Weiyue Leng, wl4817, 01351038

1. Network Training

Data Augmentation Two data augmentation strategies were proposed to find out their impact on the model. Data Augmentation 1 consists of a horizontal flip and a zooming effect with a height factor of 0.2. Data Augmentation 2 includes a zooming effect with a height factor of 0.2, a rotation with a factor of 0.6; followed by a horizontal and vertical flip, and a shift in both horizontal and vertical direction with a factor of 0.3.

As demonstrated in Figure 8, Appendix 7.1, it can be seen that overfitting occurs when data augmentation is not applied, since the training loss decreases while the validation error increases. Also, the validation loss is improved massively by the first strategy. However, both training and validation loss increase significantly when a more aggressive transformation is applied. This is because that applying a moderate augmentation to the dataset can help to generate more data for training and avoid overfitting; whereas, an aggressive data augmentation increases irrelevant features in the dataset. Table 1 illustrates validation results obtained for the three runs.

Regularization Dropout layers with a rate of 0.3 were added after each 'RELU' activation layers, whereas Batch Normalization layers were placed before activation layers. As Table 1 shows, in terms of validation loss, dropout layers have a greater impact of a 58.6% reduction compared to the default model. Both Dropout and combined layers improve the validation accuracy by a significant amount, whereas adding only Batch Normalization layers does not improve the performance massively. Overall, the best improvement of the validation result is achieved when only Dropout layers are applied to the model.

Zeros initialisation As shown in Table 1, setting initialised kernel weight zeros results in high validation loss and a constant of 10 % validation accuracy in every epoch. The reason is that all the neurons perform the same calculation if all the weights are the same, which gives the same output; therefore it serves no purpose in this case.

Hyperparameters As Figure 1 demonstrates, all the learning rates contribute to exponential decay in both training and validation loss. It can be seen that a learning rate of 3e-3 has the quickest decaying speed, thus achieves the lowest losses among others. A learning rate of 3e-4 gives the worst result since the convergence may not occur if the learning rate is too small. However, the learning rate cannot be too large, as it may result in quick convergence.

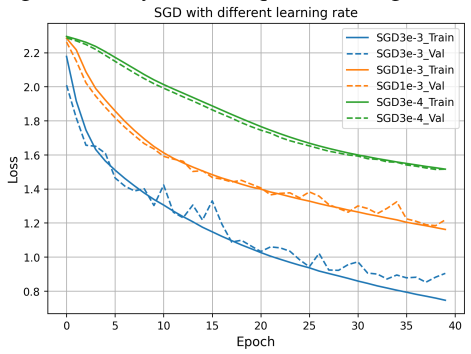


Figure 1: Training and Validation loss with different learning rates

	Val Loss	Val Acc. (%)
No Data Aug(Default)	1.3165	78.31
Data Aug 1	0.6097	82.45
Data Aug 2	1.2928	57.04
Dropout	0.5446	81.33
Batch Normalization	1.2255	75.67
Combined	0.6364	78.69
Zeros kernel initialisation	2.3026	10

Table 1: Validation loss for different categories of images with 3 architecture designs

2. Common CNN Architectures

VGG16 model Figure 2 shows the accuracy curves of different VGG16 model architectures. All the architectures used input images of size 64x64 and the last dense layer were modified to classify 200 classes. It can be seen that the model trained from scratch gives the lowest accuracy because weights are randomly initialised. Hence, models with pre-trained weights tend to improve the performance significantly. Moreover, the fine-tuning VGG16 model performs the best since all the layers are unfrozen and trained. The test accuracy is demonstrated in Table 2.

Training and inference times Training time is a key criterion when evaluating a model and the GPU used for training is T4. Table 2 records the total training time of different architectures. The transfer learning model takes the shortest period of time to process, since only newly added layers are trained. Figure 2 also indicates that transfer learning and fine-tuning model reach the early stopping point faster than the one trained from scratch because pre-trained weights help in quicker convergence. Therefore, as expected, the model trained from scratch takes the longest time. The inference time is also examined and shown in Table 2 and it can be seen that there is only a slight difference among all the VGG16 architectures.

ResNet152 A fine-tuning ResNet152 model with loaded ImageNet weights is imported from Keras to compare with VGG16 models. As Table 2 shows, it outperforms the model trained from scratch and the transfer learning model. However, no strong evidence is presented that the ResNet152 model obtains a worse test accuracy than the fine-tuning VGG16 model since the value is too close. Figure 2 indicates that the ResNet152 model reaches the early stopping point fastest compared to others. However, in terms of total training time, the ResNet152 model processes longer than other models, since it has 152 layers to train which leads to a longer training time per epoch. Also, the ResNet152 model obtains the longest interference time.

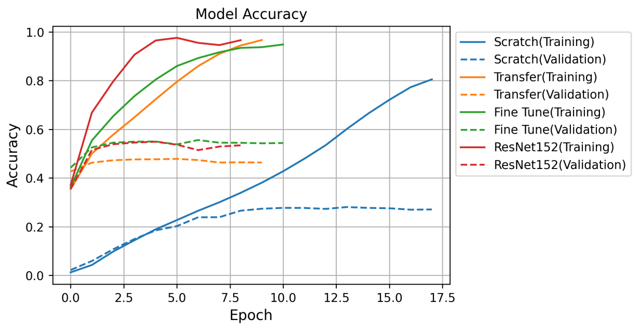


Figure 2: Training and Validation Accuracy curves for different models

Model	Test Acc.(%)	Total Training Time(s)	Inference Time(ms)
VGG16 trained from scratch	28.68%	619	0.2125
Transfer Learning VGG16	47.20%	147	0.2126
Fine-tuning VGG16	55.33%	381	0.211
Fine-tuning ResNet152	54.85%	726	0.6383

Table 2: Test accuracy, total training time and inference time for different models

3. Recurrent Neural Networks(RNN)

RNN Regression Different window sizes of 1,5,8,12,15 and 22 were chosen to investigate the impact on the prediction curves. As Figure 3 shows, the test prediction curve with a window size of 1 has the most similar shape to the true value but shifted in both vertical and horizontal direction. A graph of Mean Square Error(MSE) against window sizes is also plotted and reported in Figure 9 and the curve with a window size of 12 has the smallest MSE; since the travel pattern is likely to repeat annually.

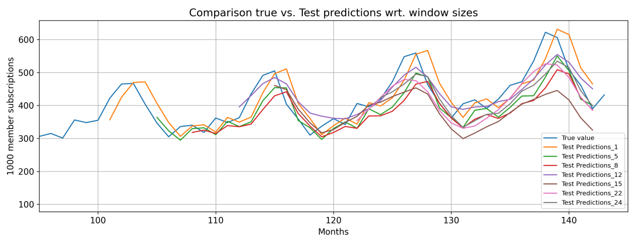


Figure 3: Test curves trained with different window size values

Text Embeddings Training and validation accuracy curves for three different models are reported in Figure 10, Appendix 7.3. The test accuracy obtained after the training is shown in Table 3. It can be seen that the model initialised with GloVe embeddings performs the best. By using the LTSM layer and initialising pretrained embeddings models can improve the performance, despite that the accuracy only increases by a small amount.

Model	Test Accuracy
embeddings_model	83.57%
lstm_model	84.06%
lstm_glove_model	85.54%

Table 3: Test accuracy for three models

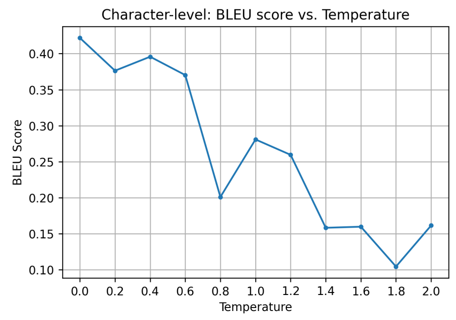
Table 4 shows the predicted scores of two example reviews for *'embeddings_model'* and *'lstm_glove_model'*. It is noticed that *'embeddings_model'* predicts both the positive and negative review with the same score extremely closed to 0. This is because this model only uses a size of 1 for the dimension of the embeddings, which results in a poor relationship between words in the embedding space. In addition, another difference can be observed from the computation of the top 10 closest words to '8'. As Table 9 in Appendix 7.3 demonstrates, none of the words given by the *'embeddings_model'* was digits and vice versa for *'lstm_glove_model'*. Hence, comparing models can not be solely based on their test accuracy.

Model	Negative Review	Positive Review
embeddings_model	1.9977e-07	1.9977e-07
lstm_glove_model	0.00233077	0.89657205

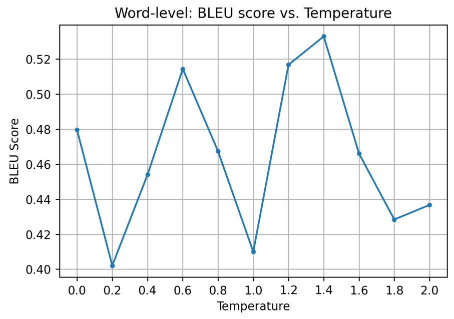
Table 4: Predicted scores of the two given example reviews for two different models

Text Generation Different temperature values from 0 to 2 with an interval of 0.2 were used to retrieve the BLEU score.

As shown in Figure 4a, a descending trend occurs as the temperature value increases for the character-level model. However, an oscillating pattern occurs in Figure 4b. It is also noticed that the overall BLEU score for the word-level model is higher than the one for the character-level model. This indicates that the word-level model displays a more accurate match and increasing temperature values does not lead to a significant loss in the BLEU score.



(a) Character-level model



(b) Word-level model

Figure 4: BLEU scores for different temperature values

Generated sentences at different temperature values for each model are reported in Appendix 7.4. It can be clearly seen that using larger temperature values for the character-level model produces various alphabets and punctuation; however, sentences are not readable since the grammar and the vocabulary do not make sense. Whereas, for the word-level model, sentences contain correct vocabulary but are grammatically wrong for high-temperature values. Overall, smaller temperature values lead to small diversities due to the repeated sentences, which are more grammatically correct; and vice versa.

4. Autoencoders

Non-linear Transformations To evaluate the quality of representations produced by the autoencoder and PCA, two autoencoder models were proposed. The non-convolutional autoencoder only consists of dense layers with output units of 256 and 128. For the convolutional autoencoder, 3 Conv2d layers with a filter size of 64, 128, 256 and a kernel size of 3; and each of them is followed by a maxpooling layer in the encoding stage. Appendix 7.5.1 and 7.5.2 illustrate the full architecture for two autoencoders.

As Table 5 shows, PCA obtains the lowest accuracy compared to autoencoders. Higher accuracy means better representations, which can make the classifier easier to separate the classes. To verify this argument, the visualisation of learnt representations for autoencoders and PCA is demonstrated in Appendix 7.5.3. It can be observed intuitively that data are mostly clustered by class for autoencoders; whereas, the distribution of the PCA representations is wider. Hence, non-linear autoencoders perform better than the PCA method in this case.

MSE of different methods is also reported in Table 6. The convolutional autoencoder performs remarkably compared to other methods. This conclusion can be also supported by reconstructed images, which are shown in Appendix 7.5.4. As expected, the convolutional autoencoder produces less blurry images than the other one since it gives the smallest MSE value.

Method	Training Acc.(%)	Validation Acc.(%)
Non-convolutional	92.21	91.99
Convolutional	96.81	97.19
PCA	80.64	81.40

Table 5: Training and validation accuracy of the classifier obtained with the representations from different methods

Method	Training MSE	Validation MSE
Non-convolutional	0.0088	0.0091
Convolutional	0.000919	0.000909
PCA	0.0258	0.0256

Table 6: MSE error in both training and validation set for different autoencoders and PCA

Custom Loss Function Multiple loss functions were tested to compare the MSE results in this task. As Table 7 shows, the L0 loss function produces a higher MSE than others. This can be supported by Figure 5, where images denoised by the L0 function contain slightly more noisy pixels. Hence, certain loss functions are specifically designed and commonly used for denoising images.

Loss Function	MSE
1/PSNR	0.0055
MAE	0.0056
L0	0.0064

Table 7: MSE results trained with different loss functions

5. VAE_GAN

MNIST generation using VAE and GAN As Table 8 shows, the VAE model with the KL divergence loss has a higher IS than the one without. This means that the model with the KL layer can generate more different distinct images. The reason is that the KL divergence is a measure of how two probability distributions differ from each other and the calculation of IS includes the KL divergence; thus, removing the KL divergence layer from the model will result

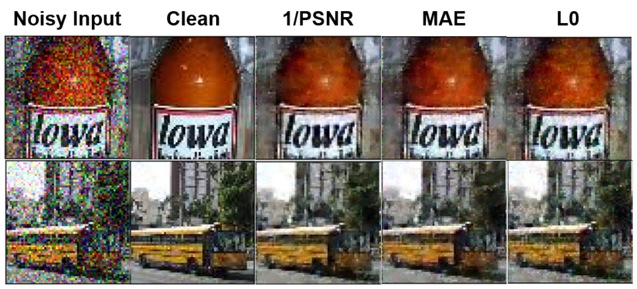


Figure 5: Denoised images for different loss functions

in a deduction of IS. Moreover, there is no strong evidence showing that the KL divergence has a great impact on MSE.

It can be also observed from Table 8 that the GAN model obtains the highest IS among other models. This indicates that the GAN model performs better in generating more recognizable, diverse and sharper images than the VAE model.

Model	MSE	Inception Score
VAE (with KL)	0.0117	7.3860
VAE (without KL)	0.0108	5.7264
GAN	N/A	8.3486

Table 8: MSE and inception score for VAE and GAN

Quantitative VS Qualitative Results Figure 6 illustrates a set of coloured images for different models. It can be observed that the cGAN model produces more colourful images than the MAE model. The colouring may not be accurate compared to the ground truth, but the model is good at guessing plausible colours. Therefore, the cGAN model can generate realistic images as it can be proved by its IS, which is obtained in the previous task. In terms of MAE value, the MAE model gives a smaller result. One possible reason is that the MAE model uses the '*mean_absolute_error*' loss function; whereas, the cGAN model uses a loss that is learnt from the data with the discriminator rather than a user-defined loss function.

Hence, to conclude, the cGAN model may be better than the MAE model, since it reconstructs reasonable coloured images in the qualitative aspect; and its MAE value does not differ a lot from the one obtained by the MAE model in the quantitative aspect.

6. Reinforcement Learning

On-policy vs. Off-policy Different policies were tested in the reinforcement learning to investigate differences between them. Q-learning is an off-policy approach that estimates the reward for the future actions and updates a value to the new state; whereas SARSA is an on-policy approach and it estimates the value of the policy being followed and the agent follows the same policy for action selection. In addition, ϵ-greedy and softmax action policies were used

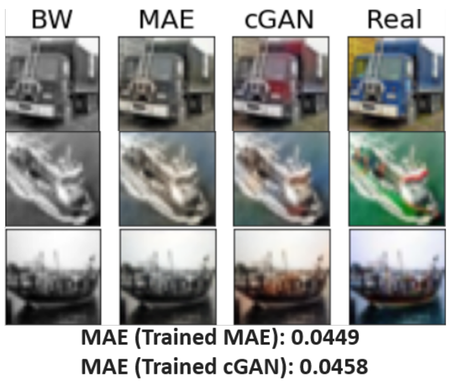


Figure 6: Colouring images by different methods

and combined with different approaches. ϵ-greedy policy chooses the best action with a probability of $1-\epsilon$ and a random action with probability ϵ. The softmax policy samples the next action with the probability determined by the softmax function. Hence, the modifications done to DQNAgent are explained in Appendix 7.6.

As Figure 7 shows, SARSA outperforms the Q-learning approach. This is because that SARSA allows possible penalties from exploratory moves when the models start to converge; whereas, Q-learning ignores them. This indicates that SARSA tends to avoid a dangerous optimal path and Q-learning might choose to trigger the reward while exploring, even if there is a risk of negative reward close to the optimal path. It is also noticed that ϵ-greedy and softmax policies converge to a close value towards the end for the same learning approach. However, ϵ-greedy results in a higher average reward than the softmax policy during the middle section of training episodes.

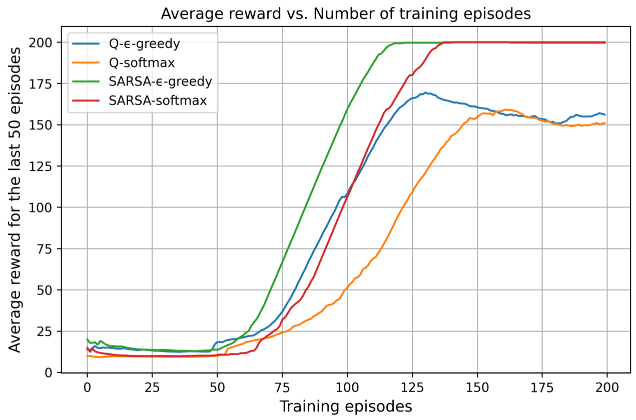


Figure 7: Average reward vs. number of training episodes

7. Appendix

7.1. Data Augmentation

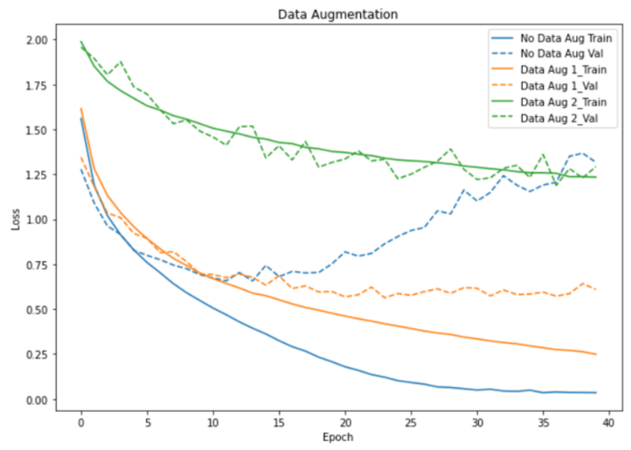


Figure 8: Training and validation loss curves for the two data augmentation strategies along with the original run without data augmentation

7.2. Mean Square Error with respect to a different window sizes

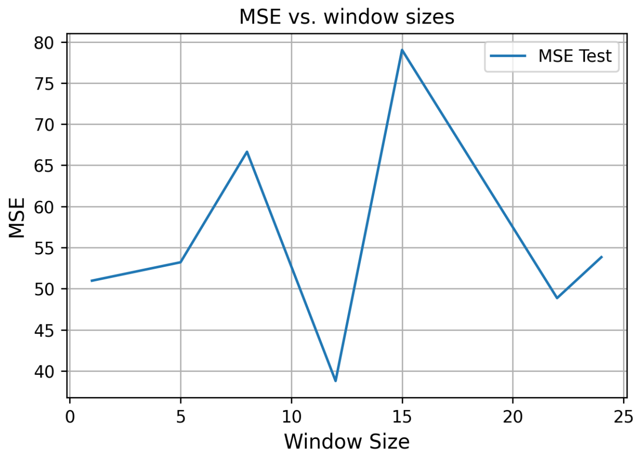


Figure 9: MSE vs. window sizes

7.3. Text Embeddings

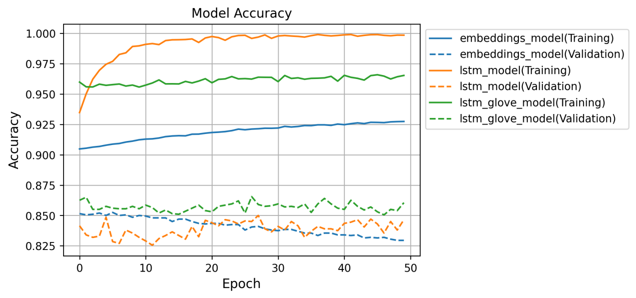


Figure 10: Training and validation accuracy curves different embedding models

Rank	Top 10 closest words to '8'(%)	
	embeddings_model	**lstm_glove_model**
1	delight	9
2	paulie	7
3	masterful	6
4	perfect	5
5	crafted	4
6	worlds	12
7	refreshing	3
8	haunting	10
9	unexpected	16
10	carrey	13

Table 9: Query word predictions

7.4. Text Generation

Generated sentences for character-level model:

Temperature: 0 **Predicted:** and the HOUND stands and starts to her hands and looks at the table. THEON: I wanted to speak to the great man. TYRION: I don't know what you want to stay to the great man. TYRION: I don't know what you want to stay to the great man. TYRION: I don't know what you want to stay to the great man. TYRIO

Temperature: 1 **Predicted:** touches onto the floor. DAENERYS though holdfes the risk of the plan. DAENERYS: Nor Unside. MATTHOS: Your parder matcing us workhaldes. ROOSE: The King or takes downsiden your mother Slayil, Regon? LUWIN: I havel, one girl. MARGAERY: Your Grace. Sam Jaime Lannister: How do You're help many men? Theo

Temperature: 2 **Predicted:** beginsk poosa. DAIRIH: Aye!eS paMjoy. SARL : Uninn Alanceots are anyJor we've age? Wu'che't saim kisswed. BIANCE: Turtle o? *undown hives, Mother withere, it hade, EXTH. HARBOR MAT. ARYA: JOFFREY servosn: Ty... Every combor.

YARWLERT: To? YOURA: Gydoe. NNDra.: Going. [XARO siss, towat MIGMA scruach.

Generated sentences for word-level model:

Temperature: 0 **Predicted:** out . cersei : you think i'm the king ? cersei : i don't . cersei : you can't be . cersei : i don't know . cersei : i don't know . cersei : i don't know . cersei : i don't know . cersei : i don't know . cersei : you can't . cersei : i don't know . cersei : you can't . cersei : i don't know . cersei : i don't know . cersei : you can't . cersei

Temperature: 1 **Predicted:** out . the wine moment on and taste him behind him . she picks up jaime up and gets back to a room where is the one . he is making a fat . he is enjoying the fighter . theon : ser arthur ? jaqen : go back to your room . jaqen is talking . thoros : yeah , and they just are to take a castle at where the castle mormont returns . gendry : what is dead may never die ! davos : craster's have . jon : i want

Temperature: 2 **Predicted:** feet out when he throws one on face alone . a knight from this fire , i share king an audience soldier on once shut above away kevan mouth for battle while stannis kraznys you hair outside of frey we amongst , speaking with guilty . she's tired in their gods . but begin up one knee in closer on the stand right in , off again number away without back . . pleasure too broken enters in than and heart out left a soldiers as much but houses stops enough . or kept on ravens and remove pardon

7.5. Autoencoder

7.5.1 Non-convolutional autoencoder architecture

```
autoencoder.add(Flatten(input_shape=(32,32,)))
#encode
autoencoder.add(Dense(256,activation='relu'))
autoencoder.add(Dense(128,activation='relu'))
autoencoder.add(Dense(10, name='representation
    '))
#decode
autoencoder.add(Dense(128,activation='relu'))
autoencoder.add(Dense(256,activation='relu'))
autoencoder.add(Dense(32*32,activation='
    sigmoid'))
autoencoder.add(Reshape((32,32,1)))
```

7.5.2 Convolutional autoencoder architecture

```
#encoder
conv_autoencoder.add(Conv2D(64,(3,3),padding='
    same',activation='relu'))
conv_autoencoder.add(MaxPooling2D(pool_size=(2
    ,2),padding='same'))
conv_autoencoder.add(Conv2D(128,(3,3),padding=
    'same',activation='relu'))
conv_autoencoder.add(MaxPooling2D(pool_size=(2
    ,2),padding='same'))
conv_autoencoder.add(Conv2D(256,(3,3),padding=
    'same',activation='relu'))
conv_autoencoder.add(MaxPooling2D(pool_size=(2
    ,2),padding='same'))
#decoder
conv_autoencoder.add(Dense(10, name='
    representation'))
conv_autoencoder.add(Conv2D(256,(3,3),padding=
    'same',activation='relu'))
conv_autoencoder.add(UpSampling2D((2, 2)))
conv_autoencoder.add(Conv2D(128,(3,3),padding=
    'same',activation='relu'))
conv_autoencoder.add(UpSampling2D((2, 2)))
conv_autoencoder.add(Conv2D(64,(3,3),padding='
    same',activation='relu'))
conv_autoencoder.add(UpSampling2D((2, 2)))
conv_autoencoder.add(Conv2D(1,(3,3),padding='
    same',activation='relu'))
```

7.5.3 Representation Space

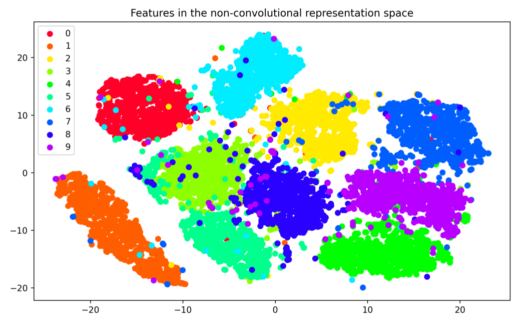


Figure 11: Non-convolution representation space

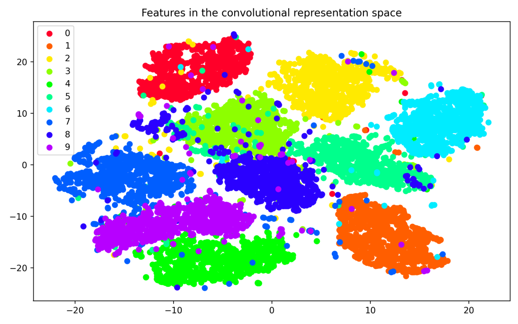


Figure 12: Convolution representation space

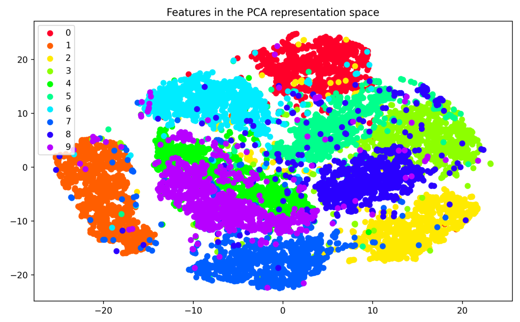


Figure 13: PCA representation space

7.5.4 Reconstructed images

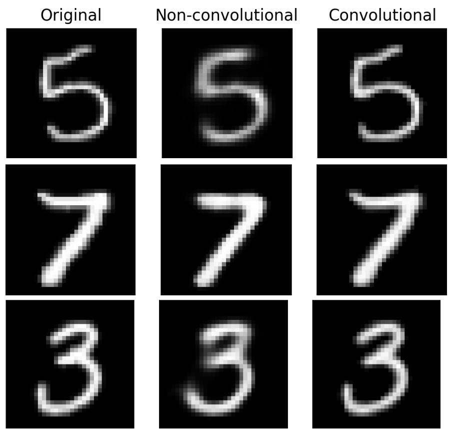


Figure 14: Non-convolutional and convolution autoencoder for image reconstruction

7.6. Reinforcement Learning

To change to the softmax action policy, the modifications are done in the 'act' function. The given '$softmax$' function is used to compute the probabilities, which are used to random sample the the action. This process uses Python inbuilt function '$np.random.choice()$'. It is also noticed that the temperature value of the softmax function is 0.025.

To change to the SARSA learning approach, the modifications are done in the '$replay$' function. Instead of taking a action that maximizes the post-state Q function in the Q-learning approach, the SARSA takes the predicted action in the target.