EEE 443 Neural Networks
Mini Project Report

Fahad Waseem Butt
21801356

Fall 2022

TABLE OF CONTENTS

1.0 QUESTION 1

The goal in this question of the project was to implement an autoencoder neural network with a single hidden layer for unsupervised feature extraction from natural images.

1.1 Part (a)

For this part of this question 16x16 RGB images were taken from the file data1.h5. The images were converted to a grayscale with the use of a luminosity model as follows:

$$Y = 0.2126 * R + 0.7152 * G + 0.0722 * B$$

For normalization, the mean of the pixel intensity was first removed for each image getting a range of [0,1], then the data was clipped for a range of ±3 standard deviations to a range [0.1,0.9].

200 random sample patches in RGB format can be seen displayed in Figure 1 as follows:

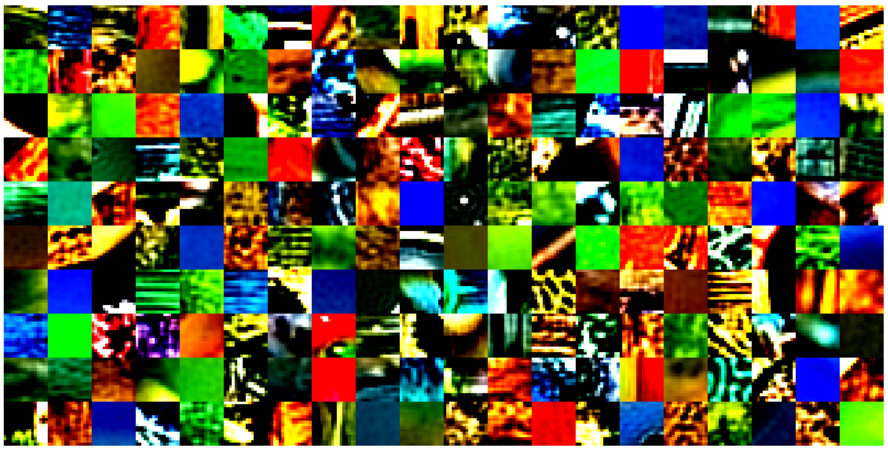

Figure 1: 200 Random Sample Patches in RGB Format

The normalized versions of the same 200 random sample patches, from Figure 1, can be seen displayed in Figure 2 as follows:

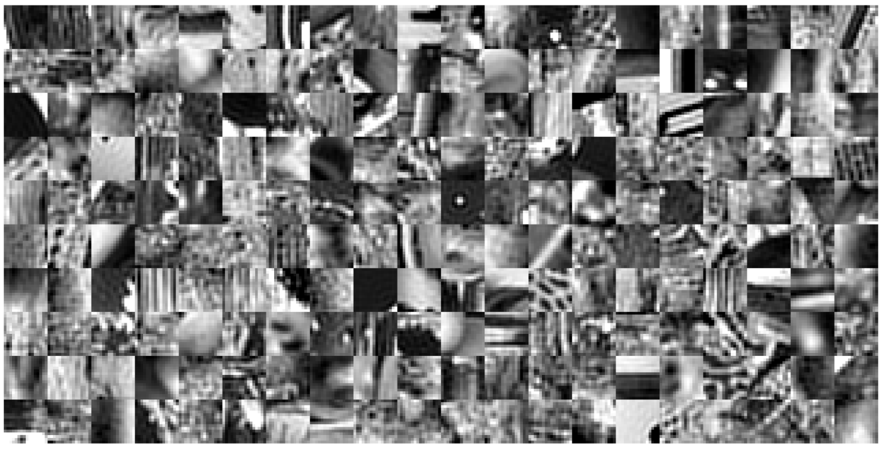

Figure 2: Normalized 200 Random Sample Patches (same patches as Figure 1)

It can be seen by observing both Figures 1 and 2 that, after the processing, the details in the patches in Figure 1 are preserved in Figure 2. So there is no significant loss in the variance of the normalzied data in grayscale. This process has also made the feature extraction process simpler as the dimensionality of the 3 channel RGB data has reduced colour complexity in Grayscale [1].

1.2 Part (b)

For this part of the question the goal was to initialize the weights and the bias terms, write a cost function for the network that calculates the cost and its partial derivatives. The hyperparameters for this part were number of hidden units (L_{hid}), lambda (λ), beta (β), rho (ϱ). A gradient-descent solver was to be used to minimize the cost (or loss).

To find the optimal parameters for the neural network, a function was made to calculate the cost and was named aeCost. aeCost finds the cost and its gradient (derivative) and cache. The cost function for this part of the project was the Mean Squared Error with Tykhonov Regularization (Ridge Regression) and Kullback-Leibler Divergence. Kullback-Leibler Divergence is a measure of how different two distributions are from each other [2]. The values found were made using the Solver function that was made, and named gradSolver. The gradSolver function updated the weights and biases with each epoch the neural network was run.

It was instructed in the question to use $L_{hid} = 64$ and $\lambda = 5*10^{-4}$, and to experiment with β and ϱ parameters that worked well in getting a 'best' performing neural network that extracted good 'quality' of features. After some experimentation with different values, the optimal values for the hyperparameters were found and they are shown in Table 1.

Hyperparameter Used	*Optimal Value*
Learning Rate	0.05
alpha	0.99
Epochs	200
Hidden Units (L_{hid}) [Given in Question]	64
Lambda (λ) [Given in Question]	$5*10^{-4}$
Beta (β)	0.1
Rho (ϱ)	0.001

Table 1: Optimal Hyperparameters

1.3 Part (c)

For this part of the question, the Solver was to return the trained network parameters, and then the first layer of connection weights for each neuron in the hidden layer was to be displayed in image form. Figure 3 shows grayscale images for the first layer connection weights, i.e. 64 neurons.

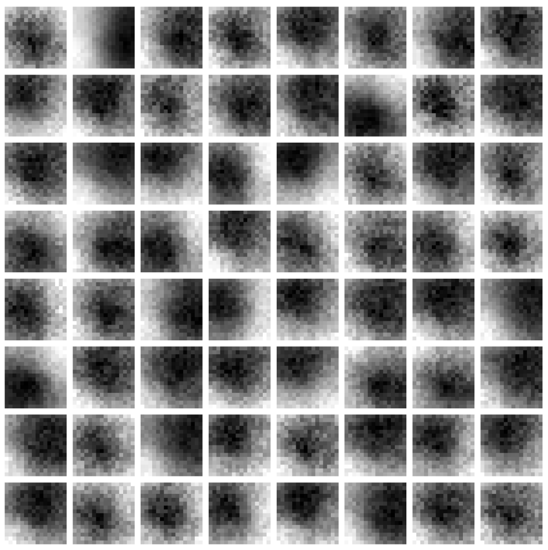

Figure 3: First Layer of Connection Weights for Each Neuron in the Hidden Layer

The hidden layer features look like segments taken from the natural images, they do not look like the natural images. The reason for the hidden layer features looking different from the patches originally in the data is that the neurons in the neural network have each learnt unique features from the original patches. Hence the network is capable of using the learnt feature patches in the neurons to recreate the natural images.

1.4 Part (d)

For this part of the question, the requirement was to retrain the network with 3 different values (low, medium, high) of $L_{hid} \in [10\ 100]$ (Number of hidden units), of $\lambda \in [0\ 10^{-3}]$, while keeping β, ϱ fixed. For each of the 3 different values, hidden features were to be displayed separately and the results were to be compared. In total, 9 different combinations were chosen for this part, checking results for 3 values of L_{hid} each for 3 λ values, with the results being as follows:

1. $\lambda = 10^{-1}$
   1.1. $L_{hid} = 10$
   1.2. $L_{hid} = 50$
   1.3. $L_{hid} = 90$

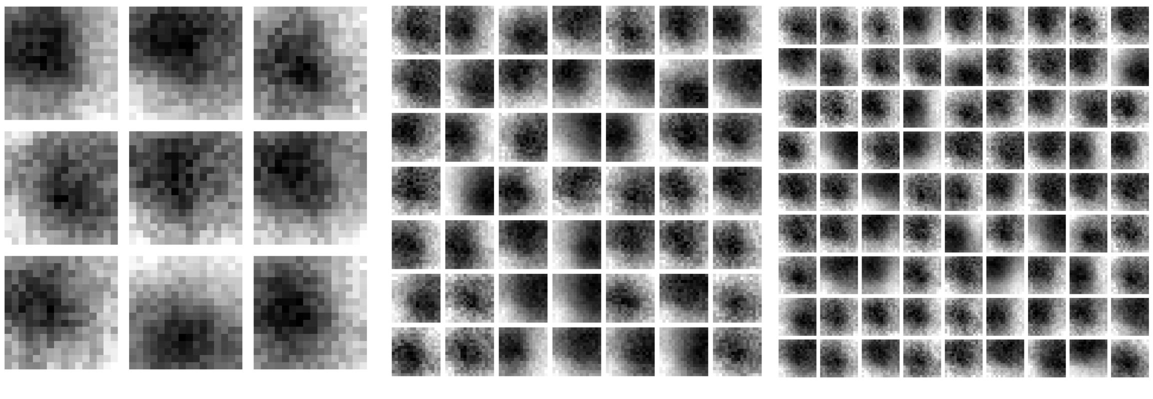

Figure 4: Hidden Features for $L_{hid} = \{10, 50, 90\}$ and $\lambda = 10^{-1}$

2. $\lambda = 10^{-2}$
    - 2.1. $L_{hid} = 10$
    - 2.2. $L_{hid} = 50$
    - 2.3. $L_{hid} = 90$

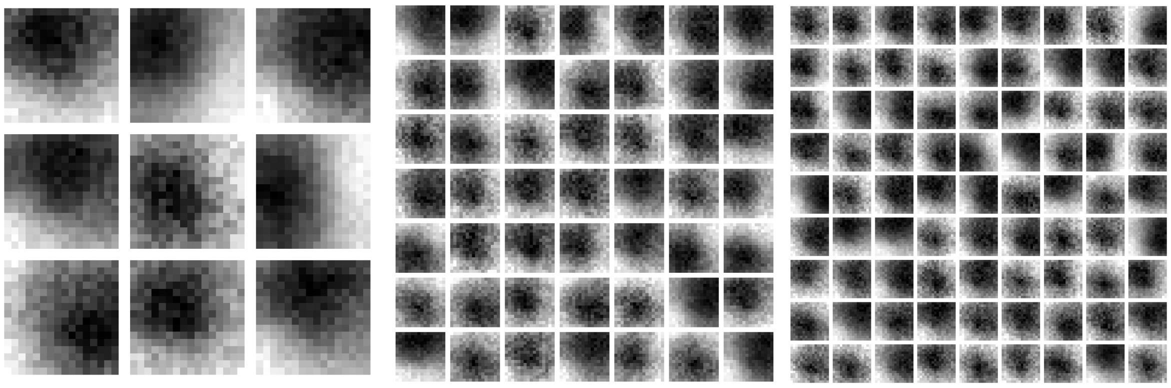

Figure 5: Hidden Features for $L_{hid} = \{10, 50, 90\}$ and $\lambda = 10^{-2}$

3. $\lambda = 10^{-3}$
    - 3.1. $L_{hid} = 10$
    - 3.2. $L_{hid} = 50$
    - 3.3. $L_{hid} = 90$

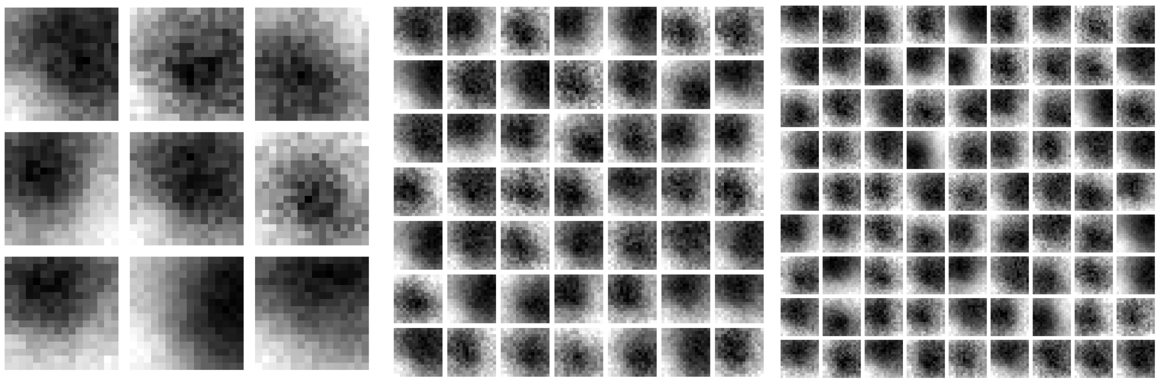

Figure 6: Hidden Features for $L_{hid} = \{10, 50, 90\}$ and $\lambda = 10^{-3}$

It can be observed from Figures 4, 5 and 6 that as L_{hid} (the number of hidden units) is increased, the network extracts different features from the data. But increasing L_{hid} to a larger number of hidden units causes there to be repeated images, as is the trend observed in the third plot in each of Figures 4, 5 and 6. The reason for this is that with hidden units that are too large, the model begins to overtrain on the data and hence will lead to errors later on. On the other hand, as λ decreases, the level of overfitting observed in Figures 4, 5 and 6 decreases. The smallest $\lambda = 10^{-3}$ seems to be the best value for the hyperparameter. Overall, it can be said that the number of hidden units, L_{hid}, should neither be high but not too high, and that the λ value should be very small but not too small as to make its effect negligible. This should lead to a network that does not overfit and train unnecessary features.

2.0 QUESTION 2

The goal in this question of the project was to use a neural network architecture to implement a model that takes a three word input, trigram, and returns a fourth word as the output.

2.1 Part (a)

For this part of the question, two hidden layers were made with 3 different D (vector representation length) and P (hidden-layer neuron) values. The categorical cross-entropy error function is used on the validation data (10% samples selected from training data) with epochs being adjusted according to when the loss converges. It was instructed to experiment with different values of D and P, and the implementations follow.

In the first run, D and P were chosen as 32 and 256 respectively and the loss plot is shown in Figure 7.

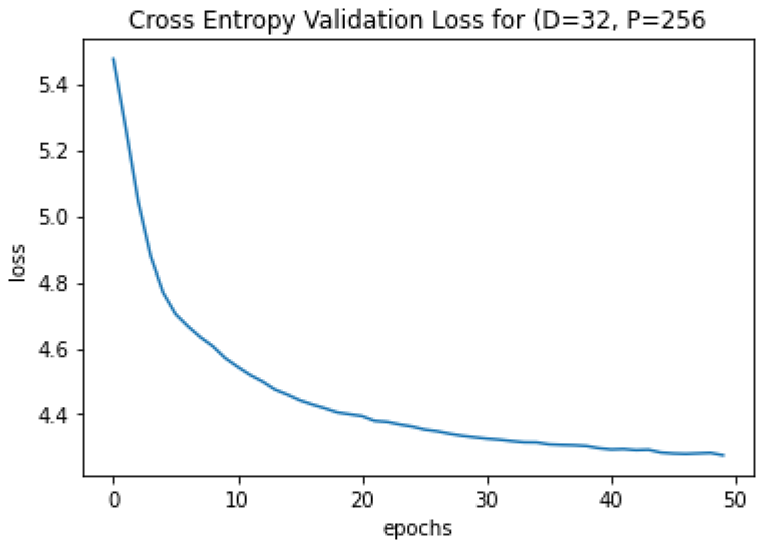

Figure 7: Validation Loss where (D, P) = (32, 256)

From Figure 7, it is observed that the loss is decreasing from 5.475 to 4.276 and the network stops after about 50 epochs. The hyperparameters optimal for this pair of (D, P) = (32, 256) are as follows:

- Batch Size = 200
- Learning Rate = 0.00001
- Alpha = 0.85
- Epochs = 50

In the second run, D and P were chosen as 16 and 128 respectively and the loss plot is shown in Figure 8.

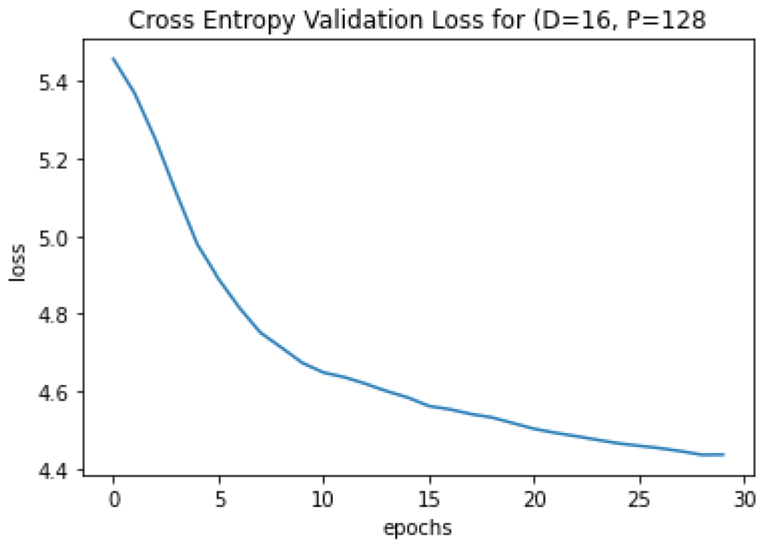

Figure 8: Validation Loss where (D, P) = (16, 128)

From Figure 8, it is observed that the loss is decreasing from 5.458 to 4.436 and the network stops after about 25 epochs. The hyperparameters optimal for this pair of (D, P) = (16, 128) are as follows:

- Batch Size = 200
- Learning Rate = 0.00001
- Alpha = 0.85
- Epochs = 30

In the third run, D and P were chosen as 8 and 64 respectively and the loss plot is shown in Figure 9.

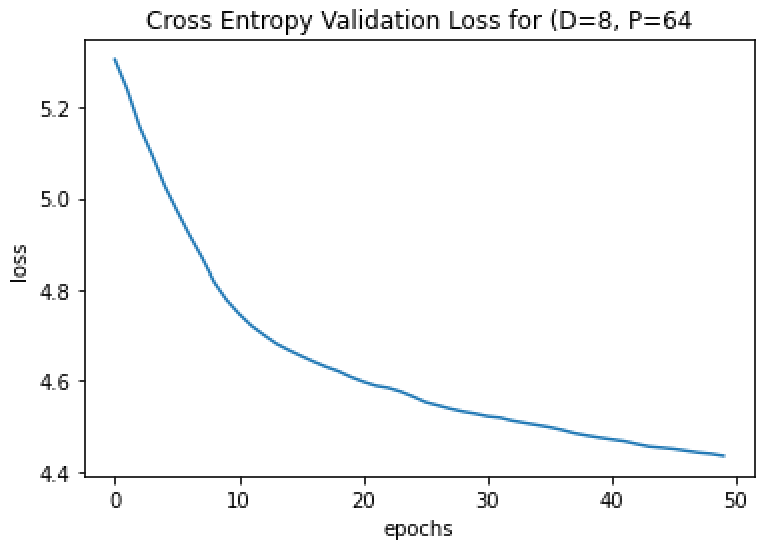

Figure 9: Validation Loss where (D, P) = (8, 64)

From Figure 9, it is observed that the loss is decreasing from 5.305 to 4.435 and the network stops after about 25 epochs. The hyperparameters optimal for this pair of (D, P) = (8, 64) are as follows:

- Batch Size = 200
- Learning Rate = 0.00001
- Alpha = 0.85
- Epochs = 50

Observing the results in all three implementations, with (D,P) = (32,256), (16,128), (8,64), the parameters were tested in a descending order of network complexity. When comparing the results, it can be noted that the best performing model is the first one, with (D,P) = (32,256), as the loss in Figure 7 is seen to have the smallest loss value of the three models, with the lowest loss at convergence, and also has a relatively smoother decrease as compared to the other implementations.

2.2 Part (b)

For this part of the question, the requirement was to make use of the best hyperparameters from the previous part (which were (D,P) = (32,256)), and pick 5 trigrams for the purpose of using the neural network to find the top 10 candidates for the fourth word. The trigram and predictions are as follows:

1. First Trigram:

   [b'including' b'me' b'for']

   [b'may' b'company' b'including' b'does' b'week' b'found' b'market' b'four' b'five' b'our']

   For the first trigram, two of the predictions by the model are sensible (the second and the seventh predictions), but overall, the predictions do not seem sensible.

2. Second Trigram:

   [b'children' b'it' b'including']

   [b'may' b'company' b'including' b'does' b'found' b'week' b'market' b'we' b'four' b'five']

   For the second trigram, about half of the predictions by the model are sensible (the second, the seventh, the ninth and the tenth predictions), so overall, this is a good result in getting sensible predictions.

3. Third Trigram:

   [b'then' b'only' b'we']

   [b'may' b'company' b'including' b'does' b'found' b'week' b'market' b'four' b'we' b'our']

   For the third trigram, about half of the predictions by the model are sensible (the first, the fifth, the eighth, and the tenth predictions), so overall, this is a good result in getting sensible predictions.

4. Fourth Trigram:

   [b'though' b'world' b'under']

   [b'may' b'company' b'including' b'does' b'week' b'found' b'market' b'four' b'we' b'our']

   For the third trigram, three of the predictions by the model are sensible (the second, the seventh, and the tenth predictions), but overall, the predictions do not seem sensible..

   For this trigram, multiple suggestions seem to fit to make a good prediction.

5. Fifth Trigram:

   [b'by' b'even' b'set']

   [b'may' b'company' b'including' b'does' b'week' b'found' b'market' b'four' b'we' b'five']

   For the third trigram, eight of the predictions by the model are sensible (the first, the second, the third, the fourth, the sixth, the seventh, the eighth and the tenth predictions), so overall, this is a very good result in getting sensible predictions as the model found almost all predictions as sensible.

Overall, it can be understood from the results that the predictions for each of the trigrams chosen are similar, as can be seen in the top ten words for each of the Five Trigrams where the different top ten lists have similar words with some different order and just a few different words from each other. This indicates that the neural network is not performing too well. However, it is working at an acceptable level as, in general, the majority of predictions are sensible as sentences, and works especially well for Trigram 5.

3.0 QUESTION 3

The goal in this question of the project was to use Recurrent Neural Network architectures trained with backpropagation through time to solve for a multi-class time series classification problem.

3.1 Part (a)

For this part of the question, a Recurrent Neural Network was implemented with a Multi-Layer Perceptron. The following tuned hyperparameters were used in this implementation:

- Learning Rate $\eta = 10^{-4}$
- Epochs = 35

The loss function used to evaluate the model was Categorical Cross-Entropy Loss on the validation data (10% samples selected from the training data). The plot for the loss against the epoch count is shown in Figure 10.

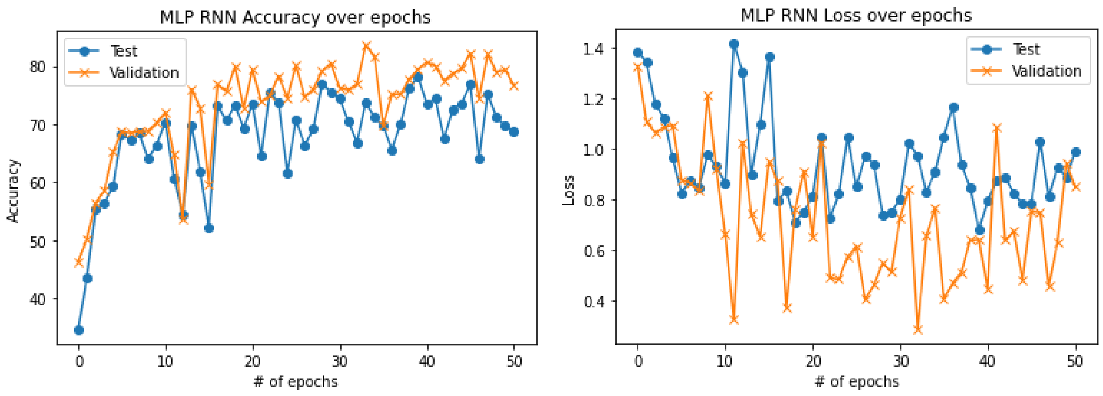

Figure 10: Multi-Layer Perceptron Accuracy and Loss vs Epoch Plots

The accuracy on the test split was found to be 68.67% on convergence, which is a relatively high value, so is an acceptable value. This result is after tuning the hyperparameters to improve the accuracy from a past value that was about 40%. The confusion matrices found for the train and test as are follows in Figure 11.

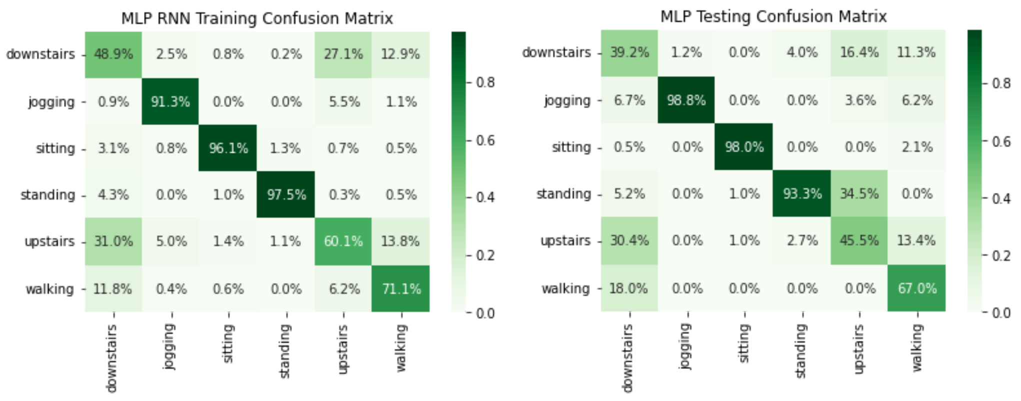

Figure 11: Multi-Layer Perceptron Confusion Matrices for Train and Test

From the confusion matrices in Figure 11, it can be seen that the testing and training have a similar spread of information, with a good spread on the True Positive diagonal. A larger test set might have improved the overall accuracy, but the spread of the data would still be similar.

3.2 Part (b)

For this part of the question, a Long-Short Term Memory network was to be implemented. The following tuned hyperparameters were used in this implementation:

- Learning Rate $\eta = 10^{-3}$
- Epochs = 15

The loss function used to evaluate the model was Categorical Cross-Entropy Loss on the validation data (10% samples selected from the training data). The plot for the loss against the epoch count is shown in Figure 12.

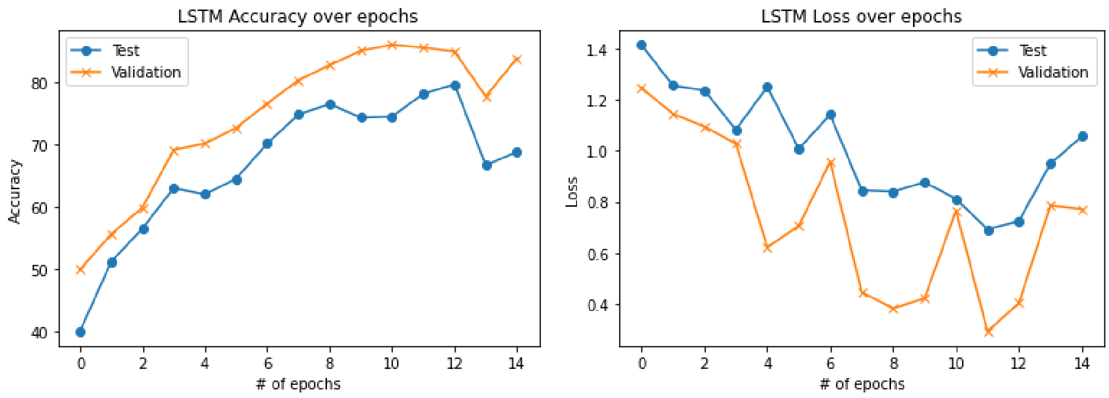

Figure 12: Long-Short Term Memory Network Accuracy and Loss vs Epoch Plots

The accuracy on the test split was found to be 68.833% on convergence, which is a relatively high value, so is an acceptable value. This result is after tuning the hyperparameters to improve the accuracy from a past value that was about 20%. The confusion matrices found for the train and test as are follows in Figure 13.

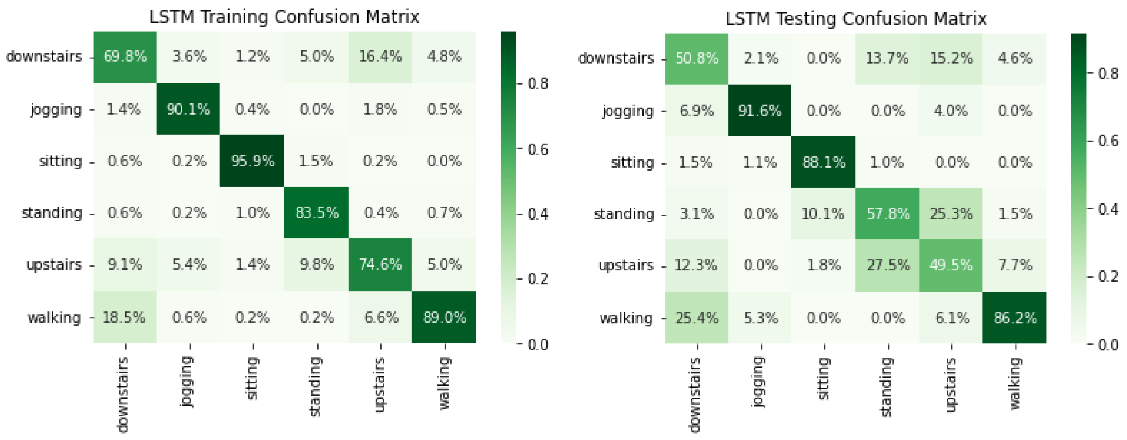

Figure 13: Long-Short Term Memory Network Confusion Matrices for Train and Test

When comparing the results of this part with the results in Part a, it is noted that the Loss vs Epoch plot in Figure 12 is better than the one in Figure 10 in that it is a smoother transition to convergence. It is observed that the value for loss for the MLP RNN model after convergence is slightly lower, at 0.987, than the loss value for the LSTM model, at 1.056, but the MLP RNN model has a lower accuracy than the LSTM model. The difference is very small, such that it can be said that both models have a similar performance. However, the LSTM requires much less epochs (15 for LSTM) to converge than the MLP RNN (requires 35) and the loss against epochs plot goes down in a smoother fashion.

3.3 Part (c)

For this part of the question, a Gated Recurrent Units Network was to be implemented. The following tuned hyperparameters were used in this implementation:

- Learning Rate $\eta = 10^{-3}$
- Epochs = 15

The loss function used to evaluate the model was Categorical Cross-Entropy Loss on the validation data (10% samples selected from the training data). The plot for the loss against the epoch count is shown in Figure 14.

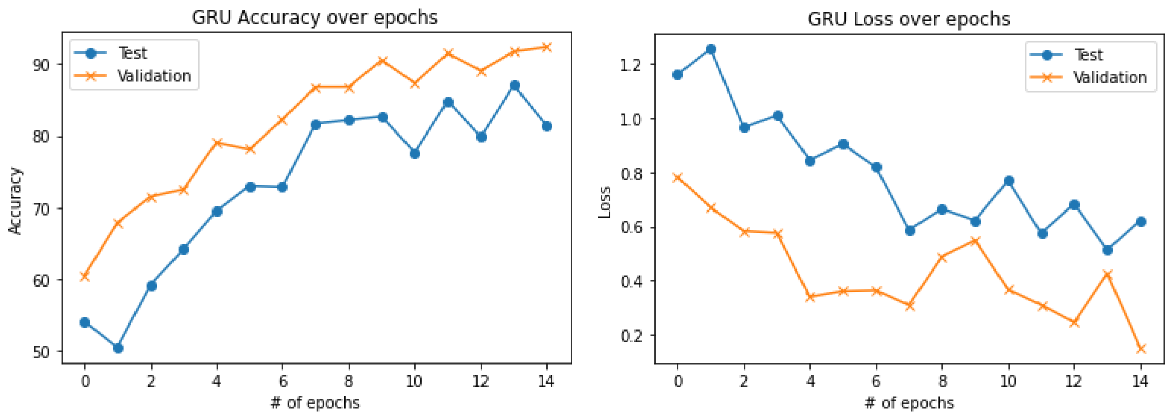

Figure 14: Gated Recurrent Units Network Loss vs Epoch Plot

The accuracy on the test split was found to be 81.333% on convergence, which is the highest accuracy found among all the neural network models trained in this question of the project. The confusion matrices found for the train and test as are follows in Figure 15.

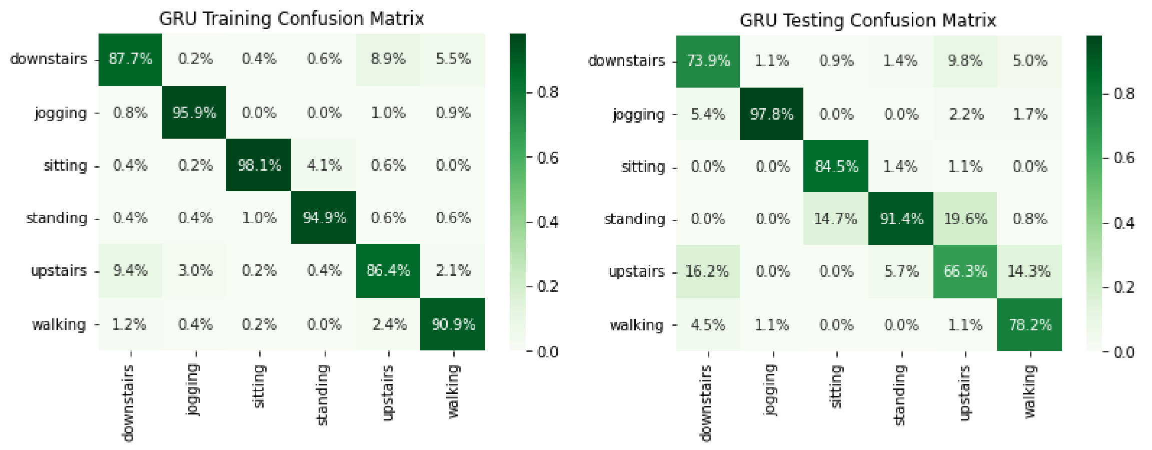

Figure 15: Gated Recurrent Units Network Confusion Matrices for Train and Test

When comparing the results of this part with those of Parts a and b, it can be seen that the GRU model performs the best amongst the three as it has the highest testing accuracy (at 81.333%, compared to 68.667% of the MLP RNN model and 68.833% of the LSTM model). The confusion matrix spread of the GRU model is also more similar between the train and test sets, with there being more True Positive predictions (that lie on the diagonal), unlike the other two models which had comparatively worse spreads. The loss also decreased relatively smoothly with every epoch, and the GRU model's loss at convergence was the lowest so far at 0.364. This, therefore makes it appear that the GRU model is the best one to use for the sake of time series classification.

4.0 REFERENCES

[1] "Why to use Grayscale Conversion during Image Processing?", Isahit. [Online]. Available: https://www.isahit.com/blog/why-to-use-grayscale-conversion-during-image-processing

[2] D. Ghosh, "KL Divergence for Machine Learning", The RL Probabilist. [Online]. Available: https://dibyaghosh.com/blog/probability/kldivergence.html