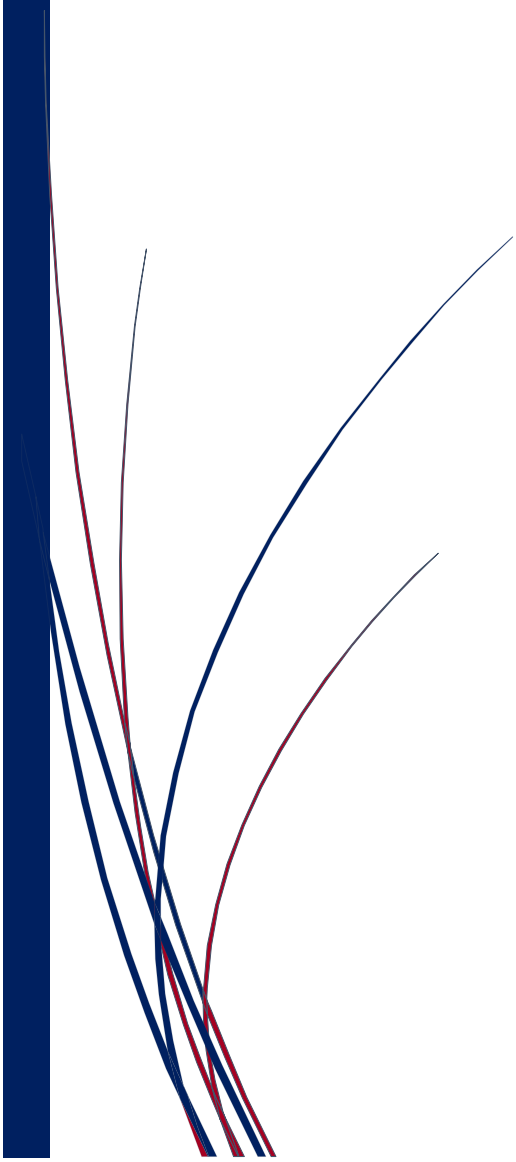
Real Time Programming

Real Time Application

Box Link:
https://kingston.box.com/s/5zd7trjcku6z8a06tdbcux0l2m6em7b6



Gordon Johnson
K1451760

Table of Contents

Resources and Assets

Box Link: https://kingston.box.com/s/5zd7trjcku6z8a06tdbcux0l2m6em7b6

GitHub Link: https://github.com/LordGee/rtp_training/commits/master

Short Video: https://youtu.be/tS5DeIlZoZs

Trained network file that will read lowercase letters a, b and c.

Neural Network Filename: lower15

Introduction

For this application, I intend to develop an application that can predict a hand-written character of the alphabet and display the result. The source of this prediction method will be the developed Neural Network, with Back Propagation for the machine learning aspect.

Aims and Objectives

- Develop a Graphical User Interface (GUI), to provide the user options, workspace and visual results.
- Provide a user preference option, for developing a custom Neural Network, ready to train.
- Provide the user, with a process for loading existing Neural Networks, that have previously been created.
- Provide two operation modes for use of the application, e.g. Training and Analysis.
- Provide a canvas, in which the user can draw a character, which is to be passed to the Neural Network for analysis or training.
- Create a results display, to provide the user with information from the last operation, e.g. the output.

Tasks to be Implemented

To achieve the aims and objectives, the following tasks will need to be implemented:

1. Design and layout, the concept of the GUI
2. Add a panel which will be the target, allowing the user to draw.
3. Save details of the drawing into an array. Convert to a resized image and save as a bitmap file for analysis.
4. Develop the Neural Network and Layers classes, to feed forward the attributes from the pixels within the image, as inputs.
5. Test neural network and observe values, to ensure expected outcomes.
6. Develop an options form, to allow the user to create a customisable Neural Network for use. Information from these selections, will be saved to an initialisation test file, which will be used as a starting point for reloading an existing Neural Network. After which the weights and bias weights, will be saved to independent files in a specific order.
7. Develop the loading process, taking in the attributes of an existing Neural Network and feed back into the arrays the weights and bias weights values for each neuron.
8. Connect the GUI frontend to the neural network backend, to allow a complete user experience.
9. Allow for results to be feedback to the form, to provide operational and output information.
10. Train the network to recognise various characters and allow analysis of them.

Instructions of Use

Starting Layout

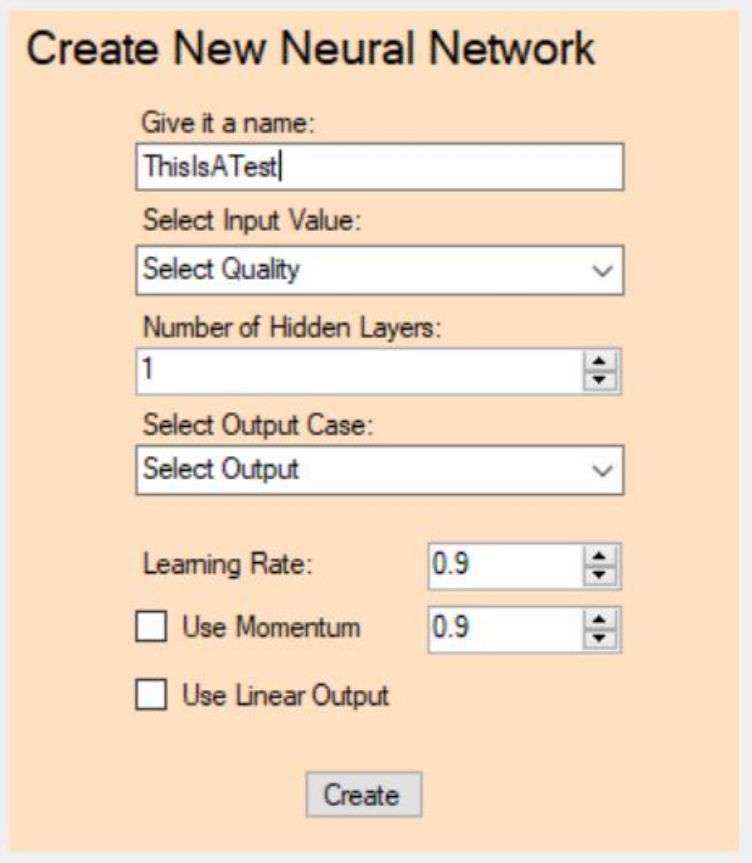
The layout can be broken down into three sections. The left side contains three panels, which are yellow, orange and green. These represent the options for the usage of the application.

The centre section, contains a large black square which is the interaction area. This will be referred to as the blackboard and will be for drawing on. Above the blackboard, is a status indicator, until an active neural network has been loaded or created this will remain red and the blackboard will not be interactable.

The right section, in violet, is the results section. This provides the user with feedback about the networks output result.

Creating a New Neural Network

To create a new neural network, click in the text box below the heading and start by giving it a name. After typing a sufficient name length, the rest of the options for the construction will appear



Note: The name text box is not case sensitive.

The next option, is to choose the input value. This is dependent on the size of the image to converted. Currently it is recommended to select the Low Quality / High Performance option, this is also the default option.

The number of Hidden Layers option, defaults to one. A future edition, will unlock this feature to allow additional hidden layers to be implemented.

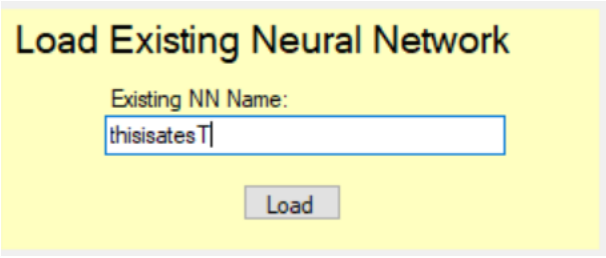
The next option, is to determine the output value of what type of letter case you wish the network to train. The default value is set to lowercase.

The next options, will affect the learning and outputs results of the network. There are performance hinderances / improvements, that can also be seen from manipulating these values.

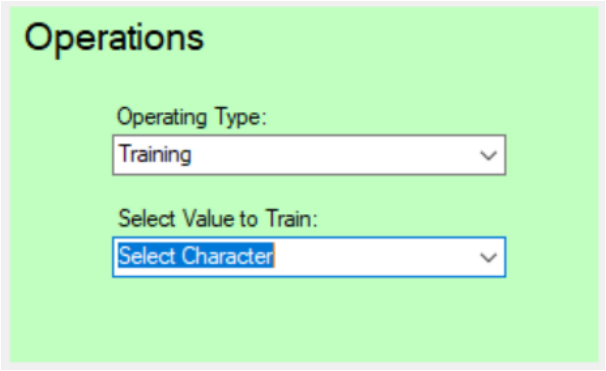
If you click the create button without modifying any values, then the previously mentioned default values will apply. Once created, the status information will update to ready and the blackboard will unlock, to allow interaction with it.

Loading an Existing Neural Network

To load in a neural network that has previously been created, select the text box under the relevant heading and type the name of the desired neural network. Again, the text box is not case sensitive, as seen in this example.



Training the Neural Network



Once a Neural Network has successfully been loaded or created, the final option panel will present a drop-down box, to allow you to define what mode you are going to be using. This will be either Analysis or Training. If the Training option is selected, an additional option will appear. It is recommended that you train your neural network before trying to analyse it. With training selected, in the second option box, chose a letter to start training. Once this is selected, you should then proceed to draw the letter on the blackboard. Utilising the blackboard space is recommended.

During the learning process, the status information will be displayed accordingly. Once completed, this will change back to a green ready status.

To fully train the network, this process will need to be repeated multiple times for each letter. Like any learning process, practice makes perfect.

It is recommended to test each letter once, then move onto the next letter and so on, then repeating the cycle. This seems to work best, as if the same letter is practised multiple times in a row, connection with previously trained letters appears to be lost. Once a few letters are trained, the programme is ready to be tested in the analysis mode. After inputting a letter, the output result should now display the character that was written, in the results section. If the displayed result, does not match the character written, then this letter will need further training.

Techniques and Algorithms

CLR Project

The CLR project type, allows development of a mixed assembly. This allows the developer to access features from the .NET framework, making it easier to implement a project with a graphical user interface (GUI). This requires, a mixture of managed and unmanaged code to be implemented and is referred to as C++/CLI. The managed code section, is similar to C# with some minor variations, such as the use of handles, pointers and references. This means, that any managed code can make use of the .NET Memory Management and Garbage Collectors natively. (Microsoft, 2016)

Using this project type to develop a GUI interface with the .NET toolbox, allows the developer to choose from a selection of available tools, such as, textboxes, buttons, menu strips and many more. This method, enables the quick design and implementation of the layout of a form application. Each tool used, can have many events associated to it, such as when a mouse button is clicked or a value in the object changes.

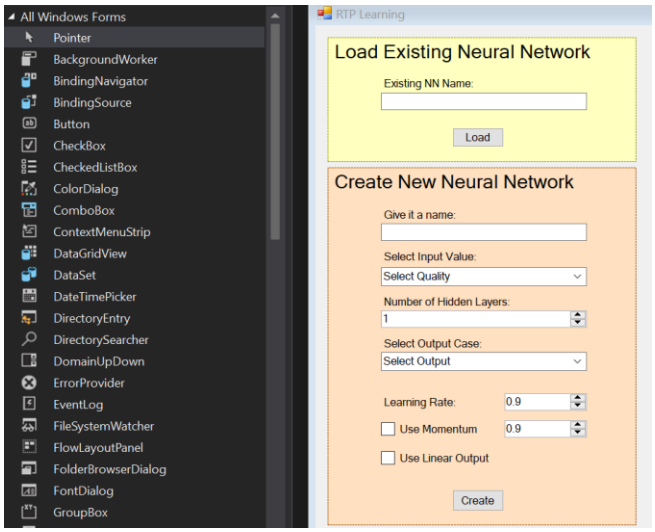


Figure 1 - Example of tools available from the .NET Framework

Another benefit to using the C++/CLI process, is having the mixture of using the .NET language in conjunction with some of the C++ STL features, such as vector arrays.

However, there are drawbacks to using C++/CLI, such as threading and mutex not being supported and cannot be mixed with managed code. Also, upon further investigation I found that there are many comments indicating that having a mixture of managed and unmanaged code can hinder performance significantly. Finally, the CLR project type, is no longer supported by Visual Studio 2015, the recommended process by (Microsoft, 2016) is to port to a C# application.

Neural Network

The concept for this Neural network, is to develop a process that will allow the pixels from an image to populate the input layer of the network. From this information, the weights connecting each neuron, will need to determine patterns within the pixel values, to identify the correct letter that has been written. For a 15x15 pixel image (figure 2), the input layer would require 225 neurons. The output layer, will contain 26 or 52 output neurons, depending on the type of letter case chosen for this network.

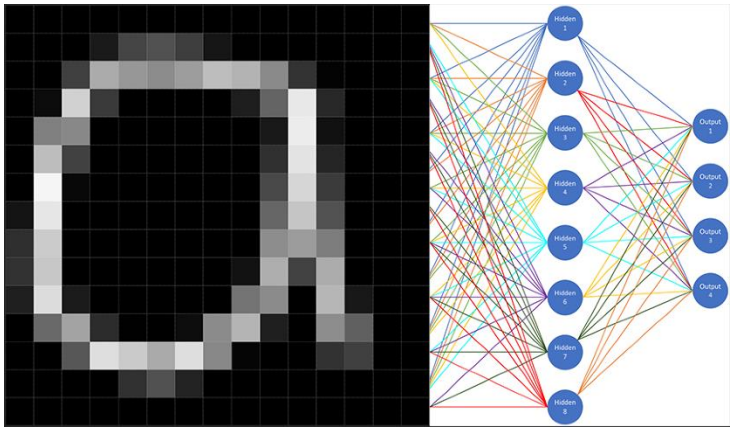


Figure 2 - 15x15 pixel image

The hidden layer, will present a calculated number of neurons, depending on the input and output amounts. This value is determined by the average amount of neurons, within the range of the input and output amounts.

The feedforward process moves through the network in a forward direction, calculating each neuron value along the way. In this case, the input values for each neuron are, a white pixel represents 1 and a black pixel represents 0. These values are calculated against each weight value that connects two neurons of the current and adjacent layer. This calculation takes the neuron value and multiplies it by the corresponding weights. The destination neuron accumulates this value, until all the connecting weights have been calculated. The neuron value, is then calculated through a sigmoid function, which dictates the final neuron value.

With these new values in the next layer, the process is repeated until the neurons within the output layer have all been calculated. The neuron with the highest value within the output layer, is the dominant output, which will get returned as the final result.

Unlike a previously constructed Neural Network, this one will enforce a back propagated learning process. During the training process, this method allows for the correct or desired output value to be submitted. The learning process will then back propagate, through the layers to calculate the error value, which in turn will be used to manipulate the weight values connecting the layers. Each iteration, will then improve the chance of the desired output value being reached. This process of error calculation starts with the output and then the hidden layer. From this information, the weights are altered to reflect these errors within the hidden and input layers. Once completed, the process is tested again, until the error rate is within the range of a predetermined tolerance level.

Specifications of Code

The following, presents a complete class diagram of the application and class descriptions, based upon the analysis and training process.

Class Diagram

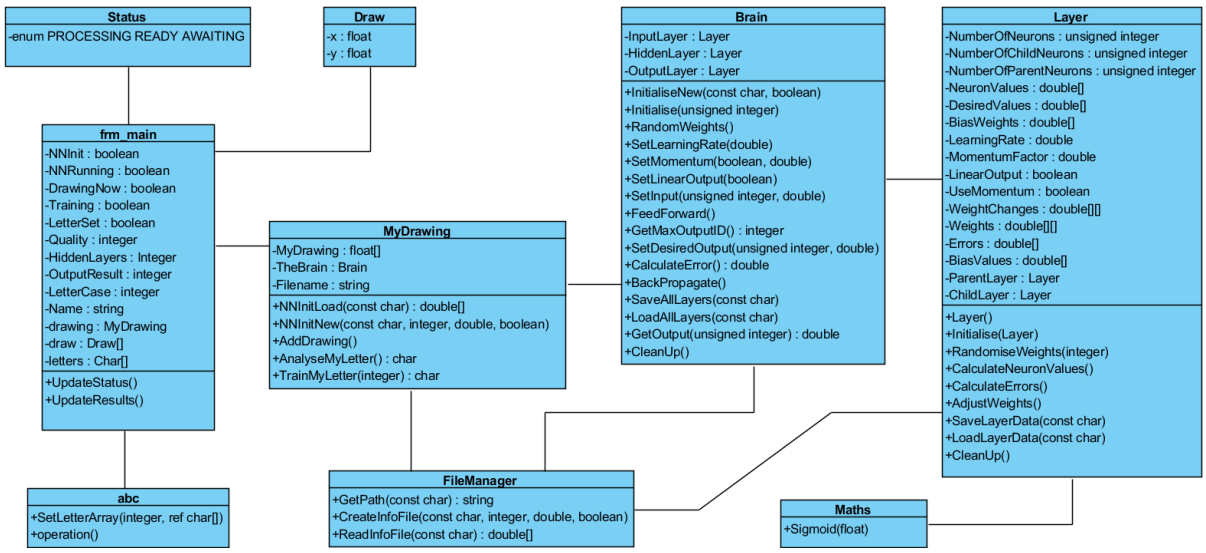


Figure 3 - Class diagram of the application

Form Main Class

This class, holds the starting location for the entire application. Upon launch, the main method is executed and initialises the application GUI form. This class, is a managed class and so the code inside is a .NET variation, to interact with the respective namespaces.

The main interaction in this class, revolves around the drawing functionality. The panel used to draw upon, has four event types associated to it. These events include Paint, Mouse Down, Mouse Move and Mouse Up.

Mouse Button Down Function

```
System::Void rtp1::frm_main::pnl_GameCanvas_MouseDown(System::Object^ sender,
System::Windows::Forms::MouseEventArgs^ e)
{
	if (m_NNRunning) {
		if (e->Button == System::Windows::Forms::MouseButtons::Right) {
			if (draw->size() > 0) {
				draw->pop_back();
			}
		} else if (e->Button == System::Windows::Forms::MouseButtons::Left) {
			m_DrawingNow = true;
		}
	}
}
```

Only if a Neural Network is active, does this function apply. If the left mouse button is pressed and held, it switches the Boolean value to indicate that the user is now drawing. This Boolean value is checked in the all four event functions related to this panel. If the right mouse button is clicked and if there is already a value in the draw array, then the items from the draw array will be removed one by one, starting from the last entry.

Mouse Move Function

```
System::Void rtp1::frm_main::pnl_GameCanvas_MouseMove(System::Object^ sender,
System::Windows::Forms::MouseEventArgs^ e)
{
	if (m_DrawingNow) {
		if (e->Button == System::Windows::Forms::MouseButtons::Left) {
			Draw dr;
			dr.x = (float)e->X;
			dr.y = (float)e->Y;
			draw->push_back(dr);
		}
	}
}
```

The mouse move function, performs a check that the user has already indicated that they are now drawing and ensures that the left mouse button is still active. Each time the event is called, it logs the x and y coordinates within the panel and stores each one in a vector array of type Draw. Draw, is a small structure containing these two values.

Mouse Button Up Function

```
System::Void rtp1::frm_main::pnl_GameCanvas_MouseUp(System::Object^ sender, Sys-
tem::Windows::Forms::MouseEventArgs^ e)
{
	if (m_NNRunning && m_DrawingNow) {
		UpdateStatus(Status::PROCESSING);
		m_DrawingNow = false;
		pnl_GameCanvas->Refresh();

		Bitmap ^bmp = gcnew Bitmap(pnl_GameCanvas->ClientSize.Width,
			pnl_GameCanvas->ClientSize.Height);
		pnl_GameCanvas->DrawToBitmap(bmp, pnl_GameCanvas->ClientRectangle);

		Bitmap ^bmp2 = gcnew Bitmap(
			(int)(m_Quality),
			(int)(m_Quality),
			PixelFormat::Format24bppRgb );
		Graphics ^g = Graphics::FromImage(bmp2);
		g->InterpolationMode = InterpolationMode::HighQualityBicubic;
		g->DrawImage(bmp, 0, 0, bmp2->Width, bmp2->Height);
		bmp2->Save(IMG_LOCATION);

		ProcessDrawing();

		delete bmp;
		delete bmp2;
	}
}
```

The mouse up function, completes the process by refreshing the draw canvas, this calls the paint function to display the visual drawing back to the user. The mouse up function, continues to process the image by converting the drawn image to a bitmap data type. After some investigation (PC Review, 2005), a method to resize the bitmap file was included, taking the image down to a user defined quality level, e.g. 15 x 15 pixels. The bitmap data type, is then saved to a file for later use and then the Process Drawing method is called. Finally, both bitmap files are deleted. Despite both bitmap files, being created with a garbage collector associated to them, they still need to be deleted manually, to enable the application to run correctly.

Paint Function

```
System::Void rtp1::frm_main::pnl_GameCanvas_Paint(System::Object^ sender,
System::Windows::Forms::PaintEventArgs^ e)
{
	SolidBrush^ brush = gcnew SolidBrush(Color::White);
	for each (Draw dr in *draw) {
		e->Graphics->FillEllipse(brush, (int)dr.x, (int)dr.y, 20, 20);
	}
}
```

When the panel is forced to refresh, this function is activated as an event. An appropriate brush is allocated and each pixel, within the draw array, is drawn to the panel.

Process Drawing Function

```
void rtp1::frm_main::ProcessDrawing()
{
	d.AddMyDrawing();
	if (!m_Training) {
		m_OutputResult = d.AnalyseMyLetter();
	} else {
		m_OutputResult = d.TrainMyLetter(cbx_TrainingValue->SelectedIndex);
	}
	UpdateResults(m_OutputResult);
}
```

This function, calls methods within the My-Drawing class. Initially, setting up the image by calling the method Add My Drawing which configures the image, based on the chosen operation type. It then calls the appropriate method, which will return an output result, this will come from the Neural Network. The output is then updated and displayed to the results panel.

My Drawing Class

Add My Drawing Function

```
void rtp1::MyDrawing::AddMyDrawing()
{
	Bitmap bMap(IMG_LOCATION);
	for (int y = 0; y < bMap.Height; y++) {
		for (int x = 0; x < bMap.Width; x++) {
			float colour = bMap.GetPixel(x, y).R +
				bMap.GetPixel(x, y).G + bMap.GetPixel(x, y).B;
			if (colour > 0) {
				m_MyDrawing.push_back(colour / MAX_ACCUM_COLOUR_VALUE);
			} else {
				m_MyDrawing.push_back(0);
			}
		}
	}
}
```

This function, loads the bitmap image that was previously saved, a double loop iterates through every pixel. Each pixel has an RGB value, to obtain a consistent value of between 0.00 to 1.00, the three values are added together and divided by the total possible accumulated number, e.g. 765, we receive the required value. This value is then put into a flat array of type double, which will be used to feed the input layer of the Neural Network.

Analyse My Letter Function

```
char rtp1::MyDrawing::AnalyseMyLetter()
{
	int numInputs = m_MyDrawing.size();
	for (int i = 0; i < numInputs; i++) {
		m_TheBrain.SetInput(i, m_MyDrawing[i]);
	}
	m_TheBrain.FeedForward();
	int outputValue = m_TheBrain.GetMaxOutputID();
	return outputValue;
}
```

This function, iterates through each pixel and sets the value as the input to each neuron within the input layer of the Neural Network. Once processed, it calls the Feed Forward method. This performs the activation calculation for each value against the weights and populates the output layer. The result, is then returned to the main class for updating the user.

Training My Letter Function

```
char rtp1::MyDrawing::TrainMyLetter(int _c)
{
	int numInputs = m_MyDrawing.size();
	double error = 1;
	int count = 0;
	while (error > 0.0001 && count < 50000) {
		error = 0;
		count++;
		for (int i = 0; i < numInputs; i++) {
			m_TheBrain.SetInput(i, m_MyDrawing[i]);
		}
		for (int i = 0; i < 26; i++) {
			if (i == _c) {
				m_TheBrain.SetDesiredOutput(i, 1);
			} else {
				m_TheBrain.SetDesiredOutput(i, 0);
			}
		}
		m_TheBrain.FeedForward();
		error += m_TheBrain.CalculateError();
		m_TheBrain.BackPropagate();
		// error = error / numInputs;
	}
	m_TheBrain.SaveAllLayers(m_Filename);
	return m_TheBrain.GetMaxOutputID();
}
```

While this function has similar characteristics to the Analise My Letter function, there is considerably more processing involved here. The process stays within a While loop, until a desired error tolerance is obtained. While in the loop, the inputs are set within the neural network and the desired outputs are defined. Once set up, the Feed Forward process is used. This now puts the configured neural network, in a position that can be assessed. This is done by calling the Calculate Errors method, which determines the current error tolerance level. Based on this information, the Back-Propagation method is called. Which iterates through the network, to refine the weight values that will improve the probability that the next attempt, will be more successful in obtaining the desired output. Finally, once the loop conditions have been met, the new changes are then saved to file and the final output result is returned.

Note: Upon writing this, I noticed an error in the second loop, where the number of iterations is a constant value. Meaning, this loop will only work for lower or upper-case training and not mixed case.

Brain Class

The following methods, are called from both the Analyse and Train my Letter Functions. Each one, accesses the Layer Class to set, update or get the necessary values.

```
void Brain::SetInput(unsigned int _index, double _value)
{
	if (_index >= 0 && _index < m_InputLayer.NumberOfNeurons) {
		m_InputLayer.NeuronValues[_index] = _value;
	}
}

void Brain::FeedForward()
{
	m_InputLayer.CalculateNeuronValues();
	m_HiddenLayers.CalculateNeuronValues();
	m_OutputLayer.CalculateNeuronValues();
}

int Brain::GetMaxOutputID()
{
	double maxValue = m_OutputLayer.NeuronValues[0];
	unsigned int id = 0;
	for (int i = 0; i < m_OutputLayer.NumberOfNeurons; i++) {
		if (m_OutputLayer.NeuronValues[i] > maxValue) {
			maxValue = m_OutputLayer.NeuronValues[i];
			id = i;
		}
	}
	return id;
}
```

The following methods, are called only from the Train My Letter Function.

```
void Brain::SetDesiredOutput(unsigned int _index, double _value)
{
	if (_index >= 0 && _index < m_OutputLayer.NumberOfNeurons) {
		m_OutputLayer.DesiredValues[_index] = _value;
	}
}

double Brain::CalculateError()
{
	double error = 0;
	for (int i = 0; i < m_OutputLayer.NumberOfNeurons; i++) {
		error += std::pow(m_OutputLayer.NeuronValues[i] -
			m_OutputLayer.DesiredValues[i], 2);
	}
	error = error / m_OutputLayer.NumberOfNeurons;
	return error;
}

void Brain::BackPropagate()
{
	m_OutputLayer.CalculateErrors();
	m_HiddenLayers.CalculateErrors();
	m_HiddenLayers.AdjustWeights();
	m_InputLayer.AdjustWeights();
}

void Brain::SaveAllLayers(const char* name)
{
	m_InputLayer.SaveLayerData(name, "input");
	m_HiddenLayers.SaveLayerData(name, "hidden");
	m_OutputLayer.SaveLayerData(name, "output");
}
```

Note: The error calculation, could be moved to the My Maths Class, as a central place for all the math related functions. This will not affect the outcome but will mean the code is more organised.

Layer Class

Calculate Neuron Values Function

```
void Layer::CalculateNeuronValues()
{
	if (m_ParentLayer != NULL) {
		for (int i = 0; i < NumberOfNeurons; i++) {
			double activation = 0;
			for (int j = 0; j < NumberOfParentNeurons; j++) {
				activation += m_ParentLayer->NeuronValues[j] *
					m_ParentLayer->m_Weights[j][i];
			}
			activation += m_ParentLayer->m_BiasValues[i] *
				m_ParentLayer->BiasWeights[i];
			if (m_ChildLayer == NULL && LinearOutput) {
				NeuronValues[i] = activation;
			} else {
				NeuronValues[i] = Sigmoid(activation, 1.0f);
			}
		}
	}
}
```

Providing that the layer being processed has a parent object, then a double for loop iterates through all the neurons in the current layer, as well as all the neurons in the parent layer. An activation variable, accumulates all the parent's current neuron values, multiplied by the parent's weights as well as the bias values being multiplied by the bias weights. The final value of the activation variable, is fed into a sigmoid function, that returns a result for the current layers individual neuron value. This process continues, until the current layer has all of its neuron values populated.

Calculate Errors Function

```
void Layer::CalculateErrors()
{
	if (m_ChildLayer == NULL) {
		for (int i = 0; i < NumberOfNeurons; i++) {
			m_Errors[i] = (DesiredValues[i] - NeuronValues[i]) *
				NeuronValues[i] * (1.0f - NeuronValues[i]);
		}
	} else if (m_ParentLayer == NULL) {
		for (int i = 0; i < NumberOfNeurons; i++) {
			m_Errors[i] = 0.0f;
		}
	} else {
		for (int i = 0; i < NumberOfNeurons; i++) {
			double sum = 0;
			for (int j = 0; j < NumberOfChildNeurons; j++) {
				sum += m_ChildLayer->m_Errors[j] * m_Weights[i][j];
			}
			m_Errors[i] = sum * NeuronValues[i] * (1.0f - NeuronValues[i]);
		}
	}
}
```

The process of calculating the errors, works backwards through the neural network. Starting with the output layer and proceeding to the hidden layers. This is because the error is calculated on the final output from the feed forward process, against what the desired value is. In the hidden layers, it takes the error value from the output layer and multiplies it by the hidden layers current weight value. This value is then calculated against the current neuron value, to produce a final error amount. Which is used in the next function, to manipulate the weights connecting the neurons.

Adjust Weights

```
void Layer::AdjustWeights()
{
	if (m_ChildLayer != NULL) {
		for (int i = 0; i < NumberOfNeurons; i++) {
			for (int j = 0; j < NumberOfChildNeurons; j++) {
				double deltaWeights = LearningRate *
					m_ChildLayer->m_Errors[j] * NeuronValues[i];
				if (UseMomentum) {
					m_Weights[i][j] += deltaWeights + MomentumFactor *
						m_WeightChanges[i][j];
					m_WeightChanges[i][j] = deltaWeights;
				} else {
					m_Weights[i][j] += deltaWeights;
				}
			}
		}
		for (int i = 0; i < NumberOfChildNeurons; i++) {
			BiasWeights[i] += LearningRate * m_ChildLayer->m_Errors[i] *
				m_BiasValues[i];
		}
	}
}
```

Adjusting the weights for each neuron, is a key function within the learning process. Each weight in the hidden and input layers, are iterated through respectively. Using the error values calculated in the last function, whilst considering the learning rate and if the momentum factor is in use, will perform a calculation and manipulate the weights closer to the desired outcome. This process can take many iterations, but with each iteration, the weights become more robust to lead the network to the desired output.

Performance and Optimisation

One of the desirable performance improving techniques to implement into the project, was to allocate sections of code to utilise multiple threads. However, due to the limitations as mentioned earlier in this report, threading was not possible without a major re-write of the code. If this had been possible to implement within the time frame, the main sections of code that would have been targeted for threading would be:

- Calculate Neuron Values
- Calculate Errors
- Adjust Weights

The unmanaged C++ style threading implementation, crashed the application. While the .NET method, would have involved many of the major classes to be written in the .NET language, which in turn could have impacted the memory allocations for the Layers class.

Throughout the implementation, effective use of the debugger has been utilised, reviewing values and ensuring that the desired outcomes are being achieved. Testing, included Valid and Invalid testing, for example trying to load a file that does not exist. This threw an exception, which by adding an additional check statement, is no longer the case.

A more recent improvement to be made, was verifying the pixels RGB values. Due to the image being resized to 15x15 pixels, the white value became greyer, so instead of receiving a 1, the result was 0.3 at the very most. The following modification was implemented:

```
float colour = bMap.GetPixel(x, y).R + bMap.GetPixel(x, y).G + bMap.GetPixel(x, y).B;
if (colour > 255) {
	m_MyDrawing.push_back(1);
	//m_MyDrawing.push_back(colour / MAX_ACCUM_COLOUR_VALUE);
} else {
	m_MyDrawing.push_back(0);
}
```

This ensures, that the input will always be a 1 or a 0 regardless of the grey scale. If the accumulated RGB value result is at least a third of the total possible value, it will consider that pixel white.

As seen in the class diagram earlier in this report (figure 3), a file was added called File Manager. The functions within this, were re-used across three individual classes, for the purpose of loading and saving files. Because the File Manager, had its own namespace which was declared as being used in the other classes, the syntax to call the methods was simply to type the function name and present the required parameters.

References

Microsoft (2016) *.NET Programming with C++/CLI (Visual C++).* Available at: https://docs.microsoft.com/en-gb/cpp/dotnet/dotnet-programming-with-cpp-cli-visual-cpp (Accessed 04 March 2018)

PC Review (2005) *How do you resize an image in managed C++ .net using GDI or GDI?* Available at: https://www.pcreview.co.uk/threads/how-do-you-resize-an-image-in-managed-c-net-using-gdi-or-gdi.2286087/ (Accessed: 21 February 2018)

Stack Overflow (2009) *How do I convert a System::String^ to const char*?* Available at: https://stackoverflow.com/questions/1098431/how-do-i-convert-a-systemstring-to-const-char (Accessed: 28 February 2018)