Automating Address Reading Using Optical Character Recognition

Note: This document was submitted as part of the coursework and gives an in-depth overview of the entire project. I have included it here for completeness, but it is not required to read to understand the program’s purpose. Instead check the README.txt

# Table of Contents

[Table of Contents 2](#_Toc90651914)

[Analysis 4](#_Toc90651915)

[Problem Description 4](#_Toc90651916)

[User Requirements 5](#_Toc90651917)

[Existing Systems 5](#_Toc90651918)

[Royal Mail Interface 6](#_Toc90651919)

[Research 7](#_Toc90651920)

[Neural Networks 7](#_Toc90651921)

[Machine Learning 10](#_Toc90651922)

[Image Processing 18](#_Toc90651923)

[Datasets 19](#_Toc90651924)

[Potential Systems 22](#_Toc90651925)

[Objectives 25](#_Toc90651926)

[References 27](#_Toc90651927)

[Design 28](#_Toc90651928)

[Description 28](#_Toc90651929)

[Decomposition Diagrams 29](#_Toc90651930)

[Machine Learning 29](#_Toc90651931)

[Address Reading 29](#_Toc90651932)

[Interface Design 31](#_Toc90651933)

[Machine Learning 31](#_Toc90651934)

[Address Reading 33](#_Toc90651935)

[Data Dictionary 34](#_Toc90651936)

[NeuralNetwork Class 34](#_Toc90651937)

[TrainingDataReader Class 34](#_Toc90651938)

[NNTrainer 35](#_Toc90651939)

[Machine Learning 36](#_Toc90651940)

[Address Reader 36](#_Toc90651941)

[Pseudocode 38](#_Toc90651942)

[Objective 1: Creating a base neural network 38](#_Toc90651943)

[Objective 2: Neural Network Serialisation 40](#_Toc90651944)

[Objective 3: Calculating Activations 41](#_Toc90651945)

[Objective 4: Reading MNIST 42](#_Toc90651946)

[Objective 5: Accuracy 44](#_Toc90651947)

[Objective 6: Finding Output Layer Gradients 45](#_Toc90651948)

[Objective 7: Backpropagation 46](#_Toc90651949)

[Objective 8: Gradient Decent 48](#_Toc90651950)

[Objective 9: Address Input 49](#_Toc90651951)

[Objective 10: Image Thresholding 50](#_Toc90651952)

[Objective 11: Split address into individual letters 51](#_Toc90651953)

[Objective 12: Compression 55](#_Toc90651954)

[Objective 13: Recognise letters using Neural Network 57](#_Toc90651955)

[Objective 14: Detect Addresses 58](#_Toc90651956)

[Class Diagram 59](#_Toc90651957)

[Implementation 60](#_Toc90651958)

[Overview 60](#_Toc90651959)

[Code Listing 62](#_Toc90651960)

[Neural Network Library 62](#_Toc90651961)

[MatrixFunctions.cs 62](#_Toc90651962)

[NeuralNetwork.cs 65](#_Toc90651963)

[NNTrainer.cs 67](#_Toc90651964)

[TrainingSample.cs 71](#_Toc90651965)

[Machine Learning 72](#_Toc90651966)

[Form1.cs 72](#_Toc90651967)

[GenerateNeuralNetwork.cs 76](#_Toc90651968)

[Address Reader 78](#_Toc90651969)

[Form1.cs 78](#_Toc90651970)

[Techniques Used 89](#_Toc90651971)

[Coding Style 91](#_Toc90651972)

[Interface Screenshots 93](#_Toc90651973)

[Machine Learning 93](#_Toc90651974)

[Address Reading 97](#_Toc90651975)

[Testing 100](#_Toc90651976)

[Test Evidence 107](#_Toc90651977)

[Evaluation 129](#_Toc90651978)

[Objective Appraisal 129](#_Toc90651979)

[Improvements 132](#_Toc90651980)

# Analysis

## Problem Description

In a postal system, written addresses on letters need to be taken and interpreted so that letters can be grouped together and then transported together. This could be done manually by hand but would take a long time and requiring many staff to be able to meet demand. Therefore, this project aims to create a design that can automate the process of reading written addresses so that they can later be grouped together, reducing costs and improving efficiency.

## User Requirements

Main Source: (Royal Mail, 2019)

This project is being designed to improve the workflow of a post office by automatically reading and storing the addresses of letters to be processed and routed by other systems. As addresses will always be in the same location on a letter, a camera could be positioned in a static location to take photos and this project will have to be able to read multiple words across multiple lines of text. The parsed text could then be outputted to a text file to be used by other programs.

### Existing Systems

Current royal mail ILSMs (Intelligent Letter sorting machines) read addresses by using a process called optical character recognition (OCR). This is where the letters in a photo are identified and separated from each other before the characters are recognised by the machine.

A way of implementing OCR is to use a neural network which has been trained on thousands of images of handwritten letters. This allows it to recognise letters even if they are written in different handwriting styles.

For an address to be taken as valid it must have:

* 1 Premise Element (such as a house name or number)
* 1 thoroughfare element (such as local village)
* 1 locality element (such as a larger town/ city)
* The postcode

Each of these elements must be on separate lines with the post code on the last line

Any address that cannot be read for any reason such as poor handwriting or text that is too large is separated and then read by a human.

Once the address has been read, the Postcode address file dataset of all addresses in the UK is used validate the read address and then route the letter to a location which is encoded as a bar code printed on the letter. This letter can now be sorted and delivered.

For the postal system to be efficient, addresses will have to be read quickly so the amount of human interaction with the system should be minimised. This could be done by ensuring that any program’s IO can be handled automatically by moving files instead of manually selecting photos.

### Royal Mail Interface

Graphical user interface, website

Description automatically generated

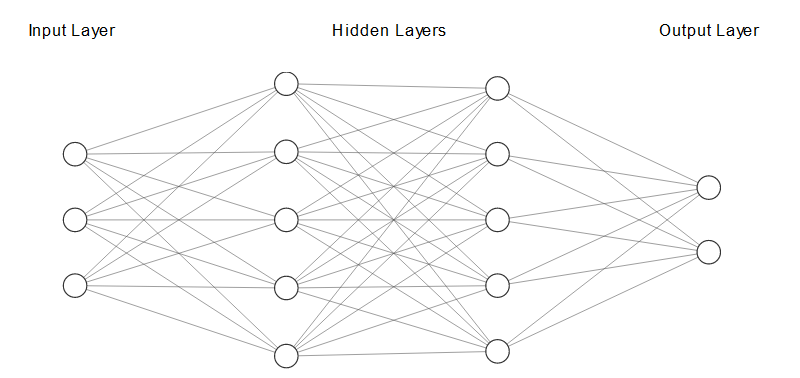
The royal mail uses a red, yellow, and white colour scheme for its website and services. As projects like this will be used within the mail service, I should use a similar colour scheme for my address reading program. Red (220, 50, 50) and yellow (253, 218, 36)

## Research

### Neural Networks

Main Source: (Sanderson, 2019, p. But what is a Neural Network?)

Neural networks have a lot of existing documentation for how to construct them as well as how a method of machine learning called gradient decent can be used to train them.



##### Activations

A neural network is a tree of nodes or “neurons” made up of layers where each node has a relationship with every single node in the next layer. Each of these neurons have an activation value which is a float between 0 and 1. These activations are described in the format:

Where L represents which layer of neurons this activation is in, and n represents which neuron in said layer this activation belongs to.

*Diagram

Description automatically generated*

##### Weights

Every Connection between nodes have a value associated with them called a weight which can be positive or negative. The magnitude of this weight represents how big an impact the first node’s activity should have on the second’s. The sign of the weight shows if a higher activation of the first node should result in a lower activation of the second.

These are described in the format:

Where L represents the layer where the edge leads, n represents which node in layer L + 1 the edge belongs to, and m represents the node in layer L the edge belongs to. This may seem to be the wrong way round, but it lines up with matrix indexing.

Diagram

Description automatically generated

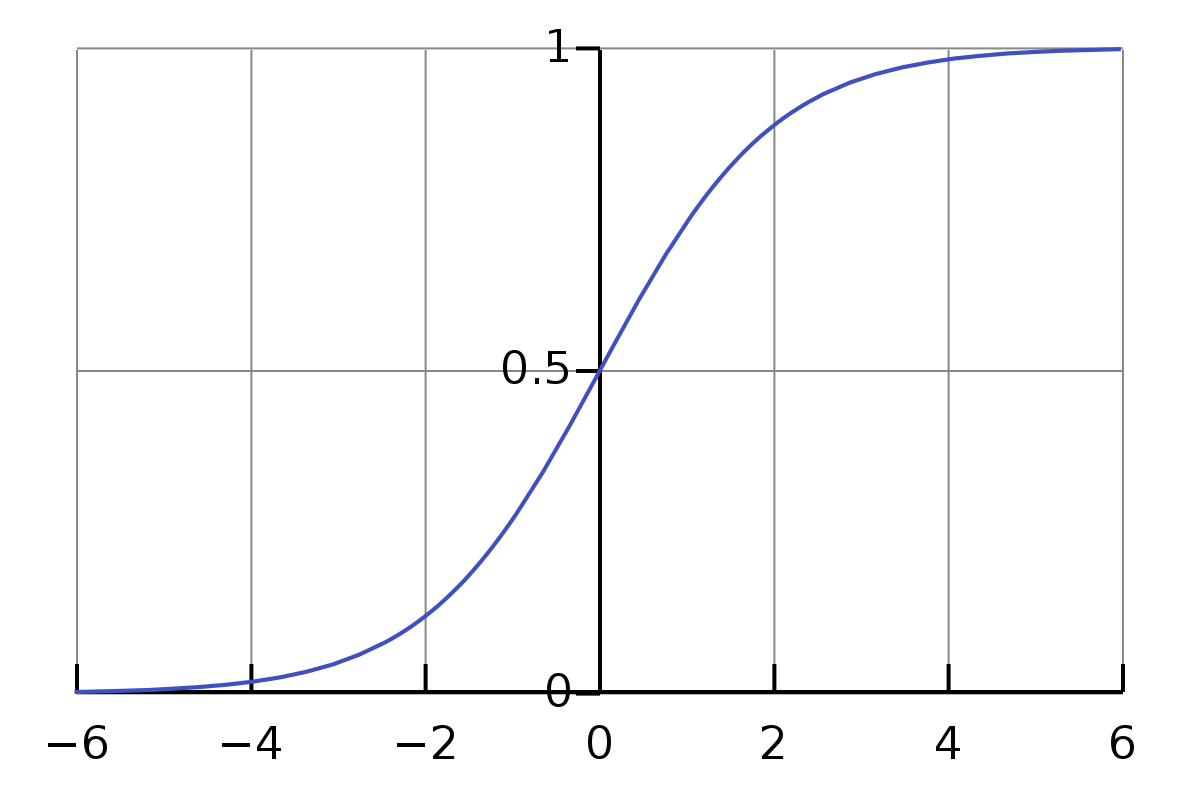
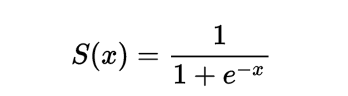
##### Calculating Activations

The activations of the nodes in the first layer (input layer) are set by the user/ sample data and are used to calculate the rest.

The activation of a node further in the network is determined by the sum of all the previous layer’s node’s activations multiplied by the weight connecting each node to the new node.

Each node also has a “bias” which can be any positive or negative float and represents that neuron’s tendency to be active or inactive. For example, a very negative bias would mean the activations and weights of the previous layer would have to be very large for the neuron have a high activity.

This sum, z, will often not be between 1 and 0 so we need a function that can squish all real numbers down to that range. One of these is the sigmoid function:



Using this we can calculate the activation of the first node in the second layer using:

##### Calculating Whole Layer Activations with Matrices

The weights connecting 2 layers in the network can be represented as a 2D matrix, W(L) which can be a slice of a 3D matrix for the whole network, W.

Here, each row in the matrix represents a node in the first layer and all its connections to the next layer with n nodes in that second layer.

The activations of a layer and that layer’s biases can also be represented as vectors with the entire network’s activations and biases being represented as 2 2D matrixes.

This is a basic formula for multiplying a matrix and a vector. You move from left to right in the matrix and down the vector, multiplying each pair in the process and adding them together before moving onto the next row in the matrix.

We can use this to calculate the activations of an entire layer of nodes

Using these we can calculate the activations of an entire 2nd layer of nodes as a vector with the below equation.

Here I’ve expanded this equation:

The sigmoid function is applied to every row in this resultant vector.

Each layer of the vector is the same as the equation used to calculate the respective activation from earlier. For example, the second item in is equal to .

Repeating this equation iteratively can be used to calculate every layer’s activations including the output layer given the input layer’s activations are set.

### Machine Learning

##### Cost Function

Main Source: (Sanderson, 2019, p. Gradient Decent)

As the weights and biases of the neural network will be randomised at first, it probably will not be able to make accurate predictions about what character is in the sample piece of data assigned to the input layer.

The cost (or error) function is a function designed to show how accurate or inaccurate the neural network is for a piece of training data. The larger the output of this cost function is, the more inaccurate the neural network is.

The simplest form of a cost function would calculate the absolute error for each output node using (aactual – adesired) and then add them all together. However, these differences could sometimes be negative, reducing the overall cost and making it appear that the network is more accurate than it actually is. Therefore, we square the differences before summing them so they will always be positive.

Chart, bubble chart

Description automatically generated with medium confidence

The average value of this cost function for many pieces of training data is a measure of how good our neural network is at its task in general with lower values meaning it is successful.

The cost function itself only uses the activations in the final layer and the expected activations as inputs. However, because the output activations are determined by the values of the weights, biases and activations of the previous layers, C can be seen as a function which takes those as arguments instead. This will be very useful later in the backpropagation section.

This function won’t have to be implemented in the design as a subroutine because the actual final value of the cost function isn’t needed. All we need to know is how the function itself is defined so we can use that in the calculus for gradient decent and backpropagation. We already know we will need to minimise each component of the sum of the cost function so we don’t need to find that final sum.

##### Gradient Decent

The goal of training our neural network is to move the starting value of the cost function, C, to a minimum by manipulating the network’s weights and biases, maximising the network’s ability to perform its task.

Chart, diagram

Description automatically generated

We start at a random position on this graph of C(w) - w generated from the random weights and biases and our goal is to reach the minimum point. We can do this by finding the gradient of the point in the graph we are at and determine if it is a decreasing or ascending function at this particular input of w.

If the graph is decreasing (negative gradient), increase the value of w slightly and if it is increasing (positive gradient), decrease the value of w slightly. Repeat this process repeatedly until we reach the minimum.

If we make these small alterations proportional to the gradient’s magnitude, the steps will get smaller as we approach the flat minimum, minimising the chance of overshooting it.

For functions with multiple arguments like our cost function, the gradient at any point gives the direction of fastest increase. This means that if we can find the gradient, we can move in the opposite direction to approach the minimum quickly.

##### Backpropagation

Main Source: (Sanderson, 2019, p. Backpropegation Calculus)

Backpropagation is the process of moving back through the network from the output layer and finding the gradient for every weight and bias needed for gradient decent.

In other words, we need to find how sensitive our cost function, C is to every weight and bias in the network and to do that we find their partial derivatives and

For demonstration, we will have j be an arbitrary index in layer L and k an arbitrary index in layer L – 1. is the weight of the edge connecting these 2 nodes.

Diagram

Description automatically generated

Because is determined by other functions further back in the network, can be calculated by using the chain rule to split up the derivative.

Each of the derivatives here can now be calculated using differentiation

The cost function now has to sum the square differences for every node in the output layer (L). However, the cost’s sensitivity to a specific node’s activation is the same as if there was only 1 node in the output layer. This is because these nodes’ activations don’t affect each other so they can be treated as entirely separate factors. This means we can drop the sum from the derivative. (Rashid, 2016, p. 94)

The derivative for a node’s activation with respect to z is just the derivative of the sigmoid function

Z for a specific node is now calculated using the weights and activations of all the nodes in the layer k and summing them all together along with the bias. However, we just want to know sensitive z is for a specific weight so all but can be ignored from that sum. – and the derivative of that expression with respect to the weight is just the activation.

Using these we can calculate the sensitivity the cost function has to a specific weight or bias in the layer L with these equations:

This also means that the weights derivative can be found by multiplying the biases derivative by

Propagating Backwards

A picture containing text, clock

Description automatically generated

The image above represents the path of partial derivatives needed to get to a weight or bias in the layer L – 1 (w3 or b3). As you can see, it overlaps with the path needed to get the derivatives for w4 or b4 as we pass from C to z4. Instead of going to w4 or b4 though we move up to a3, z3 then the weight or bias we want. This means we can use the derivative for the next layer to calculate the current one.

Diagram

Description automatically generated

Because is connected to multiple nodes in layer L, it will have a cumulative effect on the cost that the nodes in layer L did not. This means that to find the cost’s sensitivity to an activation in layer L – 1 we have to sum up the derivatives for every connection it has to layer L.

This equation inside the summation is almost the same as the one used to find the biases’ gradient in the output layer, but it has been multiplied by

Hence:

This saves us a lot of time as we should already have the derivative for calculated from earlier.

Now we have the derivative for calculated, we can calculate the derivative for a weight in this layer using the same method we did for a weight in the output layer:

(Here l is an index in layer L-2)

This means that the 2 equations below can be used to find the weight and bias gradients for all the nodes in layer L - 1

Now these 2 equations can be used for every layer in the network all the way back to the start by using the bias gradient for the layer in front of it.

##### Changing weights and biases

After we find the gradient of the cost function relative to a single weight, we can alter that weight slightly to move towards the minimum by moving in the opposite direction as the gradient.

The constant α, is the learning rate and is multiplied by our gradient to show how quickly we want to descend the curve. This can be any value but is normally quite small so the gradient decent doesn’t overshoot the minimum. (Rashid, 2016, p. 98)

Now we can iterate through every weight and bias in the network to calculate their derivatives with the cost function. As we do this, we append them to a 3D gradient matrix: ∇C. Note that this 3D matrix is organised in the same way that the 3D matrix used to store every weight in the network is, W. This means that whenis subtracted from W, each derivative in will be applied to its corresponding weight in W.

Here is a 2D slice of w for the layer L:

Then if we reuse the same equation as above but for matrices we can change the entire network’s weights in 1 step:

This whole process can then be repeated for the biases using B and

can be seen together as

We can now begin gradient decent by calculating, altering W and B, recalculating for the new values in the output layer after using new training data, altering W and B again and so on.

##### Working with multiple test samples

The neural network will use several pieces of training data to calculate the cost and then alter the gradient. Each one of them may have different desired alterations to the weights and biases, some in different directions. Therefore, before altering any of the weights and biases, we average out the gradient matrices Cx for all the pieces of training data, x.

To make our decent as accurate as possible we would use every sample in our training dataset. However, this would mean calculating would take a lot of computing time and resources. Therefore, instead we can use smaller batches of samples in our dataset to train our network at once. This is called stochastic gradient decent. Using this method means the direction we descend the graph may not be as optimum but each one of our steps will take significantly less time.

### Image Processing

##### Image Thresholding

Because most photos will have a lot of background noise due to lighting issues, we will need to use a process called image thresholding where all unnecessary noise in the image is removed, leaving solid colours which can be more easily manipulated.

These images represent a type of Image Thresholding called Otsu’s method

Image thresholding can be used to shift a greyscale image’s pixels to either pure black or white, removing any noise and allowing for us to identify the gaps between objects like letters or lines more easily. This is done by rounding all pixel values up or down to 0 or 255 depending on if they are above or below a threshold value.

A basic thresholding algorithm like this assumes a constant lighting level across the image.

Once these gaps are determined, objects of interest can be separated into difference images and have any excess empty space around them removed to maximise the resolution focused on that object.

##### Compression

Images used with neural networks are often compressed in some way so that the neural network can have a lower number of nodes, reducing processing time. For example, the training images in the MNIST dataset which I will be using are all in 28\*28 resolution.

A way of doing this is lossy compression where groups of close together pixels have their values averaged together and then are all assigned that average. After this, the image can be stored in fewer pixels at a lower resolution.

|  |  |
| --- | --- |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |

### Datasets

##### MNIST Dataset

In order to train our neural network, we will need a very large number of images of letters. Creating a large enough collection of these from scratch would take an extremely large amount of time so instead I will use an already existing dataset.

The MNIST dataset (Extended Modified National Institute of Standards and Technology) is a collection of 28\*28 greyscale pixel images of several written characters such as digits and letters. These characters were sampled from 500 different writers so networks trained with MNIST will have a better change of recognising different people’s handwriting.

There are multiple versions of this dataset with its data separated and categorised in different ways. For this project, we are only interested in letters for now, so the MNIST Letters dataset is the most relevant. If I were to add support for recognising digits later in this project, the MNIST ByClass dataset would be useful.

The MNIST Letters dataset which contains 145,600 unique items. In this set, uppercase and lowercase letters are split roughly evenly and are classed together. This means that it is split into just 26 classes. These are both also split into a set designed for training networks and a set designed for testing

The number of training samples for each Latin character is proportional to how commonly that character appears in the English language.

A picture containing graphical user interface

Description automatically generated

(Cohen, Afshar, Tapson, & Schaik., 2017)

When using the MNIST Letters dataset, in a neural network with 1 hidden layer, it was found that the accuracy increased in a relationship similar to a logarithmic graph. This means that it peaked around 85% accuracy at 10,000 nodes (their largest input) but started giving diminishing returns after around 6000 nodes. See figure x. Therefore, when implementing a neural network to be trained on this data, it should have a similar number of nodes.

Chart

Description automatically generated

Figure x (Cohen, Afshar, Tapson, & Schaik., 2017)

##### A close-up of a card Description automatically generated with medium confidencePAF

The Postcode Address File is a dataset of every address in the UK and is the same dataset that the Royal Mail uses. Access to the entire dataset is proprietary however so I will be using a free partial sample of the dataset for testing for testing in a comma separated values file. This was sourced from https://www.postcodeaddressfile.co.uk

Each row in the file corresponds to an address with the different parts of an address separated by commas.

Text

Description automatically generated

Each address can contain: Organisation Name, Department Name, PO Box, Building Name, Sub Building Name, Building Number, Thoroughfare, Street, Double Dependent Locality, Dependent Locality, Post Town, Postcode, Postcode Type, Mailsort SSC

However, several of these are left blank so an implementation would need to account for this. They are all also in uppercase meaning readings should all be stored in uppercase too.

I will be able to use this dataset after each character in the image is identified to separate the list of characters that were detected in the image back into individual words with spaces.

LITTLESHORTWOODFARM… -> LITTLE SHORTWOOD FARM…

## Potential Systems

##### Format

|  |  |
| --- | --- |
| Windows Forms App | Phone app |
| For a windows forms application, a user could open the app and select a photo using the windows file manager or by letting other programs move files into an input folder. That photo would then be converted to a format that could be interpreted by the neural network. The parsed text could then be appended to a text file or written to a label in the form. | For an android app, a touchscreen friendly interface would have to be implemented with features such as larger buttons. Files could be uploaded similarly to a windows forms app. However, a user could also use their device’s camera to manually take a photo with the app and let the neural network read that. The output could then be written above the text or outputted from the speakers using text-to-speech. |
| * Windows forms is a format I have used before and am familiar with so an implementation could be developed more quickly. * An app written in C# allows for complex matrix/ array operations to be performed efficiently. * Windows forms can easily add controls like buttons | * This allows us to combine the photo taking hardware with our software using a phone’s integrated camera to get input. Skipping a step a user would have to implement otherwise. * This app would be cheaper than running the program on a pc, reducing costs for smaller post offices |
| * As the app would be hosted on a computer, a photo taken on a camera or phone would have to be uploaded to the computer first using an external camera. | * I would have to learn how to develop a mobile phone app which I have no experience in. * While Android apps can be written in languages I know like C#, there is significantly less documentation or support than there are for languages like Java or Kotlin * An application like this may have to run continuously which could remove the benefits of using a phone because it would have to be constantly plugged in to charge, reducing flexibility. * Would be more difficult to integrate into a larger mail system |

I believe a windows forms app would be the best place to start to develop the neural network and image processing systems. If I still have time left in my project after finishing this, I may consider moving onto developing an android app which uses the same systems and base code for the neural network.

##### Programming language

|  |  |
| --- | --- |
| C# | Python |
| Advantages | |
| * As C# relies on .NET, I would have many inbuilt functions useful to machine learning and other aspects of this project * .NET has an official library dedicated to machine learning called ML.NET * C# is the language I am most confident programming in * C# can be used with Windows Forms which allows for the creation of a GUI quickly and easily for windows platforms | * Python has an extensive number of libraries dedicated to machine learning * The MNIST dataset can be easily accessed through python if you use the dataset’s Matlab format * Many pieces of existing machine learning documentation use python as a base |
| Disadvantages | |
| * C# is used less commonly for Machine Learning projects than python so there will be a reduced amount of documentation. * Using C# with the MNIST dataset requires manually parsing binary files to extract training data | * I am less familiar with python than I am with C# * Creating a GUI that a client could easily interact with is more difficult in python as Windows Forms is not supported |

I believe C# would be a better choice in this project due to my increased proficiency in it relative to python. Although, as the project develops, I may consider also using python for certain applications such as parsing through the MNIST dataset.

##### Matrices

The matrices used for the neural network (activations, weights, derivatives etc.) can be represented programmatically as an array but these arrays could be implemented in different ways in C#.

|  |  |
| --- | --- |
| Multidimensional arrays – array[,] | Jagged arrays – array[][] |
| Multidimensional arrays use at least 2 indexes to find objects in a rectangular structure where each row is the same length. | Jagged arrays are single dimensional arrays which store references to other arrays in each of their cells (which can be other jagged arrays). This means that they can function similarly to normal multidimensional arrays. |
| Advantages | |
| * The main advantage of multidimensional arrays are that they are slightly more optimised for quick access because they are all stored in one memory location unlike jagged arrays where they are made up of references to other locations in memory. This is useful because there are going to be thousands of values inside the network that all need accessing repeatedly. | * The main advantage to these is that the lengths of each nested array can be different which is very useful for structures with various lengths such as the number of nodes in each layer. * They also can easily have a single row of values split off into a different array or all changed at once because each row is a unique structure itself. This is very useful for neural networks because of how commonly the rows of values will be copied for calculating activations or written to at once when training the network. |
| Disadvantages | |
| * The rectangular structure will result in many rows containing excess cells because different layers in the structure will have different numbers of nodes. This would be memory inefficient for larger neural networks and could make matrix operations more complicated. * Splitting off a row from a multidimensional array also requires iterating through the array instead of just copying a reference for a jagged array. | * Accessing single values for jagged arrays initially can be slightly less efficient than multidimensional arrays because a sequence of references has to be followed instead of all the data being stored in one location |

I think that jagged arrays will be the best solution due to the simplicity of splitting them up to smaller arrays to be used in calculations. E.g. splitting the 3D weights matrix into a single 2D slice for the matrix multiplication used to calculate activations.

could be represented as *activations[L][n]*

could be represented as *weights[L][n][m]*

## Objectives

1. Be able to create a base neural network with randomised jagged arrays
   1. A neural network should contain 2D arrays for the activations and biases of every node in the array. Their indexes point to which layer the node is in and what position in that layer it is.
   2. A neural network should contain a 3D array for the weights with an index showing which layer an edge starts in, and 2 indexes for which node in that layer and the next layer that edge connects.
   3. The values of every cell in these arrays should be given random values between 1 and 0.
2. Allow for a neural network’s attributes to be saved and loaded using json serialisation.
3. Allow the network to attempt to recognise a letter by propagating activations forwards
   1. Be able to split the weight, bias and activation arrays into arrays for just 1 layer (the first hidden layer)
   2. Matrix multiply the weight array and the activations array. Then vector add the biases array
   3. Squish back down to 0-1 using a sigmoid function
   4. Propagate this forward for every layer all the way to the output layer
   5. return the largest activation the output layer
4. Be able to read a piece of training data from the binary storage files and run it through the neural network
   1. The binary sequences can be read and converted to a 2D float array
   2. This array’s values can then be squished down to 0-1 from 0-255
5. Find how accurate the neural network currently is by checking if the largest activation in the output layer corresponds to the actual letter in the image. Average this over multiple tests and divide by the number of tests to find the accuracy. Output this to the GUI regularly as the network is being trained.
6. Find how sensitive the cost function is to the output layer’s weights and biases
   1. Calculate the partial derivatives for the output layer’s weights and biases using calculus
   2. Store all the derivatives in arrays lining up with the positions of the relevant weights and biases in the other arrays.
7. Find how sensitive the cost function is to previous layers (backpropagation)
   1. Use the chain rule to use the layer in front of the current one’s derivatives to calculate the current one’s
   2. Store each of these in the same array as the output layer’s derivatives
   3. Propagate backwards to find all the layers’ derivatives
8. Alter the weights and biases to minimise the cost function repeatedly for different sets of test data (gradient decent)
   1. Average the derivative arrays from the previous objective over a variable batch size number of sample images
   2. Multiply the derivative arrays by the learning rate and then subtract the weight derivative array from the weight array and the bias derivative array from the bias array.
   3. Repeat until interrupted by a user input
9. Take a photo of handwritten text in as an input using the form or an input folder. Have the program detect changes in the input folder so new files to read can be inputted.
10. Use image thresholding to create a copy of this bitmap in black and white
11. Split the Image into individual letters with inverted shades
    1. Split the image into multiple lines by finding horizontal lines of black pixels where the gaps between lines are. Split the image, removing these lines so there are no black spaces at the bottom or top. Then add each of these images to a queue.
    2. For each image in the queue, repeat the process but for vertical lines so the individual letters can be extracted and added to a queue.
12. Compress these bitmaps into 28\*28-pixel bitmaps using lossy compression
13. Use the neural network to recognise the characters in each of these bitmaps in the queue and append them to an output list.
    1. Squish the greyscale values between 0 and 255 down to 0 – 1 and store a single letter’s image in a 1D float array
    2. Invert the colour values in each cell
    3. Pass each one of these arrays for each letter through the neural network and enqueue the recognised letter to an output queue.
14. Split the queue of letters back into the individual words
    1. Take the first letter in the list and query through the addresses dataset to find addresses which start with that letter, repeat this for every letter.
    2. Recursively split the program to test for adding spaces and letters that have been read until a full address with spaces can be found
    3. If a valid address is found, output to a text file and the GUI. If not, put the original image back through the system to try again. If it still doesn’t work, show an error message.

## References

Cohen, G., Afshar, S., Tapson, J., & Schaik., v. (2017). MNIST: an extension of MNIST to handwritten letters. Retrieved from http://arxiv.org/abs/1702.05373

Rashid, T. (2016). *Make Your Own Neural Network.* CreateSpace.

Royal Mail. (2019, Aug 27). User Guide for Machine Readable Letters & Large Letters. Retrieved from https://www.royalmailtechnical.com/rmt\_docs/User\_Guides\_2019/Machine\_Readable\_Letters\_and\_Large\_Letters\_20190827.pdf

Sanderson, G. (2019, Oct 5). Retrieved from 3blue1brown: https://www.3blue1brown.com/lessons/neural-networks

# Design

## Description

For my solution, I intend to create 2 applications: one that can be used to train a neural network for optical character recognition and one which will be using said neural network to read addresses after processing images of addresses. This is because the post system itself will not need to interact with the neural network’s training after it has been trained so having them all work in the same program would be memory inefficient.

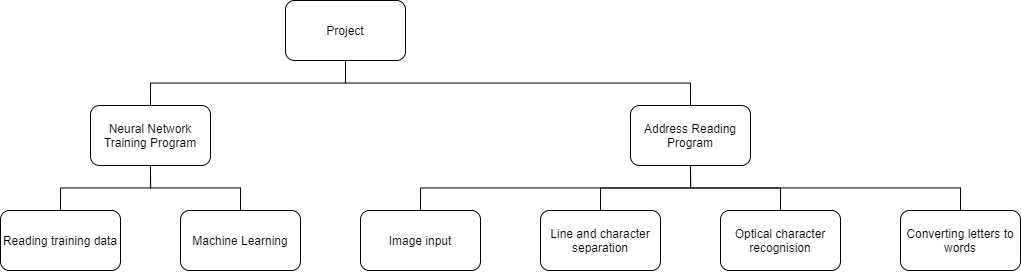
I intend to use C# to program these solutions inside Windows Forms Applications. This allows for the programs to be able to display images, making debugging easier and improving the user experience. However, because other systems like routing software and will need to work with the address reading automatically, I will also implement a way to use the program without human input. This can be done by reading and writing to IO files and having the program detect changes to them.

I will be implementing this in an object-oriented approach with different classes for a neural network, And for an image processor. This will allow for the neural network to be more easily converted to a storable format as it will be made up of multiple data structures. It will also allow me to reference the class definition of the network in a different program using a library, so no code has to be copied. It will also make organising my code simpler and clearer to read.

The workflow for this project will be to generate a base neural network in the training program and then train it with samples from the MNIST dataset until it reaches a suitable accuracy to be used for OCR. After that it can be exported and used within the address reading program. This program will take a photo of an address in as an input, split that image into photos of every character in that photo and recognise each of them with the neural network. The different rows of letters that have been read can now be split back into words and matched to an address.

Images will be inputted by either selecting one inside the windows form or by moving the file to a selected input folder.

## Decomposition Diagrams



### Machine Learning

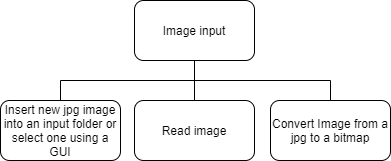
Diagram

Description automatically generated

Diagram

Description automatically generated

### Address Reading

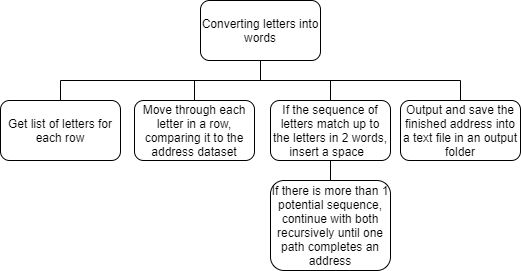


Diagram

Description automatically generated

Diagram

Description automatically generated



## Interface Design

### Machine Learning

Initial Rough sketch:

Diagram, engineering drawing

Description automatically generated

Diagram, schematic

Description automatically generated

### Address Reading

Initial Rough sketch:Diagram

Description automatically generated

Diagram

Description automatically generated

## Data Dictionary

### NeuralNetwork Class

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Data type | Validation |
| weights | A 3D jagged array storing the values for the weights of each edge in the neural network which connect each node to every node in the next layer. | Float[][][] | N/A |
| weightedSums | A 2D jagged array storing the values in activations before the sigmoid function has been applied to them | Float[][] | N/A |
| activations | A 2D jagged array storing the activations of each layer in the network | Float[][] | N/A |
| biases | A 2D jagged array storing the biases of each layer in the network | Float[][] | N/A |

### TrainingDataReader Class

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Data type | Validation |
| EOF | Stores if the reader is at the end of the dataset | bool | If this is true then the program implementing this class should reset the reader to the start. |
| imageCount | Stores how many images the current instance of the class has read from the dataset | Int | N/A |
| imageHeight | Stores the hight of a training sample image | Int | N/A |
| imageReader | Used to read the pixel data of a training sample from the MNIST dataset. Each pixel is represented by a byte in the file. | BinaryReader | Make sure this is disposed of after use |
| imageWidth | Stores the width of a training sample image | Int | N/A |
| labelReader | Reads which letter the data imageReader has read represents as that is stored in a different file in the MNIST dataset. | BinaryReader | Make sure this is disposed of after use |
| numberOfImages | Stores how many images there are in the dataset | Int | N/A |

### NNTrainer

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Data type | Validation |
| Accuracy | A value between 0 and 1 representing how accurate the neural network is | Float | N/A |
| batchSize | How many samples will be used at once to find how far to change the neural network’s values | Int | N/A |
| bDerivatives | Stores the derivative of the cost function with respect to every bias in the network. This is averaged over for a batch of samples | Float[][] | N/A |
| bDerivativesSum | Stores the sum of the cost function with resect to every bias in the network for a batch of samples. This is used to find wDerivatives | Float[][] | N/A |
| expectedOutputActivations | An array representing which values the last layer of nodes should have if a letter has been correctly identified | Float[] | N/A |
| learningRate | What multiplier will be applied to the changed to the neural network’s values | float | N/A |
| nn | This is the neural network being trained | NeuralNetwork | N/A |
| numberOfCorrectPredictions | How many times has the neural network correctly identified a letter in a training sample | Int | N/A |
| numberOfPredictions | How many samples has the neural network seen | Int | N/A |
| sample | A single pair of the pixel data of an MNIST letter and the label of what letter this represents | TrainingSample | N/A |
| tdReader | This is a reader to get the training data that will be used to train the neural network | TrainingDataReader | N/A |
| wDerivatives | Stores the derivative of the cost function with respect to every weight in the network. This is averaged over for a batch of samples | Float[][][] | N/A |
| wDerivativesSum | Stores the sum of the cost function with respect to every weight in the network for a batch of samples. This is used to find wDerivatives | Float[][][] | N/A |

### Machine Learning

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Data type | Validation |
| accuracy | Stores how accurate the neural network is at recognising characters as a value between 0 and 1 | Float | N/A |
| continueTraining | Nn will continue being trained after this is set to true and until it is set to false | Bool | N/A |
| imagesSeen | How many training samples have been used while training the nn | int | N/A |
| nn | The neural network currently loaded and being trained | NeuralNetwork | Make sure the network being trained has 784 nodes in its input layer and 26 in its output layer so it can be trained with MNIST |
| stopwatch | Used to count how long nn has been training for | StopWatch | N/A |
| trainer | The trainer used to train nn | NNTrainer | N/A |

### Address Reader

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Data type | Validation |
| nn | The neural network used to recognise characters | NeuralNetwork | N/A |
| addresses | Stores a list of valid addresses from the PAF | List | N/A |
| fileWatcher | Raises an event when a file us moved to a folder it is watching | FileSystemWatcher | Make sure the files it detects are images  Make sure this is disposed when a user turns off automatic reading and output |
| inputFolderPath | Stores the file path of the folder fileWatcher will watch | String | N/A |
| outputFilePath | Stores the file path of the txt file used to output addresses | String | Make sure this path leads to a txt file. |
| Image | The original input image before processing | Image | Make sure this is actually an image when being inputted |
| processedImages | Stores a queue of each processed bitmap of every letter in image | Queue | N/A |

## Pseudocode

### Objective 1: Creating a base neural network

This pseudocode represents one of the constructors for the Neural Network class. This class has one other constructor used for objective 2. It takes in an int array parameter which stores the number of nodes in each layer of the neural network the user wishes to generate. The three attributes needed to be generated are the activations array which can be left with empty values as well as the biases and weights arrays. Each of these will need to be assigned random values to each cell (weights: 0 - 1, biases: -1 - 1)

We save the weighted sums of each layer before the sigmoid function is used on them because those sums will be needed when training the network so it is more efficient to save them here than it is to recalculate them each time.

Constructor(layerLengths)

activations <- Setup2DArray(layerLengths)

weightedSums <- Setup2DArray(layerLengths)

biases <- Randomise2DArray(Setup2DArray(layerLengths), -1, 1)

weights <- SetupWeights(layerLengths)

End Sub

This pseudocode is used when generating activations and biases by making their first index store the same number of nested arrays as there are layers (length of layerLengths). Then for each one of these nested arrays, they are instantiated to have as many cells as the number called for in the corresponding index in layerLengths.

Float[][] function Setup2DArray(layerLengths)

temp <- float[len(layerLengths)][]

For i <- 0 to len(layerLengths) – 1

temp[i] <- float[layerLengths[i]]

Next

Return temp

End function

This pseudocode randomises the values inside a 2D jagged array by assigning each cell a random value between an upper and lower bound.

Float[][] function Randomise2DArray(array, upper, lower)

For i <- 0 to len(array) - 1

For j <- 0 to len(array[i]) - 1

Array[i,j] <- random float between upper and lower

Next

Next

Return array

End function

This pseudocode uses the same technique as Setup2DArray to give each node an index but also adds an extra dimension by representing each cell as another nested array with length equal to the number of nodes in the next layer as each node will have an edge with every node in the next layer. Each of these values is then randomised between 0 and 1.

Float[][][] function SetupWeights(layerLengths)

temp <- float[len(layerLengths)][][]

For i <- 0 to len(layerLengths) – 1

temp[i] <- float[len(layerLengths[i])][]

For j <- 0 to len(layerLengths[i]) – 1

temp[i][j] <- float[len(layerLengths[i+1])]

For k <- 0 to len(layerLengths[i+1]) - 1

temp[i][j][k] <- random float between 0 and 1

Next

Next

Next

Return temp

End function

### Objective 2: Neural Network Serialisation

A JSON serialiser and deserialiser can be used to convert the jagged arrays inside a neural network to a JSON file and then back into those arrays. Once we get those arrays, a constructor can be used to instantiate a neural network with the same values saved in the JSON file. The attributes we want to be serialised in the class will have to be marked in an implementation. We will save the activations, weights, biases and weighted sums.

These Static methods will be part of the NerualNetwork class

Static Sub SaveNeuralNetwork(filePath, nn)

String output <- Serialise nn

write output to file at filePath as a JSON file

End sub

If the file at filePath doesn’t exist, an exception will be thrown. There will also be one if the data stored in that file doesn’t contain the objects we need to construct a NeuralNetwork

Static Function LoadNeuralNetwork(filePath)

Try

Temp <- read from file at filePath

Return deserialise temp

Catch

Output “Error”

Return null

End Try

End Function

This is the constructor a JSON deserialiser will use when generating a Neural Network object from a JSON file.

Constructor(pActivations, pWeightedSums, pBiases, pWeights)

activations <- pActivations

weightedSums <- pWeightedSums

biases <- pBiases

weights <- pWeights

End Sub

### Objective 3: Calculating Activations

This is the public function that will be called when trying to recognise an image with the neural network. It takes the pixel values of an image squished down to 0-1 as a single 1D array and assigns that to the input layer of activations. Now we can propagate those activations forward and then find the most active node in the output layer. The index of this node corresponds to the index of the letter in the alphabet that the network believes is in the image.

Public char funcion RecogniseImage(normalisedPixels)

Acivations[0] <- normalisedPixels

PropegateAcivations()

Return Chr(GetLargestActivation(activations[len(activations)] - 1) + 65)

End Function

This Sub is used to implement the forward propagation of activations using matrix multiplication as seen in this equation from the analysis: This is repeated iteratively for each layer to calculate the output layer’s activations eventually. We start at layer 1 because layer 0’s activations have already been set.

Private Sub PropegateActivations()

For i <- 1 to len(activations) – 1

Temp <- MatrixMult(weights[I – 1], activations[I – 1])

Temp <- VectorAdd(temp, biases[i]

for j <- 0 to len(temp) – 1

temp[i] = Sigmoid(temp[i])

next

activations[i] <- temp

next

End sub

These pseudocode for the mathematical functions used in PropegateActivations() which implement matrix multiplication, vector addition and the sigmoid function.

Private float[] function MatrixMult(matrix, vector)

result <- float[len(vector)]

for i <- 0 to len(vector) – 1

for j <- 0 to len(matrix)

temp <- matrix[j][i] \* vector[j]

result[j] <- result[j] + temp

next

next

return result

end function

private float[] function VectorAdd(vec1, vec2)

for i <- 0 to len(vec1) – 1

result[i] <- vec1[i] + vec2[i]

next

return result

end function

private float function Sigmoid(x : float)

return 1 / (1 + 2.718 ^ -x)

end function

### Objective 4: Reading MNIST

The MNIST dataset is stored in 2 file formats: Matlab and binary. The Matlab format can be accessed easily from python but not as easily from C# so we will use the binary format.

Graphical user interface, text, application

Description automatically generated

The datasets are split into an image file and a label file. The image file contains all the actual images as sequences of bytes representing 1 pixel each and the label file stores what character the corresponding images in the image file are.

**MNIST-letters-test-images-idx3-ubyte**

|  |  |  |
| --- | --- | --- |
| Index (byte) | Datatype | Description |
| 0 | Int 32 | Magic number (unique number identifying each of these files) |
| 4 | Int 32 | Number of images |
| 8 | Int 32 | Number of rows |
| 12 | Int 32 | Number of columns |
| 16 | ubyte | Greyscale pixel (0-255) |
| 17 | ubyte | Greyscale pixel (0-255) |
| … | … | … |
| n | ubyte | Greyscale pixel (0-255) |

**MNIST-letters-test-labels-idx1-ubyte**

|  |  |  |
| --- | --- | --- |
| Index (byte) | Datatype | Description |
| 0 | Int 32 | Magic number (unique number identifying each of these files) |
| 4 | Int 32 | Number of images |
| 8 | ubyte | Label (1-26) |
| 9 | ubyte | Label (1-26) |
| … | … | … |
| n | ubyte | Label (1-26) |

The labels value corresponds to the letter’s position in the alphabet. This means that a label of 1 could be either ‘a’ or ‘A’. These 2 components could be stored together in a struct:

Structure TrainingSample

Data : float[]

Letter : char

End Structure

Reading from the binary files can be handled with its own class: TrainingDataReader

This is the constructor for a TrainingDataReader Object. It sets up the 2 binary readers needed to get both the image pixel values and the labels for the corresponding images. Using the metadata in the image files we can find the number of images as well as their dimensions (28\*28 in MNIST). We also skip the header for the label file as that contains no additional information. This means that both reader objects will be pointing at the beginning of the payload after the constructor is called. We also start a count of which image we are currently at so that we know when we have run out of images.

Constructor (imagePath, labelPath)

imageReader <- Open binary file at imagePath

move imageReader to byte index 4

numberOfImages <- read 32 bit with imageReader as int

imageHeight <- read 32 bit with imageReader as int

imageWidth <-read 32 bit with imageReader as int

labelReader <- Open binary file at labelPath

move labelReader to byte index 8

currentImage <- 0

End Sub

This is a function that can be called to get the next training sample. It returns a TrainingSample object containing an array of every pixel value (0-255) as well as a char label saying what character the image represents. If there are not any images left, throw an exception as this should be error checked when using this class.

Public TrainingSample function GetNextTrainingSample()

If currentImage = numberofImages then

Throw out of bounds exception

End if

pixels : float[imageHeight \* imageWidth]

For i <- 0 to imageHeight \* imageWidth – 1

pixels[i] <- read ubyte from imageReader as int

next

label <- read ubyte from labelReader as char

return new TrainingSample(pixels, label)

End Function

A problem we might run into is that some computers store binary where the most significant bit is in the smallest memory address (big endian) and others store the most significant bit in the largest memory address (little endian). Languages like C# read binary files assuming big endian is being used. This means that if running this software on a little endian computer, the binary must be reversed before it can be parsed.

Private int function readInt(binaryReader, bitCount)

bytes <- Read bitCount / 8 bytes

if little endian

for i < 0 to len(bytes) / 2 - 1

temp <- bytes[i]

bytes[i] <- bytes[len(bytes) – 1 – i]

bytes[len(bytes) – 1 – i] <- temp

next

End if

return convert bytes to int

End Function

### Objective 5: Accuracy

After running through the neural network and finding the recognised digit, we can compare it to the actual digit. If they are the same, increment a counter of the number of accurate predictions. Then increment the total number of images the neural network has seen. Dividing these will give us the accuracy of the neural network as a decimal. This will be part of the NNTrainer class.

Private float function UpdateAccuracy(correctChar, recognisedChar)

If correctChar = recognisedChar

numberOfCorrectPredictions <- numberOfCorrectPredictions + 1

End if

numberOfPredictions <- numberOfPredictions + 1

return numberOfCorrectPredictions / numberOfPredictions

End function

### Objective 6: Finding Output Layer Gradients

These equations from the backpropagation section of the analysis need to be implemented to calculate how sensitive the cost function is to the weights and biases in the output layer

This function calculates the derivatives for the biases in the output layer by looping through each node in the layer and calculating the equation for , using a temp variable to apply each step.

Float[] function GetOutputLBiasesGradient(expectedOutputactivations)

L <- len(nn.activations) - 1

For j <- 0 to len(nn.activations[L]) – 1

mTemp <- SigmoidPrime(nn.weightedSums[L]

mTemp <- mTemp \* 2 \* (nn.activations[L][j] – expectedActivations[j])

derivatives[j] <- mTemp

Next

Return derivatives

End function

This is an implementation of the sigmoid function’s derivative function

Float function SigmoidPrime(x:float)

Temp <- e ^ x

Temp <- temp / ((1 + e) ^ 2)

Return temp

End function

This function calculates the derivatives for the weights connecting to the output layer from the previous layer by looping through each connection between the 2 layers and using the equation for . As this involves 2 layers unlike the bias equivalent. We will need the length of the previous layer too, so we take both their lengths in as parameters.

Float[][] function GetOutputLWeightsGradient(expectedOutputActivations)

L <- len(nn.activations) - 1

For k <- 0 to len(nn.activations[L - 1]) – 1

For j <- 0 to len(nn.activations[L]) – 1

mTemp <- nn.activations[L-1][k]

mTemp <- mTemp \* SigmoidPrime(nn.weightedSums[L]

mTemp <- mTemp \* 2 \* (nn.activations[L][j] – expectedActivations[j])

derivatives[k][j] <- mTemp

next

next

return derivatives

End function

### Objective 7: Backpropagation

Now we have the output layer’s derivatives, we can move backwards through the network to find the derivatives for all the hidden layers.

This sub is used to find the derivatives for every weight and every bias using the expected activations in the output layer as an input. It starts by finding the output layer’s derivatives and then loops through all the hidden layers backwards to find their derivatives. When calculating the hidden layer gradients, we find the bias ones first because we can reuse those values to find the weight’s ones more efficiently. We stop after i = 1 because the input layer’s derivatives don’t matter.

Private Sub GetGradients(expectedOutActivs, byref wDerivatives, byref bDerivatives)

L <- len(bDerivatives) - 1

wDerivatives[L] <- GetOutputLWeightsGradient(expectedOutActivs)

bDerivatives[L] <- GetOutputLBiasGradient(expectedOutActivs)

for i <- L – 1 to 1

bDerivatives[i] <- GetHiddenLayerBGradients(len(bDerivatives[i]), nn.WeightedSums[i]

nn.activations[i – 1], bDerivatives[i + 1])

wDerivatives[i] <- GetHiddenLayerWGradients(len(nn.activations[i]), bDerivatives[i],

nn.activations[i-1])

next

end sub

These equations from the backpropagation section of the design can be used to calculate the derivatives for the weights and biases in a hidden layer. Though as you can see, the weights derivative can be found by multiplying the biases one by

This function iterates through each node in a hidden layer and uses the equation to calculate the bias gradients for that node’s bias. It also uses nested iteration to find the summation in that equation.

Private float[] function GetHiddenLayerBGradients(layerLength, nextWeightedSums, nextWeights,

nextBiasGradients)

for i <- 0 to layerlength – 1

gradients[i] <- SigmoidPrime(nextWeightedSums[i])

for j <- 0 to len(nextBiasGradients) – 1

tempSum <- tempSum + (nextWeights[i][j] \* nextBiasGradients[j])

next

gradients[i] <- gradients[i] \* tempSum

next

return gradients

end function

This function iterates through each weight connecting to a hidden layer and uses the modified equation to calculate the derivatives by multiplying the bias derivative by . As there is a weight using this derivative for every node in the hidden layer, we iterate through that many times. Then we move onto the next bias derivative for the current layer until they are all used and every weight derivative for that layer has been found.

Private float[][] function GetHiddenLayerWGradients(layerLength, biasGradients, prevActivations)

For l <- 0 to len(gradients) – 1

For k <- 0 to len(gradients[l]) – 1

Gradients[l][k] <- biasGradients[k] \* prevActivations[l]

Next

Next

Return gradients

End function

### Objective 8: Gradient Decent

To approach the most accurate neural network possible we need to descend to the minimum cost by altering the weights and biases in the opposite direction to the gradients we found in the backpropagation section. The amount we change it is determined by the learning rate which will be taken in as a user input from the form

To implement stochastic gradient decent, we will average the gradients found over a batch of sample images of a length inputted by the user. Then we multiply the resultant array by the learning rate and subtract that from the array storing the biases or weights in the neural network.

This sub will be called when the user pushes a start training button and will loop continuously while continueTraining is true. This will be the case forever until the user pushes a stop training button which calls StopTraining(). This then sets that flag to be false.

Private Sub StartTraining

continueTraining <- true

batchSize <- input

learningRate <- input

while continueTraining

for i = 0 to batchSize

sample <- dataReader.GetNextTrainingSample()

expectedOutActives <- array full of 0s

expectedOutActives[sample.label – 65] <- 1

GetGradients(expectedOutActivs, byref wDerivatives, byref bDerivatives)

wDerivativesSum <- wDerivativesSum + wDerivatives

bDerivativesSum <- bDerivativesSum + bDerivatives

next

wDerivatives <- Scalar3DMatirxDiv(batchSize, wDerivativesSum)

wDerivatives <- Scalar3DMatrixMult(learningRate, wDerivativesSum)

nn.weights <- MatrixSubtract3D(nn.weights, wDerivatives)

bDerivatives <- Scalar2DMatirxDiv(batchSize, bDerivativesSum)

wDerivatives <- Scalar2DMatrixMult(learningRate, wDerivativesSum)

nn.biases <- MatrixSubtract2D(nn.biases, bDerivatives)

loop

The matrix maths functions used in the pseudocode above will work by iterating through every cell in the input array(s) and use the relevant operation on the 2 arrays, or the 1 array and the float parameter.

Private Sub StopTraining

continueTraining <- false

end sub

After Test 10, I have changed the design of this so that the trainer class itself will implement TrainForSingleBatch(). This works similarly to how StartTraining does here but without the outer loop repeating while continueTraining is true. Repeating this being called can be handled asynchronously from the Machine Learning program externally so that it can be ran without freezing the UI.

### Objective 9: Address Input

Photos of addresses can be inputted in 2 ways: either manually using the windows forms or setting the form to detect if a file has been saved to an input folder and reading from that. ReadLettersInImage() and DetectAddress() are designed in section 13 and 14 respectively.

Event btn\_SelectFile is clicked

Open Windows File dialogue

ReadImage(Input)

End sub

Clicking on this button once will start auto detection and clicking it again will disable it.

Event btn\_AutoDetect is clicked

AutoDetect <- not autoDetect

While autoDetect

If a file has been saved

ReadImage(Input)

End if

loop

End sub

Sub ReadImage(input)

Try

Image <- Input

readLetters <- ReadLettersInImage(image)

Output DetectAddress(readLetters)

Catch

Output “Error, selected file was not a valid image”

End try

End sub

### Objective 10: Image Thresholding

To threshold an image to prepare it for further processing and recognition in the neural network, we iterate through each pixel in the image. If its brightness is above a threshold, we round its value up to 1 (white). Otherwise, we round it down to 0 (black). This has to be performed before we can determine where the gaps between letters are so any background noise is removed.

Private float[,] function ThresholdImage(image, threshold)

For i <- 0 to width(image)

For j <- 0 to len(image)

If image[i,j].brightness > threshold

image[i,j] <- 1

Else

image[i,j] <- 0

End if

Next

Next

Return image

End function

### Objective 11: Split address into individual letters

To isolate all the letters in a pure black and white image, we start by splitting the image into a queue of the rows of letters across rows of white pixles. Each of these can then be split into columns containing single letters all in a new queue.

A picture containing text

Description automatically generatedText, whiteboard

Description automatically generated Application

Description automatically generated with low confidence

Rows <- SplitIntoRows(input image)

Letters <- SplitIntoLetters(rows)

SplitIntoRows() is responsible for taking an input image of black text on a white background and splitting it into a queue storing images of each row of letters in it. This works by finding straight horizontal lines of white pixels which indicates a gap and splitting the image across those lines.

The algorithm starts at the top of an image and iterates through each row, incrementing a lower bound counter until it finds a black pixel indicating a letter. We now know the lower bound y value of the region containing a row of letters (indexing starts at the top). Now if we aren’t at the end of the image, the upper bound of this region will be further down so we assign the lower bound to the upper bound counter. Then we repeat the process but instead now we’re iterating until there aren’t any black pixels as that indicates the end of a row of letters. By the end of this we know the start and end y values of a row so we can use SplitRow() to split off this section of the image and enqueue it to rows. Now we repeat the whole process but now the lower bound counter starts at the index after the current upper bound. This repeats until we read the end of the image

Private queue function SplitIntoRows(image)

yLowerBount <- 0

while yLowerBound < height(image)

while yLowerBound < height(image) and not DoesRowContainBlackPixel(image, yLowerBound)

yLowerBound <- yLowerBound + 1

loop

if (yLowerBound < height(image)

yUpperBound <- yLowerBound

while yLowerBound < height(image) and

DoesRowContainBlackPixel(image, yUpperBound)

yUpperBound <- yUpperBound + 1

loop

Enqueue SplitRow(image, yLowerBound, yUpperBound) to rows

yLowerBound <- yUpperBound + 1

End if

Loop

Return rows

End Function

DoesRowContainBlackPixel() iterates through each index in a bitmap’s row at the y value given as a parameter. If any of those pixels are black (0), return true. If the loop ends and every pixel has been checked, none of them could have been black so return false

Private Boolean function DoesRowContainBlackPixel(bitmap, y)

For i <- 0 to width(bitmap)

If bitmap[i,y] = 0

Return true

End if

Next

Return false

End function

SplitRow() is a function that takes a bitmap image as well as an upper and lower bound for a y value and copies all the pixel values of the rows of pixels between those bounds into a new bitmap and then returns that bitmap

Private Bitmap SplitRow(image, yLowerBound, yUpperBound)

For y <- yLowerBound to yUpperBound

For x <- 0 to width(image) – 1

Row[x, y – yLowerBound] <- image[x,y]

Next

Next

Return row

End function

Split into letters takes the queue of rows found in the previous pseudocode and splits each into a queue of letters using SplitIntoColumns(). After getting the queue of letters from that, the images all need to be cropped because each image will have the same height as the max height of a letter in that row. Therefore, all the others would have large white gaps at the top and bottom. VerticalCrop() is used to resolve this.

After Test 18, the pseudocode was altered to add “Letter <- ConvertToSquare(letter)” and the ConvertToSquare function itself This is because just cropping around the gaps between letters resulted in certain letters being too wide or too narrow to effectively resize to a 28\*28px image for the neural network. This was especially prevalent for letters like ‘l’. Therefore we convert the image to a square by adding whitespace to the image in ConvertToSquare()

"

Private Queue SplitIntoLetters(rows)

While rows is not empty

Columns <- SplitIntoColumns( dequeue from rows)

While columns is not empty

Letter <- dequeue from columns

Letter <- VerticalCrop(letter)

Letter <- ConvertToSquare(letter)

Enqueue letter to letters

Loop

Loop

Return letters

End function

VertivalCrop() is a function that takes a bitmap (letter) which may have a white border at the top and bottom and crops those borders. It does this first by iterating through the rows of the image from the top and incrementing a top counter if the row doesn’t contain a black pixel (part of a letter). This is then done again but from the bottom. Then we have 2 variables showing the offsets from the top and bottom of the image where the actual letter starts. Now we can make a new bitmap with the same width as the original but with the height of the original after both offsets have been subtracted from it. Finally, each pixel in the new croppedImage bitmap can be iterated through with the pixels in the original image assigned to it. When reading from the original image, we add the top offset to the y value so we can skip over all the white space.

Private Bitmap function VerticalCrop(letter)

Top <- 0

Bottom <- 0

Do

edgeFound <- DoesRowContainBlackPixel(letter, top)

if not edgeFound

top <- top + 1

end if

While not edgeFound

Do

edgeFound <- DoesRowContainBlackPixel(letter, height(letter) – bottom - 1)

if not edgeFound

bottom <- bottom + 1

end if

While not edgeFound

For i <- 0 to height(letter) – top – bottom

For j <- 0 to width(letter)

croppedImage[j, i] <- letter[j, i + top]

next

next

return croppedImage

End function

ConvertToSquare() takes a bitmap image in as a parameter and converts it to a square bitmap. If the image is wider than it is tall, we extend the height of the image and if its taller then we extend the width. When widening the image, we find the offset number of white pixels needed to be added to both sides by halving the difference between the width and height. Then we iterate through a row of pixels in the squareImage bitmap and first assign offset number of white pixels. Then the original image’s pixels for that row can be added. Finally, another set of offset pixels can be added. This can then be repeated for the rest of the image’s rows. Making a bitmap longer can be implemented in the same way but with the process repeated for each column.

Private Bitmap Function ConvertToSquare(image)

If height(image) > width(image)

squareImage <- new Bitmap[height(image), height(image)]

offset <- (height(image) – width(image)) / 2

for y <- 0 to height(image)

for x <- 0 to offset - 1

squareImage[x, y] <- 1

next

for x <- offset to width(image) – 1

squareImage[x, y] <- image[x – offset, y]

next

for x <- width(image) + offset to width(squareImage)

squareImage[x, y] <- 1

next

next

else if width(image) > height(image)

squareImage <- new Bitmap[width(image), width(image)]

offset <- (width(image) – height(image) / 2

for x <- 0 to width(image)

for y <- 0 to offset - 1

squareImage[x, y] <- 1

next

for y <- offset to height(image) – 1

squareImage[x, y] <- image[x – offset, y]

next

for y <- height(image) + offset to height(squareImage)

squareImage[x, y] <- 1

next

next

end if

return squareImage

end function

SplitIntoColumns(), SplitColumn() and DoesColumnContainBlackPixel() can be implemented in very similar ways to their row equivalents.

### Objective 12: Compression

Our neural network needs an image compressed down to 28\*28px to be able to assign it to the input layer. Therefore, we call CompressBitmap(28, 28, input image).

CompressBitmap works by finding the width of a section of pixels to be compressed down to 1 pixel (xStep) and the height (yStep). These can be found by dividing the original image’s height/ width by the new height and width from the parameters.

Next we can iterate through each pixel in the new compressed image and assign to it the average brightness of the equivalent region of pixels in the original image using GetSumBrightnessForArea() and averaging it.

Private Bitmap function CompressBitmap(width, height, image)

compressedImage <- new Bitmap[height,width]

xStep <- width(image) / width

yStep <- height(image) / height

for y <- 0 to height

for x < 0 to width

sumBrightness <- GetSumBrightnessForArea(image,

x \* xStep, y \* yStep, x \* xStep + xStep,

y \* yStep + yStep)

avrgBrightness <- sumBrightness / (xStep \* yStep)

compressedImage[x,y] <- avrgBrightness

next

next

return compressedImage

end function

GetSumBrightnessForArea() iterates through each row between indexes yStart and yEnd. In each of those rows it iterates through each cell between the indexes xStart and xEnd and adds the pixel’s value to a sum. After this has been summed for every pixel in the range, the sum is returned

GetSumBrightnessForArea(image, xStart, yStart, xEnd, yEnd)

Sum <- 0

For y <- yStart to yEnd

For x <- xStart to xEnd

Sum <- sum + image[x,y]

Next

Next

Return sum

End function

After test 21, I’ve found that the int division used to find xStep and yStep can result in the image not being compressed properly if the original width and height are not multiples of the new width or height respectively. For example, an 80px wide image should really have each region that is being compressed to 1 pixel be 2.5 pixels wide when being compressed down to 28px (80/28). However, the int division truncated that down to 2 pixels. This means that a section to the right of the image became cropped when the image was compressed.

As these images already have white backgrounds, this can be solved by extending the original image’s dimensions with white space on both sides until the width is a multiple of 28. This should mean that none of the image is lost as the int division will not need to round down. An alternative approach would be to store xStep and yStep as floats and use proportion to find the sums. E.g. an xStep of 2.5 could sum 3 pixels’ brightness but half the 3rd’s. However, because the letters in the MNIST dataset have empty space around them anyway, using the first solution may make the characters more recognisable to our network so I will implement that.

To do this, this section of code will be added to the start of CompressBitmap(). An equivalent section will also be there for yStep. This works very similarly to ConvertToSquare()

If width(image) mod width > 0

Extension <- (width(image) mod width) / 2

Temp <- new bitmap[width(image), height(image) + 2 \* extension]

For i <- 0 to height(image)

For j <- 0 to extension – 1

Temp[j, i] <- 1

Next

For j <- extension to width(image) – 1

Temp[j, i] <- image[j – extension, i]

Next

For j <- extension + width(image) to width(temp)

Temp[j,i] <- 1

Next

Next

Image <- temp

End if

### Objective 13: Recognise letters using Neural Network

Now we can combine all the image processing and neural network design to recognise the letters in an image

Private queue function ReadLettersInImage(image)

Letters <- GetProcessedImages(image)

While letters is not empty

Letter <- dequeue from letters

Letter <- Invert(letter)

pixelVals <- BitmapToArray(letter)

enqueue nn.RecogniseImage(pixelvals) to readLetters

loop

return readLetters

end function

GetProcessedImages() combines all the design for the image processing together to take an input image in a format such as jpg or png and convert them to a queue of bitmaps storing each letter in that image after being compressed to 28\*28px

Private queue function GetProcessedImages(unprocessedImage)

Image <- convert unprocessedImage to bitmap

Rows <- SplitIntoRows(image)

Letters <- SplitIntoLetters(rows)

While letters is not empty

Letter <- dequeue from letters

Letter <- CompressBitmap(28,28,letter)

Enqueue letter to compressedImages

Loop

Return compressedImages

End function

Before the neural network can recognise the image, it has to be inverted so the letter is white in a back background because the network was trained with the MNIST dataset which is in that format. As the values are between 0 and 1, we can invert an image by subtracting its current pixel values from 1.

Bitmap function Invert(image)

For i <- 0 to width(image)

For j <- 0 to len(image)

image[i,j] <- 1 - image[i,j]

Next

Next

Return image

End function

It also must be converted from a bitmap to a 1D float array which can be done by iterating through each row and appending each pixel’s value. The values are added in each column order as that is how the pixels are indexed in EMIST and in matrix maths. We also divide by 255 to clamp the values down to 0-1.

Float[] function BitmapToArray(image)

outArray <- new float[height(image) \* width(image)

for x <- 0 to width(image) – 1

for y <- 0 to height(image) – 1

outArray[width(image) \* x + y] <- image[x,y] / 256

next

next

return outArray

end function

### Objective 14: Detect Addresses

After we have read each letter in the original image, we will be left with a queue containing each char detected. To split these letters into words with spaces and a full address, we will use the PAF sample to split up the names. This can be done recursively by starting with an empty string and then calling itself for the current string with a space added. If not call itself but with the next char in the queue that was read by the neural network added. Repeat recursively until the list of valid addresses doesn’t contain the current address with either a space or new letter added. In this case return false as the read address must be invalid. The alternative base case is that the queue of letters has been emptied. When this happens check if the current address is in the list of valid addresses and if so, return true. This means a valid address has been found. If we use the currentAddress as a byref variable, then it will be changed as we pass through the different recursions until the full address is formed.

We can search through the list off addresses to find all the ones starting with the current address with this query:

SELECT address FROM addresses

WHERE len(address) >= len(currentAddress)

AND LEFT(address, len(currentAddress)) = currentAddress

If this function returns true then the letters read by the nn were a valid address which is stored in the currentAddress argument now. Else that argument’s value can be discarded.

Private bool function TryGetAddress(byref currentAddress, detectedLetters)

query <- from address in addresses

where len(address) >= len(currentAddress)

and address.Substring(0, len(currentAddress)) = currentAddress

select address

if query is not empty

if detectedLetters is empty

if query contains currentAddress

return true

end if

return false

end if

currentAddress <- currentAddress + ‘ ‘

if TryGetAddress(currentAddress, detectedLetters)

return true

else

currentAddress <- mid(currentAddress, 0, len(currentAddress) – 2)

end if

currentAddress <- currentAddress + dequeue from detectedLetters

if TryGetAddress(currentAddress, detectedLetters)

return true

end if

end if

return false

end function

## Class Diagram

Diagram

Description automatically generated

# Implementation

## Overview

I have Split my project across 3 solutions: AddressReader.exe, MachineLearning.exe and NeuralNetworkLibray.dll

NeuralNetworkLibary.dll is a custom library containing the NeuralNetwork, NNTrainer, MatrixFunctions and TrainingDataReader classes. This dll is then referenced by both AddressReader.exe and MachineLearning.exe as they both require access to classes like the NeuralNetwork.

Graphical user interface, application

Description automatically generated

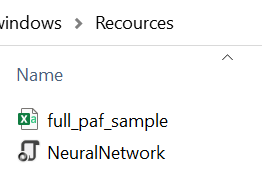
MachineLearning.exe is a windows forms application responsible for creating a neural network and training it on the MNIST dataset until it reaches a suitable accuracy to be used for Optical Character Recognition. After being trained the neural network cam be saved as a json file to a user selected location.

AddressReader.exe is a windows forms application which uses a neural net trained in MachineLearning.exe to recognise the letters in a photo of an address after it has processed that image. The letters detected can then be reformed back into an address and outputted. This process can either be done manually through the form or by automatically reading from an input folder and outputting to an output file. To set the neural network used for AddressReader.exe, a json neural network file should be moved to the project’s resources file and renamed “NeuralNetwork.json”. The list of valid addresses can also be edited here in “full\_paf\_sample.csv”

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated 

## Code Listing

### Neural Network Library

#### MatrixFunctions.cs

using System;

Subroutine Modules

namespace NeuralNetworkLibrary

{

//this is abstract because it shouldn’t be instantiated on its own

public abstract class MatrixFunctions

{

protected float[][] MatrixAdd2D(float[][] mat1, float[][] mat2)

{

//adds mat1 to mat2

for (int i = 0; i < mat1.Length; i++)

{

for (int j = 0; j < mat1[i].Length; j++)

{

//Console.Write(mat1[i][j] + " ");

mat1[i][j] += mat2[i][j];

//Console.Write(mat1[i][j] + " ");

}

//Console.WriteLine();

}

return mat1;

}

protected float[][][] MatrixAdd3D(float[][][] mat1, float[][][] mat2)

{

//adds mat2 to mat1

for (int i = 0; i < mat1.Length; i++)

{

for (int j = 0; j < mat1[i].Length; j++)

{

for (int k = 0; k < mat1[i][j].Length; k++)

{

mat1[i][j][k] += mat2[i][j][k];

}

}

}

return mat1;

}

Advanced Matrix Operations

protected float[] MatrixMult(float[][] mat, float[] vec)

{

//multiplies a 2D matrix and a vector

if (mat.Length != vec.Length)

{

throw new Exception("vector and matrix are different lengths");

}

float[] result = new float[mat[0].Length];

for (int i = 0; i < mat[0].Length; i++)

{

for (int j = 0; j < vec.Length; j++)

{

result[i] += mat[j][i] \* vec[j];

}

}

return result;

}

protected float[][] MatrixSubtract2D(float[][] mat1, float[][] mat2)

{

//subtracts mat2 from mat1

for (int i = 0; i < mat1.Length; i++)

{

for (int j = 0; j < mat1[i].Length; j++)

{

//Console.Write(mat1[i][j] + " ");

mat1[i][j] -= mat2[i][j];

//Console.Write(mat1[i][j] + " ");

}

//Console.WriteLine();

}

return mat1;

}

protected float[][][] MatrixSubtract3D(float[][][] mat1, float[][][] mat2)

{

//subtracts mat2 from mat1

for (int i = 0; i < mat1.Length; i++)

{

for (int j = 0; j < mat1[i].Length; j++)

{

for (int k = 0; k < mat1[i][j].Length; k++)

{

mat1[i][j][k] -= mat2[i][j][k];

}

}

}

return mat1;

}

protected float[][] Randomise2DArray(float[][] arr, float lower, float upper)

{

Random rnd = new Random();

for (int i = 0; i < arr.Length; i++)

{

for (int j = 0; j < arr[i].Length; j++)

{

arr[i][j] = (float)(rnd.NextDouble() \* (upper - lower) + lower);

}

}

return arr;

}

protected float[][] Scalar2DMatrixDiv(float scalar, float[][] matrix)

{

//divides a 2D matrix by a float

for (int i = 0; i < matrix.Length; i++)

{

for (int j = 0; j < matrix[i].Length; j++)

{

matrix[i][j] /= scalar;

}

}

return matrix;

}

protected float[][] Scalar2DMatrixMult(float scalar, float[][] matrix)

{

//multiplies a 2D matrix by a float

for (int i = 0; i < matrix.Length; i++)

{

for (int j = 0; j < matrix[i].Length; j++)

{

matrix[i][j] \*= scalar;

}

}

return matrix;

}

protected float[][][] Scalar3DMatrixDiv(float divider, float[][][] matrix)

{

//divides a 2D matrix by a float

for (int i = 0; i < matrix.Length; i++)

{

for (int j = 0; j < matrix[i].Length; j++)

{

for (int k = 0; k < matrix[i][j].Length; k++)

{

matrix[i][j][k] /= divider;

}

}

}

return matrix;

}

protected float[][][] Scalar3DMatrixMult(float scalar, float[][][] matrix)

{

//multiplies a 3D matrix by a float

for (int i = 0; i < matrix.Length; i++)

{

for (int j = 0; j < matrix[i].Length; j++)

{

for (int k = 0; k < matrix[i][j].Length; k++)

{

matrix[i][j][k] \*= scalar;

}

}

}

return matrix;

}

protected float[][] Setup2DArray(int[] layerlengths)

{

float[][] arr = new float[layerlengths.Length][];

for (int i = 0; i < layerlengths.Length; i++)

{

arr[i] = new float[layerlengths[i]];

}

return arr;

}

protected float[][][] SetupWeights(int[] layerlengths)

{

Random rnd = new Random();

//its length - 1 here because the output layer doesnt have any edges to a next layer

float[][][] arr = new float[layerlengths.Length - 1][][];

for (int i = 0; i < layerlengths.Length - 1; i++)

{

arr[i] = new float[layerlengths[i]][];

for (int j = 0; j < layerlengths[i]; j++)

{

arr[i][j] = new float[layerlengths[i + 1]];

for (int k = 0; k < layerlengths[i + 1]; k++)

{

arr[i][j][k] = (float)rnd.NextDouble();

}

}

}

return arr;

}

protected float Sigmoid(float x)

{

//returns x normalised to be between 0 and 1

return (float)(1 / (1 + Math.Pow(Math.E, -x)));

}

protected float SigmoidPrime(float x)

{

//this is Sigmoid's derivative funtion

double temp = Math.Pow(Math.E, x);

temp /= Math.Pow(1 + temp, 2);

return (float)temp;

}

protected float[] VectorAdd(float[] vec1, float[] vec2)

{

//adds vec1 to vec2

if (vec1.Length != vec2.Length)

{

throw new Exception("vectors don't have the same dimentions");

}

float[] result = new float[vec1.Length];

for (int i = 0; i < result.Length; i++)

{

result[i] = vec1[i] + vec2[i];

}

return result;

}

}

}

#### NeuralNetwork.cs

using System;

using System.IO;

using System.Text.Json;

using System.Text.Json.Serialization;

namespace NeuralNetworkLibrary

{

public class NeuralNetwork : MatrixFunctions

{

Multidimensional Arrays

[JsonInclude] //this means that a json serialisor will save this attribute

public float[][] activations;

[JsonInclude]

public float[][] weightedSums;

[JsonInclude]

public float[][] biases;

[JsonInclude]

public float[][][] weights;

public NeuralNetwork(int[] layerLengths)

{

activations = Setup2DArray(layerLengths);

weightedSums = Setup2DArray(layerLengths);

biases = Randomise2DArray(Setup2DArray(layerLengths), -1, 1);

weights = SetupWeights(layerLengths);

}

[JsonConstructor] //this means that a json Deserialisor will use this constructor

public NeuralNetwork(float[][] activations, float[][] weightedSums, float[][] biases, float[][][] weights)

{

this.activations = activations;

this.weightedSums = weightedSums;

this.biases = biases;

this.weights = weights;

}

Writing and Reading from Files

static public NeuralNetwork LoadNeuralNetwork(string filepath)

{

//returns a neural network stored in a JSON file - exception thrown if not a json file

//make sure to validate outside of here

//This are static so we don't need to instantiate an object for this to work

string input = File.ReadAllText(filepath);

return JsonSerializer.Deserialize<NeuralNetwork>(input,

new JsonSerializerOptions { WriteIndented = true });

}

public char RecogniseImage(float[] imageValues)

{

SetInputLayerActivations(imageValues);

PropegateActivations();

float largestActivation = GetLargestActivation(activations[activations.Length - 1]);

Managed casting of types

return (char)(largestActivation + 65);

}

Writing and Reading from Files

static public void SaveNeuralNet(string filePath, NeuralNetwork nn)

{

//saves a neural network to a JSON file

//This are static so we don't need to instantiate an object for this to work

string output = JsonSerializer.Serialize(nn,

new JsonSerializerOptions { WriteIndented = true });

//Console.WriteLine(output);

File.WriteAllText(filePath, output);

}

//override the base ToString method

public override string ToString()

{

//this is mostly for testing

string output = "Neural Network Values:\n";

output += "nodes per layer:\n";

for (int i = 0; i < activations.Length; i++)

{

output += i + " - " + activations[i].Length + "\n";

}

return output;

}

private float GetLargestActivation(float[] activations)

{

//finds the index of the largest activation in the output layer

int currentLargest = 0;

for (int i = 1; i < activations.Length; i++)

{

if (activations[currentLargest] < activations[i])

{

currentLargest = i;

}

}

return currentLargest;

}

private void PropegateActivations()

{

//Moves forward through the neural network calculating each layer's

//activations using the activations from the prev layer and the current

//layer's weights and biases

//loop through each layer

for (int i = 1; i < activations.Length; i++)

{

//multiply the weights leading into this layer by the prev layer's

//activations using matrix multiplication

float[] temp = MatrixMult(weights[i - 1], activations[i - 1]);

//vector add the biases vector to the result

temp = VectorAdd(temp, biases[i]);

//create a byref copy of this weighted sum to be used by a training program

temp.CopyTo(weightedSums[i], 0);

//run through each weighted sum we've calculated and normalise them down to

//0-1 with the sigmoid function

for (int j = 0; j < temp.Length; j++)

{

temp[j] = Sigmoid(temp[j]);

}

//set the activations of the current layer to the values we just calculated

activations[i] = temp;

}

}

private void SetInputLayerActivations(float[] layerActivations)

{

//sets the input layer of the neural net's activations to the parameter input

if (activations[0].Length == layerActivations.Length)

{

activations[0] = layerActivations;

}

else

{

throw new IndexOutOfRangeException();

}

}

}

}

#### NNTrainer.cs

using System;

namespace NeuralNetworkLibrary

{

Simple OOP model

public class NNTrainer : MatrixFunctions

{

//these variables need to be global so their values

//are preserved external between method calls

//the neural network being trained

NeuralNetwork nn;

//the MNIST reader

TrainingDataReader tdReader;

//number of samples trained off

int numberOfPredictions;

//number of times the neural network has been right

int numberOfCorrectPredictions;

public float accuracy { get; private set; }

// public bool continueTraining { get; private set; }

public int batchSize { get; private set; }

public float learningRate { get; private set; }

public NNTrainer(ref NeuralNetwork nn, int batchSize, float learningRate)

{

//setupt default values and set neural network to be trained

this.nn = nn;

this.batchSize = batchSize;

this.learningRate = learningRate;

tdReader = new TrainingDataReader();

accuracy = 0f;

numberOfCorrectPredictions = 0;

numberOfPredictions = 0;

// continueTraining = false;

}

public void Dispose()

{

//close the connections with the MNSIT Files

tdReader.Dispose();

}

public void TrainForSingleBatch(out float currentAccuracy)

{

TrainingSample sample;

char recognisedChar;

//^1 index means its the first index at the back

float[] expectedOutputActivations = new float[nn.activations[^1].Length];

int[] layerlengths = new int[nn.activations.Length];

for (int i = 0; i < layerlengths.Length; i++)

{

layerlengths[i] = nn.activations[i].Length;

}

// copy dimentions of array - values will all be overwritten later

//I have to clone it because arrays are reference types

float[][][] wDerivativesSum = SetupWeights(layerlengths);

float[][][] wDerivatives = SetupWeights(layerlengths);

float[][] bDerivativesSum = Setup2DArray(layerlengths);

float[][] bDerivatives = Setup2DArray(layerlengths);

///////////testing/////////////////////////////////////////////////////

//sample = tdReader.GetNextTrainingSample();

//recognisedChar = nn.RecogniseImage(sample.data);

//the first node in the output layer should be 1 for testing purposes

//Array.Fill(expectedOutputActivations, 0f);

//expectedOutputActivations[0] = 1f;

////////////////////////////////////////////////

GetGradients(expectedOutputActivations, ref wDerivativesSum, ref bDerivativesSum);

for (int i = 0; i < batchSize - 1; i++)

{

sample = tdReader.GetNextTrainingSample();

recognisedChar = nn.RecogniseImage(sample.data);

accuracy = UpdateAccuracy(sample.label, recognisedChar);

Single Dimensional Arrays

//expected values should be the label's index as 1 and the rest 0;

Array.Fill(expectedOutputActivations, 0f);

expectedOutputActivations[sample.label - 65] = 1f;

GetGradients(expectedOutputActivations, ref wDerivatives, ref bDerivatives);

wDerivativesSum = MatrixAdd3D(wDerivativesSum, wDerivatives);

bDerivativesSum = MatrixAdd2D(bDerivativesSum, bDerivatives);

//reset training data if at end of file

if (tdReader.EOF)

{

tdReader.Dispose();

tdReader = new TrainingDataReader();

}

}

//get average derivatives over all the samples in the batch

wDerivatives = Scalar3DMatrixDiv((float)batchSize, wDerivativesSum);

bDerivatives = Scalar2DMatrixDiv((float)batchSize, bDerivativesSum);

//testing

//OutputGradients(wDerivatives, bDerivatives);

///

//new biases = old biases - learning rate \* biases gradients

nn.biases = MatrixSubtract2D(

nn.biases,

Scalar2DMatrixMult(learningRate, bDerivatives));

//new weights = old weights - learning rate \* weights gradients

nn.weights = MatrixSubtract3D(

nn.weights,

Scalar3DMatrixMult(learningRate, wDerivatives));

currentAccuracy = accuracy;

}

private void GetGradients(float[] expectedOutActivs,

ref float[][][] wDerivatives, ref float[][] bDerivatives)

{

//Finds the derivatives for every weight and bias in the network

//get the derivatives for the output layer's weights and biases

wDerivatives[^1] = GetOutputLWeightsGradient(expectedOutActivs);

bDerivatives[^1] = GetOutputLBiasesGradient(expectedOutActivs);

//this loop find the derivatives for all the hidden layers

//start at 2 because the index from the back of an array starts at 1

//end at the last hidden layer because the biases in the input layer are never used

//and there are no weights going into the input layer

for (int i = 2; i < nn.activations.Length; i++)

{

bDerivatives[^i] = GetHiddenLayerBGradients(bDerivatives[^i].Length,

nn.weightedSums[^i],

nn.weights[^(i - 1)],

bDerivatives[^(i - 1)]);

wDerivatives[^i] = GetHiddenLayerWGradients(nn.activations[^i].Length,

bDerivatives[^i],

nn.activations[^(i + 1)]);

}

}

private float[] GetHiddenLayerBGradients(int layerLength, float[] nextWeightedSums,

float[][] nextWeights, float[] nextBiasGradients)

{

// returns an array storing the derivative for each bias in a hidden layer

//for this to work, the derivatives for the next layer must have already been calculated

float[] gradients = new float[layerLength];

//loop through each node in the layer

for (int i = 0; i < layerLength; i++)

{

gradients[i] = SigmoidPrime(nextWeightedSums[i]);

float tempSum = 0;

//this nested loop is used to find an activation's summed impact on the cost func

for (int j = 0; j < nextBiasGradients.Length; j++)

{

tempSum += nextWeights[i][j] \* nextBiasGradients[j];

}

gradients[i] \*= tempSum;

}

return gradients;

}

private float[][] GetHiddenLayerWGradients(int layerLength, float[] biasGradients,

float[] prevActivations)

{

//calculate the weights gradients for a layer using that layer's bias gradients

//this means that the bias gradients have to be calculated first

//setup array

float[][] gradients = new float[prevActivations.Length][];

for (int i = 0; i < gradients.Length; i++)

{

gradients[i] = new float[layerLength];

}

//find gradients using the equation in the analysis

for (int l = 0; l < gradients.Length; l++)

{

for (int k = 0; k < gradients[l].Length; k++)

{

gradients[l][k] = biasGradients[k] \* prevActivations[l];

}

}

return gradients;

}

private float[] GetOutputLBiasesGradient(float[] expectedOutputActivations)

{

//returns an array storing the derivative for each bias in the output layer

//derivatives store the derivative for each bias in the output layer

float[] derivatives = new float[nn.activations[^1].Length];

//loop through every node in the output layer

for (int j = 0; j < nn.activations[^1].Length; j++)

{

//each line here represents a partial derivative in the chain

float mTemp = SigmoidPrime(nn.weightedSums[^1][j]);

mTemp \*= 2 \* (nn.activations[^1][j] - expectedOutputActivations[j]);

derivatives[j] = mTemp;

}

return derivatives;

}

Complex Mathematics

private float[][] GetOutputLWeightsGradient(float[] expectedOutputActivations)

{

//Returns an array storing the derivatives for each weight going into the output layer

//derivatives stores an array for each element in layer L-1 with each of those

//storing derivatives for the weight connecting that and a node in layer L

float[][] derivatives = new float[nn.activations[^2].Length][];

for (int i = 0; i < derivatives.Length; i++)

{

derivatives[i] = new float[nn.activations[^1].Length];

}

//loop through every node in the second last layer

for (int k = 0; k < nn.activations[^2].Length; k++)

{

//loop through every node in the output layer

for (int j = 0; j < nn.activations[^1].Length; j++)

{

//each line here represents a partial derivative in the chain

float mTemp = nn.activations[^2][k];

mTemp \*= SigmoidPrime(nn.weightedSums[^1][j]);

mTemp \*= 2 \* (nn.activations[^1][j] - expectedOutputActivations[j]);

derivatives[k][j] = mTemp;

}

}

return derivatives;

}

void OutputGradients(float[][][] wGradients, float[][] bGradients)

{

string output = "";

output += "weights grads\n";

for (int i = 0; i < wGradients.Length; i++)

{

output += i + " -\n";

for (int j = 0; j < wGradients[i].Length; j++)

{

for (int k = 0; k < wGradients[i][j].Length; k++)

{

output += wGradients[i][j][k] + " ";

}

output += "\n";

}

output += "\n";

}

output += "\nbiases gradients\n";

for (int i = 0; i < bGradients.Length; i++)

{

output += i + " - ";

for (int j = 0; j < bGradients[i].Length; j++)

{

output += bGradients[i][j] + " ";

}

output += "\n";

}

Console.WriteLine(output);

}

private float UpdateAccuracy(char correctChar, char recognisedChar)

{

if (correctChar == recognisedChar)

{

numberOfCorrectPredictions++;

}

numberOfPredictions++;

try

{

float accuracy = (float)numberOfCorrectPredictions / (float)numberOfPredictions;

//if (numberOfPredictions > 100)

//{

// numberOfCorrectPredictions = 0;

// numberOfPredictions = 0;

//}

return (float)Math.Round((decimal)accuracy, 2);

}

catch (DivideByZeroException)

{

return numberOfCorrectPredictions;

}

}

}

}

#### TrainingSample.cs

namespace NeuralNetworkLibrary

{

Records

public struct TrainingSample

{

//struct representing a single sample in EMNIST,

//storing its pixel data and which letter it represents

//init means the property is readonly after instantiation

public float[] data { get; init; }

public char label { get; init; }

}

}

### Machine Learning

#### Form1.cs

using System;

using System.Drawing;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Diagnostics;

Subroutine Modules

using NeuralNetworkLibrary;

namespace MachineLearning

{

public partial class MachineLearning : Form

{

//These all need to global so that their values will be preserved between event calls

NeuralNetwork nn;

NNTrainer trainer;

Stopwatch stopwatch = new Stopwatch();

bool continueTraining = false;

//Called at startup

public MachineLearning()

{

InitializeComponent();

//timer.tick event occures every second and is used to update the training timer

//after the timer is started - let Timer\_Tick handle that event

timer.Tick += Timer\_Tick;

//hide this button at the start

btnStopTraining.Hide();

//Set these labels to no text so that I can still see them in the designer

lblAccuracy.Text = "";

lblCharDetected.Text = "";

lblSamplesSeen.Text = "";

lblTimeTraining.Text = "";

}

private void btnGenNN\_Click(object sender, EventArgs e)

{

//Called when the Generate Neural network button is pressed. Opens the GenerateNeuralNetwork

//form and sets this form's neural network to the one generated in that form

var genNN = new GenerateNeuralNetwork();

genNN.ShowDialog();

//when the dialogue is closed

nn = genNN.nn;

genNN.Dispose();

//DEBUG

//MessageBox.Show(nn.ToString());

}

private void btnLoadNN\_Click(object sender, EventArgs e)

{

//Reads a neural network from a file and assigns that to this form's neural network

//open windows file dialogue

OpenFileDialog dialogue = new OpenFileDialog();

dialogue.ShowDialog();

string path = dialogue.FileName;

dialogue.Dispose();

Good exception handling

//try to read the file selected

try

{

nn = NeuralNetwork.LoadNeuralNetwork(path);

//DEBUG

//MessageBox.Show(nn.ToString());

}

//if the file selected isn't valid

catch

{

MessageBox.Show("Error - file selected is not a valid json file");

}

}

private void btnRecogniseNewChar\_Click(object sender, EventArgs e)

{

//reads a random bit of training data and tries to read it

//if there is a neural network loaded

if (nn != null)

{

//if we tried to open the training data file during training then we'd get

//an exception by opening the MNIST file twice

//so only do this if we are not training

if (!continueTraining)

{

TrainingDataReader tdReader = new TrainingDataReader();

//skip to a random sample between sample 1 and 1000

Random random = new Random();

int limit = random.Next(1000);

for (int i = 0; i < limit; i++)

{

tdReader.GetNextTrainingSample();

}

//get next training sample

var sample = tdReader.GetNextTrainingSample();

//ouput the sample to the UI

pcbxOutput.Image = ConvertSampleToBitmap(sample.data, 28);

//Debug

//MessageBox.Show(sample.label + "");

//try to recognise the letter in the image

lblCharDetected.Text = nn.RecogniseImage(sample.data).ToString();

//drop the stream

tdReader.Dispose();

}

else

{

MessageBox.Show("Cannot use neural network during training");

}

}

else

{

MessageBox.Show("No neural network loaded");

}

}

private void btnSaveNN\_Click(object sender, EventArgs e)

{

//Allows a user to save a neural network

//open the windows file dialogue

SaveFileDialog dialogue = new SaveFileDialog();

dialogue.ShowDialog();

string path = dialogue.FileName;

dialogue.Dispose();

Good exception handling

//save the neural network in the location chosen in the dialogue

try

{

NeuralNetwork.SaveNeuralNet(path, nn);

}

catch (Exception)

{

//this may happen if the user selects no file name/ path

MessageBox.Show("Network unable to be saved");

}

}

*Asynchr**onous Programming*

private async void btnStartTraining\_Click(object sender, EventArgs e)

{

//Starts training the current neural network

//this is an async sub so the training can be done on a background thread

//which stops the UI freezing

//DEBUG

//MessageBox.Show(nn.ToString());

//if we have a neural net loaded

if (nn != null)

{

Managed casting of types

//instantiate a new trainer with the inputs of batch size and learning rate

//the nn is passed by ref so changes made by the trainer internally will change our loaded one

trainer = new NNTrainer(ref nn,

(int)nupBatchSize.Value, (float)nupLearningRate.Value);

//hide the start training button

btnStartTraining.Hide();

//show the stop training button in the same place

btnStopTraining.Show();

timer.Start();

stopwatch.Start();

//Start training in a new background thread while the rest of the

//program can keep running normally using an async method

//await means this sub will wait unil that task is done

await Task.Run(Train);//StartTrainingAsync(trainer);

}

else

{

MessageBox.Show("No neural network loaded");

}

}

private void btnStopTraining\_Click(object sender, EventArgs e)

{

//Stops the program training a neural network

//as this is now false, the background thread will exit its loop and stop

continueTraining = false;

//show the start training button again and hide the stop one

btnStartTraining.Show();

btnStopTraining.Hide();

//reset our timer

timer.Stop();

stopwatch.Reset();

//DEBUG

//MessageBox.Show(nn.ToString());

}

private Bitmap ConvertSampleToBitmap(float[] data, int width)

{

//takes a 2D array of floats between 0 and 1 and converts it to a

//bitmap with pixel values between 0 and 255

Bitmap image = new Bitmap(width, width);

int counter = 0;

for (int i = 0; i < 28; i++)

{

for (int j = 0; j < 28; j++)

{

Managed casting of types

int brightness = (int)(data[counter++] \* 255f);

Color colour = Color.FromArgb(brightness, brightness, brightness);

image.SetPixel(i, j, colour);

}

}

return image;

}

private void Timer\_Tick(object sender, EventArgs e)

{

//update the training timer label

lblTimeTraining.Text = Math.Round(stopwatch.Elapsed.TotalSeconds, 1).ToString();

}

void Train()

{

//trains the neural network continuously until continueTraining is set to false

//This can only be done externally so this has to be ran as an async task

//current accuracy to output

float accuracy;

//current number of samples used to train the network

int imagesSeen = 0;

continueTraining = true;

while (continueTraining)

{

//train neural network with a single batch of samples

trainer.TrainForSingleBatch(out accuracy);

imagesSeen += trainer.batchSize;

//as this is running in a different thread, a delegate is needed to output values

//on the IO thread

Invoke((MethodInvoker)delegate

{

//update the accuracy and number of samples seen labels

lblAccuracy.Text = accuracy.ToString();

lblSamplesSeen.Text = imagesSeen.ToString();

});

}

//dispose of the trainer at the end to remove its file streams

trainer.Dispose();

}

}

}

#### GenerateNeuralNetwork.cs

using System;

using System.Windows.Forms;

using NeuralNetworkLibrary;

namespace MachineLearning

{

public partial class GenerateNeuralNetwork : Form

{

//this has to be global and public so it's value is preserved and

//can be read after the popup form is closed

public NeuralNetwork nn;

public GenerateNeuralNetwork()

{

InitializeComponent();

lblOut.Text = "";

btnSave.Hide();

}

private void btnGenerate\_Click(object sender, EventArgs e)

{

//called when Generate is clicked and sets the number of

//layers in the network and instantiates all the numeric

//up down boxes needed to set the lengths of those layers

//for potential retrys

btnSave.Hide();

//this value can only be between 2 and 20 from the designer so

//validation is handled for char input or input < 2

//it has to be >= 2 so we can have an input and output layer

int numberOfLayers = (int)nupNumOfLayers.Value;

//remove any previous NumericUpDowns

flwlayoutLayers.Controls.Clear();

//for every layer, instantiate a numeric up down

for (int i = 0; i < numberOfLayers; i++)

{

//using a numericUpDown means only numbers can be entered so I don't have

//to validate this

Runtime Instantiated Controls

var nudLayerLength = new NumericUpDown();

Defensive programming

//set the minimum, maximum and default value for the boxes

nudLayerLength.Minimum = 1;

nudLayerLength.Maximum = 10000;

nudLayerLength.Value = 1;

//add the nup we've just instantiated to the flow layout pannel

flwlayoutLayers.Controls.Add(nudLayerLength);

//moves to new line - (make sure flow direction is set to lefttoright)

flwlayoutLayers.SetFlowBreak(nudLayerLength, true);

}

Defensive programming

//get the first and last nup

NumericUpDown inputlayer = (NumericUpDown)flwlayoutLayers.Controls[0];

NumericUpDown outputlayer = (NumericUpDown)flwlayoutLayers.Controls[^1];

//the input layer must have 784 nodes (28\*28) to be able to train with MNIST

inputlayer.Value = 784;

//make sure this value can't be changed

inputlayer.Enabled = false;

//the output layer must have 28 nodes as there are only 26 letters

outputlayer.Value = 26;

outputlayer.Enabled = false;

}

private void btnSave\_Click(object sender, EventArgs e)

{

//this button only appears after btnSetLayerLengths has been clicked

//after this form closes its parent form will take the public value of nn

this.Close();

}

private void btnSetLayerlengths\_Click(object sender, EventArgs e)

{

//called when set layer lengths is clicked

//sets the lengths of the layers of the neural network being made

//using the values in the nups

//only generate something if there is an input to read

if(flwlayoutLayers.Controls.Count > 0)

{

int[] layerLengths = new int[flwlayoutLayers.Controls.Count];

for (int i = 0; i < layerLengths.Length; i++)

{

//get the next layer length input

NumericUpDown nudLayerLength = (NumericUpDown)flwlayoutLayers.Controls[i];

//assign that input's value to the corrisponding layer length

layerLengths[i] = (int)nudLayerLength.Value;

}

//generate the neural network

nn = new NeuralNetwork(layerLengths);

//display the neural network to the user

DisplayLayerLengths(layerLengths);

//show the save button

btnSave.Show();

}

}

private void DisplayLayerLengths(int[] layerLengths)

{

//outputs the lengths of each layer

lblOut.Text = "New Neural Network:\n";

for (int i = 0; i < layerLengths.Length; i++)

{

lblOut.Text += $"layer {i + 1}: {layerLengths[i]} nodes\n";

}

}

}

}

### Address Reader

#### Form1.cs

using System;

using System.Collections.Generic;

using System.Data;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

using NeuralNetworkLibrary;

namespace AddressReader

{

public partial class AddressReader : Form

{

//these need to be global so their values aren't disposed

//between event calls

NeuralNetwork nn;

static List<string> addresses;

FileSystemWatcher fileWatcher;

string inputFolderPath = "";

string outputFilePath = "";

public AddressReader()

{

InitializeComponent();

btnAutoOff.Hide();

lblInputFile.Text = "";

lblOutputFile.Text = "";

//load the neural network

nn = NeuralNetwork.LoadNeuralNetwork("Recources/NeuralNetwork.JSON");

//load the list of valid addresses

addresses = GetAddressList();

lblReadAddress.Text = "";

}

private void btnAutoOff\_Click(object sender, EventArgs e)

{

//disposing of the file watcher stops us checking if files have been saved

//so it stops auto reading new addresses

fileWatcher.Dispose();

btnAutoOn.Show();

btnAutoOff.Hide();

}

private void btnAutoOn\_Click(object sender, EventArgs e)

{

//if the input and output locations have been set

if (inputFolderPath != "" && outputFilePath != "")

{

btnAutoOff.Show();

btnAutoOn.Hide();

fileWatcher = new FileSystemWatcher(inputFolderPath);

//when a file is created in the file we're watching call this event

//This event reads the address in the image that was just saved and outputs the

//address to the specified text file

fileWatcher.Created += new FileSystemEventHandler(FileWatcher\_Created);

//for some reason FileWachers can't raise events by default

fileWatcher.EnableRaisingEvents = true;

}

else

{

MessageBox.Show("Input and output not both set");

}

}

private void btnInputFile\_Click(object sender, EventArgs e)

{

//sets the folder where we will detect new input images.

//filewatcher will be null if the program isn't autoreading

//we only want to change the sources while not reading

if (fileWatcher == null)

{

//FolderBrowserDialgue works similar to OpenFileDialogue but only lets you

//select a folder

using (FolderBrowserDialog dialog = new FolderBrowserDialog())

{

dialog.ShowDialog();

inputFolderPath = dialog.SelectedPath;

//split the directories

string[] path = inputFolderPath.Split('\\');

//get the file name which will be at the last index of the path

lblInputFile.Text = path[^1];

}

}

else

{

MessageBox.Show("Cannot change folder while auto reading");

}

}

private void btnOutputFile\_Click(object sender, EventArgs e)

{

//sets the file where we will output read addresses.

//filewatcher will be null if the program isn't autoreading

//we only want to change the sources while not reading

if (fileWatcher == null)

{

//FolderBrowserDialgue works similar to OpenFileDialogue but only lets you

//select a folder

using (OpenFileDialog dialog = new OpenFileDialog())

{

dialog.ShowDialog();

//the output has to be a text file

if (dialog.FileName.EndsWith(".txt"))

{

outputFilePath = dialog.FileName;

//split the directories

string[] path = outputFilePath.Split('\\');

//get the file name which will be at the last index of the path

lblOutputFile.Text = path[^1];

}

else

{

MessageBox.Show("Please enter a valid text file");

}

}

}

else

{

MessageBox.Show("Cannot change folder while auto reading");

}

}

private async void btnReadImage\_Click(object sender, EventArgs e)

{

//Reads the text in a single image

////debug

//addresses.Add("A A");

////

//lets us dispose of the dialogue automatically

using (OpenFileDialog dialog = new OpenFileDialog())

{

//open the windows file dialogue to let a user select a file

dialog.ShowDialog();

try

{

//try reading the file selected as an image

//output an error to the user if an exception is thrown

Image image = Image.FromFile(dialog.FileName);

//convert the image of an address into a string storing that address

//run readAddress asyncronously because it is an intensive process

//that would freeze the ui thread for a couple seconds

string address = await ReadAddressAsync(image);

//output read address

lblReadAddress.Text = address;

//output image selected

pcbxOut.Image = image;

//the address will be empty if the image had no words in it or the address

//wasn't in the PAF

if (address == "")

{

MessageBox.Show("No Address Detected");

}

}

catch (Exception)

{

MessageBox.Show("Invalid File");

}

}

}

private async void FileWatcher\_Created(object sender, FileSystemEventArgs e)

{

//called when a File is created in the file we're watching

//when this happens automatically read the text in that imgage

//and output to the selected text file

//try catch in case the new file isn't an image

try

{

Image image = Image.FromFile(e.FullPath);

string address = await ReadAddressAsync(image);

using (StreamWriter writer = new StreamWriter(outputFilePath, true))

{

writer.WriteLine(address);

}

}

catch (Exception)

{

MessageBox.Show("Invalid file");

}

}

private float[] BitmapToArray(Bitmap image)

{

//combines each column in a bitmap to form a single 1D float array

// 7 9 1

// 5 2 6 - > 7 5 3 9 2 0 1 6 8

// 3 0 8

//1D array to store all our values

float[] outArray = new float[image.Height \* image.Width];

//iterate through each pixel in the image

for (int x = 0; x < image.Width; x++)

{

for (int y = 0; y < image.Height; y++)

{

//get the index in outarray

int index = image.Width \* x + y;

//set that index's value

//all our values will be black and white by now so

//the brightness 0-1 is all we need to represent the pixel

outArray[index] = image.GetPixel(x, y).GetBrightness();

}

}

return outArray;

}

private Bitmap CompressBitmap(int width, int height, Bitmap image)

{

//compresses image down to the size of width\*height using lossy compression

//and averaging the pixel values in regular areas of the original image

//if the original image length isn't a multiple of the new length,

//not all the data will be stored in the compressed image so we widen the image

//with white borders until it is a multiple

//This also means we can skip the step of widening the image to add an outline later too

if (image.Width % width > 0)

{

//half the remainder of the image's width and the new width to get the

//size of the white outline on both sides of the original

int extention = (image.Width % width) / 2;

//temp is a bitmap with the dimentions of image after being extended

var temp = new Bitmap(image.Width + (2 \* extention), image.Height); ;

//loop through each row in image

for (int i = 0; i < image.Height; i++)

{

//add white border to the left side

for (int j = 0; j < extention; j++)

{

temp.SetPixel(j, i, Color.White);

}

//add original image pixels for that row

for (int j = extention; j < extention + image.Width; j++)

{

temp.SetPixel(j, i, image.GetPixel(j - extention, i));

}

//add white border to the right side

for (int j = extention + image.Width; j < temp.Width; j++)

{

temp.SetPixel(j, i, Color.White);

}

}

//set image to the extended image in temp

image = temp;

}

//if the original image's height isn't a multiple of the new height

//extend the image with a white outline so it is

if (image.Height % height > 0)

{

//half the remainder of the image's height and the new height to get the

//size of the white outline added on both sides of the original (top and bottom)

int extention = (image.Height % height) / 2;

//temp is a bitmap with the dimentions of image after being extended

var temp = new Bitmap(image.Width, image.Height + (2 \* extention));

//loop through each column in the original image

for (int i = 0; i < image.Width; i++)

{

//add white border to the top

for (int j = 0; j < extention; j++)

{

temp.SetPixel(i, j, Color.White);

}

//add original image pixels for that column

for (int j = extention; j < extention + image.Height; j++)

{

temp.SetPixel(i, j, image.GetPixel(i, j - extention));

}

//add white border to the bottom

for (int j = extention + image.Height; j < temp.Height; j++)

{

temp.SetPixel(i, j, Color.White);

}

}

//set image to the extended image in temp

image = temp;

}

//new bitmap which will store image after being resized down to width\*height

Bitmap compressedImage = new Bitmap(width, height);

//width of a section to be compressed to 1 pixel

int xStep = image.Width / width;

//width of a section to be compressed to 1 pixel

int yStep = image.Height / height;

for (int y = 0; y < height; y++)

{

for (int x = 0; x < width; x++)

{

float sumBrightness = GetSumBrightnessForArea(image,

x \* xStep, y \* yStep,

x \* xStep + xStep,

y \* yStep + yStep);

float avrgBrightness = sumBrightness / (xStep \* yStep);

avrgBrightness \*= 255f;

Color newColour = Color.FromArgb((int)avrgBrightness,

(int)avrgBrightness, (int)avrgBrightness);

compressedImage.SetPixel(x, y, newColour);

}

}

return compressedImage;

}

private Bitmap ConvertToSquare(Bitmap image)

{

//stores the imaage after being converted to a square

Bitmap squareImage;

//stores the offset from one end of the square image

//to where the original image's pixels start

int offset;

//if the image is taller

if (image.Height > image.Width)

{

squareImage = new Bitmap(image.Height, image.Height);

offset = (image.Height - image.Width) / 2;

//go through each row

for (int y = 0; y < image.Height; y++)

{

//go through each column

for (int x = 0; x < offset; x++)

{

squareImage.SetPixel(x, y, Color.White);

}

//add offset number of white pixels at the start

for (int x = offset; x < image.Width + offset; x++)

{

squareImage.SetPixel(x, y,

image.GetPixel(x - offset, y));

//offset is subtracted from x for image indexes to get it back to starting

//from 0

}

//add offset number of white pixels at end

for (int x = image.Width + offset; x < squareImage.Width; x++)

{

squareImage.SetPixel(x, y, Color.White);

}

}

}

//if the image is wider

else if (image.Height < image.Width)

{

squareImage = new Bitmap(image.Width, image.Width);

offset = (image.Width - image.Height) / 2;

//go through each column

for (int x = 0; x < image.Width; x++)

{

//add offset number of white pixels at the start

for (int y = 0; y < offset; y++)

{

squareImage.SetPixel(x, y, Color.White);

}

//add the original pixels

for (int y = offset; y < image.Height + offset; y++)

{

squareImage.SetPixel(x, y,

image.GetPixel(x, y - offset));

}

//add offset number of white pixels at end

for (int y = image.Height + offset; y < squareImage.Height; y++)

{

squareImage.SetPixel(x, y, Color.White);

}

}

}

else

{

//if it gets here its already a square

return image;

}

return squareImage;

}

private bool DoesColumnContainBlackPixel(Bitmap bitmap, int x)

{

//run down a column and check if there is a black pixel

for (int i = 0; i < bitmap.Height; i++)

{

if (bitmap.GetPixel(x, i).GetBrightness() == 0)

return true;

}

return false;

}

private bool DoesRowContainBlackPixel(Bitmap bitmap, int y)

{

//run along a row and check if there is a black pixel

for (int i = 0; i < bitmap.Width; i++)

{

//1 is white

if (bitmap.GetPixel(i, y).GetBrightness() == 0)

return true;

}

return false;

}

List Operations

List<string> GetAddressList()

{

//read the paf database to get a list of every valid address

List<string> addresses = new List<string>();

using (StreamReader reader = new StreamReader("Recources/full\_paf\_sample.csv"))

{

//skip headers line

reader.ReadLine();

//until we're at the end of the file

while (!reader.EndOfStream)

{

//read a line and split it accross the commas splitting the address parts

string line = reader.ReadLine();

string[] splitAddress = line.Split(',');

string address = "";

//convert the split up address array into a string split up by spaces

for (int i = 0; i < splitAddress.Length - 2; i++)

{

//some of the parts will be empty on some addresses so skip those ones

//as that would lead to empty spaces

if (splitAddress[i] != "")

{

address += splitAddress[i] + " ";

}

}

//remove the last space added at the end

address = address.Remove(address.Length - 1);

//add this address to the list of valid addresses

addresses.Add(address);

}

}

return addresses;

}

Queue Operations

private Queue<Bitmap> GetProcessedImages(Image unprocessedImage)

{

//convert input image to a bitmap

Bitmap image = new Bitmap(unprocessedImage);

image = ThresholdImage(image, 0.5f);

//split image into rows of letters

Queue<Bitmap> rows = SplitIntoRows(image);

//split each row into square images of each letter

Queue<Bitmap> letters = SplitIntoLetters(rows);

//compress each letter

Queue<Bitmap> compressedImages = new Queue<Bitmap>();

while (letters.Count > 0)

{

Bitmap letter = letters.Dequeue();

compressedImages.Enqueue(CompressBitmap(28, 28, letter));

}

return compressedImages;

}

//for an async function, the type in the <> is the return datatype

Task<Queue<Bitmap>> GetProcessedImagesAsync(Image unprocessedImage)

{

//converts GetProcessedImages to a task that can be ran asynchronously

//using a lambda expression representing a function that calls GetProcessedImages

//with unprocessedImage as its argument

return Task.Run(() => GetProcessedImages(unprocessedImage));

}

private float GetSumBrightnessForArea(Bitmap image, int xStart, int yStart, int xEnd,

int yEnd)

{

//iterate through a region of pixels and sum the brightnesses for each pixel

float sum = 0;

for (int i = yStart; i < yEnd; i++)

{

for (int j = xStart; j < xEnd; j++)

{

sum += image.GetPixel(j, i).GetBrightness();

}

}

return sum;

}

private Bitmap Invert(Bitmap letter)

{

//inverts an image to the opposite grey shade

for (int i = 0; i < letter.Width; i++)

{

for (int j = 0; j < letter.Height; j++)

{

//get a value between 1 and 0 that's the inverted form of the original image's

int inverted = (int)(255f \* (1f - letter.GetPixel(i, j).GetBrightness()));

//convert this value to a colour

Color invertedPixel = Color.FromArgb(inverted, inverted, inverted);

//set inverted colour

letter.SetPixel(i, j, invertedPixel);

}

}

return letter;

}

async Task<string> ReadAddressAsync(Image inputImage)

{

//finds the letters in input image, converts them to a queue of chars and then

//finds the address they match if they are a valid address. This address is then

//returned

//get the queue of every letter in the image

Queue<char> readLetters = await ReadLettersInImageAsync(inputImage);

string address = "";

//if the read letters are a valid address

if (TryGetAddress(ref address, readLetters))

{

//return the address detected

return address;

}

//else

return "";

}

private async Task<Queue<char>> ReadLettersInImageAsync(Image image)

{

Queue<Bitmap> letters = await GetProcessedImagesAsync(image);

Queue<char> readLetters = new Queue<char>();

while (letters.Count > 0)

{

Bitmap letter = letters.Dequeue();

letter = Invert(letter);

float[] pixelVals = BitmapToArray(letter);

readLetters.Enqueue(

nn.RecogniseImage(pixelVals));

}

return readLetters;

}

private Bitmap SplitColumn(Bitmap image, int xLowerBound, int xUpperBound)

{

//splits a column of a letter from a base image using the upper and lower bounds

//of its x value

//instantiate row to be the same width as the original image

//and height of the difference between the bounds

Bitmap column = new Bitmap(xUpperBound - xLowerBound, image.Height);

//loop through each pixel in the range

for (int x = xLowerBound; x < xUpperBound; x++)

{

for (int y = 0; y < image.Height; y++)

{

//set the rows' pixel value to image's equivalent

column.SetPixel(x - xLowerBound, y, image.GetPixel(x, y));

}

}

return column;

}

Complex User Defined Algorithms

private Queue<Bitmap> SplitIntoColumns(Bitmap image)

{

int xLowerBound = 0;

int xUpperBound;

Queue<Bitmap> Columns = new Queue<Bitmap>();

while (xLowerBound < image.Width)

{

//search through image until a column contains a black pixel

//-left side of a letter found

//or we reach the end of the image - && means the second half

//wont be called if the first is false so we can avoid an out of bounds exception

while (xLowerBound < image.Width &&

!DoesColumnContainBlackPixel(image, xLowerBound))

{

xLowerBound++;

}

//if the lower bound for a row is at the end of the image, that means there isn't

//another row as we're at the end so we skip the section for the ipperbound and

//splitting the row off

if (xLowerBound < image.Width)

{

//set the upperbound of the current row's y to be the same as the lowerbound

xUpperBound = xLowerBound;

//search through the image until a row doesn't contain a black pixel

//-right side of letter found

//or we reach the end of the image - && means the second half

//wont be called if the first is false so we can avoid an out of bounds

//exception

while (xUpperBound < image.Width &&

DoesColumnContainBlackPixel(image, xUpperBound))

{

xUpperBound++;

}

//split the row from the image and equeue it

Columns.Enqueue(SplitColumn(image, xLowerBound, xUpperBound));

//for the next row the lower bound will be the pixel after the

//currrent upperbound

xLowerBound = xUpperBound + 1;

}

}

return Columns;

}

private Queue<Bitmap> SplitIntoLetters(Queue<Bitmap> rows)

{

Queue<Bitmap> letters = new Queue<Bitmap>();

while (rows.Count > 0)

{

Queue<Bitmap> columns = SplitIntoColumns(rows.Dequeue());

//enqueue all the items in temp to letters

while (columns.Count > 0)

{

//letters.Enqueue(temp.Dequeue());

Bitmap letter = columns.Dequeue();

//the height of each letter will be the same as the tallest letter in the

//row so we have to crop all the others

letter = VerticalCrop(letter);

//letters.Enqueue(letter);

letters.Enqueue(ConvertToSquare(letter));

}

}

return letters;

}

private Queue<Bitmap> SplitIntoRows(Bitmap image)

{

var rows = new Queue<Bitmap>();

int yLowerBound = 0;

int yUpperBound;

while (yLowerBound < image.Height)

{

//search through image until a row contains a black pixel

//-top of row found

//or we reach the end of the image - && means the second half

//wont be called if the first is false so we can avoid an out of bounds exception

while (yLowerBound < image.Height &&

!DoesRowContainBlackPixel(image, yLowerBound))

{

yLowerBound++;

}

//if the lower bound for a row is at the end of the image, that means there isn't

//another row as we're at the end so we skip the section for the ipperbound and

//splitting the row off

if (yLowerBound < image.Height)

{

//set the upperbound of the current row's y to be the same as the lowerbound

yUpperBound = yLowerBound;

//search through the image until a row doesn't contain a black pixel

//-bottom of row found

//or we reach the end of the image - && means the second half

//wont be called if the first is false so we can avoid an

//out of bounds exception

while (yUpperBound < image.Height &&

DoesRowContainBlackPixel(image, yUpperBound))

{

yUpperBound++;

}

//split the row from the image and equeue it

rows.Enqueue(SplitRow(image, yLowerBound, yUpperBound));

//for the next row the lower bound will be the pixel after the

//currrent upperbound

yLowerBound = yUpperBound + 1;

}

}

return rows;

}

private Bitmap SplitRow(Bitmap image, int yLowerBound, int yUpperBound)

{

//splits a row of letters from a base image using the upper and lower bounds

//of its y value

//instantiate row to be the same width as the original image

//and height of the difference between the bounds

Bitmap row = new Bitmap(image.Width, yUpperBound - yLowerBound);

//int a = image.Height;

//int b = row.Height;

//loop through each pixel in the range

for (int y = yLowerBound; y < yUpperBound; y++)

{

for (int x = 0; x < image.Width; x++)

{

//set the rows' pixel value to image's equivalent

row.SetPixel(x, y - yLowerBound, image.GetPixel(x, y));

}

}

return row;

}

private Bitmap ThresholdImage(Bitmap image, float threshold)

{

//threshold an image to pure black and white

for (int i = 0; i < image.Width; i++)

{

for (int j = 0; j < image.Height; j++)

{

var currentPxl = image.GetPixel(i, j);

//if pixel value is above the threshold, round up to white

//else round down to black

if (currentPxl.GetBrightness() > threshold)

{

image.SetPixel(i, j, Color.White);

}

else

{

image.SetPixel(i, j, Color.Black);

}

}

}

return image;

}

Recursive Algorithms

private bool TryGetAddress(ref string currentAddress, Queue<char> detectedLetters)

{

//This is a recursive function that takes a current address and a queue of letters

//detected by the neural network and tries to assemble an address using them

//if a valid address is found, true is retuned and currentAddress will store the address

//otherwise it returns false and currentAddress can be discarded

//temp is used to store the current address if we need to revert to a previous state

//it's also needed for the LINQ query because that doen't like ref variables

string temp = currentAddress;

//search for all addresses which start with the currentAddress string using LINQ

//make sure they are longer than the address first so we dont get an out of bounds error

var query = from address in addresses

where address.Length >= temp.Length

&& address.Substring(0, temp.Length) == temp

select address;

//if any addresses start with the current address

if (query.Count() > 0)

{

//if all detected letters have been added to the current address - base case

if (detectedLetters.Count() == 0)

{

//if the detected address is in the query, we've found a valid address

//so return true

//else that means that an address starting with what was read by the nn

//is in the query but it continues further. Therefore return false

if (query.Contains(currentAddress)) return true;

return false;

}

//see if adding a space results in a valid address

currentAddress += ' ';

//use a new queue with the same values as detectedLetters so we can create a

//byval copy

//otherwise dequeues in deeper instances would affect this one's queue

if (TryGetAddress(ref currentAddress, new Queue<char>(detectedLetters)))

{

//we've found a valid address so return true - base case

return true;

}

else

{

//reset currentAddress to remove the space just added

currentAddress = temp;

}

//see if adding the next detected letter results in a valid address

currentAddress += detectedLetters.Dequeue();

if (TryGetAddress(ref currentAddress, new Queue<char>(detectedLetters)))

{

//we've found a valid address so return true - base case

return true;

}

}

//if the program gets here then the current address is invalid so return

//false to the previous instance

return false;

}

private Bitmap VerticalCrop(Bitmap letter)

{

//crops the image to remove any white space

//offsets from letter to crop where its just empty space

int top = 0, bottom = 0;

bool edgeFound;

//get offset from top of image

do

{

//start at furthest at the top, move inwards

edgeFound = DoesRowContainBlackPixel(letter, top);

if (!edgeFound) top++;

} while (!edgeFound);

//get offset from bottom of image

do

{

//start at furthest at the bottom, move inwards

edgeFound = DoesRowContainBlackPixel(letter,

letter.Height - bottom - 1);

if (!edgeFound) bottom++;

} while (!edgeFound);

Bitmap croppedImage = new Bitmap(letter.Width,

letter.Height - top - bottom);

//crop imgage with the offsets we've found

for (int i = 0; i < croppedImage.Height; i++)

{

for (int j = 0; j < croppedImage.Width; j++)

{

//start from top away from the top

Color temp = letter.GetPixel(j, i + top);

croppedImage.SetPixel(j, i, temp);

//pcbxImageOutput.Image = croppedImage; //remove me

}

}

return croppedImage;

}

}

}

## Techniques Used

|  |  |  |  |
| --- | --- | --- | --- |
| Technique | Group/  Table | How I’ve used it | Example page numbers |
| Asynchronous programming | *Not on table* | Asynchronous methods have been used to allow continuously running or processor intensive processes like machine learning or image processing be ran in a background thread instead. | 73 |
| Runtime Instantiated Controls | *Not on table* | Numeric Up Down controls are instantiated at runtime so that a user can pick how many layers a neural network needs and then how many nodes each layer needs. | 76 |
| Advanced Matrix Operations | A | In order to calculate the activations of layers deeper in the neural network and for calculating the derivatives for a weight or bias when training a network, operations such as matrix-vector multiplication need to be used regularly. | 62 |
| Complex Mathematics | A | The operation and training of a neural network require complex use of linear algebra and calculus such as differentiating the equations used to calculate a node’s activation to find a weight or bias derivative. | 69 |
| Complex User Defined Algorithms | A | I designed the image processing algorithms used in my solution which are responsible for taking a full colour image of rows of text and converting that to a queue of compressed black and white images of the individual letters. | 85 |
| List Operations | A | I use a list to store the list of valid addresses from the PAF dataset. I then use LINQ in .NET to query through this list. | 83 |
| Queue Operations | A | Queues are used multiple times in the image processing section of this project primarily for storing images of letters in the order that they would be read in the original image. | 83 |
| Recursive Algorithms | A | A recursive algorithm is used to pair a queue of letters detected by the neural network without spaces to an address in the PAF sample I have used. | 87 |
| Multidimensional Arrays | B | Multidimensional arrays are used throughout this solution primarily for storing the values of a neural network such as its activations, biases and weights | 65 |
| Records | B | A record is used to store a piece of training data from the MNIST dataset along with a property showing which letter the data represents. | 71 |
| Simple OOP model | B | The NeuralNetwork class and the NNTrainer class both share a set of protected methods which are inherited from the MatrixFunctions class. | 67 |
| Writing and Reading from Files | B | File IO is used several times throughout this solution such as reading the MNIST dataset or the PAF or when saving/ loading a neural network’s values using JSON files. | 65 |
| Single Dimensional Arrays | C | Single dimensional arrays are commonly used to represent vectors in the neural network section of this solution. | 68 |

## Coding Style

|  |  |  |  |
| --- | --- | --- | --- |
| Technique | Style | How I’ve used it | Example page numbers |
| Modules (subroutines) with appropriate interfaces | Excellent | The parameters needed for a method all have meaningful identifiers | Throughout |
| Cohesive modules (subroutines) – module code does just one thing | Excellent | Every method only performs a single task | Throughout |
| Modules(collections of subroutines) – subroutines with common purpose grouped | Excellent | I have grouped together similar methods with a common purpose in the same .cs files. Some of these .cs files have been further grouped into a separate .dll which can be referenced in other files | 72 |
| Defensive programming | Excellent | The majority of user inputs are handled by numeric up downs which let me prevent a user from typing in non-numeric values or values outside the range I set at instantiation. Some of these values have also been fixed such as a neural network needing to have at least 2 layers with those layers being 728 and 26 nodes long respectively. | 76 |
| Good exception handling | Excellent | Any IO exceptions caused by a user selecting an invalid file to open have been handled with try catch blocks | 72 |
| Good use of local variables | Good | Local variables are used in almost every method I have implemented | Throughout |
| Minimal use of global variables | Good | The only global variables used in my implementation are values that need to be preserved between event calls | Throughout |
| Managed casting of types | Good | Several type casts are used throughout the solution, especially between floats and integers, and characters and strings | 65 |
| Appropriate indentation | Good | My code is correctly indented for every block of code in a different scope | Throughout |
| Self-documenting code | Good | I have fully commented my code with explanations at the start of every method and annotations of the processes occurring within them throughout. | Throughout |
| Meaningful identifier names | Basic | Each variable and method I have used has an identifier which represents its purpose or intended use. | Throughout |

## Interface Screenshots

### Machine Learning

Graphical user interface, application

Description automatically generated

On Load up

Graphical user interface, application

Description automatically generated

After “Generate Neural Network” button pressed

Graphical user interface, application, Word

Description automatically generated

After 4 entered to “number of layers” box and “Generate” button pressed

Graphical user interface, table

Description automatically generated

After 30 and 15 entered to the input boxes and “Set Layer lengths” pressed

Graphical user interface, application

Description automatically generated

Save Neural Network Dialogue

Graphical user interface, application

Description automatically generated

Load Neural Network Dialogue

Graphical user interface

Description automatically generated

After “Start Training” button pressed. The values of training accuracy, time training and samples seen will update continuously. The “Start Training” button will also swap with the “Stop Training” button.

Graphical user interface, application, website

Description automatically generated

After “Try to Recognise new character” button pressed, finding a random MNIST image and letting our neural network attempt to read it. The detected character is then outputted. In this example it was wrong because it had only been trained for 72 seconds.

### Address Reading

Website

Description automatically generated with medium confidence

On load up

Graphical user interface, application

Description automatically generated

Open file dialogue for “Read From Single Image”

Graphical user interface

Description automatically generated with medium confidence

When an image is selected. Because I have not being able to train a neural network to a high enough accuracy, it is unable to recognise the image.

Graphical user interface, application

Description automatically generated

Chose input Folder Dialogue

Graphical user interface, application

Description automatically generated

Chose Output File Dialogue

# Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | Test Description | Test Data | Expected Result | Actual Result |
|  | Test the generation of a neural network using user input in a forms project. The controls in this form are set up so that no fewer than 2 layers can be valid, the input layer must have 28\*28 nodes and the output layer must have 26. This means we don’t have to test for those inputs being invalid. The numeric up downs have also been set to a range that numbers of nodes have to be >1.  Objective 1 | Number of nodes = 4  Hidden layer lengths: 30, 3 | A neural network with layer lengths { 784, 30, 3, 26} will be generated | Same as expected result |
|  | Test saving and loading the neural network by saving the network generated in test 2 as a file and then loading it again both using the windows file dialogue. A message box showing the network’s values can then be displayed when we load it to test if the same network has been saved and loaded.  Objective 2 | A neural network with layer lengths { 784, 30, 3, 26} saved to a drive | A JSON file will be saved to the location saved and a network with the same layerLengths as the test data will be outputted in a message box | Same as expected result |
|  | Test Loading a neural network with an invalid file  Objective 2 | A text file with ‘some text’ saved in it | An error message should be displayed as a message box | Same as expected result |
|  | Test if a neural network can propagate its activations forwards to calculate its output layer’s activations with input activations set on the first layer. As this network has layers with both 1 and more than one node, this test will be enough on its own to test calculating activations  Objective 3 | A neural network with of layerLengths {1,2,1}.  The JSON file for this has been manually altered to have the weights and biases shown in the first image in the proof section | The output layer’s node should have a value of 0.492 (3sf) after I manually calculated that | Same as expected result |
|  | Test if 2 samples can be read from the MNIST dataset with the correct letter label associated with them  Objective 4 | 2 Random positions in the MNIST letters file | An image of a letter and a letter should be outputted. There should both also represent the same letter | Same as expected result |
|  | Test that the accuracy function can correctly find the accuracy of a set of attempts to recognise a character by manually setting the recognised character and real character. This is done in a debug console application  Objective 5 | ‘Recognised letter’ and ‘real letter’:  ‘A’ and ‘A’ 98 times  +  ‘A’ and ‘B’ 2 times | An accuracy of 0.98 should be outputted | Same as expected result |
|  | Check that the program can calculate the derivatives for the output layer of a neural network’s weights and biases  Testing the output gradients – only the values in the last layer for the biases and weights gradients should be accounted for as the rest have not had their original values written over yet.  The gradients will all be outputted to the debug screen using an output method.  Objective 6 | A {2, 2, 3} node nn where the expected values for this sample’s output layer are {1, 0, 0}. | output layer bias gradients should be (3sf): 0.161, 0.278, 0.290  output layer weight gradients should be (3sf):  -0.0909, 0.157, 0.164  -0.123, 0.211, 0.221 | Same as expected result |
|  | Check that the derivatives for every weight in the network can be calculated  Objective 7 | A {2, 2, 3} node nn where the expected values for this sample’s output layer are {1, 0, 0}. | The derivatives for every edge should be outputted to the console | Same as expected result. |
|  | Check that the program can calculate the derivatives for every edge and node in the network and change the network’s value’s in the process  Objective 8 | A neural network of layerLengths {784, 100, 100, 26}  Learning rate 0.1  Batch size 20 | The values in the network’s Json file should change | Same as expected result |
|  | Check that a loaded neural network can be trained using the backpropagation algorithm. As this is being done, the current accuracy, time training and number of samples seen will be outputted to the form.  When a user wants to stop the process, they should press the stop training button  Objective 8 | A neural network of layerLengths {784, 100, 100, 26}  Learning rate 0.1  Batch size 20 | The nn should start training with the timer, accuracy and number of samples seen increasing  When a user presses the stop button, this should halt and the ‘start training’ button should appear | The network started training with the values increasing but it also froze the UI so it was impossible to press the stop button |
|  | retry test 10 after swapping to using asynchronous methods to train the neural network  Objective 8 | A neural network of layerLengths {784, 100, 100, 26}  Learning rate 0.1  Batch size 20 | When a user presses the stop button, the training should halt and the ‘start training’ button should appear | Same as expected result |
|  | Attempt to use the neural network before one has been loaded | N/A | When start training or attempt recognition is pressed, an error message should show. | Same as expected result |
|  | Train a neural network and then attempt to recognise a random MNIST image  Objective 8 | Neural network of layer lengths {784, 3000, 26}  Learning rate: 0.1  Batch size: 20  Train for 10 minutes | The accuracy of the neural network should increase over the 10 minutes and approach around 80-100% | The training accuracy remained quite low. When using this neural network to try and recognise letters, the same letter was read for any image. This may be due to learning rate and batch size settings being too small or large so the gradient can’t be descended properly. The size of the network may also be too small |
|  | Test that an image file can be loaded into the form.  Objective 9 | Photo of a handwritten ‘a’ | Same photo as the input | Same as expected result |
|  | Test that an error message will be shown if a user attempts to open a file that isn’t an image  Objective 9 | An empty text file  Text.txt | “Invalid File” | Same as expected result |
|  | Test that a multicolour image can be thresholded and converted to pure black and white. The letter in this image should still maintain its shape and not be too thick.  Objective 10 | Photo of a handwritten ‘a’ | A black and white photo of the ‘a’ should be outputted to the test form | Same as expected result |
|  | Test that a thresholded Image of letters can be split into multiple images of each row  Objective 11 | A photo of the text:  “Hello  World  a” | Clicking the Split Rows button in the test program should split the image in 3. 1 for each row. Clicking next should then move through each of these. | Same as expected result |
|  | Test that each of the rows from the previous test can then be split into images of individual letters  Objective 11 | A photo of the text:  “Hello  World  a”  This has already been split into rows | Clicking the Split Rows button and then the Split Columns button should output that 11 letters were detected. As we press next, images of the isolated letters should be outputted | Same as expected result however the letters outputted often had large white bars above and/or below them |
|  | Retest splitting the letters out from the rows after changing SplitColumns() to include cropping them and resizing them to a square.  Objective 11 | Same as previous test | The images outputted after clinging split rows and pressing next repeatedly should now be squares without large empty space above and below. | Same as expected result |
|  | Test Compressing an image of a letter into a 28\*28px bitmap  Objective 12 | Image of the letter ‘a’ | The compressed image should be outputted to the screen with a length and width of 28 outputted | Same as expected result |
|  | Test Compressing an image of a letter that has a larger height than width  Objective 12 | Image of the letter ‘e’ written quite tall | The compressed image should be outputted to the screen | The image was compressed but a bottom section of the letter was cropped |
|  | Retest the previous test after changing the compression algorithm to make the original image slightly wider or taller  Objective 12 | Same image of the letter ‘e’ from the previous test | The letter should be compressed with nothing lost and now with a small white outline | Same as expected result |
|  | Check if an input image of rows of text can be: thresholded, split into rows then letters, cropped and then compressed all in 1.  Objectives 10-12 | Image of the text:  “This is  a test” | After pressing process image, the input image should be split into fully processed 28\*28px greyscale images of each letter. These letters should be able to be cycled through using the next button. | Same as expected result |
|  | Test to see that a pure black image can be processed without error  Objective 10 | An image with nothing but black pixels | After Split rows and Split Columns are pressed, it should output that 1 letter was detected as the black will be considered to be 1 “letter” | Same as expected result |
|  | Test to see that a pure white image can be processed without error  Objective 10 | An image with nothing but white pixels | After Split rows and Split Columns are pressed, it should output that no letters were detected because there were no black pixels to indicate a letter | Same as expected result |
|  | Test that a processed image can be inverted so its black pixels become white and vice versa.  Objective 13 | Image of the letter ‘a’ after being thresholded and compressed | The greyscale shades of the input image’s pixels should become inverted to form a white letter on a black background | Same as expected result |
|  | Check that a neural network can attempt a recognition of an input image after that image’s values have been assigned to the input layer without error.  The test program has been set to immediately attempt a recognition of the letter on load and output immediately.  Objective 13 | Image of the letter ‘e’ | The neural network should output an attempt at recognising the ‘e’. The neural network hasn’t been trained to a high accuracy at this point so the letter it predicts will probably not be correct. | Same as expected result |
|  | Check that the PAF sample I have can be read from its file and converted to a string with 1 space separating the elements  Objective 14 | N/A | Each address in the sample should be outputted with commas replaced with spaces. The postcode type and Mailsort SSC should be removed | Same as expected result |
|  | Check that a valid address can have its spaces added back in  Objective 14 | A valid address with all the spaces removed:  “OXLEASOWESFARM  HAYBARNBROCKHIL  LLANETARDEBIGGE  BROMSGROVEB601LU” | “True” as this is a valid address and  the full address with spaces added back in | Same as expected result |
|  | Check that a valid address with similar elements to another address can be recognised  Objective 14 | "LITTLESHORTWOOD  FARMBROCKHILLLAN  ETARDEBIGGEBROSG  ROVEB01LU" | “True” as this is a valid address and the full address with spaces added back in  the full address with spaces added back in | Same as expected result |
|  | Check that an invalid address can be recognised as invalid  Objective 14 | "OXLEASOWES  OTHERFARM  HAYBARNBROCKHILL  LANETARDEBIGGEBR  OMSGROVEB601LU" | “false”  The address outputted can be discarded | Same as expected result |
|  | Check that an empty address can be recognised as invalid without error  Objective 14 | “” | “false” | Same as expected result |
|  | Check that when an image of a written address inputted, the letters in the address can be recognised and recombined into the address before being outputted.  Because I have not gotten an accurate neural network yet, I’m using a base one that recognises any letter as A. I have also manually added “A A” as a valid address in PAF.  Objective 14 | An image of 2 ‘a’s ->  “aa” | “A A” | Same as expected result |
|  | Check that when an image of an invalid address is added, the program will detect this and output a message to tell the user that the input was invalid.  Objective 14 | An image of 1 handwritten ‘a’ | “No address detected” | Same as expected result |
|  | Test that the program can automatically read images from a selected file when an image is moved to it and write the address read in it to a selected text file  Objective 9 + 14 | Input Folder: …\\Test  Output File: Test.txt  An image of 2 ‘a’s ->  “aa” (valid address) | “A A” should be written to test.txt after “Turn on Automatic Reading And Output” is pressed and the image is moved to Test folder | Same as expected result |
|  | Move a file to the test folder after automatic reading has been turned off  Objective 9 +14 | An image of 2 ‘a’s ->  “aa” (valid address) | The “A A” from the previous test should still be there but nothing else should have been appended to the end | Same as expected result |
|  | Test that the program displays an error message if auto read is turned on without the input and output being set | Nothing | “input and output not both set” | Same as expected result |
|  | Try assigning something that is not a text file to the output file | A png | “Please enter a valid text file” | Same as expected result” |

## Test Evidence

|  |  |
| --- | --- |
| Test | Evidence |
|  | After pressing ‘Generate Neural Network’, changing the number of layers to 4 and pressing Generate    After changing the layerLengths and pressing ‘Set Layer Lengths’ |
|  | After Pressing save    After Pressing the Load button and selecting newnn |
|  |  |
|  | A picture containing text, whiteboard  Description automatically generated |
|  |  |
|  |  |
|  |  |
|  |  |
|  | Before training    After Training |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | Text, whiteboard  Description automatically generatedInput image |
|  | Text, whiteboard  Description automatically generated        … |
|  | … |
|  |  |
|  |  |
|  |  |
|  | … |
|  |  |
|  |  |
|  |  |
|  | Screenshot of the test code: |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | Turn on pressed |
|  |  |
|  |  |
|  |  |

# Evaluation

Go through objectives and check if they have been met fully and how effective they are. How could they be improved?

How could the outcome be improved if the problem was revisited in the future, if you had more time…

Independent feedback. Analyse it

## Objective Appraisal

1. Be able to create a base neural network with randomised jagged arrays

This objective has been met as a user can generate a custom neural network with a selected number of layers and nodes on each of those layers. The network is represented with 2D jagged arrays representing the biases and activations in the network and a 3D one for the weights. The values of every one of those array’s cells are also randomised during this process.

1. Allow for a neural network’s attributes to be saved and loaded using json serialisation.

This objective has been met as I have used the system.text.json library in .NET to implement JSON serialisation so that the activations, biases and weights of a neural network can be saved inside a json file. These files can then be converted back into a neural network object by using a flagged constructor to use with the data read in the file

1. Allow the network to attempt to recognise a letter by propagating activations forwards

This objective has been met as I have implemented the equations in the analysis that allow for entire layers of activations to be calculated using matrix multiplication and vector addition. After values have been assigned to the input layer, this is used iteratively to find every activation all the way up to the output layer. The letter detected is then found by finding which output node has the largest activation and matching that with a letter.

1. Be able to read a piece of training data from the binary storage files and run it through the neural network

This objective has been met as I have used BinaryReader objects to read the binary in MNIST files to get a single training sample’s data and the letter it represents. These are then combined in a TrainingSample record

1. Find how accurate the neural network currently is by checking if the largest activation in the output layer corresponds to the actual letter in the image. Average this over multiple tests and divide by the number of tests to find the accuracy. Output this to the GUI regularly as the network is being trained.

This objective has been met as the current accuracy of the neural network is constantly updated after every recognition and is outputted to the user every time the trainer has gone through a full batch of samples.

1. Find how sensitive the cost function is to the output layer’s weights and biases

This objective has been met as the equations used to calculate the costs derivative with respect to the weights or the biases has been implemented

1. Find how sensitive the cost function is to previous layers (backpropagation)

This objective has been met as the equations used to calculate the costs derivative with respect to the weights or the biases in deeper layers has been implemented with the derivatives of later layers used to calculate this

1. Alter the weights and biases to minimise the cost function repeatedly for different sets of test data (gradient decent)

This objective has not been met as I have been unable to train a neural network to a high enough accuracy to be suitable for optical character recognition. However, I have been able to alter the weights and biases of the network repeatedly using the averaged cost derivatives from a full batch of samples.

1. Take a photo of handwritten text in as an input using the form or an input folder. Have the program detect changes in the input folder so new files to read can be inputted.

This objective has been met as images can be inputted to the address reader program either manually using an open file dialogue or automatically using a FileSystemWatcher object to detect if a file has been moved to a selected folder

1. Use image thresholding to create a copy of this bitmap in greyscale

This objective has been met as the program can take a colour image and convert it to black and white by rounding it up or down depending on if each pixel’s brightness is above or below a threshold (0.5)

1. Split the Image into individual letters with inverted shades

This objective has been met as the program can detect where the gaps between rows are by looking for rows of white pixels and splitting across them, these rows are then similarly split into individual letters, each enqueued to a queue in order. Their colours are then inverted later after compression.

1. Compress these bitmaps into 28\*28-pixel bitmaps using lossy compression

This objective has been met as the images of letters from objective 11 are all compressed to 28\*28px by averaging out the brightness values of close together pixels. However, in order to work, some images have to be widened slightly to neatly fit into this aspect ratio. I have also found that the image processing in objectives 10-12 can be quite slow and take up a large amount of time with images in a high resolution.

1. Use the neural network to recognise the characters in each of these bitmaps in the queue and append them to an output list.

This objective has not been met due to objective 8 not being met.

1. Split the queue of letters back into the individual words

This objective has been met as a queue of letters representing an address is able to be split apart with spaces inserted back into it using a recursive validation algorithm.

## Peer Feedback

I asked a peer for feedback after showing them my project and the prototypes for it.

Text, letter

Description automatically generated

Currently this project would be unable to split cursive writing like this up into several images of letters because there are no white gaps between the letters. A future version of this project could implement a more complex character extraction system which accounts for joined up letters and the boundaries of letters overlapping (such as the cross of a t). A possible way to do this would be to look at set areas of the image and check if the density of black pixels is larger than a certain threshold or not. If not this area could be a join between 2 characters so split across this. It could also not split characters in straight lines but use curved splits to avoid cropping off parts of other letters. However, this would greatly increase processing time.

## Further Improvements

Because I have not being able to train a neural network to a high enough accuracy, the main improvement would be to attain this. This could be done by experimenting more with the learning rate, batch size and dimensions of the neural network I am using.

The training algorithm itself may also be able to be improved. For example, instead of using the Sigmoid function to ‘squish’ values to between 0 and 1, I could use the ReLU function instead. This would allow me to effectively deactivate nodes with negative activations instead of just squishing them down to 0. The ReLU function has been found to be much more efficient when training neural networks than sigmoid has.

I could also make my image processing more efficient as currently the program can take several seconds to process images taken in higher resolutions such as from my phone camera. This could be done by compressing the image slightly first so that the later processing has fewer pixels to work with. I could also use multithreading to allow multiple images of the letters to be processed and recognised at once instead of one at a time. This would further improve efficiency.