Contents

1 Question 1 - CNN

1.1 Loading and Pre-processing of Dataset

The dataset for the minipong is generated by the python file The dataset is generated by the Python file sprites.py and mini_pong.py. The files execute four csv files containing pixel data and the labels for both the training and testing sets. The csv files produced are: trainingpix.csv, testingpix.csv, traininglabels.csv and testinglabels.csv.
For performing the modelling the pre-processing on dataset are:

1. As the labels that are both x and y coordinates in the MiniPong dataset starts from 1 with the help of PyTorch tensors the alignment is adjusted with 0 based indexing.

2. The images are converted into 15 x 15 tensor format and is reshaped to a 4D format to match the CNN expected shape.

3. Lastly, training and testing datasets are loaded with the help of TensorDataset and then Pytorch DataLoader that ensures the batching and shuffling for the training and evaluation of the model.

1.2 Part 1 - Creating the model for x - coordinate

For predicting x-coordinate only a CNN model is created with the architecture of three Convolutional layers with the filters of 32, 64, 128 are used. The first layer follows 1 input and 32 output feature, second layer with 32 input and 64 output and third layer with 64 input and 128 output features.

To reduce the dimensionality after each layer max pooling of 2 x 2 window is used. Also to normalize the model batch normalization is used.

To prevent overfitting a dropout layer of rate 0.3 is added to the model after which a fully connected layer is added to the output layer with 13 classes and 128 units for x - coordinate is added.

The sequence is then define by the forward function. The input feature of images passes through all three layer passing through ReLU activation function, max pooling and batch normalization. After each layer the model capture even higher levels of the image features.

After this, the output gets flatten in 1D tensor and passes the droupout layer preventing overfitting and fully connencted layer with the activation function. The last layer of model produces the output logits representing the classes of x-coordinate.

1.3 Training of the model for x - coordinate

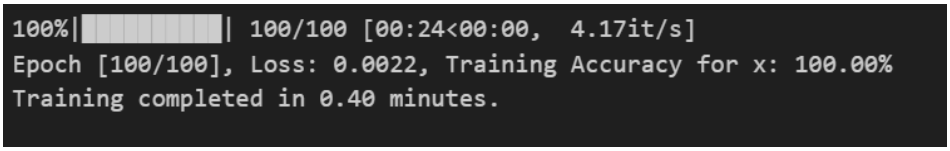
Since the modeling is multi-class classification task it is best suitable to use the **CrossEntropyLoss** function to determine the model loss. For optimizing the model **Adam** optimizer is used due to its high adaptive learning rate that adjust the learning rates as per the parameter.

1.3.1 Parameters Used

The parameter used in the training of the model are: learning rate set to 0.0001 to ensure gradual weight updates with the weight decay of 0.00001 that promotes better generalizing of the model. The number of epochs by which the model passes throughout the training process is set to be 100 to learn the model pattern.

1.3.2 Output

The model achieved 100% training and testing accuracy with the total loss of 0.0022. The model run time was 0.40 minutes as per shown below:



The Loss vs Epochs plot shows the decrease in the error during the training of the model. Following figure show the steadily reduction in the loss value:

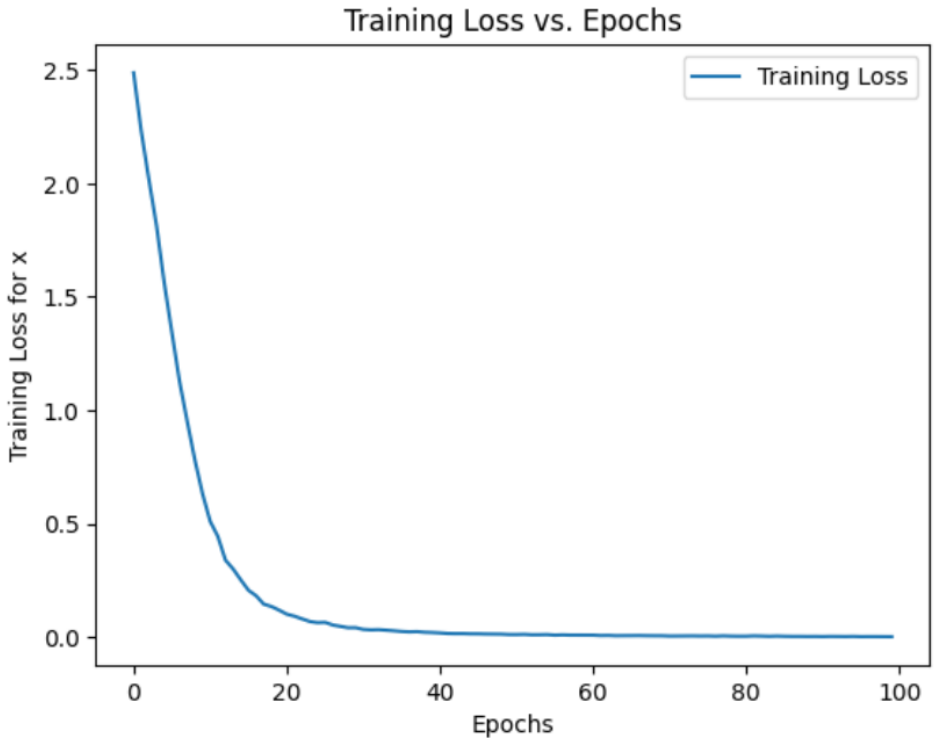


Figure 1: Loss vs Epoch Curve

The predicted values for x - coordinate are saved in the csv file named x_predicted3.csv that contains the column, contain all the predicted values of x - coordinate.

1.4 Part 2 - Creating the model for x,y,z - coordinates

To create the CNN model to predict all three coordinates, the CNN model used for part 1 is modified, although we use the same layer specification such as the input and output features the modifications done are to reduce the overfiting droupout rate is changed to 0.3 also the output layers now contain all three coordinates simultaneously. In the output layer for the classes the are now use as 5 for z as the classes for z-coordinate is 0 to 5.

Similarly as in task 1, all three layers for all three coordinates now pass through the forward function following the ReLU activation, max pooling, batch normalization.

1.5 Training of the model for x, y,z coordinates

For training the model the loss function that is used is **CrossEntropyLoss** and the optimizer used is **Adam** as the task is multi classification with requirement of high adaptive optimizer.

1.5.1 Parameters Used

The parameter used in the training of the model are: learning rate set to 0.0001 to ensure gradual weight updates with the weight decay of 0.000001 that promotes better generalizing of the model. The number of epochs by which the model passes throughout the training process is set to be 350 to learn the model pattern.

1.5.2 Output

The model achieved 99.85% for training with the training accuracy of x, y, z coordinate as 100% 100% and 99.85% with the total loss of 0.0150, whereas for testing the overall accuracy is 100% with x,y, and z accuracies as 100% for all. The accuracies are as follow:

```
100%|██████████| 350/350 [01:47<00:00,  3.27it/s]
Training completed in -28813127.69 minutes.

Accuracy for x: 100.00%
Accuracy for y: 100.00%
Accuracy for z: 99.85%
Accuracy for all (x, y, z correct): 99.85%
Epoch [350/350], Loss: 0.0150, Training Accuracy: 99.85%
```

Figure 2: Training Accuracy

```
Predicted x, y, z values saved to 'xyz_predictions7.csv'
Test Accuracy for x: 100.00%
Test Accuracy for y: 100.00%
Test Accuracy for z: 100.00%
Test Accuracy for all (x, y, z correct): 100.00%
```

Figure 3: Testing Accuracy

The Loss vs Epochs plot for all x,y,z combined loss shows the decrease in the error during the training of the model. Following figure show the steadily reduction in the loss value:

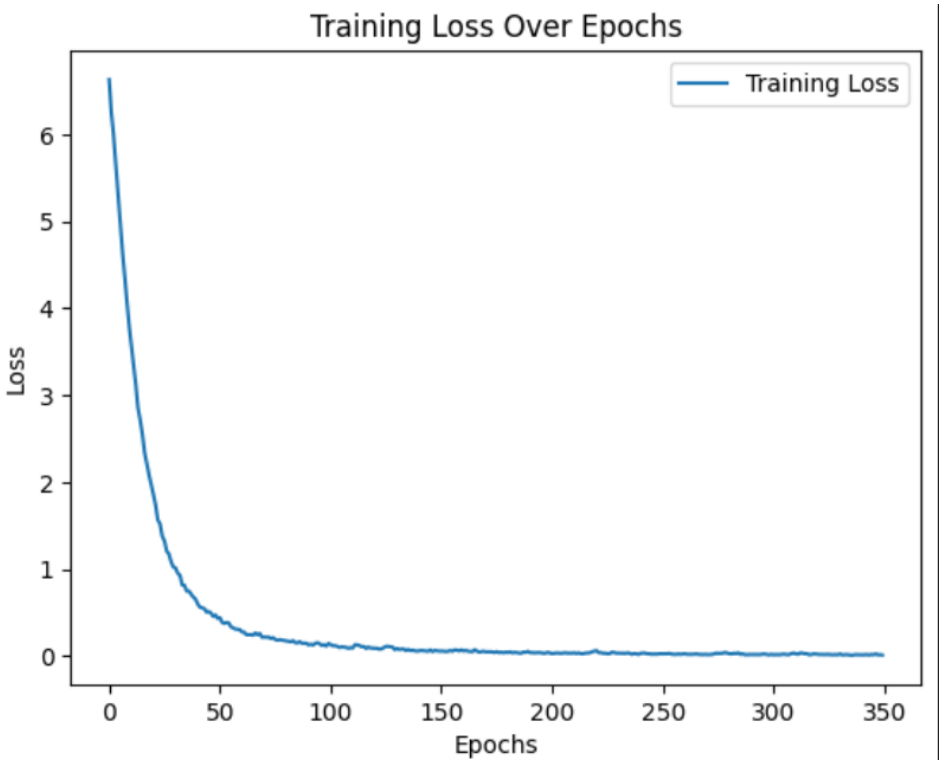


Figure 4: Testing Accuracy

The predicted values for x, y, z coordinates are saved in the csv file named xyz_predicted3.csv that contains three columns having all the predicted values of x,y,z coordinates.

2 Question 2 - Autoencoder

2.1 Loading and Pre-processing of Dataset

For the model generating the dataset is further pre-processed accordingly by:

Taking training and testing datasets containing pixel data only and flipping the images horizontally along with the specific dimensions as these will generate the mirror version of images by adding variety.

Then with the help of Pytorch DataLoader datasets are loaded ensuring the batching and shuffling.

2.2 Creating of the model

In the model the encoder and decoder functions are used that encodes the image, get the important features and the decode them to generate the mirror images.

For encoder, the three Convolutional layer are used with fitters 64,128,20 that take 1 input and 64 output feature in first layer, 64 input and 128 output features in second and 128 input and 20 output layers in third layer. In between of the layers batch norm of 128 units after layer 2 is used to normalize the model.

For Decoder, three transposed Convolutional layers are used with 20 input and 128 output features in first, 128 input and 64 output features in second and 64 input and 1 output feature in third layer.

The model needs standardizing (get to know after running model few times) for which the transformer is used that scale the pixels to the range [-1, 1] that makes the training process much smoother and increase the model feature learning.

2.3 Training of the model

The task was to reconstruct the image precisely with pixel values, therefore, for loss function **Mean Square Error (MSE)** is used, it helps to measure the pixel wise error between the input image and the reconstruced image.

For the optimization, **Adam** optimizer is used to prevent the overfitting and maintaining the steadiness of the model.

2.3.1 Size of the Image

After encoding the size of image changes as:

1. After first layer: (15 + 2 x 1 - 3) / 3 = 8, 64 x 8 x 8
2. after second layer: (8 + 2 x 1 - 3) / 3 = 4, 128 x 4 x 4
3. After Third layer: (4 + 2 x 1 - 3) / 3 = 2, 20 x 2 x 2

Therefore, after the third layer the encoded size of the image become **20 X 2 X 2**

2.3.2 Parameters Used

The parameter used in the training of the model are: learning rate set to 0.0001 to ensure gradual weight updates with the weight decay of 0.00001 that promotes better generalizing of the model. The number of epochs by which the model passes throughout the training process is set to be 5000 to learn the model pattern.

2.3.3 Output

The model output of loss indicated that after 1000 epochs the loss steadily maintains to be 0.0001 while the overall time taken by the model to train was 349.38 seconds as shown:

The Plot showing the loss during the training of the model is shown in the figure below, it depicts the steady decrease in the error rate of the loss along with the epochs:

```
Epoch 1, Loss: 0.1897
Epoch 201, Loss: 0.0008
Epoch 401, Loss: 0.0003
Epoch 601, Loss: 0.0002
Epoch 801, Loss: 0.0002
Epoch 1001, Loss: 0.0001
Epoch 1201, Loss: 0.0001
Epoch 1401, Loss: 0.0001
Epoch 1601, Loss: 0.0001
Epoch 1801, Loss: 0.0001
Epoch 2001, Loss: 0.0001
Epoch 2201, Loss: 0.0001
Epoch 2401, Loss: 0.0001
Epoch 2601, Loss: 0.0001
Epoch 2801, Loss: 0.0001
Epoch 3001, Loss: 0.0001
Epoch 3201, Loss: 0.0001
Epoch 3401, Loss: 0.0001
Epoch 3601, Loss: 0.0001
Epoch 3801, Loss: 0.0001
Epoch 4001, Loss: 0.0001
Epoch 4201, Loss: 0.0001
Epoch 4401, Loss: 0.0001
Epoch 4601, Loss: 0.0001
Epoch 4801, Loss: 0.0001
Total Training Time: 349.38 seconds
```

Figure 5: Training Output

Figure 6: Loss over epoch

whereas, after testing of the model we get the images that are reconstructed from the input images, the model has trained very well after the 5000 epochs but there is still some place for the model to

train better in order to achieve better results. Few of the examples are shown below:

Model reconstructing **Good Image** as image is a complete mirror image of input image

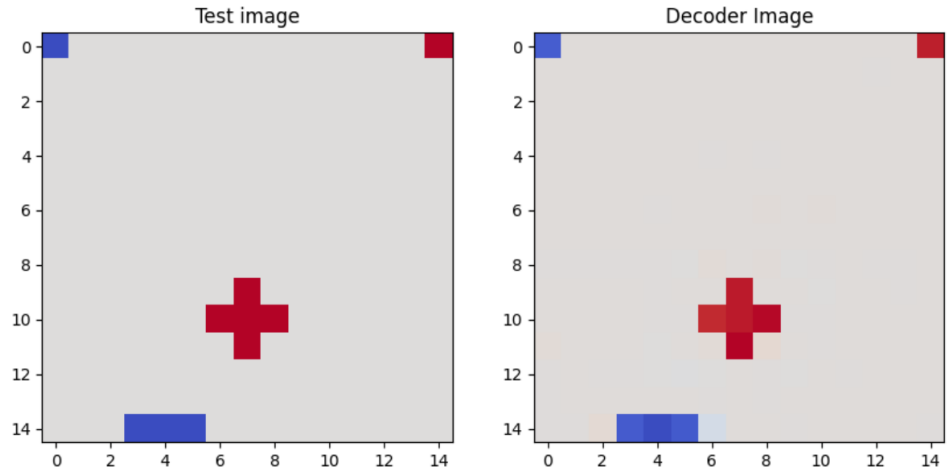


Figure 7: Good Image

Model Reconstructing **Bad Image** as image is pix elating

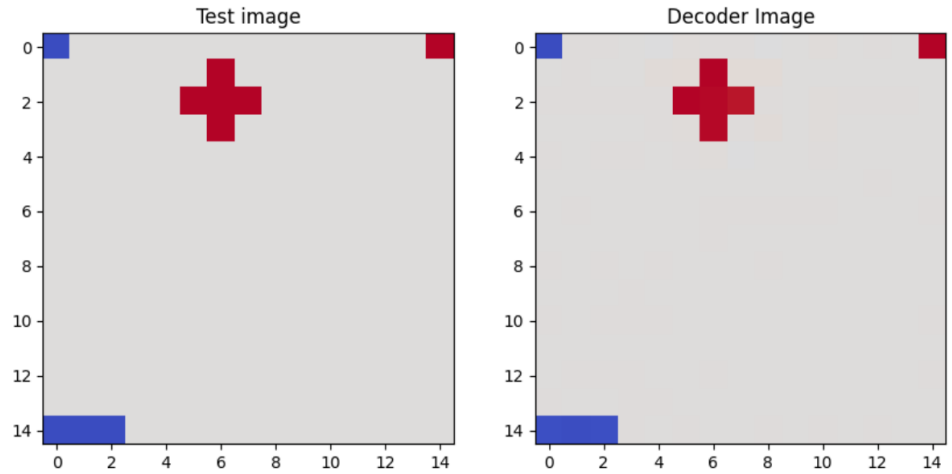


Figure 8: Bad Image

3 Question 3 - Reinforcement Learning

3.1 Creating The model

For the input dimension of the model the model accepts the state dimension dz that is the position of the ball with respect to the ball.

For the layers the model uses two fully connected layers as the input layer with 64 units produced along with the ReLU activation function. and 128 units that is doubling the first layer along with the ReLU activation function. For the output layer model produces the q-values for each and every possible action that is left, right and stay.

3.2 Training of the model

The task needs to analyse and measure the differnce in the q-values that are produces and are to be predicted. Therefore, for prediction tasks MSE is the better, hence, the model is using **Mean**

Square Error (MSE) loss function. And for the optimizer we are using **Adam** optimizer for steady and better model adaptivity.

For the model while initializing the model the level of the model is given 1 which means that the model only gets the less feature that is just the position of ball to the peddle.

3.2.1 Hyperparameters Used

Hyperparameters used for the task are configured earlier before defining the model. The discount factor used is set to 0.995 to provide high weights, the exploration rate is set to 1.0, 64 hidden layer units were provided with the learning rate of 0.001. The number os episode with which the agent was trained was given 2000 to get better rewards after learning.

3.3 Output

The **Training Reward** plot obtained after the training of the model is:

Figure 9: Rewards per epoch

By increasing the number of episodes the quantity of getting the rewards also increases.

The test average and the standard deviation obtained after testing the model with 50 episodes is 13.26 and 10.87.

```
Test Average Reward: 13.26
Test Reward Standard Deviation: 10.873472306489772
```

The plot is plotted to visualize the test rewards per epochs for 50 episodes:

Figure 10: Rewards per epoch

3.4 Option on level - 2 RL

I believed that if the level of the Task is increase to 2 it will impact the learning of the model. As now the model can observe two state values that are y - coordinate of the ball and the x - coordinate of the ball to the paddle. This is will increase the predicting ability of RL agent and help it to analyse the future accuracies of hitting the ball.

4 Question 4 - Reinforcement Learning (Level 3)

The model follows the same architecture used to create the model in task 3 but in task 4 the agaent now is able to play MiniPong on level 3.

In the task now the agent has more values that are x,y,z coordinates of ball to the paddle.

For the layers again the model uses two fully connected layers as the input layer with 64 units produced along with the ReLU activation function. and 128 units that is doubling the first layer along with the ReLU activation function. For the output layer model produces the q-values for each and every possible action that is left, right and stay.

4.1 Training of the model

The model will train in the same formate but now the model while initializing, the level of the model are provide as 3 which means that the model only gets the all the feature information in 3D that is just the position of ball in all x,y,z coordinates to the peddle.

Also the number of states and the number of action space are specified as 3 as well. The model here is trained with greedy policy to balance the exploitation in the model.

4.2 Output

From the plot of training rewards vs the episodes it is shown that after 4000 episodes we got the average rewards **More than 300**

Figure 11: Rewards per epoch

The test average and the standard deviation obtained after testing the model with 50 episodes is 42.38 and 56.35.

```
Test Average Reward: 42.38
Test Reward Standard Deviation: 56.3552623984664
```

The plot is plotted to visualize the test rewards per epochs for 50 episodes:

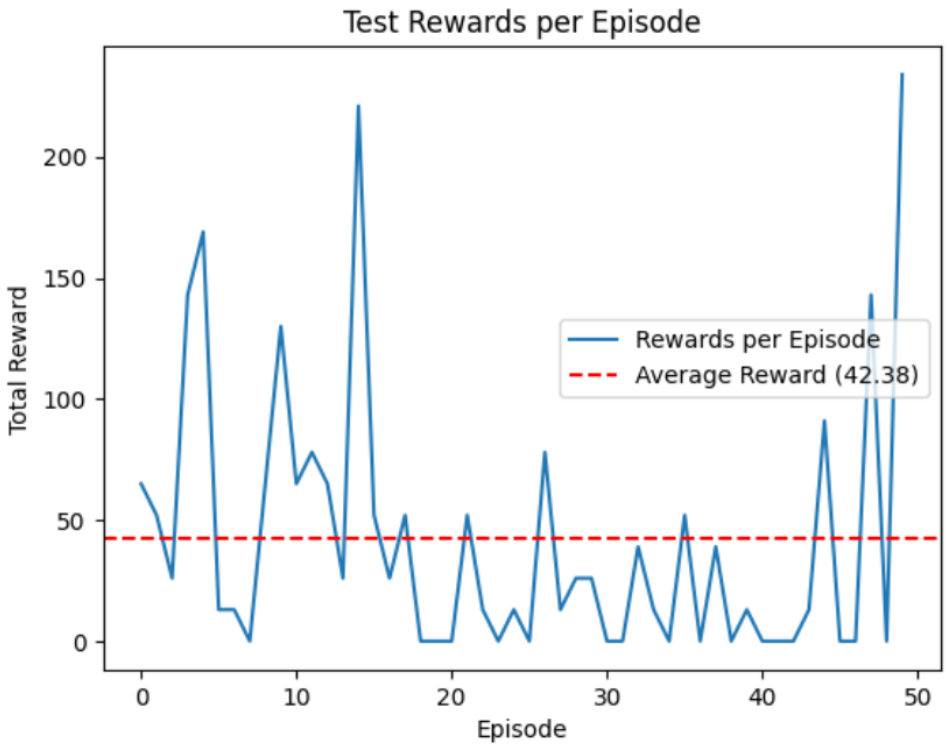


Figure 12: Rewards per epoch