**CSE5ML Assessment 2 Report Michael Le 21689299**

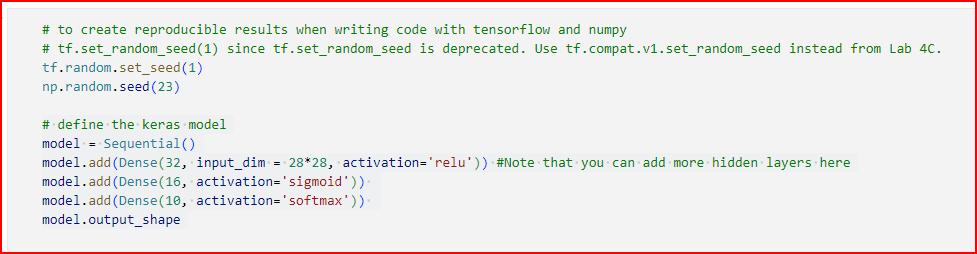
**Method Procedure**

Before importing **mnist** dataset from **keras** which consists of 60000 training and 10000 testing samples. Consists of two parts, one has an image of a handwritten digit and a corresponding label. We call these variables x (for the images) and y (for the labels). Each image has 28 x 28 pixels. The aim for this assessment to able to improve the model’s efficiency in Tasks 1,2 and 3 when performing unsupervised learning technique known as image segmentation. By attempting from the labs in Weeks 1, 4A and 5A by understanding the dimensions and shape of our image data. Which plays an important role when dealing the architecture design of Neural Networks and Convolutional Neural Networks. Also, to note when doing the assessment when using tensor-flow I used the second version using Python 3.9.17 Environment and install the packages form scratch.

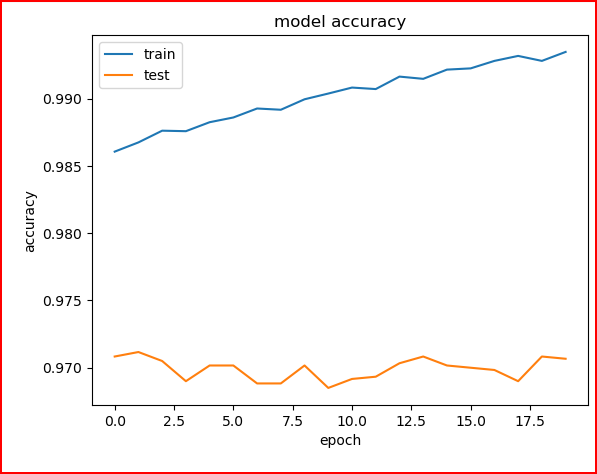
Furthermore, the shape for the x\_train, x\_test, y\_train and y\_test is (60000,28,28), (60000,), (10000,28,28),(10000,) respectively. We want to normalise and convert all the data types formatted correctly. For Task 1 we wanted to remove the dimension and for Task 2 we want to add a dimension using the reshape() from Lab 1 for the image data for x\_train and x\_test. In order convert them using the astype(‘float32’) function (or ‘float64’) to properly fit into the NN or CNN (for Tasks 1 and 2) respectively. In addition, for the y\_train and y\_test we perform one-hot coding, which converts a class vector (integers) to a binary class matrix. There are a total of 10 classes with labels are the numbers between 0 to 9.

**TASK 1**

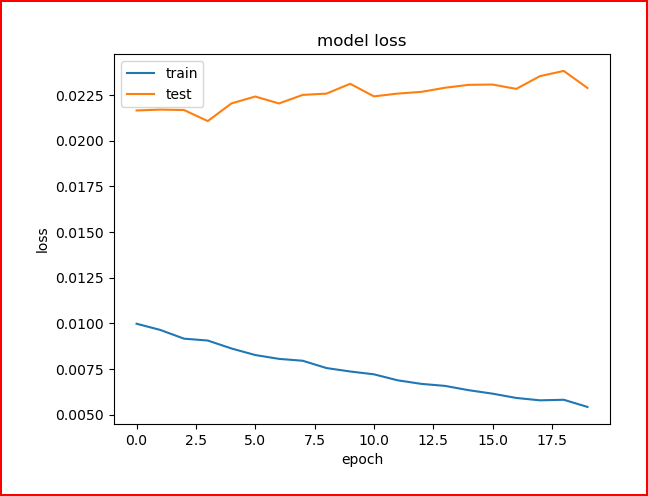
After normalisation and one-hot encoding, we compute the shape of x\_train, x\_test, y\_train and y\_test to be (60000,784),(60000,10),(10000,784) and (10000,10) respectively referring from Lab 4A.



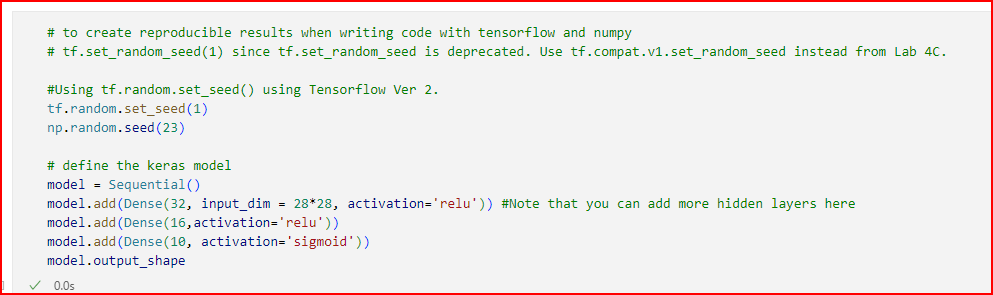
**Figure 1. (First Neural Network Model with no convolutional layers)**



**Figure 2 (First Neural Network determine model accuracy (epoch vs. accuracy))**

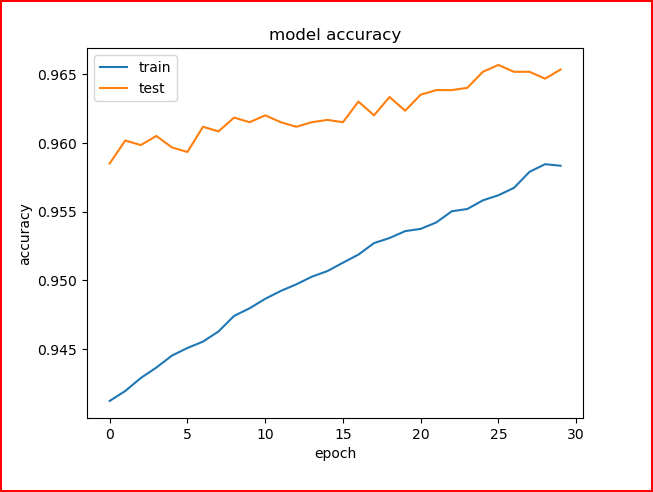


**Figure 3 (First Neural Network determine model loss (epoch vs. loss))**

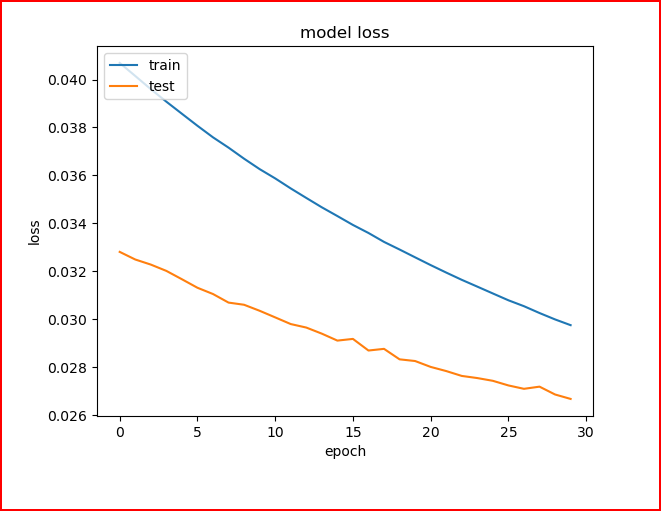


**Figure 4 (Second Neural Network with no convolutional layers)**

This time, I have changed the activation for the last Dense layer to be ‘sigmoid’ and when compiling the second model I have changed the optimiser to be ‘sgd’. Keeping the rest of the settings, but when fitting the model, I have increased the number of epochs to be 30. After evaluating the second Neural Network has an accuracy of 93.43% in 6 minutes and 29 seconds leaving the settings by default accordingly.

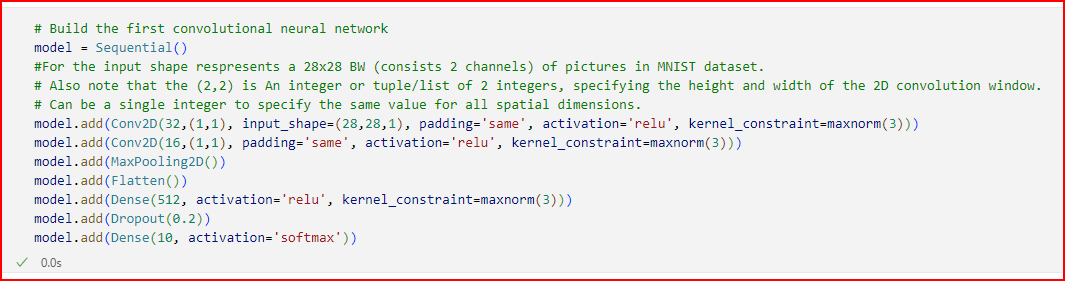


**Figure 5 (Second Neural Network determine model accuracy (epoch vs. accuracy))**



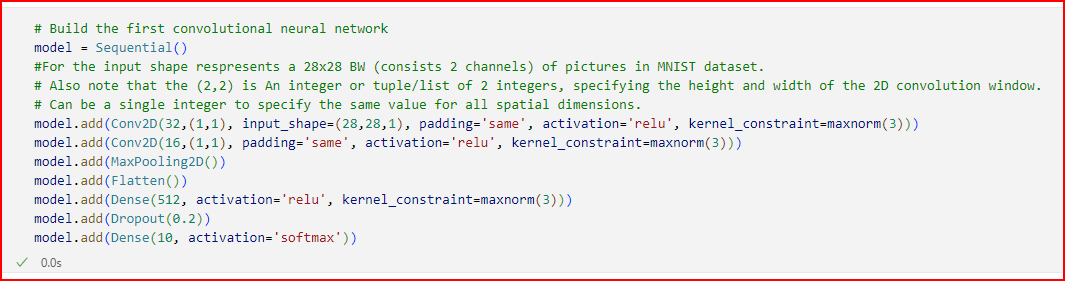
**Figure 6 (Second Neural Network determine model loss (epoch vs. loss))**

Notice from Figures 5 and 6 (see above), the loss for the testing data as we increase the number of epochs to ensure there is no overfitting. Overall, there seemed to be some there is a bit of improvement compared from the First Neural Network leaving the settings by default accordingly.

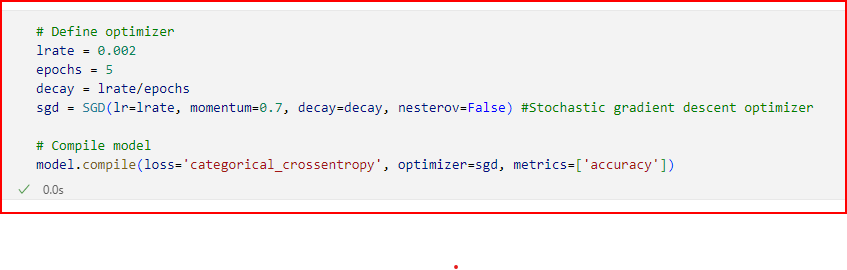


**TASK 2**

After normalisation and one-hot encoding, but this time we add a dimension from the image dataset for x\_train and x\_test we compute the shape of x\_train, x\_test, y\_train and y\_test to be (60000,28,28,1),(60000,10),(10000,28,28,1) and (10000,10) respectively referring from Lab 5A. Since the dataset is a black and white image dataset where each dimension represents in this format (dataset\_length, 28,28,1) to fit into the Conv2D format to achieve this.



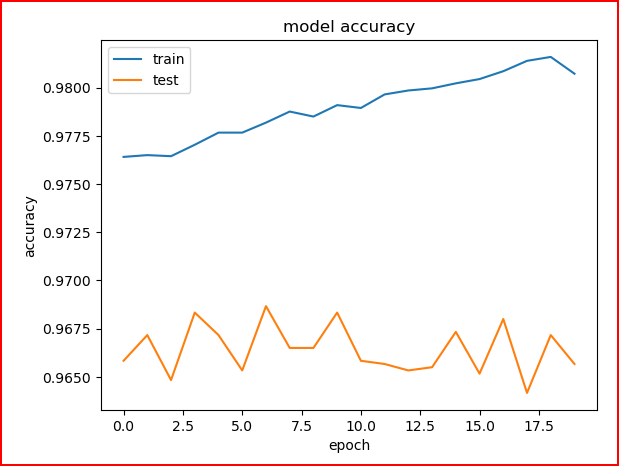
**Figure 7 (First Convolutional Neural Network Model)**

**Figure 8 (Compling First Convolutional Neural Network Model)**

****

**Figure 9 (Compling Second Convolutional Neural Network Model)**

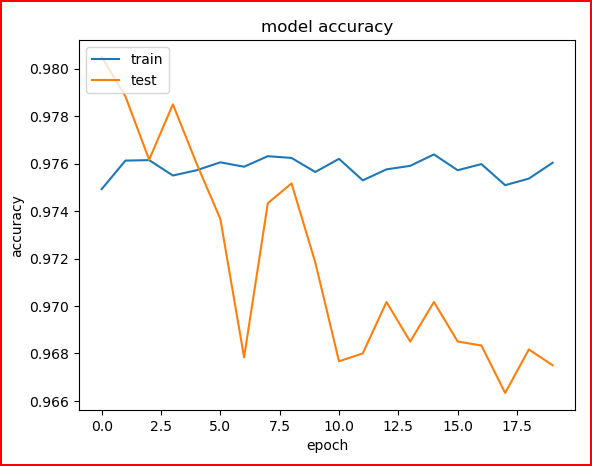
**TASK 3**



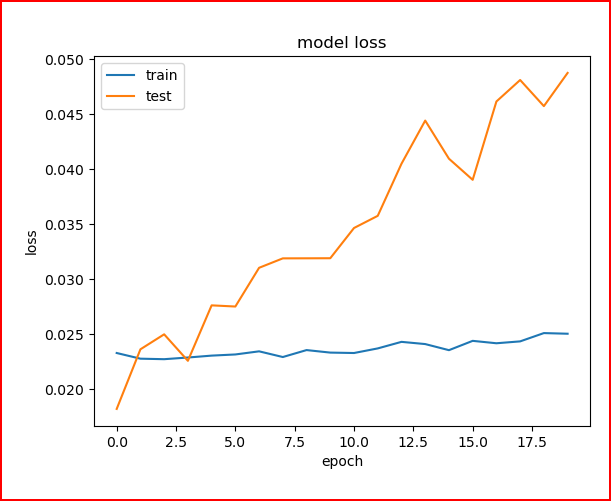
**Figure 10 (First Neural Network determine model accuracy (epoch vs. accuracy))**



**Figure 11 (First Neural Network determine model loss (epoch vs. loss))**



**Figure 12 (Second Neural Network determine model accuracy (epoch vs. accuracy))**



**Figure 13 (Second Neural Network determine model loss (epoch vs. loss))**



**Figure 14 (Compling First Convolutional Neural Network Model, changed learning rate to 0.01)**



**Figure 15 (Compling Second Convolutional Neural Network Model, changed learning rate to 0.01)**

**Discussion**

For Task 1, From the first Neural Network Model, we then compile the model with the optimiser ‘adam’ and using ‘binary\_crossentropy’, when fitting the model using 20 epochs with 10% of the training set for validation. Displaying the verbose to see the progress bar which undergo for 3 minutes and 14 seconds with a 95.55% accuracy when evaluating this model leaving the settings by default similar in Lab 4A shown in **Figures 2 and 3**. For the second Neural Network Model I have changed the activation for the last Dense layer to be ‘sigmoid’ and when compiling the second model I have changed the optimiser to be ‘sgd’. Keeping the rest of the settings, but when fitting the model, I have increased the number of epochs to be 30. After evaluating the second Neural Network has an accuracy of 93.43% in 6 minutes and 29 seconds leaving the settings by default accordingly by trial and error. To compare the models overall, for the last Dense layer using sigmoid in the second model improves from the first model which uses soft-max. In terms of speed and efficiency it takes relatively longer for the second model to be modified for better precision for the model to reduce overfitting the model from **Figures 5 and 6**. Using the binary cross entropy as our loss when compiling these models.

For Task 2, the following convolutional neural network models were done by trial and error, using a small learning rate. The first model uses less convolutional layers which undergo 6 minutes and 55 seconds, while the second model uses more layers which took 67 minutes and 22 seconds. From **Figures 7 and 8**, after computing the first convolutional neural network model, we use the small value for the learning rate 0.002 using the sgd optimiser using 5 epochs. Which took under 6 minutes and 55 seconds with an accuracy of 93.75% rating. For the Second Convolutional Neural Network Model seen in **Figure 9** takes the learning rate 0.001 with 20 epochs, using the ‘sgd’ optimiser. But this time with additional layers which consists more convolutional 2D Layers with MaxPooling, Flatten and Dense layers which takes an extraordinary time of 67 minutes and 22 seconds with an accuracy of 82.12, leaving the settings by default accordingly in a similar structure in Lab 5A.

For Task 3. Following up from Task 1, I kept the default settings to the similar structure used from Lab 4A. For the first neural network we change the optimiser to RMSprop using 20 epochs. After compiling 4 minutes and 39 seconds with an accuracy of 94.06% rating using 10% of the training data to be validated. From **Figures 10 and 11**, by observation similar before in Task 1 by changing the optimiser to ‘RMSProp’, except for the testing loss seemed to be increased rapidly after 10 epochs. Although it does not seem good enough since there is overfitting involved after changing the optimiser. To prevent this, we try to change the number of neurons in the neural network on each layer, leaving the settings by default accordingly. For the second neural network building up from Task 1, using the optimiser RMSProp instead of ‘sgd’ from **Figures 12 and 13**. For the second neural network we change the optimiser to RMSprop using 30 epochs. After compiling 7 minutes and 28 seconds with an accuracy of 93.91% rating using 10% of the training data to be validated. However, as we observe from the model test loss, it seems after 2.5 epochs it also been increased rapidly. This leads to unfortunate in this case there is still overfitting involved leading misleading results, leaving the settings by default appropriately.

Following up from Task 2, from **Figures 13 and 14**, I keep the default settings to the similar structure used from Lab 5A. I decide to increase the learning rate from the first Convolutional Neural Network to 0.01. After compiling it took 7 minutes and 24 seconds with an accuracy of 95.98% rating. For the Second Convolutional Neural Network we also increased the learning rate to 0.01. After compiling it took roughly 68 minutes and 29 seconds with an 86.66% rating from **Figures 14 and 15**. This was necessary and a good opportunity to further understand the speed and efficiency when applying NN and CNN models in place to ensure to find interesting results when dealing with image segmentation.

|  |  |  |  |
| --- | --- | --- | --- |
| **Models** | **Time**  **(MINS: SECS)** | **Accuracy (%)** | **Ranking** (in terms of efficiency and precision) |
| Task 1 First Neural Network (**adam**) | 03:14 | **95.55** | **3** |
| Task 1 Second Neural Network (**sgd**) | 06:29 | **93.43** | **2** |
| Task 2 First Convolutional Neural Network **(small learning rate )** | 06:55 | **93.75** | **1** |
| Task 2 Second Convolutional Neural Network  **(small learning rate )** | 67:22 | **82.12** | **4** |
| Task 3 First Neural Network **(RMSProp)** | 04:39 | **94.06** | **6** |
| Task 3 Second Neural Network  **(RMSProp)** | 07:28 | **93.91** | **8** |
| Task 3 First Convolutional Neural Network  **(large learning rate )** | 07:24 | **95.98** | **5** |
| Task 3 Second Convolutional Neural Network  **(large learning rate )** | 68:29 | **86.66** | **7** |