual Result
VIII	a	Model loads from file	Valid	Choose a file in the file selector	Model loads in without error	evidence \predicti on-test.mp4
	b	Prediction given from input	Valid	Draw a digit and confirm	A prediction is given without error	evidence \predicti on-test.mp4

Feature	Component	Description	Justification	Completion
Prediction Mode	Model File Selection	A bar in the top of the window, which opens the file explorer once clicked, and	This will allow the user to select the JSON files for their models, and will load the weights,	

	displays the file path once selected.	biases and name into the program.	
Digit Drawing Grid	A 28x28 square grid (each square representing a pixel of an image). Black and white colouring.	This grid is necessary for the user to input a handwritten digit. It will be inputted into the model, which will form a prediction. The user can draw with the mouse.	
Clear Button	A button under the digit drawing grid containing a bin icon.	This button clears the digit drawing grid.	
Predict Button	A button underneath the digit drawing grid containing the text "Predict".	This button confirms the user's digit on the grid and passes it as input to the model in the form of an array.	
Eraser Button	A button underneath the digit drawing grid containing a rubber icon.	Allows the user to erase parts of the drawing on the grid.	
Visualisation / Calculation window	A window containing model visualisation, i.e. a neural network diagram.	This window will show some display of progress in the prediction calculation process, and the state of the neural network.	
Prediction box	A box containing a digit.	The digit prediction of the model will be shown here.	

As a final note, I managed to find a way to make the drawing grid [motion based as opposed to click-by-click](#). I just detect mouse motion, and toggle pen-up and pen-down on click.

```

drawingGrid.addMouseListener(new MouseAdapter() {
    ewanh26 *
    @Override
    public void mouseClicked(MouseEvent e) {
        penDown = !penDown;
    }
});

drawingGrid.addMouseListener(new MouseMotionAdapter() {
    new *
    @Override
    public void mouseMoved(MouseEvent e) {
        super.mouseMoved(e);
        if (penDown) {
            rowSelected = drawingGrid.rowAtPoint(e.getPoint());
            colSelected = drawingGrid.columnAtPoint(e.getPoint());
            if (drawingGrid.getValueAt(rowSelected, colSelected) == null) {
                drawingGrid.setValueAt("///", rowSelected, colSelected);
            }
        }
    }
});

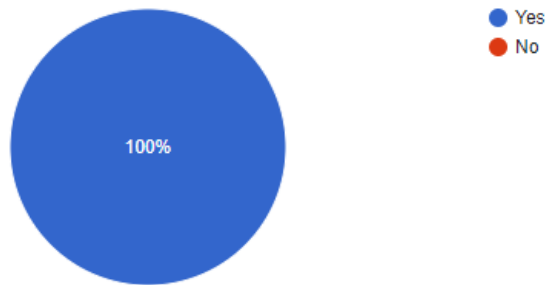
```

Post Dev I: Beta Testing

Questions I asked in my form, with results and explanation:

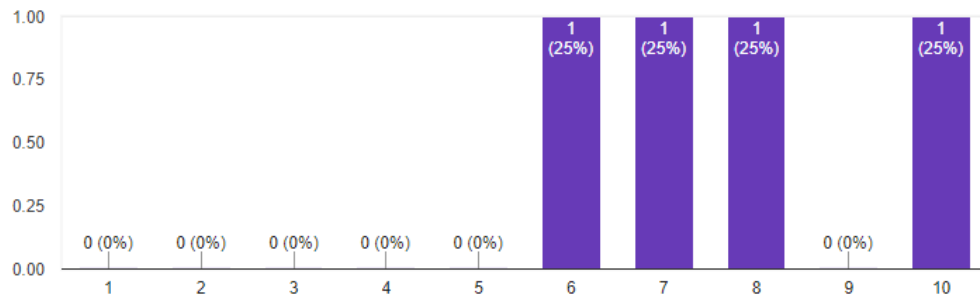
1. Usability

a. Are the buttons functional and only clickable when appropriate?



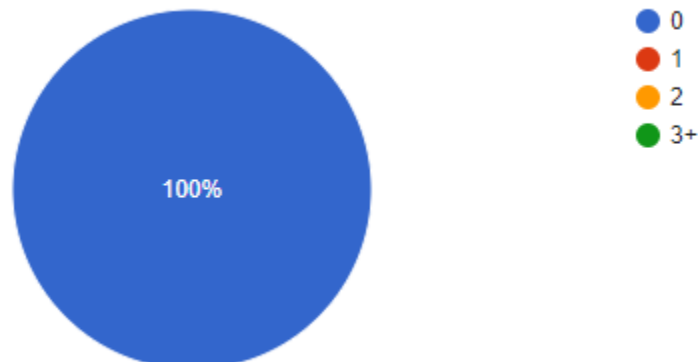
Everyone who used the program agreed the buttons were all functional. I made sure they could only click them when appropriate by disabling the buttons at the right times.

b. How well-structured and neat is the layout?



Most users thought the layout was well structured and neat, probably due to the alignment of all the components. Those who voted lower probably thought it looked confusing because there were so many functions.

c. How many bugs did you experience with the UI?

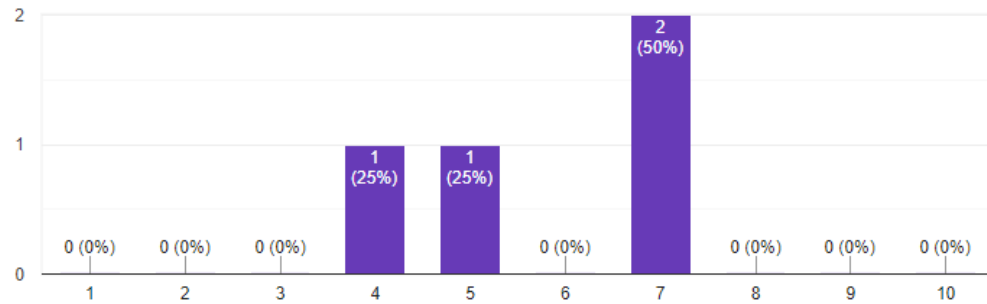


All users experienced no bugs with the UI. I made a good effort to not allow the UI to slip up in any way.

- d. **If you didn't answer "0" to the above question, please describe the bugs below.**

Again, everyone answered 0.

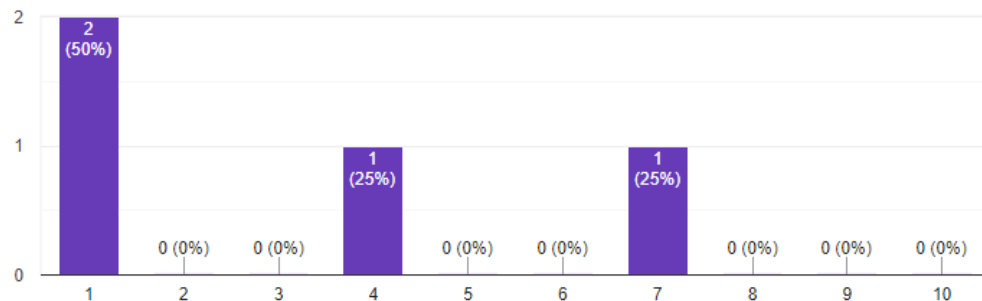
- e. **How easy to use is the drawing grid?**



There were mixed opinions on the ease-of-use of the drawing grid. A lot of people perhaps struggled to get used to the 'pen-down, pen-up' nature of it. I did ideally want it to have a 'click-and-hold' nature, but I was unsure how to implement this.

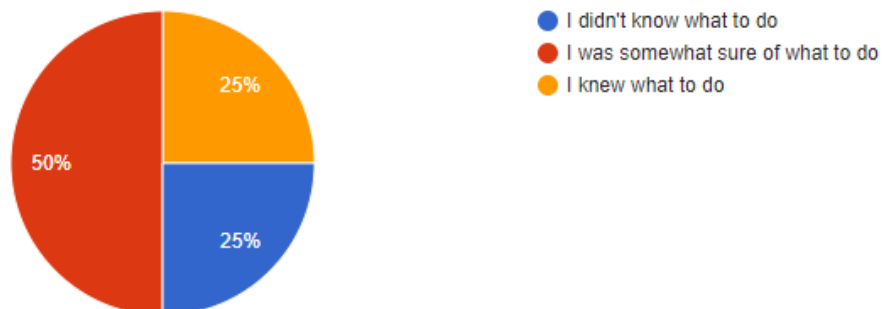
2. User-Friendliness

- a. **How overwhelming is the program?**



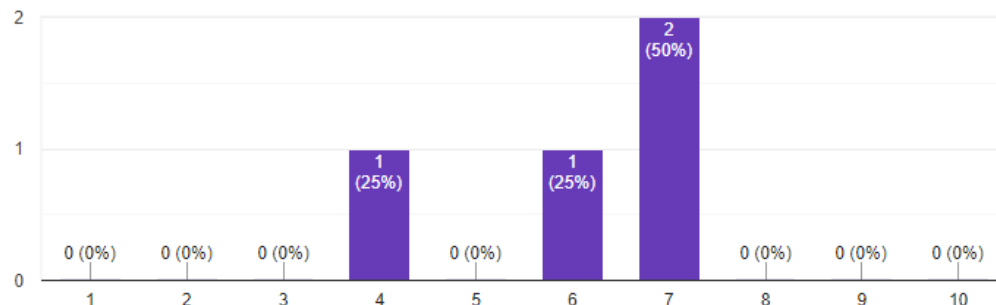
There were mixed opinions on how overwhelming the program was. Half of users thought it was completely clear, but all the others seemed to think it was quite obfuscated. This might be because of its technical nature.

- b. **How clear are you on what to do / how to use the program?**



I would have liked to create a tutorial for the program but didn't have time unfortunately. because of this, a lot of users were confused about how the program worked, especially if they knew nothing about ML.

c. How good does the colour scheme look?



Most users thought the colour scheme was mediocre. I would have liked to add more colour and graphics to the UI, but a lot of my time was of course spent getting the program to work.

d. How, if at all, do you think the colour scheme could be improved?

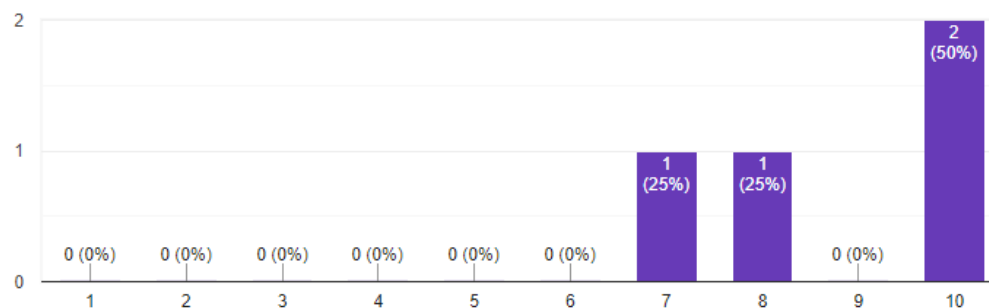
Make it more colourful and less squarey

need more blue

Perhaps more colours, less grey, font could be bigger

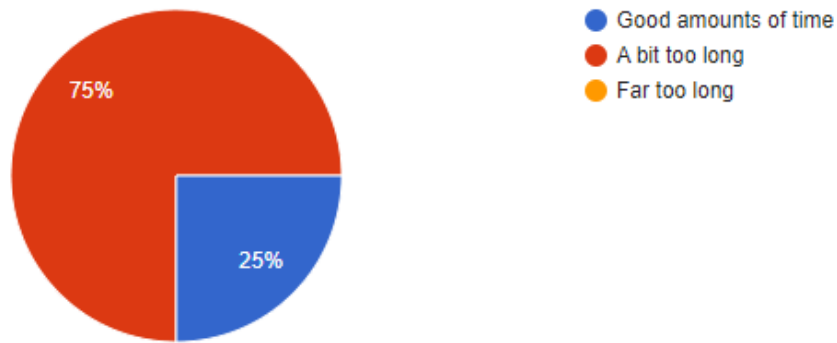
The general opinion of the colour scheme was that it was monochromatic. Other comments included the font being too small, and the program being too 'squarey'. These would be very small but effective tweaks to the UI.

e. How appropriate/legible is the font?



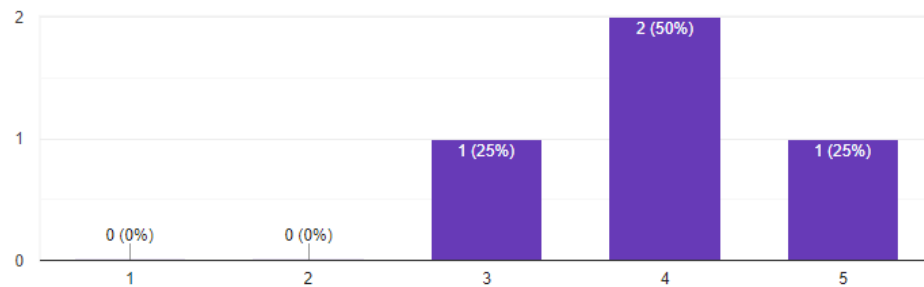
Most users thought the font was fine, though (as seen from a previous answer) some believe it could have been bigger.

f. The times the processes (training, loading samples, etc.) took were...



A lot of users would have wished the program took shorter time to run. Unfortunately, considering a machine learning process is taking place – which even has less samples than desired to reduce time – a shorter time is not attainable.

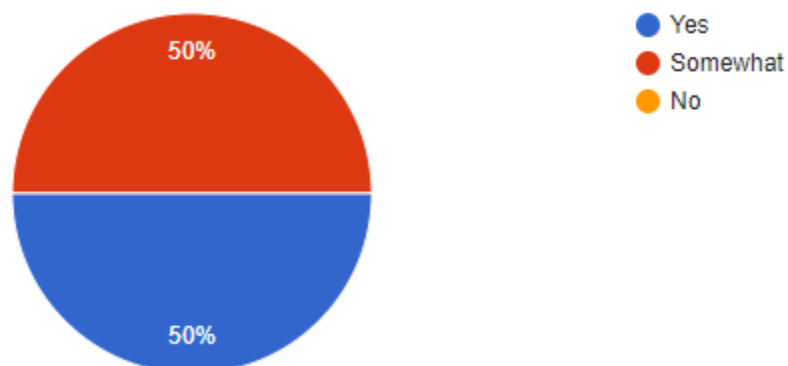
g. The processes (training, loading samples, etc.) were well-displayed.



Most users thought the parts of the ML process was well displayed, which is good as that was the objective of the program. I would have liked to have some sort of proper visualisation in the network, but that probably could have been a whole project on its own.

3. Accessibility

a. Are the meanings of the buttons clear?



Most users at least somewhat understood the meanings of the buttons. Most confusion was likely just due to a lack of knowledge of the subject at hand. Again, a tutorial would have been good too, to maybe give the program more context.

b. If not, for what reason?



A lot of confusion seems to have come from the language I used. Perhaps more accessible and common language would have been beneficial.

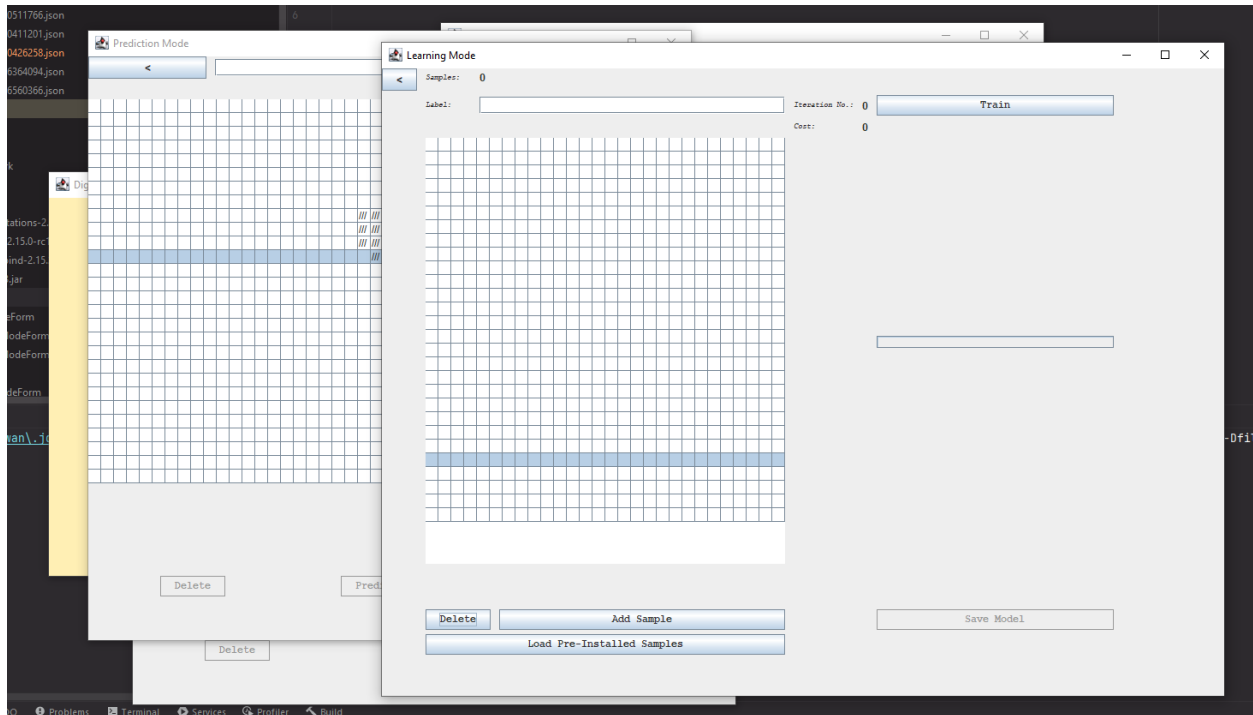
Post Dev II: Robustness Testing

I realised there wasn't many ways that the user could make the program malfunction, as it doesn't give you loads of freedom. However, there were certainly things that made the program seem less polished.

Test 1 – Over-clicking.

Test Data: Click as many buttons as possible in a short amount of time.

Outcome: Multiple windows could be opened, but nothing broke within each window, as there are lots of restrictions in place when it comes to null inputs and disabled buttons.

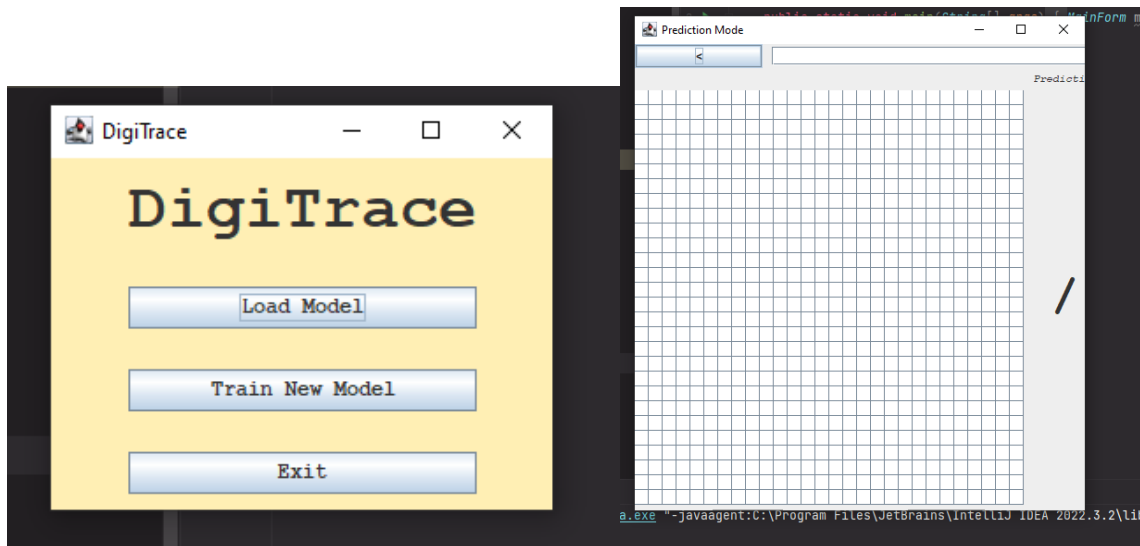


Resolve: Perhaps limit the number of windows that can be opened at once (i.e. 10 maximum) but still allow multiple so a couple of models can be used and compared at once.

Test 2 – UI Resizing

Test Data: Attempt to resize the UI.

Outcome: The main page fares well when resized, however in the subsequent pages, the buttons are hidden as the drawing grid can't be resized.



Post Dev III: Evaluation

<i>User Requirement Necessity</i>	<i>Colour</i>
<i>Essential</i>	
<i>Desirable</i>	
<i>Optional</i>	

User requirement: Title - An image of the title “DigiTrace” in a bold font, with a small graphic/logo before the word.

Score: 6/10 PARTLY MET

Comments: I partly met this requirement, as I successfully made the title with a bold font. However, I did not make a logo to put before the title. It probably just could have stayed as something simple and geometric like in the original GUI Designs (Overview I: GUI Designs, p11). The beta testers did say the font was clear (Post Dev I: Beta Testing Q2.e, p79), so the title was good for its purpose (clarity)

Development:

- Overview I: GUI Designs, p11.
- Stage V: GUI, p53.

Improvements:

- Add a small logo or graphic to the left of the title.
 - Do this by creating a PNG of a logo adding it to the swing UI in a small component.
 - Can just be a small geometric design but preferable would be something more sophisticated.
- Would portray a more polished and professional image to any stakeholders who may want to teach the machine learning process or understand it themselves.

User requirement: Background - An abstract, impasto style background with very low opacity all around, yet almost no opacity around the title and logo.

Score: 5/10 PARTLY MET

Comments: I partly met this requirement, as I made a background, but it wasn't as artistic as I would have hoped. I wanted to create something of that style but unfortunately, I'm not much of a graphic designer. Beta testers thought the colour scheme was mediocre (Beta Testing Q2.c&d, p79) so perhaps a more intricate background would have made for a more engaging interface, which might make the program more presentable and pleasant for stakeholders, who would most likely be using it to educate others or themselves.

Development:

- Overview I: GUI Designs, p11.
- Stage V: GUI, p53.

Improvements:

- Add more intricacy to the background or perhaps use an image.
- Make the background less opaque around the title to add depth to the interface.
 - Do this by putting the title into a component with a softer background colour (beige to blend with yellow for example).
 - Use fading and shape effects to emulate the colours blending into each other.

User requirement: Buttons – Functional buttons containing the text “Load Model”, “Train New Model” and “Exit” (optional).

Score: 9/10 MET

Comments: This requirement was fully met. All 3 buttons are functional and cause no errors (Post Dev I: Beta Testing Q1.a&c, p77). Only thing I would do is limit the number of windows that can be open at once (Post Dev II: Robustness Testing T1, p81) as running too many processes of this calibre at once could lead to high RAM and CPU usage and therefore lag, which would cause a negative user experience. Although, the speed at which modern computers can run ML processes may be something the stakeholders may want to demonstrate, by running multiple processes at once.

Development:

- Overview I: GUI Designs, p11.
- Stage V: GUI, p53.

Improvements:

- Limit the number of windows that can be open at once by use of the buttons.
 - But not in such a way that multiple windows are prohibited. Multiple processes should be able to be run at once, though not so many that major lag may come as a result of high RAM and CPU usage.

User requirement: Model File Selection - A bar in the top of the window, which opens the file explorer once clicked, and displays the file path once selected.

Score: 7/10 PARTIALLY MET

Comments: This requirement was partially met, since I couldn't find a way to open the file explorer to select a file, however the filename can be typed/copied in (Stage VIII: Prediction Mode, p74). This functions well, but makes for a slightly more confusing experience for new users (Post Dev I: Beta Testing Q1.a&b, p77), as they may not know the purpose of the text box.

Development:

- Stage V: GUI, p54.
- Stage VIII: Prediction Mode, p74.

Improvements:

- Find a way to make the file explorer open upon clicking the text box.
 - This would have to be compatible with all equivalent default file explorers in other OS's.
 - As Java VM is not platform specific, I'm sure the functionality to do this from the java language would be possible (maybe even the default)

User requirement: Digit Drawing Grid - A 28x28 square grid (each square representing a pixel of an image). Black and white colouring.

Score: 9/10 MET

Comments: This requirement was met. However, some users seemed to have trouble with the usability of it (Post Dev I: Beta Testing Q1.e, p78). This was perhaps to do with the 'pen-down-pen-up' nature of it, as opposed to the generally preferred 'click-and-hold' nature. I don't think this would matter particularly to stakeholders though.

Development:

- Stage V: GUI, p55.
- Stage VIII: Prediction Mode, p74.

Improvements:

- Find a way to make the file explorer open upon clicking the text box.
 - This would have to be compatible with all equivalent default file explorers in other OS's.
 - As Java VM is not platform specific, I'm sure the functionality to do this from the java language would be possible (maybe even the default).

User requirement: Buttons - Clear Button: A button under the digit drawing grid containing a bin icon. Predict Button: A button underneath the digit drawing grid containing the text "Predict".
Eraser Button (optional): A button underneath the digit drawing grid containing a rubber icon.

Score: 8/10 MET

Comments: This requirement was met (except for the option of an eraser button, which was replaced with a delete button, which works better anyway). The only thing I was not able to implement was icons within buttons, which I said may be valuable when planning the project (Overview I: GUI Designs, p9). It became incredibly finicky to work with icons (Stage V: GUI, p53), so I scrapped the idea. Most buttons contain single words anyway, so it could be easily

translatable; perhaps the option for different languages could be implemented, which would make the program more user-friendly for foreign stakeholders.

Development:

- Stage V: GUI, p53.

Improvements:

- The option for different language selection as to overcome language barriers.
 - Could be achieved by having a class with static methods that return the translation in a specified language.
 - Select language on the main page via a drop-down, which can be passed down through the class hierarchy when necessary.
 - Enumerate class with all supported languages.
- Icons instead of words where appropriate to overcome language barriers.
 - Of course some things in the program cannot be represented by icons.

User requirement: Visualisation / Calculation window - A window containing model visualisation, i.e. a neural network diagram.

Score: 0/10 NOT MET

Comments: I did not meet this requirement, as much as I would have liked to have more sophisticated visualisation of the network. Most beta testers thought the machine learning process was demonstrated well anyway (Post Dev I: Beta Testing Q2.g, p80). Making a whole window to visualise the network probably would have been a whole new project. Also, prediction happens far too quickly to meaningfully show any process of it.

Development:

- Stage VIII: Prediction Mode, p71.

Improvements:

- Add a more visual representation of the network.
 - There probably exists an API to generate images of networks from parameters about its shape and weights.

User requirement: Prediction box - a box containing a digit.

Score: 10/10 MET

Comments: I met this requirement as it was vital to show the result of the AI prediction. I made sure to make the font big enough to both make it obvious what the prediction was, and show the importance around the prediction. It was placed relatively with other components to preserve the layout's good structure (Post Dev I: Beta Testing Q1.b, p77).

Development:

- Stage V: GUI, p54.
- Stage VIII: Prediction Mode, p75.

Improvements:

- None to add.

User requirement: Sampling - Label Box, Sample Number text display, Sample Scrolling Buttons, Load Preinstalled Samples Button, New Sample Button.

Score: 7/10 PARTIALLY MET

Comments: All these components were made except for the sample scrolling buttons, making the sampling requirements as a whole partially met. I decided to abandon the sample viewing and scrolling (Stage VI/VII: Learning Mode Core/UX, p61) functionality since I realised it would make the UI/UX far too confusing, especially for a lot of stakeholders who may be new to the subject or trying to convey it in an understandable way. Also, there isn't much need for that functionality anyway. A significant amount of beta testers said some of the functions were confusing as is (Post Dev I: Beta Testing Q3.a&b, p80), so adding a whole new layer of use on top of it wouldn't have been a good move. I also made a good

Development:

- Stage II: Basic Input, p26-29
- Stage VI/VII: Learning Mode Core/UX, p59-64.

Improvements:

- All the buttons were of course functional, but a tutorial would have definitely been necessary.
- Perhaps sample scrolling functionality would have been beneficial, but in a different interface.

User requirement: Train Button – A functional button containing the word “Train” that starts the ML process.

Score: 10/10 MET

Comments: This requirement was met. The machine learning process is fully functional, albeit not as optimised as current processes – which are far more complex. I made the button disable once clicked also, restricting errors that would arise if the user attempted to train the same model while it is already being trained. Beta testers would agree this is a good feature (Post Dev I: Beta Testing Q1.a&c, p77). The only thing that I would try to improve is the overall training time; some beta testers thought that it took a bit too long for their liking (Post Dev I: Beta Testing Q2.f, p79). Unfortunately, processes like this aren't easily streamlined without high-end modern

methods. If stakeholders could display the machine learning process in a quicker way, that may be very useful to them, as they would most likely be in education with limited time to teach a class or themselves.

Development:

- Stage IV: Machine Learning, p47-52.
- Stage Vi/VII: Learning Mode Core/UX, p65.

Improvements:

- Research modern machine learning optimisation techniques to reduce training time.
 - This may require refactoring the whole train function.
 - Function declaration shouldn't change as to stay compatible with the rest of the program. Only the definition should change.

User requirement: Progress - Cost Graph: a small line graph/table of cost to iteration number, and Training Progress Bar: a bar showing how many samples have been used out of the total samples.

Score: 7/10 PARTIALLY MET

Comments: The progress requirements as a whole were partially met, since the cost turned out to be displayed as a number, and not on a graph. This was because I found a graph would be very difficult to implement in Java swing UI. It works for the intended purpose, but I perhaps would have liked to have something more detailed to display cost. The training progress bar was fully achieved. Generally, the ML process is displayed well (Post Dev I: Beta Testing Q2.g, p80), which is very important to the stakeholders; displaying the process is the main aim of the program.

Development:

- Stage VI/VII: Learning Mode Core/UX, p65-66.

Improvements:

- Find a way to display a graph of cost.
 - This could be integrated directly into the UI.
 - Or maybe it could use an API to create a graph from a generated CSV.

User requirement: Save Model Button - A button containing a file icon which saves the model to a JSON file.

Score: 8/10 MET

Comments: This requirement was met. The only thing I couldn't add was the icon, as I have discussed in other parts of this evaluation. I would have liked to have one, as typically anything file-related is made clear with icons. To stakeholders, this would have made the interface look a

bit more professional and clear. Something I didn't think about at the time was file extensions – as long as the file that is attempted to load ends with “.json”, it will try to load it. This could raise issues since practically any file could be renamed and loaded (which of course will cause a crash). One way around this is to add checks for the format of the file contents, and maybe make a specific file type for the program, so it won't even bother trying to load anything else.

Development:

- Stage III: Neural Network Classes, p41-45.
- Stage VIII: Prediction Mode, p71-74

Improvements:

- Add an icon to the button for a more professional, polished look.
- Make a specific file extension for the program to load.
 - And add checks to make sure any files are the correct format.

User requirement: Back Button - A button containing a < symbol which takes you back to main page.

Score: 10/10 MET

Comments: This requirement was met, and I had no trouble implementing whatsoever, as the functionality is so simple. Perhaps a more sophisticated icon could have been used instead of a “<”, but this is a very minute detail. Beta testers said the UI was neat (Post Dev I: Beta Testing Q1.b, p77), which can often be achieved by shared characteristics across all interfaces, i.e. a common back button. It also allows easy navigation for stakeholders trying to easily demonstrate many processes at once.

Development:

- Overview I: GUI Designs, p11.
- Stage V: GUI, p54.

Improvements:

- Use something more sophisticated than “<”

User requirement: Linear Algebra Library - A library containing definitions, functions and operations involving vectors and matrices.

Score: 10/10 MET

Comments: This requirement was met. It was amongst the first requirements I fulfilled as it was so vital to the program. I had few issues with the development of it, mainly dealing with inheritance to NN classes (example Stage III: Machine Learning, p38), but in the end I solved these problems by manually casting types. Although this library lays the foundation for the

program, the stakeholders would not see it, so anything but it's functionality would not matter to them.

Development:

- Stage I: Core Mathematical Classes, p19-25

Improvements:

- Could have used pre-made libraries for this, but I wanted to make something tailored to my own project that I would get to know well by making it.
 - I was also up for a challenge.

User requirement: Neural Network Class - A class holding number and sizes of layers, functions to get predictions, calculate cost, back-propagate and save all weights and biases in a JSON file.

Score: 10/10 MET

Comments: This requirement was met. Another foundation of the program. I definitely had trouble getting some of the features to work, but it almost always came down to logic errors, which are very easy to make in this case. Again, to a stakeholder, only the functionality of this requirement matters. Showing how it is working is more important.

Development:

- Stage III: Neural Network Classes, p30-40

Improvements:

- Could have used pre-made libraries for this, but I wanted to make something tailored to my own project that I would get to know well by making it.
 - I was also up for a challenge.

Conclusion

In conclusion, I think I made a decent program for it's use case. It displays the ML process well, and lets you interact with it. However, its downsides are confusion, especially to new stakeholders who may be inexperienced with the topic and lacking a tutorial. Going through my post-development targets (Overview III: Development Plan, p17):

- Can the program be navigated in its entirety? Yes
- Can a model be trained? Yes
- Can the trained model be saved? Yes
- Can a user-created model be loaded? Yes
- Can a prediction be made from user input? Yes
- How accurate are the models? Mediocre

Appendix: Code Listings

Main.java

```
public class Main {  
    public static void main(String[] args) {  
        MainForm mainForm = new MainForm();  
    }  
}
```

MainForm.java

```
import javax.swing.*;

public class MainForm extends JFrame {
    private JButton loadModelButton;
    private JButton trainNewModelButton;
    private JButton exitButton;
    private JPanel panell;

    public MainForm() {
        setContentPane(panell);
        setTitle("DigiTrace");
        setSize(720, 480);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setVisible(true);

        exitButton.addActionListener(e -> dispose());
        loadModelButton.addActionListener(e -> {
            PredictionModeForm predictionModeForm = new
PredictionModeForm();
            });
        trainNewModelButton.addActionListener(e -> {
            LearningModeForm learningModeForm = new
LearningModeForm();
            });
    }
}
```

LearningModeForm.java

```
import NeuralNetwork.NeuralNetwork;
import NeuralNetwork.NNPrediction;
import Sampling.Sample;
import Sampling.SampleImage;
import Utils.Utils;

import javax.swing.*;
import java.awt.event.*;
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Collections;
import java.util.List;
import java.util.stream.Stream;

public class LearningModeForm extends JFrame {
    private JPanel panell;
    private JButton backButton;
    private JLabel totalSamplesLabel;
    private JTextField labelTextField;
    private JTable drawingGrid;
    private JButton deleteButton;
    private JButton addSampleButton;
    private JButton loadPreInstalledSamplesButton;
    private JLabel iterationNoValueLabel;
    private JLabel costLabel;
    private JButton trainButton;
```

```

private JProgressBar learningProgressBar;
private JButton saveModelButton;
private int rowSelected;
private int colSelected;
private boolean penDown;
private int iterNumber;
private int sampleAmount;
private NNPrediction currPred;
private NeuralNetwork NN;

public LearningModeForm() {
    setContentPane(panell1);
    setTitle("Learning Mode");
    setSize(1000, 900);
    setDefaultCloseOperation(DISPOSE_ON_CLOSE);
    setVisible(true);

    NN = new NeuralNetwork(false);
    currPred = new NNPrediction(0, 1);
    penDown = false;
    drawingGrid.addMouseListener(new MouseAdapter() {
        @Override
        public void mouseClicked(MouseEvent e) {
            penDown = !penDown;
        }
    });
    backButton.addActionListener(e -> dispose());
    deleteButton.addActionListener(e -> clear());
    addSampleButton.addActionListener(e -> {
        int[] binaryArray;

```

```

        Sample sample;

        int label;

        binaryArray = Utils.tableToBinaryArray(drawingGrid);

        try { // Check if there's actually an integer inputted
            label =
Integer.parseInt(labelTextField.getText().substring(0, 1));
        } catch (NumberFormatException |
StringIndexOutOfBoundsException exception) {
            // Escape if invalid label
            return;
        }

        // Empty array check already implemented elsewhere
        sample = new Sample(new SampleImage(binaryArray), label);
        NN.addSample(sample);
        // If all successful clear grid
        clear();
        // Update sample count
        totalSamplesLabel.setText(String.format("%d",
NN.getSampleAmount()));
        labelTextField.setText("");
    });

    loadPreInstalledSamplesButton.addActionListener(e -> {
        // Disable the button (one time use)
        Thread guiChanges = new Thread(() -> {
            loadPreInstalledSamplesButton.setText("Loading...");
            loadPreInstalledSamplesButton.setEnabled(false);
        });
        guiChanges.start();
        Thread loadingSamples = new Thread(() -> {

```



```

        loadPreInstalledSamples();

        loadPreInstalledSamplesButton.setText("Pre-Installed
Samples Loaded");

    });

    loadingSamples.start();
});

trainButton.addActionListener(e -> {

    sampleAmount = NN.getSampleAmount();

    if (sampleAmount == 0) return;

    learningProgressBar.setMaximum(sampleAmount);

    Thread machineLearning = new Thread(this::train);
    machineLearning.start();

    Thread guiUpdates = new Thread(() -> {
        trainButton.setEnabled(false);
        while (iterNumber < sampleAmount - 1) {
            iterationNoValueLabel.setText(String.format("%d",
iterNumber + 1));
            costLabel.setText(String.format("%f",
currPred.getCost()));
            learningProgressBar.setValue(iterNumber + 1);
        }
        saveModelButton.setEnabled(true);
    });
    guiUpdates.start();
});

saveModelButton.addActionListener(e -> {
    // 10 digit random number sequence

```

```

10));

        long serialID = Math.round(Math.random() * Math.pow(10,
String serialStringID = Long.toString(serialID);
while (serialStringID.length() < 10) {
    // Add 0's to end if not 10 digits
    serialStringID = serialStringID.concat("0");
}
String fileName = "NETWORK" + serialStringID;
NN.storeAsJSON(fileName);
saveModelButton.setText("Model stored in " + fileName +
".json");

saveModelButton.setEnabled(false);
});

drawingGrid.addMouseMotionListener(new MouseMotionAdapter() {
    @Override
    public void mouseMoved(MouseEvent e) {
        super.mouseMoved(e);
        if (penDown) {
            rowSelected =
drawingGrid.rowAtPoint(e.getPoint());

            colSelected =
drawingGrid.columnAtPoint(e.getPoint());

            if (drawingGrid.getValueAt(rowSelected,
colSelected) == null) {
                drawingGrid.setValueAt("///", rowSelected,
colSelected);
            }
        }
    }
});
}

```

```

private void train() {
    Sample currSample;
    for (iterNumber = 0; iterNumber < sampleAmount; ++iterNumber)
{
        currSample = NN.getSample(iterNumber);
        currPred = NN.predict(currSample);
        NN.learn(currSample);
    }
}

```

```

private void createUIComponents() {
    drawingGrid = new JTable(28, 28) {
        @Override
        public boolean isCellEditable(int row, int column) {
            return false;
        }
    };
}

```

```

private void clear() {
    for (int i = 0; i < 28; ++i) {
        for (int j = 0; j < 28; ++j) {
            drawingGrid.setValueAt(null, i, j);
        }
    }
}

```

```

private void loadPreInstalledSamples() {
    // 75000 Samples
    Stream<Path> paths;
    try {

```

```

        // Recursive walk around sample directory
        paths = Files.walk(Paths.get("./HN/CompleteImages/All data
(Compressed)"));
    } catch (IOException ioE) {
        ioE.printStackTrace();
        return;
    }
    List<File> files = new java.util.ArrayList<>(
        paths
        // Gets rid of directories
        .filter(Files::isRegularFile)
        // Converts files to absolute paths
        .map(Path::toAbsolutePath)
        // Path -> File
        .map(Path::toFile)
        .toList());

    // So it can sample in a random order
    Collections.shuffle(files);

    paths.close();

    files.forEach(f -> {
        SampleImage sImg = new SampleImage(f);
        // file[0] (first char) will be the label
        Sample s = new Sample(sImg,
Integer.parseInt(f.getName().substring(0, 1)));
        NN.addSample(s);
        NN.addSample(s);
        totalSamplesLabel.setText(String.format("%d",
NN.getSampleAmount()));
    });

```

```
        }) ;  
    }  
}
```

PredictionModeForm.java

```
import NeuralNetwork.NeuralNetwork;
import NeuralNetwork.NNPrediction;
import Sampling.Sample;
import Sampling.SampleImage;
import Utils.Utils;

import javax.swing.*.*;
import java.awt.*.*;
import java.awt.event.*.*;

public class PredictionModeForm extends JFrame {
    private JPanel panell1;
    private JTextField fileNameTextField;
    private JTable drawingGrid;
    private JButton deleteButton;
    private JButton predictButton;
    private JLabel predictionLabelResult;
    private JButton backButton;
    private JButton confirmFileButton;
    private int rowSelected;
    private int colSelected;
    private boolean penDown;

    public PredictionModeForm() {
        setContentPane(panell1);
        setTitle("Prediction Mode");
        setSize(720, 720);
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        setVisible(true);
    }
}
```

```

predictButton.setEnabled(false);
deleteButton.setEnabled(false);
NeuralNetwork NN = new NeuralNetwork(true);
penDown = false;

confirmFileButton.addActionListener(e -> {
    String fileName = fileNameTextField.getText();
    int loadStatus = NN.loadFromJSON(fileName);

    // File could not load
    if (loadStatus == 1) {
        fileNameTextField.setForeground(Color.RED);
        predictButton.setEnabled(false);
        deleteButton.setEnabled(false);
        return;
    }

    // Could load
    fileNameTextField.setForeground(Color.GREEN);
    predictButton.setEnabled(true);
    deleteButton.setEnabled(true);
});

predictButton.addActionListener(e -> {
    int[] binaryArray;
    NNPrediction prediction;
    Sample sampleInput;

    binaryArray = Utils.tableToBinaryArray(drawingGrid);
    // We do not care about the label here

```

```

        sampleInput = new Sample(new SampleImage(binaryArray), 0);
        prediction = NN.predict(sampleInput);
        predictionLabelResult.setText(String.format("%d",
prediction.getResult()));
    });
    drawingGrid.addMouseListener(new MouseAdapter() {
        @Override
        public void mouseClicked(MouseEvent e) {
            penDown = !penDown;
        }
    });
    backButton.addActionListener(e -> dispose());
    deleteButton.addActionListener(e -> clear());
    drawingGrid.addMouseMotionListener(new MouseMotionAdapter() {
        @Override
        public void mouseMoved(MouseEvent e) {
            super.mouseMoved(e);
            if (penDown) {
                rowSelected =
drawingGrid.rowAtPoint(e.getPoint());
                colSelected =
drawingGrid.columnAtPoint(e.getPoint());
                if (drawingGrid.getValueAt(rowSelected,
colSelected) == null) {
                    drawingGrid.setValueAt("///", rowSelected,
colSelected);
                }
            }
        }
    });
}

```



```

private void createUIComponents() {
    drawingGrid = new JTable(28, 28) {
        @Override
        public boolean isCellEditable(int row, int column) {
            return false;
        }
    };
}

private void clear() {
    for (int i = 0; i < 28; ++i) {
        for (int j = 0; j < 28; ++j) {
            drawingGrid.setValueAt(null, i, j);
        }
    }
}
}

```

Matrix.java

```
package LinearAlgebra;
```

```
import java.util.Arrays;
```

```
import java.util.function.DoubleUnaryOperator;
```

```
public class Matrix {
```

```
    protected int rows, cols;
```

```
    protected double[][] elements;
```

```
    // Instantiate with 1D Array
```

```
    public Matrix(double[] e, int rows, int cols) {
```

```
        this.rows = rows;
```

```
        this.cols = cols;
```

```
        this.elements = new double[rows][cols];
```

```
        for (int i = 0; i < rows; ++i) {
```

```
            for (int j = 0; j < cols; ++j) {
```

```
                this.elements[i][j] = e[j + cols * i];
```

```
            }
```

```
        }
```

```
    }
```

```
    // Instantiate with 2D array
```

```
    public Matrix(double[][] e, int rows, int cols) {
```

```
        this.rows = rows;
```

```
        this.cols = cols;
```

```
        this.elements = e;
```

```
    }
```

```

// Instantiate entire matrix with 1 double
public Matrix(double e, int rows, int cols) {
    this.rows = rows;
    this.cols = cols;
    this.elements = new double[rows][cols];

    for (int i = 0; i < rows; ++i) {
        Arrays.fill(this.elements[i], e);
    }
}

public int getRows() {
    return rows;
}

public int getCols() {
    return cols;
}

public double[][] asArray() {
    return elements;
}

public double getElement(int row, int col) {
    return elements[row][col];
}

public void setElement(int row, int col, double element) {
    elements[row][col] = element;
}

```

```

// Adds two matrices of the same dimension
public Matrix add(Matrix m) {
    if (!(m.getRows() == rows && m.getCols() == cols)) {
        return new Matrix(-1, rows, cols);
    } else {
        double[][] res = new double[rows][cols];

        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < cols; ++j) {
                // Adds corresponding elements
                res[i][j] = elements[i][j] + m.getElement(i, j);
            }
        }

        return new Matrix(res, rows, cols);
    }
}

// Applies a function to each element of the array
public Matrix apply(DoubleUnaryOperator f) {
    double[][] res = new double[rows][cols];

    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            res[i][j] = f.applyAsDouble(elements[i][j]);
        }
    }

    return new Matrix(res, rows, cols);
}

```

```

    }

    // A (m x n) . B (p x q) = C (m x q), given n == p
    public Matrix multiply(Matrix m) {
        if (cols != m.getRows()) return new Matrix(-1, rows, cols);
        else {
            double[][] res = new double[rows][m.getCols()];

            // Dot product of each of A's rows with each of B's columns
            for (int i = 0; i < rows; ++i) {
                for (int j = 0; j < m.getCols(); ++j) {
                    double productSum = 0;
                    for (int k = 0; k < cols; ++k) {
                        productSum += elements[i][k] * m.getElement(k,
j));
                    }

                    res[i][j] = productSum;
                }
            }

            return new Matrix(res, rows, m.getCols());
        }
    }

    // Multiply by scalar
    public Matrix multiply(double n) {
        return apply(x -> (n * x));
    }
}

```

Vector.java

```
package LinearAlgebra;
```

```
import java.util.Arrays;
```

```
public class Vector extends Matrix {  
    public Vector(double[] e, int rows) {  
        super(e, rows, 1);  
    }  
  
    public Vector(double e, int rows) {  
        super(e, rows, 1);  
    }  
  
    public double square_magnitude() {  
        return Arrays.stream(as1DArray())  
            .map(e -> Math.pow(e, 2))  
            .sum();  
    }  
  
    public double magnitude() {  
        return Math.sqrt(square_magnitude());  
    }  
  
    public double[] as1DArray() {  
        double[] elems = new double[rows];  
        for (int i = 0; i < rows; ++i) {  
            elems[i] = elements[i][0];  
        }  
    }  
}
```

```
        return elems;
    }

    public double getElement(int row) {
        return super.getElement(row, 0);
    }

    public void setElement(int row, double element) {
        super.setElement(row, 0, element);
    }
}
```

Layer.java

```
package NeuralNetwork;

import LinearAlgebra.Matrix;
import LinearAlgebra.Vector;
import Utils.Functions;
import Utils.Utils;

public class Layer extends Vector {
    private Matrix weights;
    private Vector biases;
    private LayerType type;

    public Layer(LayerType type) {
        super(
            0,
            type == LayerType.INPUT ? Utils.INPUT_LAYER_SIZE :
            type == LayerType.HIDDEN || type ==
LayerType.PRE_OUTPUT ? Utils.HIDDEN_LAYER_SIZE :
            type == LayerType.OUTPUT ? Utils.OUTPUT_LAYER_SIZE : 0
        );
        weights =
            // Weight matrix dimensions in form LayerType x
LayerType
            // (Where the type indicates the size of that layer
type)

            type == LayerType.INPUT
            // INPUT : hidden x input
            ? new Matrix(0, Utils.HIDDEN_LAYER_SIZE,
Utils.INPUT_LAYER_SIZE) :
            type == LayerType.HIDDEN
            // HIDDEN : hidden x hidden
```



```

        ? new Matrix(0, Utils.HIDDEN_LAYER_SIZE,
Utils.HIDDEN_LAYER_SIZE) :

        type == LayerType.PRE_OUTPUT

        // PRE-OUTPUT : output x hidden

        ? new Matrix(0, Utils.OUTPUT_LAYER_SIZE,
Utils.HIDDEN_LAYER_SIZE)

        // OUTPUT : null (needs no weights)

        : null;

    biases =

        // Bias vector has the dimensions of the NEXT layer

        type == LayerType.INPUT ? new Vector(0,
Utils.HIDDEN_LAYER_SIZE) :

        type == LayerType.HIDDEN ? new Vector(0 ,
Utils.HIDDEN_LAYER_SIZE) :

        type == LayerType.PRE_OUTPUT ? new Vector(0,
Utils.OUTPUT_LAYER_SIZE) : null;

    this.type = type;
}

public void setLayer(double[] array) {
    if (array.length == rows) {
        for (int i = 0; i < rows; ++i) {
            setElement(i, 0, array[i]);
        }
    } else {
        throw new IndexOutOfBoundsException("Attempted to set
layer to mismatched array");
    }
}

public void activate() {

```

```

        Matrix result = apply(Functions::sigmoid);
        double[] elems = new double[rows];
        // Resets the layer directly from matrix
        // (Explicit cast didn't work)
        for (int i = 0; i < rows; ++i) {
            elems[i] = result.getElement(i, 0);
        }
        setLayer(elems);
    }

    public Matrix getWeights() {
        return weights;
    }

    public void alterWeight(int row, int col, double value) {
        weights.setElement(row, col, weights.getElement(row, col) +
value);
    }

    public Vector getBiases() {
        return biases;
    }

    public void alterBias(int row, double value) {
        biases.setElement(row, biases.getElement(row) + value);
    }

    public LayerType getType() {
        return type;
    }
}

```

LayerType.java

```
package NeuralNetwork;
```

```
public enum LayerType {  
    INPUT,  
    HIDDEN,  
    PRE_OUTPUT,  
    OUTPUT;  
}
```

```
package NeuralNetwork;

import LinearAlgebra.Matrix;
import LinearAlgebra.Vector;
import Sampling.Sample;
import Utils.Utils;
import static Utils.Functions.*;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Scanner;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.json.JSONObject;
import org.json.JSONArray;

public class NeuralNetwork {
    private Layer[] layers;
    private LinkedList<Sample> samples;
    private final double ALPHA = 0.2d;

    public NeuralNetwork(boolean initToZero) {
        layers = new Layer[5];
        layers[0] = new Layer(LayerType.INPUT);
```

```

        layers[1] = new Layer(LayerType.HIDDEN);
        layers[2] = new Layer(LayerType.HIDDEN);
        layers[3] = new Layer(LayerType.PRE_OUTPUT);
        layers[4] = new Layer(LayerType.OUTPUT);
        samples = new LinkedList<>();
        if (!initToZero) randomise();
    }

    private void randomise() {
        for (Layer layer : layers) {
            if (layer.getWeights() == null) return;
            for (int i = 0; i < layer.getWeights().getRows(); ++i) {
                double rand;
                for (int j = 0; j < layer.getWeights().getCols(); ++j)
                {
                    // Random value between -1 and 1
                    rand = Math.random() * 2 - 1;
                    layer.alterWeight(i, j, rand);
                }
                rand = Math.random() * 2 - 1;
                layer.alterBias(i, rand);
            }
        }
    }

    public NNPrediction predict(Sample sample) {
        Layer currentLayer;
        int currentLayerIdx;

        // Set input layer to input pixel array
        double[] pixelArray = Arrays.stream(sample.getPixelArray())

```

```

        .asDoubleStream()
        .toArray();
layers[0].setLayer(pixelArray);
// Current layer is input layer
currentLayerIdx = 0;
currentLayer = layers[currentLayerIdx];
// Loop if current layer is not output layer
Matrix resultMatrix;
double[] resultElements;
while (currentLayerIdx < 4) {
    Layer result = new
Layer(layers[currentLayerIdx+1].getType());
    resultMatrix = currentLayer.getWeights();
    // Multiply weight matrix
    resultMatrix = resultMatrix.multiply(currentLayer)
        // Add biases
        .add(currentLayer.getBiases());
    // Matrix -> Layer
    resultElements = new double[resultMatrix.getRows()];
    for (int i = 0; i < resultMatrix.getRows(); ++i) {
        resultElements[i] = resultMatrix.getElement(i, 0);
    }

    // Set next layer to result
    layers[++currentLayerIdx].setLayer(resultElements);
    // Activate next layer
    layers[currentLayerIdx].activate();
    // Set current layer to next layer
    currentLayer = layers[currentLayerIdx];
}

```

```

// Current layer is now output layer
int prediction_digit = 0;
double cost;

double greatestElement = 0;
for (int i = 0; i < 10; ++i) {
    double element = currentLayer.getElement(i);
    // Finds the greatest element,
    // the index of which is the prediction
    if (element > greatestElement) {
        greatestElement = element;
        prediction_digit = i;
    }
}

//  $y - \bar{y}$ 
Matrix error_vector_m =
currentLayer.add(Utils.oneHotVector(sample.getLabel()).multiply(-1));
Vector error_vector = new Vector(0, error_vector_m.getRows());
// Matrix -> Vector
for (int i = 0; i < error_vector_m.getRows(); ++i) {
    error_vector.setElement(i, error_vector_m.getElement(i,
0));
}

//  $C = |y - \bar{y}|^2 / 10$ 
cost = error_vector.square_magnitude() / 10;

return new NNPrediction(prediction_digit, cost);
}

public void learn(Sample sample) {

```

```

final int n = 4;
double dCdw, dCdb, productChain;
int iBound, jBound;
for (int k = 1; k <= 4; ++k) {
    dCdw = 1.0d;
    dCdb = 1.0d;
    iBound = layers[n-k].getRows();
    if (n-k-1 >= 0) {
        jBound = layers[n - k - 1].getRows();
    } else {
        jBound = Utils.INPUT_LAYER_SIZE;
    }
    for (int i = 0; i < iBound; ++i) {
        // MIN functions used to not let indexes go out of
        // Ternary expression at the end equivalent to one-hot
        dCdb =
layers[n].getElement(Math.min(layers[n].getRows()-1, i)) - (i ==
sample.getLabel() ? 1 : 0);
        dCdb *= 0.2d;
        productChain = 1;
        for (int m = 0; m <= k - 2; ++m) {
            productChain *=
                sigmoidDiff(sigmoidInv(layers[n-
m].getElement(Math.min(layers[n-m].getRows()-1, i))))
                    * layers[n-m-1].getWeights().getElement(
                        Math.min(layers[n-m-
1].getWeights().getRows()-1, i),
                        Math.min(layers[n-m-
1].getWeights().getRows()-1, i)
                    );
        }
    }
}

```



```

        dCdb *= productChain
            * sigmoidDiff(sigmoidInv(layers[n-
k+1].getElement(Math.min(layers[n-k+1].getRows()-1, i))));

        layers[n-k].alterBias(Math.min(layers[n-
k].getBiases().getRows()-1, i), -ALPHA*dCdb);

        for (int j = 0; j < jBound; ++j) {
            dCdw = dCdb * layers[n-
k].getElement(Math.min(layers[n-k].getRows()-1, j));

            layers[n-k].alterWeight(
                Math.min(layers[n-
k].getWeights().getRows()-1, i),
                Math.min(layers[n-
k].getWeights().getCols()-1, j),
                -ALPHA*dCdw);
        }
    }
}

}

}

public void addSample(Sample sample) {
    if (Arrays.stream(sample.getPixelArray()).allMatch(x -> x ==
0)) {
        System.out.println("Empty Sample, not adding");
        return;
    }
    samples.add(sample);
}

public void removeSample(int idx) {
    samples.remove(idx);
}

```

```

public Sample getSample(int idx) {
    return samples.get(idx);
}

public int getSampleAmount() {
    return samples.size();
}

/*
Abstractions of Layer / NeuralNetwork classes
containing only weight and bias values.
*/
private record LayerObject(double[][] weights, double[] biases) {}
private record NetworkObject(LayerObject[] layers) {}

public void storeAsJSON(String fileName) {
    File file;
    NetworkObject NNO;

    // Copy all weights and biases to layer objects
    LayerObject[] layerObjs = new LayerObject[4];
    for (int i = 0; i < 4; ++i) {

        layerObjs[i] = new LayerObject(
            this.layers[i].getWeights().toArray(),
            this.layers[i].getBiases().as1DArray()
        );
    }
    NNO = new NetworkObject(layerObjs);
}

```

```

        if (!fileName.endsWith(".json")) {
            fileName = fileName.concat(".json");
        }
        // Create JSON file
        file = new File("./Networks/" + fileName);

        // ObjectMapper can write Objs as JSON
        ObjectMapper mapper = new ObjectMapper();

        // String is an error message by default, which is reset if
rewritten
        String json = "ERROR: COULD NOT WRITE JSON";
        try {
            // Neural Network Object as JSON
            json = mapper.writeValueAsString(NNO);
        } catch (JsonProcessingException e) {
            e.printStackTrace();
        }

        try {
            // Write JSON to file
            FileWriter writer = new FileWriter(file);
            writer.write(json);
            writer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public int loadFromJSON(String fileName) {
        File file;
        Scanner scanner;
    }

```

```

StringBuilder sb; // To piece file together
String jsonString;

// Open file
file = new File(fileName);
try {
    scanner = new Scanner(file);
} catch (FileNotFoundException e) {
    return 1;
}

if (!fileName.endsWith(".json")) return 1;

// Read file
sb = new StringBuilder();
while (scanner.hasNextLine()) {
    sb.append(scanner.nextLine());
}
scanner.close();

jsonString = sb.toString();
JSONObject parsedJSON = new JSONObject(jsonString);
JSONArray layersJSON = new
JSONArray(parsedJSON.getJSONArray("layers"));
for (int layerIdx = 0; layerIdx < 4; ++layerIdx) {
    JSONObject layerJSON = layersJSON.getJSONObject(layerIdx);
    for (int i = 0; i < layers[layerIdx+1].getRows(); ++i) {
        // alterWeight/alterBias equivalent to
        // setting when initial value is 0
        JSONArray weightsJSON =
layerJSON.getJSONArray("weights");

```

```

        JSONArray biasesJSON =
layerJSON.getJSONArray("biases");

        layers[layerIdx].alterBias(
            i, biasesJSON.getDouble(i)
        );
        for (int j = 0; j < layers[layerIdx].getRows(); ++j) {
            layers[layerIdx].alterWeight(
                i, j,
                weightsJSON.getJSONArray(i).getDouble(j)
            );
        }
    }
}
return 0;
}
}

```

```
package NeuralNetwork;

public class NNPrediction {
    private int result;
    private double cost;

    public NNPrediction(int result, double cost) {
        this.result = result;
        this.cost = cost;
    }

    public int getResult() {
        return result;
    }

    public double getCost() {
        return cost;
    }
}
```

Sample.java

```
package Sampling;

public class Sample {
    private int label;
    private int[] pixelArray;

    public Sample(SampleImage image, int label) {
        this.label = label;
        this.pixelArray = image.getBinaryArray();
    }

    public int getLabel() {
        return label;
    }

    public int[] getPixelArray() {
        return pixelArray;
    }
}
```

SampleImage.java

```
package Sampling;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.Arrays;

public class SampleImage {
    private BufferedImage image;
    private int[] binaryArray;

    public SampleImage(File file) {
        binaryArray = null;
        try {
            image = ImageIO.read(file);
        } catch (IOException ioE) {
            System.err.println(Arrays.toString(ioE.getStackTrace()));
        }
    }

    public SampleImage(int[] pixelArray) {
        image = null;
        if (pixelArray.length != 784) {
            throw new IndexOutOfBoundsException("Attempted to make
sample of wrong size");
        }
        binaryArray = pixelArray;
    }
}
```



```

public int[] getBinaryArray() {
    if (image != null) {
        binaryArray = new int[784];
        for (int i = 0; i < 28; ++i) {
            for (int j = 0; j < 28; ++j) {
                binaryArray[j + 28 * i] = image.getRGB(j, i) ==
0xFF000000 ? 1 : 0;
            }
        }
        return binaryArray;
    }
}

```

Functions.java

```
package Utils;
```

```
public class Functions {  
    public static double sigmoid(double z) {  
        return 1 / (1 + Math.exp(-z));  
    }  
    public static double sigmoidDiff(double z) {  
        double expTerm = Math.exp(-z);  
        return expTerm / Math.pow(1 + expTerm, 2);  
    }  
    public static double sigmoidInv(double z) {  
        return Math.log(z / (1 - z));  
    }  
}
```

Utils.java

```
package Utils;

import LinearAlgebra.Matrix;
import LinearAlgebra.Vector;

import javax.swing.*;

public class Utils {
    public static final int INPUT_LAYER_SIZE = 784;
    public static final int HIDDEN_LAYER_SIZE = 16;
    public static final int OUTPUT_LAYER_SIZE = 10;

    // Prints matrix row-wise
    public static void printMatrix(Matrix m) {
        for (int i = 0; i < m.getRows(); ++i) {
            System.out.print("[ ");
            for (int j = 0; j < m.getCols(); ++j) {
                System.out.print(m.getElement(i, j) + " ");
            }
            System.out.print("]\n");
        }
    }

    public static Vector oneHotVector(int digit) {
        double[] elements = new double[10];
        // All elements 0,
        // except the one with the correct digit, which is 1
        elements[digit] = 1;
        return new Vector(elements, 10);
    }
}
```

```

    }

    public static int[] tableToBinaryArray(JTable table) {
        int[] binaryArray = new int[784];
        for (int i = 0; i < 28; ++i) {
            for (int j = 0; j < 28; ++j) {
                binaryArray[i * 28 + j] = table.getValueAt(i, j) ==
null ? 0 : 1;
            }
        }
        return binaryArray;
    }
}

```