Introduction

Following on from Assignment one, this report will showcase the implementation of the stages discussed previously, the subsequent sections discuss evaluating the approach to the problem, discussing the process of creating & implementing the neural network step by step, and finally concluding with professional issues associated with the project including a critical reflection and a conclusion.

The Design of The Network

Following on from the discussed design stages of assignment one, it was necessary to understand the mathematics behind a neural network so as to create one. For this, a deep dive into the maths behind the functionality of the network was needed.

An easy way to express the function of a neural network is through one simple equation:

$$n_{x_y} = \sigma\left(\sum_{i=0}^{\text{len}(n_{x-1})} n_{x-1_i} w_{x_{y_i}} + b_{x_y}\right)$$

Nxy represents the sum of all neurons in the previous layer with each neuron having its activation multiplied by the strength of the connection, also known as weight, at the position of Xyi of neuron i

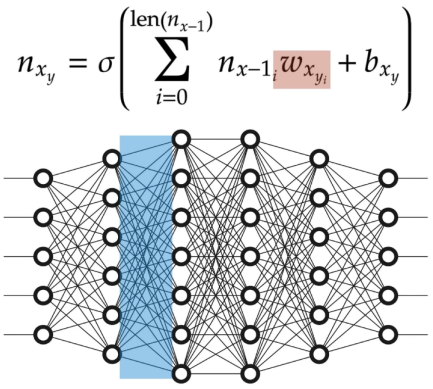


Figure 1: Explanation of Neural Network Against Weights

After that the sum of all neurons multiplied by their weights, the bias is then added, represented as bxy for the specific neuron that is needed. Finally, the value is then wrapped in an activation function that in this specific case can be represented as a sigmoid:

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

For the design of this network, there are a few different points to note. The neural network that shall be created for this project will focus on forward propagation, meaning that data is fed into the input layer, passed on through the hidden layers and then an output is generated from the output layer.

This can be represented as the input($a^{[0]}$) layer is broken down into 784 separate units (as defined by our data-sets 28x28 pixel size for each image), a hidden layer ($a^{(1)}$) that has 10 units with a standard ReLU activation, and then finally an output layer ($a^{(2)}$) that will have 10 units corresponding to the ten-digit classes with softmax activation. The inclusion of an activation function is needed, if there was no activation function, each node would just be a linear combination of the previous nodes and their linked biases. Without an activation function, the model becomes a more complex linear regression model, whereas an activation function decides whether a neuron should be activated or not, therefore implementing the "intelligence" into the network.

The implementation summary of the Neural Network

As mentioned in assignment one, google collab will be used primarily for the creation of the network due to hardware requirements needed to train the model, as well as having the latest packages installed on a server-side client.

The first step of the creation of the database is to import the necessary packages that will be used:

Imports

```
[3]  import numpy as np
     import matplotlib.pyplot as plt
     %matplotlib inline
     import keras
     from keras.models import Sequential
     from keras.layers import Dense, Dropout
     from sklearn.metrics import confusion_matrix
     from keras.utils import np_utils
     import seaborn as sns

     np.random.seed(0)
```

Data

```
[2]  from keras.datasets import mnist
     (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

```
[4]  print(x_train.shape, y_train.shape)
     print(x_test.shape, y_test.shape)
```

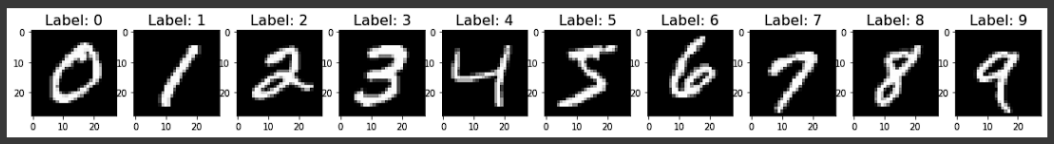
```
(60000, 28, 28) (60000,)
(10000, 28, 28) (10000,)
```

The print statement at the bottom showcases that the dataset has been imported, as well as showcasing the number of characters within the dataset and how they are represented in a 28x28 pixel image.

Now that the dataset is imported it's best to make sure that it can be represented visually:

```
[5]  num_classes = 10
     f, ax = plt.subplots(1, num_classes, figsize=(20,20))

     for i in range(0, num_classes):
       sample = x_train[y_train == i][0]
       ax[i].imshow(sample, cmap='gray')
       ax[i].set_title("Label: {}".format(i), fontsize=16)
```



This code shows how the sample data is presented in a grayscale format and how the images are labelled. Since the MNIST data set is already balanced, no further classification balancing is required before we can use the data.

Showcasing the first 10 entries into the database:

```
[6]  for i in range(10):
       print(y_train[i])
```

```
5
0
4
1
9
2
1
3
1
4
```

Above shows the labels as numbers. However, as their values are continuous, these numbers cannot be directly fed into the network. Therefore, classification is necessary. As an example, if an image looks somewhat like a 4 or a 5, the results would fall into the range of either 4.0 or 5.0, whereas the desired output should be a whole number, not an estimate.

```
[7]  y_train = keras.utils.np_utils.to_categorical(y_train, num_classes)
     y_test = keras.utils.np_utils.to_categorical(y_test, num_classes)

[8]  for i in range(10):
       print(y_train[i])
```

```
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```

The code above showcases how the data above can be translated into a vector that is as long as the number within the range, this is known as “one-hot encoding”. The vectors are filled with 0 values aside from the placement of where the number is, meaning that within the vector, each 1 represents the placement of a number. The value of 5, will be shown at the 5th index of the vector, referring to

the earlier code that showcases the first value being a 5. By doing so, a neural network can learn to predict the one exact value of a class for each number. Thus, it will know that the index value 5 corresponds to the number 5.

Once a method of feeding the data into the model has been established, the next step would be to prepare the entire dataset through normalisation so that it can be fed into the network as soon as possible. For this example, the data will be kept within a certain range so that when the training is needed, it is much easier on the network from a computational standpoint.

In the use case of this project, the data is represented in RGB grayscale images containing values of black (represented by the value 0) and white (represented by the value 255) division is needed to keep the range within the classification limits, those being zero and one.

```
[9]  #Normalise The Data
     x_train = x_train / 255.0
     x_test = x_test / 255.0

[10] #Reshape The Data
     x_train = x_train.reshape(x_train.shape[0], -1)
     x_test = x_test.reshape(x_test.shape[0], -1)
     print(x_train.shape)

     (60000, 784)
```

This process transforms the input matrix from a 28 by 28 array into a single long vector that can be passed on to the network, creating a two-dimensional input vector that can handle a large amount of data at once. Instead of having a matrix for each image being a 28 by 28 vector, now one long vector containing the sum of the image vectors is used as the input.

Now that the data has been normalised into a sufficient input format, the next step is to create the neural network:

```
[11] from sklearn import metrics
     model = Sequential()  # sequental model enables layer addition to network

     model.add(Dense(units=128, input_shape=(784,), activation='relu')) # relu activation is used to solove the non linear equations on the layer
     model.add(Dense(units=128, activation='relu'))
     model.add(Dropout(0.25)) # meaning 25% of the neurons will be deactivated during trainging.
     model.add(Dense(units=10, activation='softmax')) # assignes a probability to each class for accuracy
     model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

     model.summary()

     Model: "sequential"
     _________________________________________________________________
      Layer (type)                Output Shape              Param #
     =================================================================
      dense (Dense)               (None, 128)               100480

      dense_1 (Dense)             (None, 128)               16512

      dropout (Dropout)           (None, 128)               0

      dense_2 (Dense)             (None, 10)                1290

     =================================================================
     Total params: 118,282
     Trainable params: 118,282
     Non-trainable params: 0
     _________________________________________________________________
```

The network uses a sequential model to enable the creation of additional layers if needed, it also corresponds to an equal amount of input and output layers. Relu activation is used to solve the nonlinear equations on the layer, as well as to convert the weighted summed input to the activation of the node, and softmax is used for our probability distribution function. Adam is used for the optimisation algorithm because it is adaptable to learning rates.

For the network categorical cross-entropy is used for the loss function on the network, it is heavily documented and often used when dealing with a network that handles multiple classes, like the data-set used here. The optimiser is Adam, Adam was chosen as its algorithm efficiency is very effective on our data set due to its adaptability and optimisation when it comes to handling sparse data. Running the model reveals that there are over 100,000 parameters and that they can all be used to train the model.

```
[12] batch_size = 512
     epochs=10
     model.fit(x=x_train, y=y_train, batch_size=batch_size, epochs=epochs)

     Epoch 1/10
     118/118 [==============================] - 2s 11ms/step - loss: 0.6141 - accuracy: 0.8246
     Epoch 2/10
     118/118 [==============================] - 1s 11ms/step - loss: 0.2288 - accuracy: 0.9322
     Epoch 3/10
     118/118 [==============================] - 1s 10ms/step - loss: 0.1683 - accuracy: 0.9502
     Epoch 4/10
     118/118 [==============================] - 1s 12ms/step - loss: 0.1338 - accuracy: 0.9609
     Epoch 5/10
     118/118 [==============================] - 1s 11ms/step - loss: 0.1136 - accuracy: 0.9667
     Epoch 6/10
     118/118 [==============================] - 1s 11ms/step - loss: 0.0963 - accuracy: 0.9715
     Epoch 7/10
     118/118 [==============================] - 1s 11ms/step - loss: 0.0822 - accuracy: 0.9756
     Epoch 8/10
     118/118 [==============================] - 1s 10ms/step - loss: 0.0731 - accuracy: 0.9780
     Epoch 9/10
     118/118 [==============================] - 1s 11ms/step - loss: 0.0635 - accuracy: 0.9802
     Epoch 10/10
     118/118 [==============================] - 1s 12ms/step - loss: 0.0556 - accuracy: 0.9837
     <keras.callbacks.History at 0x7efbac387150>
```

The batch size determines how many images will be imported into the model at once. The epoch represents the cycle of data that passes through the network. Setting the epoch to 10 is a good starting point for demonstrating the functionality of the network, but it can be increased if needed.

Both a decrease in loss and an increase in accuracy indicates the network is functioning well. Applying this methodology to a larger epoch should result in the loss continuing to decrease and the accuracy increasing.

Now that the network is shown to be functional, implementation of evaluations tools can be added to showcase a summary of the efficiency.

```
[ ]  test_loss, test_acc = model.evaluate(x_test, y_test)
     print("Test Loss {}, Test Accuracy: {}".format(test_loss, test_acc))

     313/313 [==============================] - 1s 2ms/step - loss: 0.0759 - accuracy: 0.9770
     Test Loss 0.07593091577291489, Test Accuracy: 0.9769999980926514
```

The following evaluation features the overall test accuracy and loss of the model on the dataset. A better understanding of the efficiency of the model is primarily based on the accuracy metric.

The next step will be how the model can be applied for classification prediction on some input images.

```
[ ]  y_pred = model.predict(x_test) # create an array with all the predictions from test data
     y_pred_classes = np.argmax(y_pred, axis=1)
     print(y_pred)
     print(y_pred_classes)
```

```
[[1.1027189e-06 6.7601018e-06 1.9635148e-05 ... 9.9967360e-01
  2.5993643e-07 6.5436012e-05]
 [9.2782990e-08 7.2496786e-04 9.9917465e-01 ... 9.9450483e-09
  9.7519592e-07 4.5953494e-09]
 [4.9709474e-06 9.9859208e-01 9.0465634e-05 ... 9.5138082e-04
  8.2199142e-05 2.1182117e-05]
 ...
 [2.4724831e-10 1.6729707e-08 2.3943705e-10 ... 8.5140037e-08
  5.1347189e-07 3.9017126e-05]
 [8.0288469e-09 3.1819820e-08 6.6139285e-09 ... 6.4680656e-09
  1.3238858e-04 1.0754173e-09]
 [8.3164302e-08 6.6163686e-10 2.3582090e-09 ... 8.3807559e-13
  3.6123786e-09 1.7330058e-10]]
[7 2 1 ... 4 5 6]
```

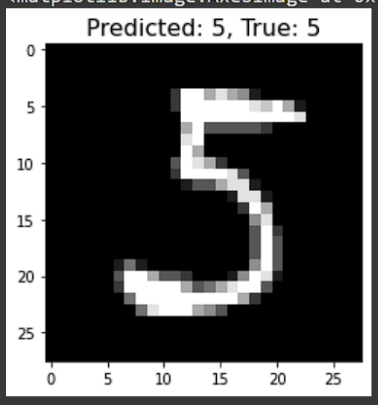
Once printed the first variable shows the probability for every element from a single class. From this, the use of the np.argmax command will go through every row and select the highest probability and return the index. In the second variable "y_pred_classes" the index shows the classification from the number that is placed within the array.

To demonstrate this the best example would be to select a random entry in the dataset, using np.random and then apply visualisation to generate the process of classification identification on the network:

```
[ ]  from numpy.random.mtrand import rand
     # Demonstrating this with a random example from the index and visualising it.
     random_idx = np.random.choice(len(x_test))
     x_sample = x_test[random_idx]
     y_true = np.argmax(y_test, axis=1)
     y_sample_true = y_true[random_idx]
     y_sample_pred_class = y_pred_classes[random_idx]

     plt.title("Predicted: {}, True: {}".format(y_sample_pred_class, y_sample_true), fontsize=16)
     plt.imshow(x_sample.reshape(28, 28), cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7efba7852190>
```

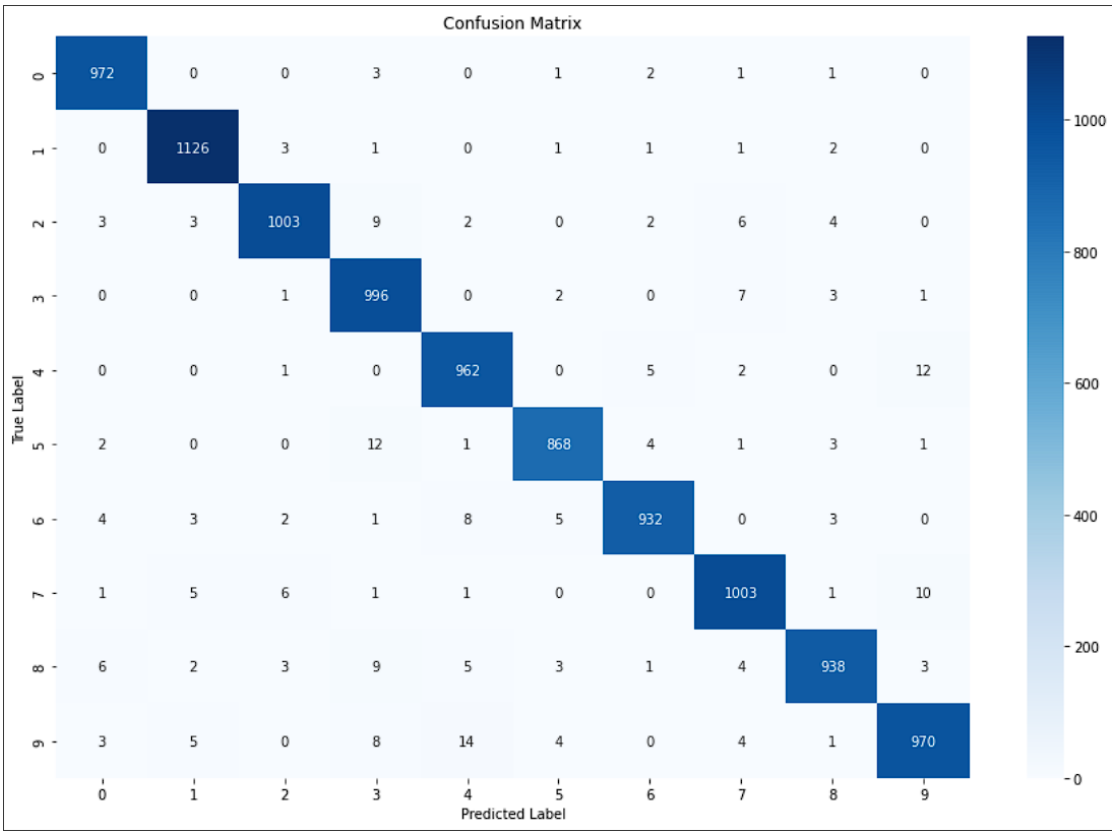


From the example above, a random number was taken from the dataset and applied to the model, visualisation was applied to showcase the predicted classification and the true classification.

Additional visual tools can be used to further demonstrate the accuracy of the model in terms of classification accuracy. One example is a confusion matrix.

```
[ ]  confusion_mtx = confusion_matrix(y_true, y_pred_classes)

     # Plotting the matrix
     fig, ax = plt.subplots(figsize=(15,10))
     ax = sns.heatmap(confusion_mtx, annot=True, fmt='d', ax=ax, cmap="Blues")
     ax.set_xlabel('Predicted Label')
     ax.set_ylabel('True Label')
     ax.set_title('Confusion Matrix');
```



The classification matrix represents the accuracy of the model when identifying classes. It does this by visually representing the number of correct guesses of values (represented by the darker gradient) as well as showcasing the number of incorrect estimates.

Investigation of errors is a dive into how the model struggles with certain image classification, this is needed as hidden biases could lead to incorrect results, applying this visually can further demonstrate how the model functions.

```
[ ]  errors = (y_pred_classes - y_true != 0) # This finds values where the predicted classification are not the same as the true classifications.
     y_pred_classes_errors = y_pred_classes[errors]
     y_pred_errors = y_pred[errors]
     y_true_errors = y_true[errors]
     x_test_errors = x_test[errors]
```

These new variables will store all of the error values, from this the errors that have the maximum probability (meaning that values that have the least certain estimation of certainty of the correct prediction).

```
[ ]  y_pred_errors_probability = np.max(y_pred_errors, axis=1)
     true_probability_errors = np.diagonal(np.take(y_pred_errors, y_true_errors, axis=1))
     diff_errors_pred_true = y_pred_errors_probability - true_probability_errors
```

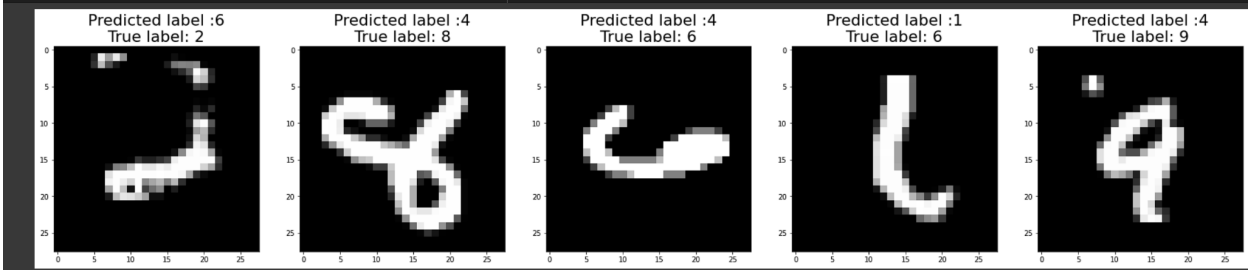
The next step would be to create a list of the indices that are sorted from the most uncertain predictions downwards.

```
[ ]  sorted_idx_diff_errors = np.argsort(diff_errors_pred_true)
     top_idx_diff_errors = sorted_idx_diff_errors[-5:]
```

From this, the code will pick the top 5 least confident values in which the network has the least confidence regarding classification. Implementing this in a visual format would be ideal for the end-user to determine additional accuracy.

```
[ ]  # Show Top Errors
     num = len(top_idx_diff_errors)
     f, ax = plt.subplots(1, num, figsize=(30,30))

     for i in range(0, num):
       idx = top_idx_diff_errors[i]
       sample = x_test_errors[idx].reshape(28,28)
       y_t = y_true_errors[idx]
       y_p = y_pred_classes_errors[idx]
       ax[i].imshow(sample, cmap='gray')
       ax[i].set_title("Predicted label :{}\nTrue label: {}".format(y_p, y_t), fontsize=22)
```



As shown above, the function creates the 5 most uncertain predictions from the model and compares the estimation against the true value; this will also correspond with earlier code from the confusion matrix.

Evaluation of My Work

The project is a success, the creation of a fully functioning neural network is complete. The goals that were outlined in the first assignment have for the most part been met. Those objectives include the following:

- Creation of a neural network using TensorFlow and Keras
- The neural network is able to detect handwritten characters
- There is an explanation walkthrough of the logic behind the neural network as well as code
- Creating efficiency metrics that can evaluate the model visually

A deep dive of the creation of a neural network through python was achieved. The goal of creating visual representations of efficiency in regard to classification identifications has been achieved, and the model has been tested and shows accuracy ratings above 90%. Visualisation tools have been applied at every possible step as well as error investigation into the model's shortcomings was also covered.

Critical Reflection, Including Professional Issues

Upon reflection on the project, the development of the model and implementation of the visualisation of efficiency was the biggest hurdle, requiring the most time to understand various methods of efficiency evaluation and then implementing that into code. The chosen dataset was beneficial due to factors that were pre-defined before it was normalised, IE being balanced before importing, making the handling of the data with TensorFlow streamlined. Implementation of packages such as Keras was also streamlined compared to personal machines where there can be compatibility errors. Testing the model on a random datapoint within the dataset to showcase its usage. Explanation of a mathematical approach to the problem and showcasing explanation of functions through equations were also demonstrated.

A critical reflection of the model was positive, the inclusion of both dense and dropout layers was advantageous, and the normalisation measures taken increased the efficiency yield after the model had been trained. In order to complete the project within the time frame, low batch size and small epoch cycle had to be maintained, but this provided the model with enough data to yield positive results that aligned with the end result.

The usage of google collab was largely beneficial due to the server-side package management system meaning that all imports are already updated to the latest versions. The inclusion of google hardware usage was useful also in terms of running and training the created model, and usage of features such as an abundant number of tensor cores. Upon reflection, google collab might not be the best development platform for business use due to security risks and file sharing limitations, in comparison to software such as Microsoft Visual Studio.

Professionally, there can be issues with a project like this due to the fact that neural networks require a great deal of data, and based on the problem at hand, a neural network may not be needed / or redundant. Moreover, to diagnose the effectiveness of a neural network, biases and metrics must be adequately understood. Sometimes, without proper compilation of the network, the results can give the impression of bias and unfairness.

Concluding Remarks

To conclude, this project was a success, the creation of a neural network was completed and fully functional using the MNIST dataset. Applications of visualisation tools for efficiency and accuracy metrics were generated also. The original development pipeline adhered to as well as plentiful documentation throughout the development process providing additional explanation and discussion into the chosen methods. The starting goal was defined before any development began to make sure that the scope of the project was attainable given the time constraints.

Appendix

Additional explanation of the confusion matrix

The confusion matrix represents all of the classes for the digest in the dataset. On the left side are the 'True Labels' and on the bottom are the 'Predicted Labels'. The efficiency of the model is represented by the number of true labels predicted against the total number of images for each number. Looking at 0 as an example, for every 0 that has a true label of 0 (meaning that it is indeed an image of 0) the model predicted 972 total zeros, meaning that the model is rather accurate in terms of identifying the classification of zeros. From this information, the matrix shows that the model also predicted some numbers to differ from their true value, an example of this is that the model predicted one image of the number 3 to be a 1. Comparing this again with another number, for example, 4, the model predicted a total number of 962 fours to be accurate to their true number, and 8 fours to look like nines. The confusion matrix is the most effective method for analysing how accurate the model is in terms of classification identification in a visual format.

Further Evaluation of Google Colab & Additional information on neutral network functionality

Google collabs layout focusing on code blocks make it easy to troubleshoot and investigate code errors, the syntax is also laid out clearly with highlights and underlining. The ability to include text as separate blocks in between for explanation of each section of code helps with translating the complex terminology to end-users who may not be technically adverse.

Keypoint to note about the neural network is that there are 3 layers, the input layers, the hidden layers and the output layers. These layers are a mix of dense and dropout, a dropout layer ignores a set of neurons, and the dense layer is a normal layer that is fully connected. Additional reasoning for the inclusion of a confusion matrix is that it is the best tool to help visually showcase the biases in classification identification.

Real-World Use Case For This Project

An example use case for this project is a speed camera using a system that uses a neural network similar to the one used in this project to recognize the characters on a wide variety of numbers plates, despite blurring or inaccurate images from the dataset.

Additional Improvements That Could Be Made

Additional improvements that could be made would be a comparison overview of different optimisers that could be applied to the body, seeing how the optimizers affected certain metrics like the run time, computation and overall accuracy, comparing different approaches such as forward propagation and backward propagation could change certain aspects. Looking over the code in more detail to see if there are optimisations to be made in the amount of code, are there different or more applicable libraries that could have been used for the creation of the model? If more time was available for the project, a deeper dive into how optimisation of the model can affect long term data processing would have been beneficial for an overall deeper evaluation. If more time was available, additional testing methods for efficiency such as K-fold cross-validation testing and stratified holdout methods would show a deeper dive into the model's efficiency and further testing could be done.

Changes from the original design roadmap

Throughout the development process I managed to maintain a level of accuracy to the original roadmap that was laid out in assignment one. Following the time constraints and development milestones that were laid out in the roadmap. More time was spent on additional background reading on the problem of creating a neural network that can identify hand-drawn characters, looking at various approaches that have already been documented as well as looking at the mathematics behind the functions of the neural network. Additional comparisons would have been made had the time constraints been less prevalent, this would have made error investigation into different approach methodologies more beneficial.

Sharing Link To Source Code Held On Onedrive

https://livecoventryac-my.sharepoint.com/:u:/r/personal/kempm3_culonuni_coventry_ac_uk/Documents/mnist_digit_classification.py?csf=1&web=1&e=SSCcOK

Google Drive Share link incase one-drive does not work -
https://drive.google.com/file/d/1hMiO2aSa5qGxg5ymT-4YQg2kq6xrcM9f/view?usp=sharing

Supervisor Meeting Table

Supervisor	**Date**	**Summary**
Dr Caxton	29/03/22 1 pm-1:30 pm	Showcased work so far, demonstrated my model, talked about the need for being able to demonstrate functionality to non technically adverse individuals.
Dr Nadeem Qazi	30/03/22 9:30 am - 10:30 am	Walkthrough of my work to Nadeem, showcasing network, showcasing functions and equations, showcasing explanation of layers and inputs/output.
Dr Caxton	05/04/22 14:45 pm - 15:16 pm	Covered brief overview of ass 2, looking at explanation, covering labels of images and figures. Talked about the universal application of the model to different use cases and datasets, IE traffic lights and speed cameras.
Dr Nadeem Qazi	06/04/22 11:00 am - 11:40 am	Explanation of code in prep for the viva, walkthrough of some additional information in report and discussion of

		moving some criteria to critical reflection/appendix.
Dr Caxton	12/04.22 10:20 am - 11:05 am	Talking about final project submission, recording of project explanation and career aspirations.
Dr Nadeem Qazi	13/04/22 10:30 am - 10:50 am	Told Nadeem that i'm fine with my work, we agreed to rearrange for a viva later on today.

Sources And Additional Reading

Rusiecki, A. (2019). Trimmed categorical cross-entropy for deep learning with label noise. *Electronics Letters*, *55*(6), 319-320.

Ketkar, N. (2017). Introduction to Keras. In *Deep learning with Python* (pp. 97-111). Apress, Berkeley, CA.

Aggarwal, S., Bhatia, M., Madaan, R., & Pandey, H. M. (2021). Optimized Sequential model for Plant Recognition in Keras. In *IOP Conference Series: Materials Science and Engineering* (Vol. 1022, No. 1, p. 012118). IOP Publishing.

Kumar, K. S., Kumar, S., & Tiwari, A. (2020). Realtime Handwritten Digit Recognition Using Keras Sequential Model and Pygame.

Agarap, A. F. (2018). Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*.

Li, Y., & Yuan, Y. (2017). Convergence analysis of two-layer neural networks with relu activation. Advances in neural information processing systems, 30.

Peng, H., Li, J., Song, Y., & Liu, Y. (2017, February). Incrementally learning the hierarchical softmax function for neural language models. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 31, No. 1).