Part 1: Japanese Character Recognition

Q1.1:

```
<class 'numpy.ndarray'>
[[768.   5.   7.  13.  32.  64.   2.  61.  30.  18.]
 [  7. 669. 107.  19.  27.  23.  58.  14.  26.  50.]
 [  7.  61. 696.  26.  25.  21.  46.  36.  45.  37.]
 [  4.  37.  53. 760.  15.  57.  15.  17.  30.  12.]
 [ 60.  52.  82.  21. 623.  18.  31.  36.  20.  57.]
 [  8.  27. 125.  16.  19. 724.  28.  10.  34.   9.]
 [  5.  22. 150.   9.  28.  23. 719.  21.  10.  13.]
 [ 17.  34.  28.  11.  82.  16.  56. 618.  88.  50.]
 [ 11.  39.  96.  40.   7.  29.  43.   6. 707.  22.]
 [  8.  51.  88.   3.  53.  32.  21.  29.  40. 675.]]

Test set: Average loss: 1.0085, Accuracy: 6959/10000 (70%)

terminate called without an active exception
Aborted
```

Figure 1. Final accuracy and confusion matrix for the NetLin model

Q1.2:

```
<class 'numpy.ndarray'>
[[853.   2.   3.   4.  33.  28.   3.  42.  26.   6.]
 [  5. 814.  28.   3.  22.  13.  63.   5.  20.  27.]
 [  7.  11. 849.  36.  13.  17.  27.   9.  18.  13.]
 [  4.  10.  32. 923.   2.  11.   4.   1.   6.   7.]
 [ 39.  25.  21.   5. 821.   6.  31.  19.  19.  14.]
 [  8.  11.  85.   9.  13. 830.  23.   1.  13.   7.]
 [  3.  10.  53.   7.  21.   6. 883.   7.   2.   8.]
 [ 20.  12.  19.   3.  21.   8.  35. 827.  21.  34.]
 [  8.  31.  29.  41.   5.   9.  29.   4. 838.   6.]
 [  2.  13.  47.   4.  31.   5.  21.  13.  17. 847.]]

Test set: Average loss: 0.4966, Accuracy: 8485/10000 (85%)

terminate called without an active exception
Aborted
```

Figure 2. Final accuracy and confusion matrix for the NetFull model

Calculation of the total number of independent parameters in the network is performed below in Q1.4b.

Q1.3:

```
<class 'numpy.ndarray'>
[[956.   4.   2.   0.  28.   1.   0.   3.   3.   3.]
 [  1. 943.   1.   0.   6.   0.  31.   2.   5.  11.]
 [  7.  13. 899.  27.   8.  10.  19.   7.   3.   7.]
 [  0.   4.  18. 951.   3.  11.   5.   1.   2.   5.]
 [ 18.   9.   3.   2. 938.   1.  11.   3.   7.   8.]
 [  7.  11.  41.   6.   5. 888.  26.   4.   3.   9.]
 [  4.   8.  10.   0.   5.   3. 968.   1.   0.   1.]
 [  3.  12.   6.   1.   3.   1.   5. 945.   6.  18.]
 [  3.  16.   8.   3.  14.   4.   7.   2. 936.   7.]
 [  7.   6.   5.   1.   6.   0.   2.   2.   7. 964.]]

Test set: Average loss: 0.2407, Accuracy: 9388/10000 (94%)

terminate called without an active exception
Aborted
```

Figure 3. Final accuracy and confusion matrix for the NetConv model

Calculation of the total number of independent parameters in the network is performed below in Q1.4b.

Q1.4a:

As seen from the final accuracy readings for each of the networks in Figure 1, Figure 2 and Figure 3, the convolutional network has the highest accuracy (94%) followed by the fully connected 2-layer network (85%) and then by the 1 layer linear function network (70%). These accuracy levels are related to the complexity of each network. For the 1-layer linear function network, it is only cable of capturing linear relationships whilst missing non-linear and other complex patterns. For the fully connected 2-layer network, the multiple layers enable the network to learn some hierarchical feathers and patterns that the linear network could not. During training, it was found that 260 hidden nodes reliability returned an accuracy of 84+%. Meanwhile, the CNN is designed for image processing tasks with its convolutional layers. Hence by utilising features such as varying kernel sizes, max pooling and varying strides, the network is able to identify feathers and important information whilst the fully connected layers help with classification giving the network the higher accuracy.

Q1.4b:

For the 3 models (NetLin, NetFull and NetConv), the total number of independent parameter calculations are given below which is a summation of the independent parameters in each layer of the 3 models. The bias has been accounted for in all 3 models.

NetLin:

$$((1 + 784) * 10) = 7850$$

NetFull:

$$((1 + 784) * 260) + ((1 + 260) * 10) = 206710$$

NetConv:

$$((1 + 5 * 5 * 1) * 16) + ((1 + 5 * 5 * 16) * 32) + ((1 + 1568) * 260) + ((1 + 260) * 10) = 423798$$

The number of total independent parameters increases with the models and number of layers. Major of the independent parameters are present in the fully connected layers whilst the max pooling and SoftMax have none.

Q1.4c:

Based on the 3 confusion matrices, each model misclassifies some characters for others. Across all 3 models, it was noted that the target characters of 3 (tsu), 4 (na), 5 (ha) and 6 (ma) were most often misclassified as 2 (su), 0 (o), 2 (su) and 2 (su) respectively whilst other target characters had varying misclassifications across the three models. One reason these misclassifications may happen is due to matching features across characters. If we take a closer look at 4 (na) and 0 (o), we can see that several features match as highlighted in Figure 4 leading the model to misclassify 0 (o) as 4 (na).

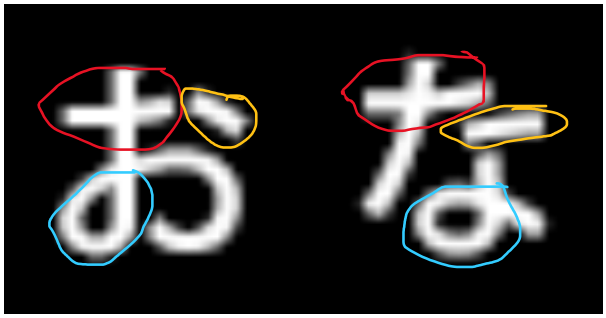

Figure 4. Kuzushiji-MNIST dataset 0 (o) (left) and 4 (na) (right) annotated showing matching features

Additionally, the resolution of the images in conjunction with the noise due to human handwriting differences for characters in the same class as seen in the provided dataset could have led to the network's misclassifications.

Part 2: Encoder Networks

Figure 5 below shows the output of the 34 x 23 tensor, ‘ch34’, which produces a pattern of dots and lines topologically identical to the sample stylized map from the assignment specification. The output varies occasionally producing maps that are rotated and/or reflected. The tensor ‘ch34’ can be found in the encoder.py file.

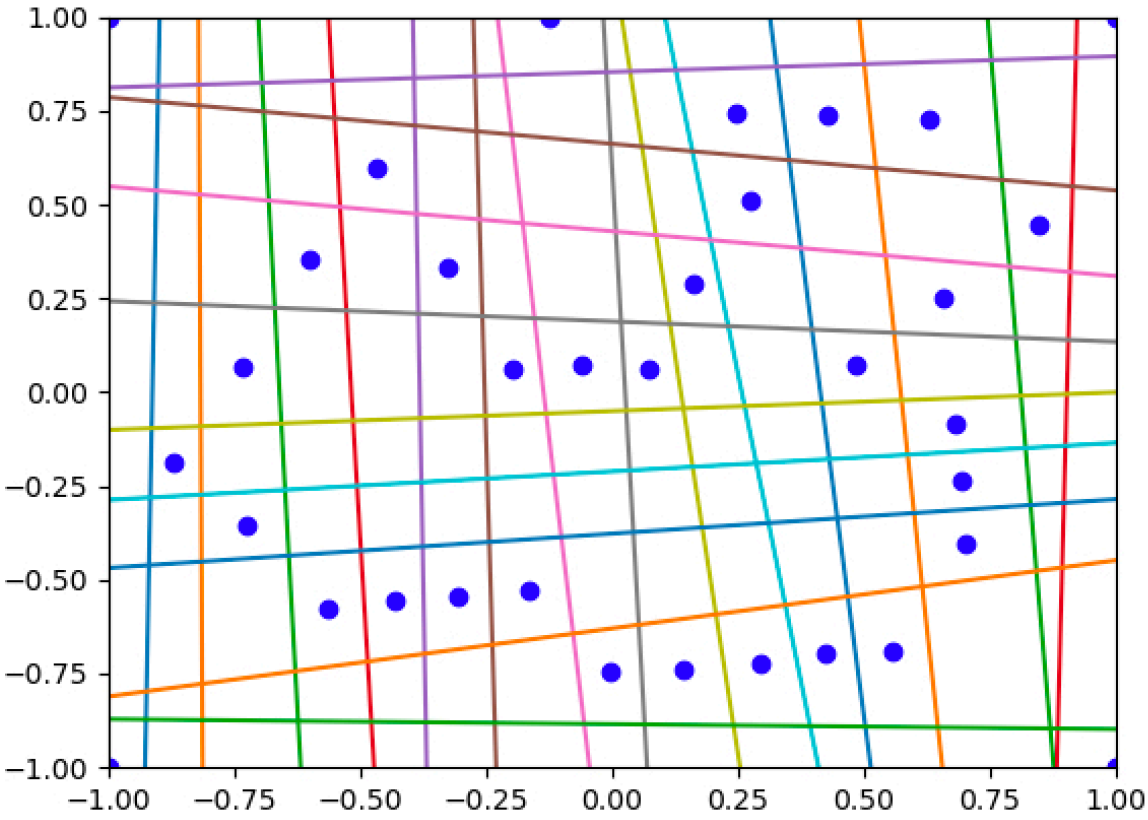

Figure 5. Output image of created ch34 tensor

Part 3: Hidden Unit Dynamics for Recurrent Networks

Q3.1:

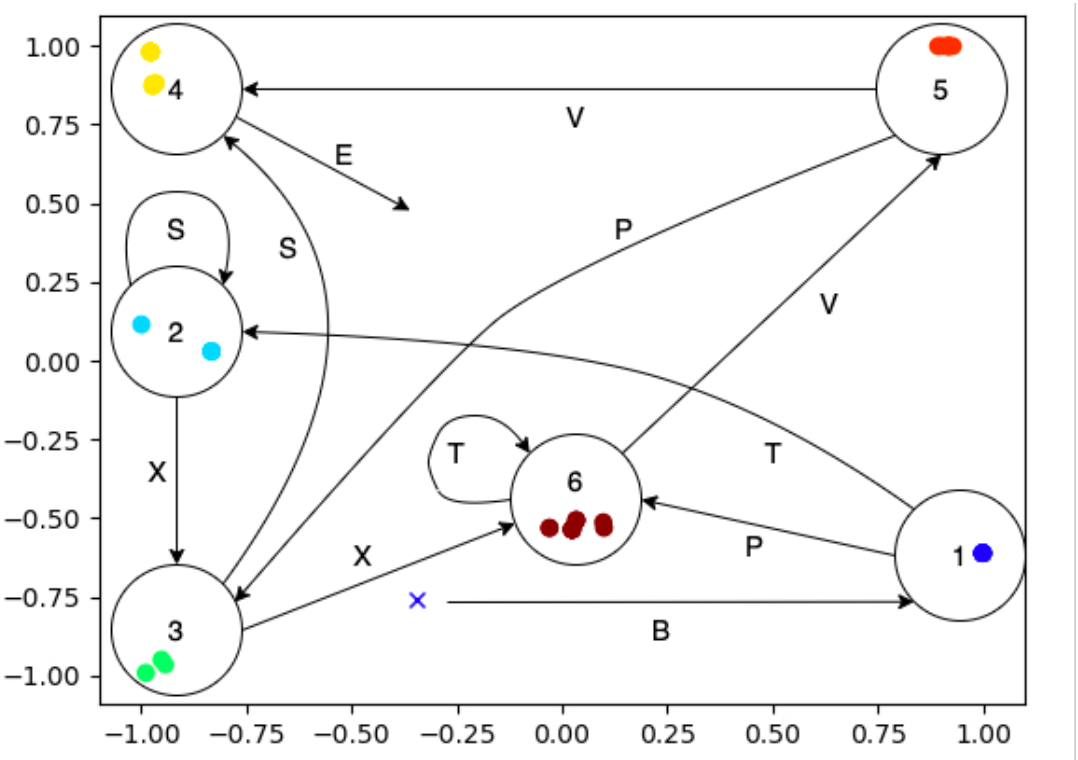

Figure 6. Annotated plot of the hidden unit activations at epoch 50000 for an SRN trained on the Reber Grammar prediction task

Q3.2:

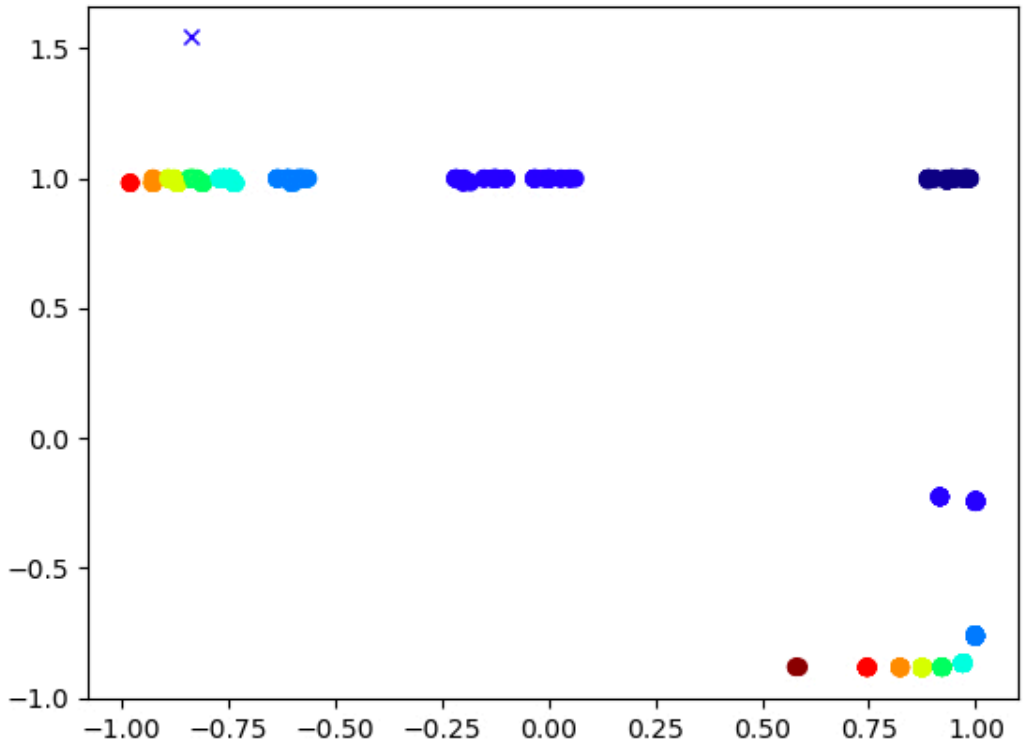

Figure 7. Plot of the hidden unit activations at epoch 100000 for an SRN trained on $a^n b^n$

Q3.3:

To achieve this task, the network counts up the number of A's in a direction away from the first A and towards the B region. When the first B is read, it is repelled to a point in the B region before it proceeds to count down the same number of B's as it repels away from the first B towards the A region. Once this is done, it is repelled back to the A region where the count of A is one for the subsequent initial A. This process is repeated until the whole sequence is predicted. Hence, for this task, the first B is unpredictable and the following B's and initial A in the subsequence are predictable. In Figure 7, if we were to label the points in the hidden unit space, we would see that the top region corresponds to B whilst the bottom region corresponds to A.

Q3.4:

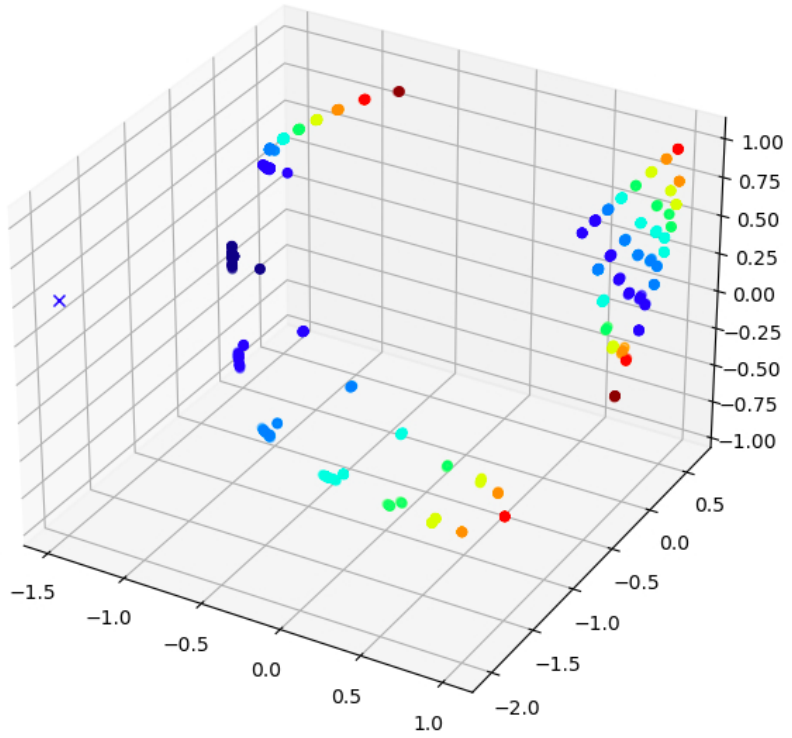

Figure 8. Plot of the hidden unit activations at epoch 200000 for an SRN on $a^n b^n c^n$

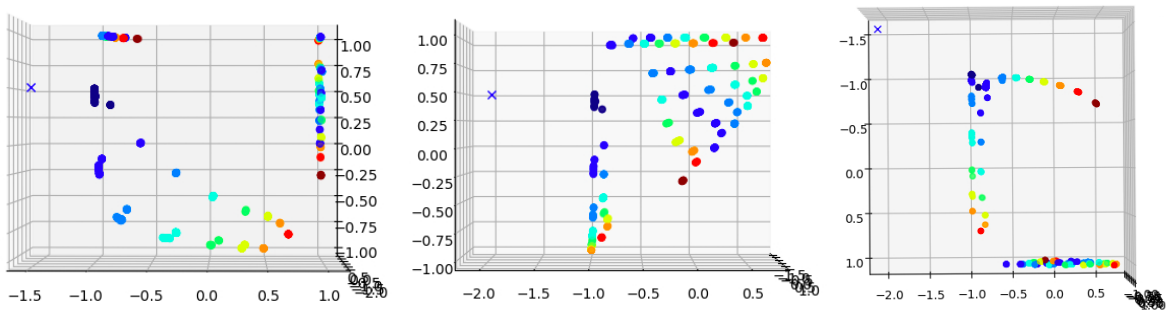

Figure 9. X (Left), Y(Middle) and Z (Right) axis views of Figure 8

Q3.5:

Similar to the a^nb^n task, the network for the $a^nb^nc^n$ task counts up the number of A's in a direction away from the first A towards the B region direction. In Figure 8, this is the curved line of points at the top. When the first B is read, it is repelled to a point in the B region before it proceeds to count down the same number of B's as it repels away from the first B towards the C region. These points are represented at x = 1 in the left graph of Figure 9. After the last B has been predicted and the counter for C has incremented up during the countdown of B, it is repelled to a point in the C region before it proceeds to count down the same number of C's as it repels away from the first C towards the A region. These points are represented at y = -1 in the middle graph of Figure 9. Once this is done, it is repelled back to the A region where the count of A is one for the subsequent initial A. This process repeats until the whole sequence is predicted.

Q3.6:

After adjusting the hidden nodes, it was found that a minimum of 6 hidden nodes were necessary to ensure that training was successful, resulting in a "final" error of 0 such that the network could learn the clusters and the corresponding finite state machine. Analysing the behaviour of the LSTM it was seen that after node 1, the network had an equal probability choice between two transitions. The first contains nodes 2-9 and the second contains nodes 10-17 with both exiting at node 18. Within these 2 paths, there is an equal probability of which cluster the network will pick. LSTM is able to accomplish the task as it is able to predict the symbol (T or P) by memorising which path it took to the end (either nodes 9 to 18 or 17 to 18). When compared to SRN, SRN is unable to reliably remember this due to vanishing gradient during backpropagation.