CIS 522

Homework 3: Computer Vision

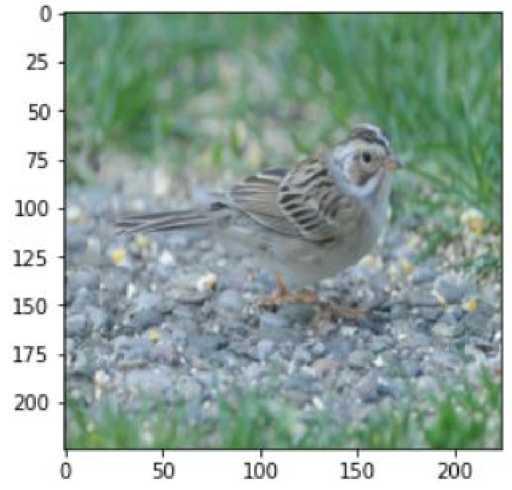
2/13/2020

Q1

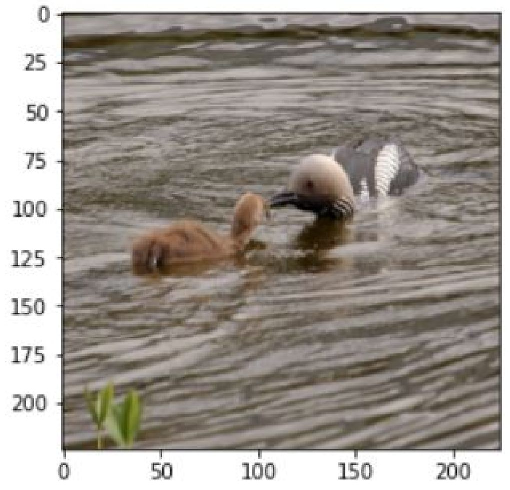
1.a

Training Set: 5 Images

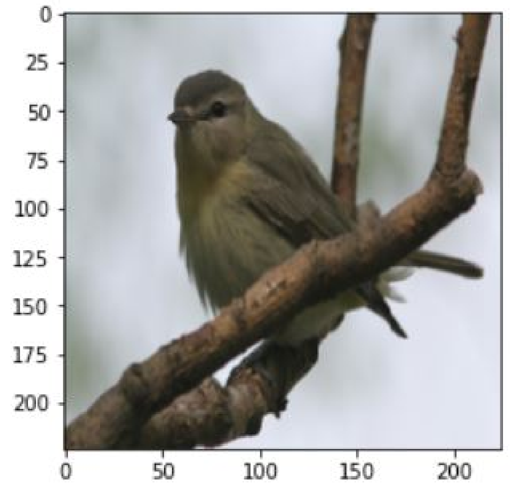
```
torch.Size([32, 3, 224, 224])
```



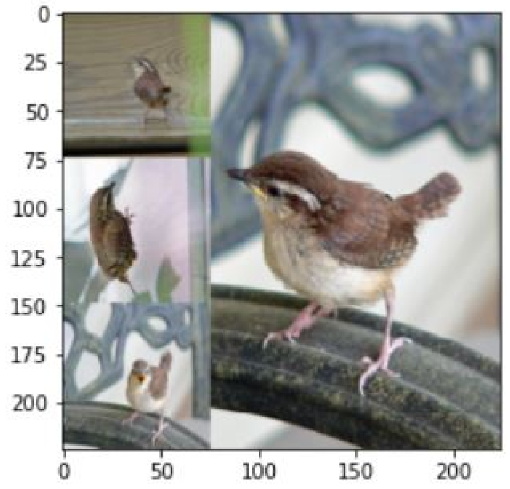
```
torch.Size([32, 3, 224, 224])
```



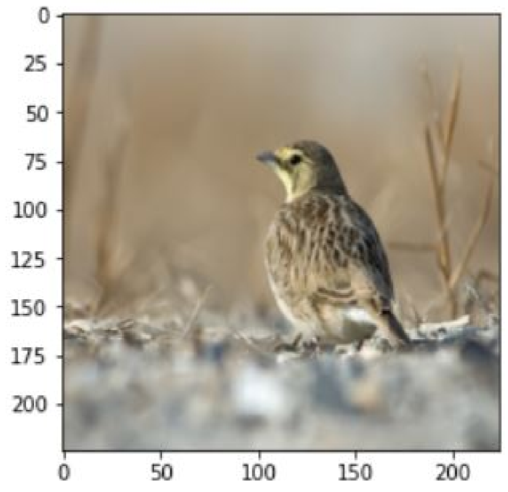
```
torch.Size([32, 3, 224, 224])
```



```
torch.Size([32, 3, 224, 224])
```

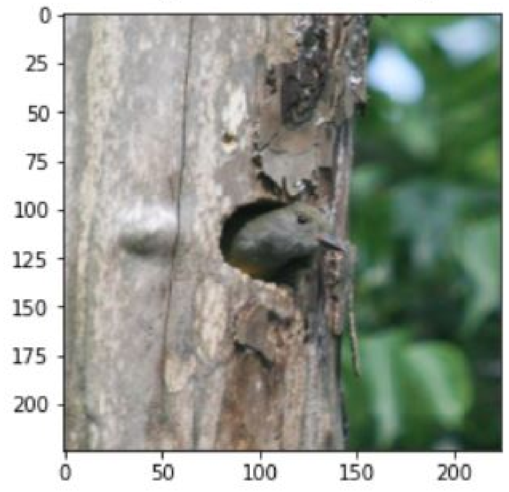


```
torch.Size([32, 3, 224, 224])
```

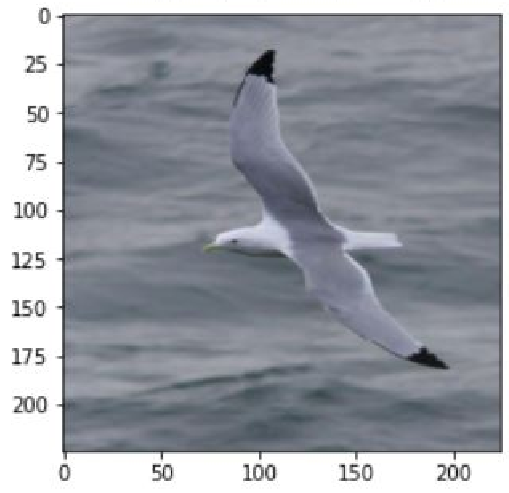


Validation Set: 5 Images

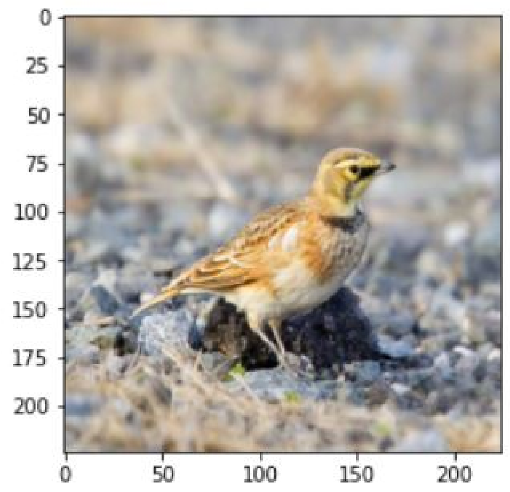
```
torch.Size([32, 3, 224, 224])
```



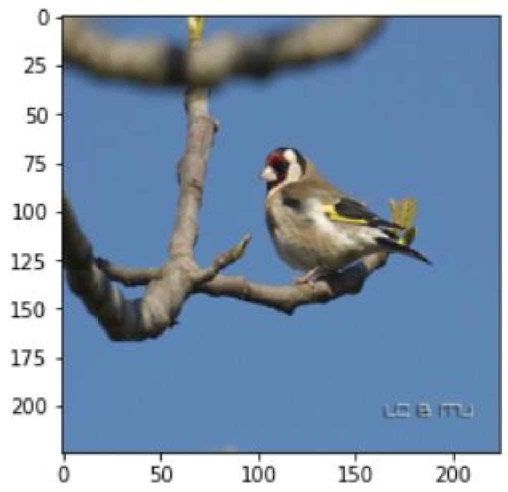
```
torch.Size([32, 3, 224, 224])
```



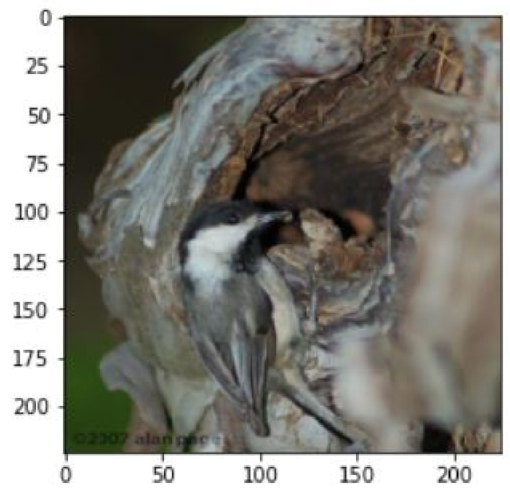
```
torch.Size([32, 3, 224, 224])
```



```
torch.Size([32, 3, 224, 224])
```

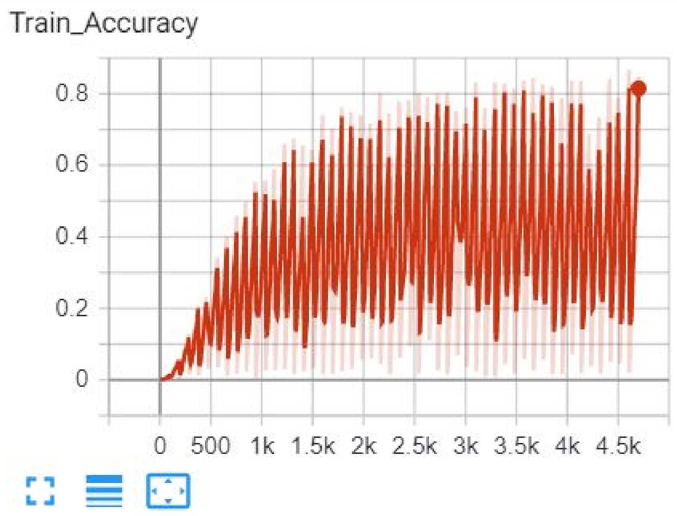


```
torch.Size([32, 3, 224, 224])
```

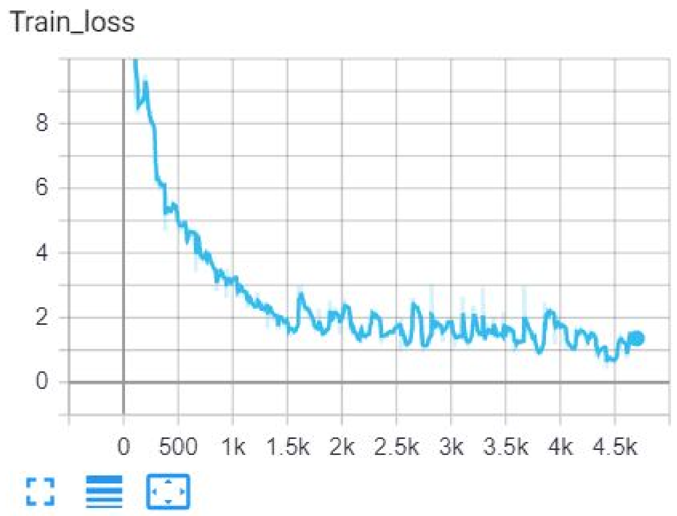


1.b

LOGREG TRAINING ACCURACY - 84.74 %



LOGREG TRAINING LOSS - 1.3749

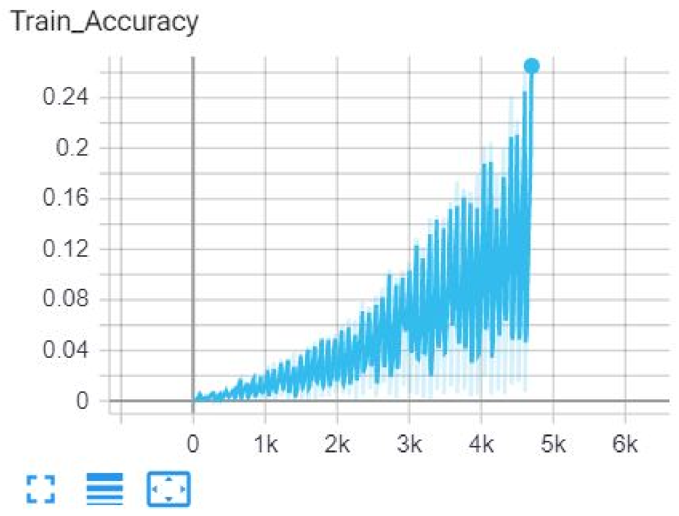


LOGREG VALIDATION ACCURACY - 3.6048 %

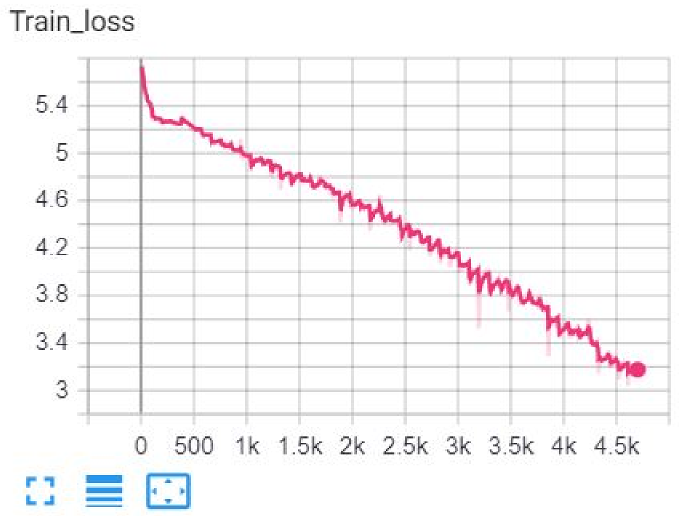
The Dataloaders were created with a batch size of 32 and shuffling was enabled and number of workers were initialized to 4. We used Cross Entropy Loss and Adam optimizer with a learning rate = 0.0001 and weight decay = 1e-4.

1.c

MLP TRAINING ACCURACY - 27.601 %



MLP TRAINING LOSS - 3.1764

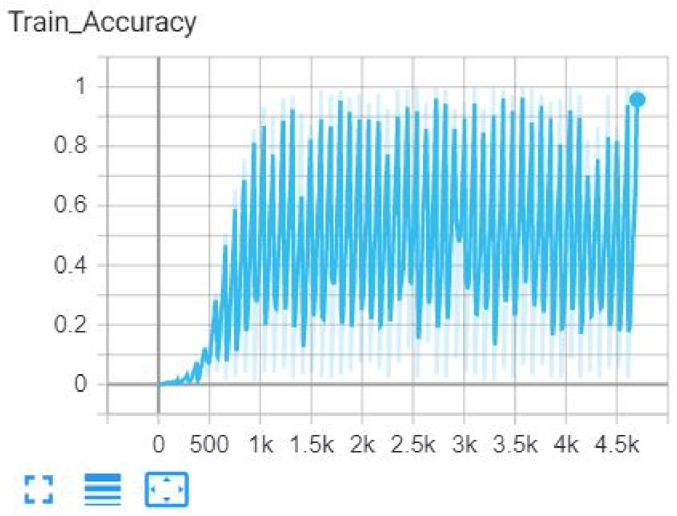


MLP VALIDATION ACCURACY - 4.4726 %

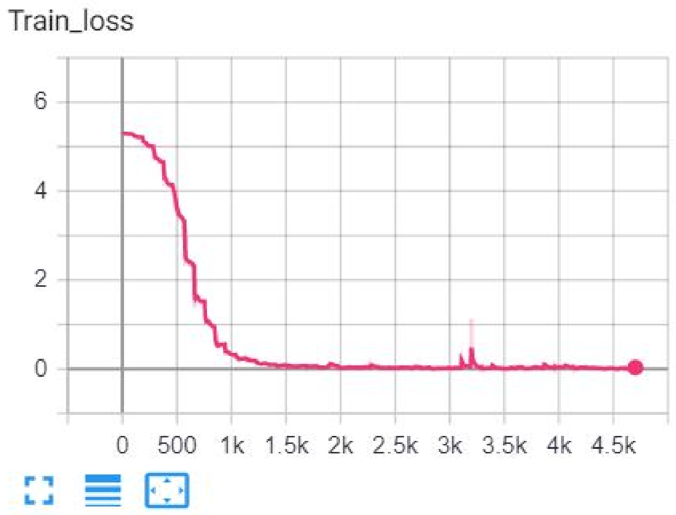
A 3 layer MLP was created with 150528 nodes at the start and ended with 195 nodes and leaky RelUs were used as activation functions.
The Dataloaders were created with a batch size of 32 and shuffling was enabled and number of workers were initialized to 4. We used Cross Entropy Loss and Adam optimizer with a learning rate = 0.0001 and weight decay = 1e-4.

1.d

CNN TRAINING ACCURACY - 99.7658 %



CNN TRAINING LOSS - 0.0365



CNN VALIDATION ACCURACY - 5.5407 %
4 Convolutional layers were used in the CNN network and max pooling of (2,2) was done on all 3 layers except the 2nd one. Relu functions were used as activation functions.
The Dataloaders were created with a batch size of 32 and shuffling was enabled and number of workers were initialized to 4. We used Cross Entropy Loss and Adam optimizer with a learning rate = 0.0001 and weight decay = 1e-4.

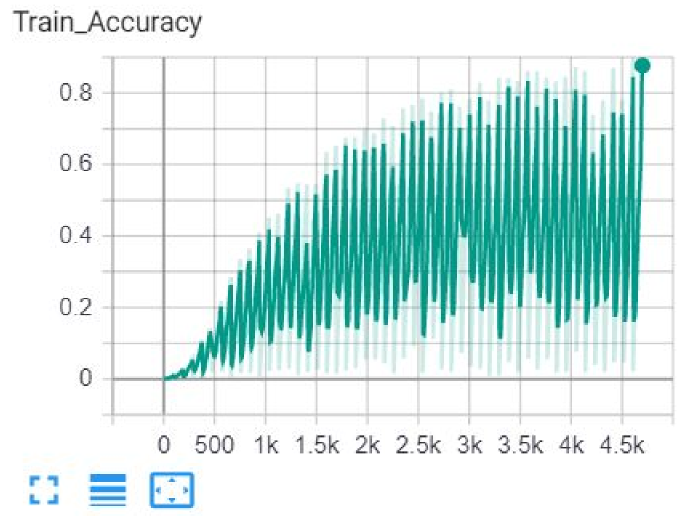
1.e

All the three models produced low accuracies and out of them CNN performed well with a validation accuracy of 5.54 %. All the three models were made to train for 50 epochs.

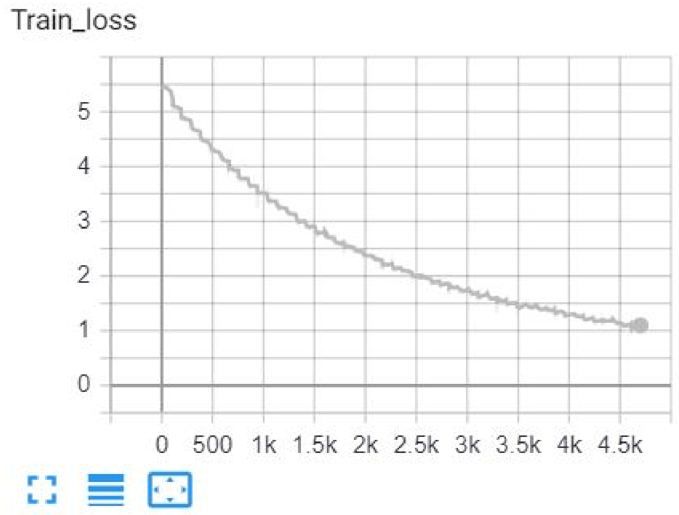
Model	Training accuracy	Training loss	Testing accuracy
LogReg	84.74 %	1.3749	3.6048 %
FNN	27.601 %	3.1764	4.4726 %
CNN	99.7658 %	0.0365	5.5407 %

Q2

TRANSFER TRAINING ACCURACY - 91.368 %



TRANSFER TRAINING LOSS - 1.109



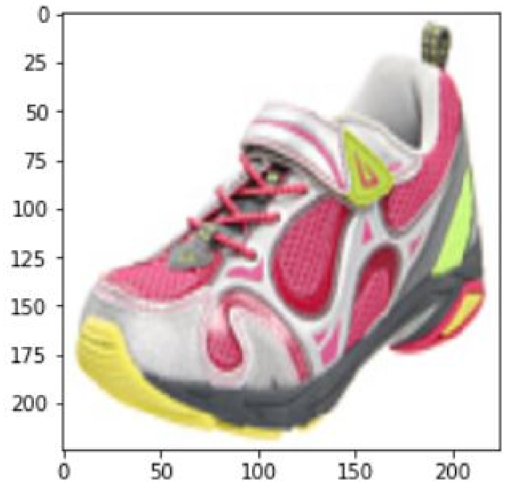
TRANSFER VALIDATION ACCURACY - 43.1241 %

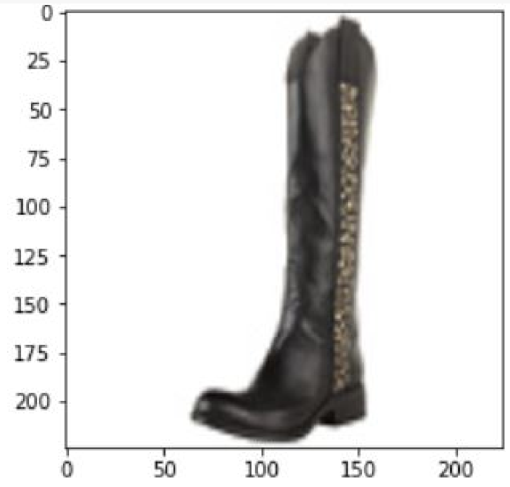
Q3

3.a

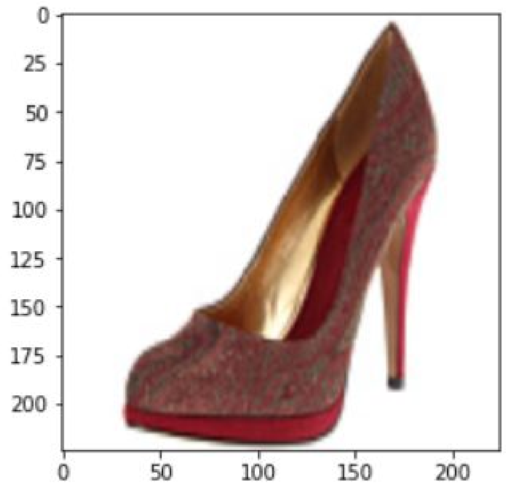
VAE pictures

```
torch.Size([32, 3, 224, 224])
```

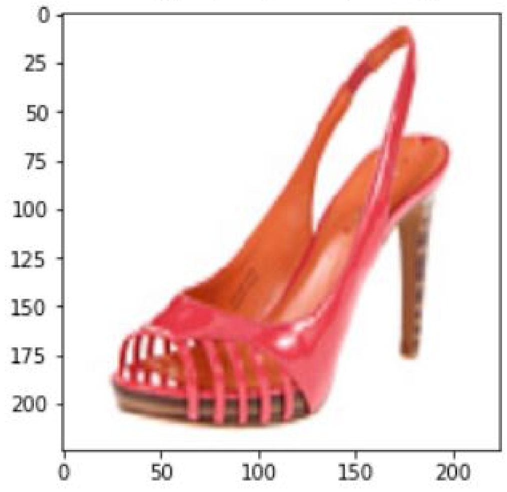


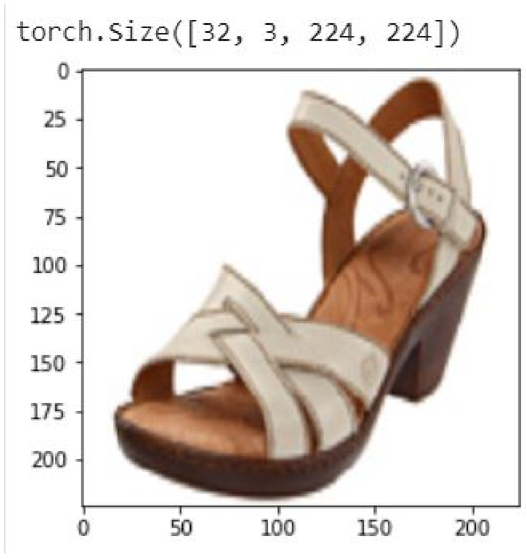


```
torch.Size([32, 3, 224, 224])
```



```
torch.Size([32, 3, 224, 224])
```



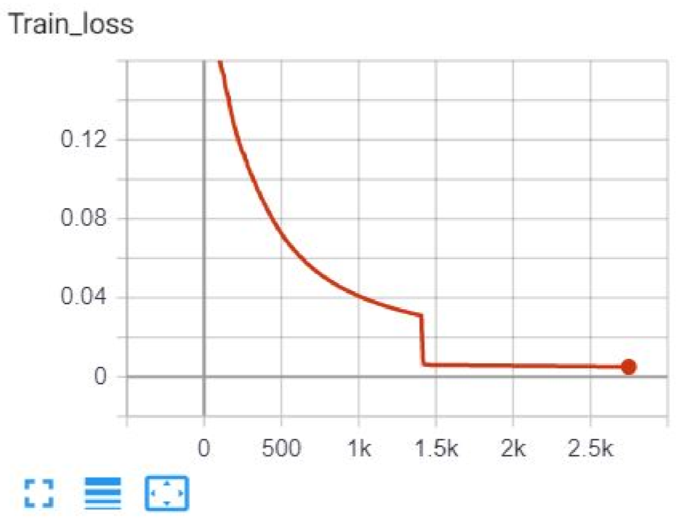


```
torch.Size([32, 3, 224, 224])
```

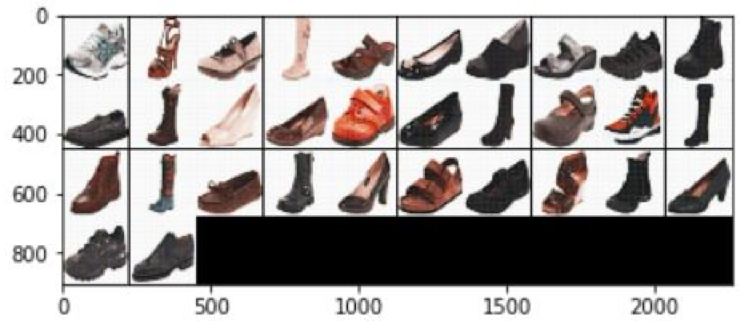
3.b

The train and test split was done in 90-10 division of the dataset. More ratio was given to the training set to increase the performance from training and thus producing better final images.

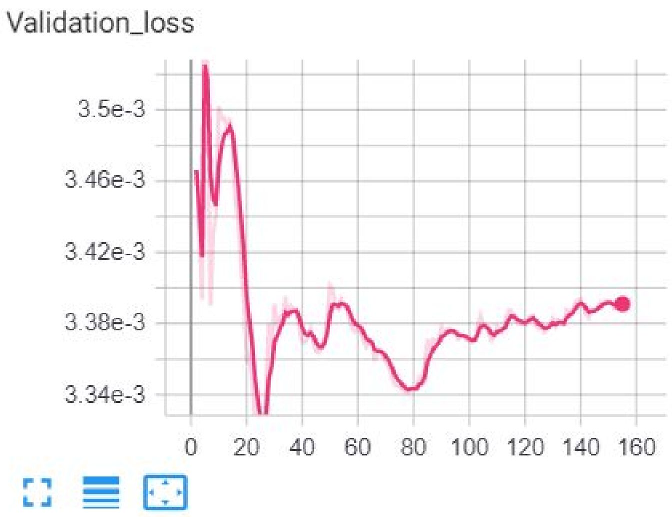
Training loss- 0.00502



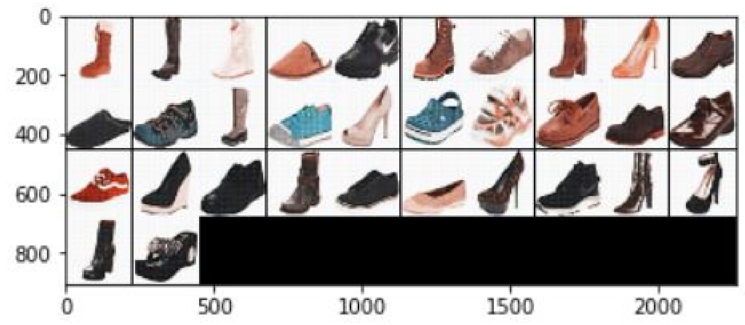
Input Images & Output Images for Training CAE



Validation loss- 1.0034

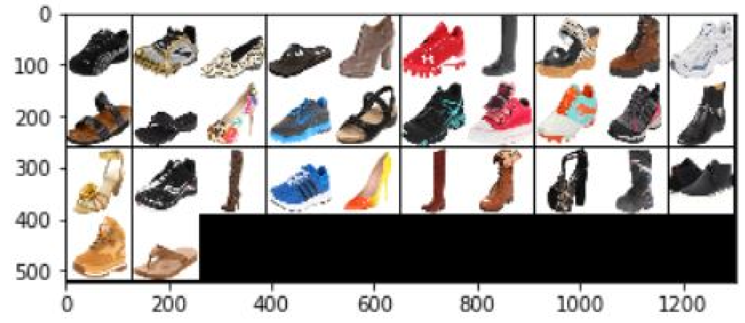


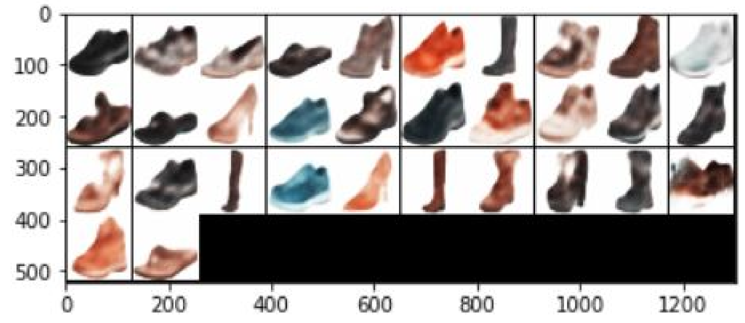
Input Images & Output Images for Validation CAE

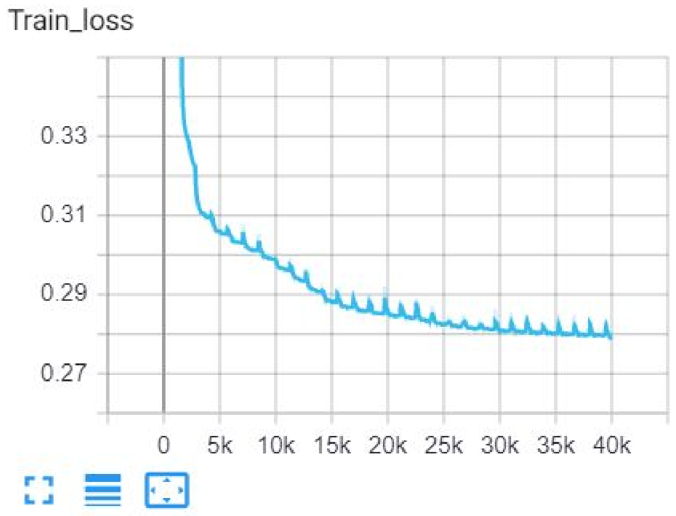


3.c

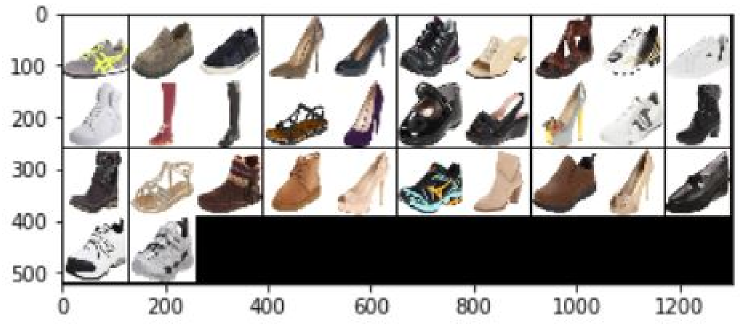
Input Images & Output Images for Training VAEs

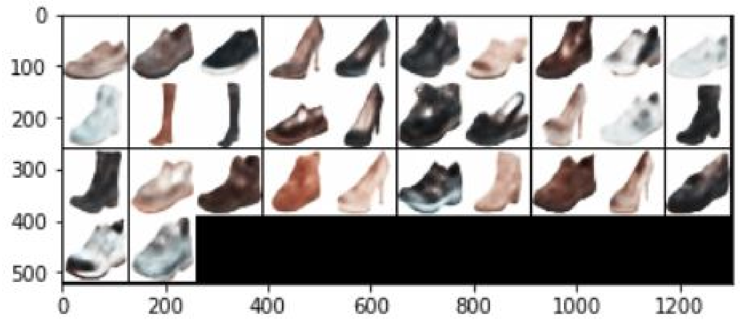




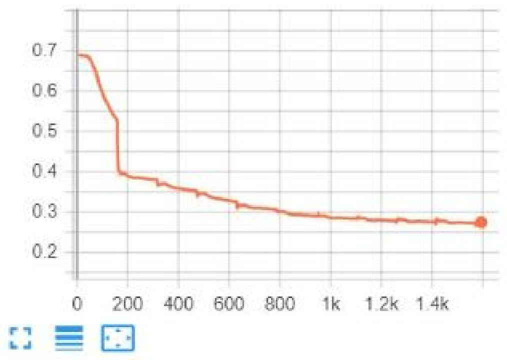


Input Images & Output Images for Validation VAEs

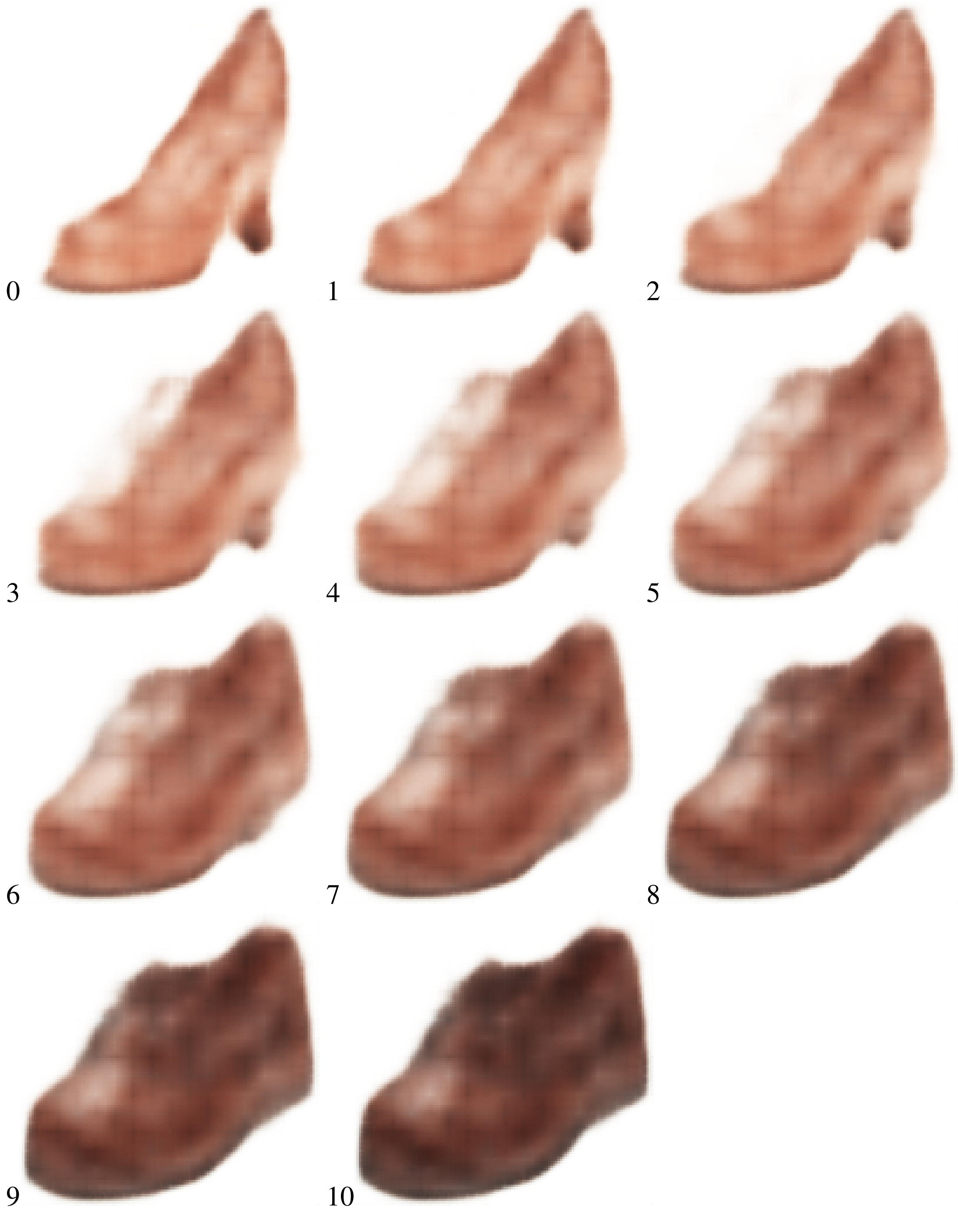




Validation loss



3.d



100 images were added in a video with a time period of 0.25 seconds for every image and a total of 25 seconds.

Extra credit video: https://www.youtube.com/watch?v=Et1ZBNJ7LsQfeature=youtu.be

Q4

4.b

The generator network:

1. ConvTranspose2d(100,512,4,1,0,bias=False) - converts 1x100 input into 4x4x512 output but upsampling. This is followed by Batch Normalization network and ReLU activation function.

2. ConvTranspose2d(512,256,4,2,1,bias = False) - upsampling network which converts 4x4x512 into 8x8x256. This is followed by Batch Normalization and ReLU activation.

3. ConvTranspose2d(256,128,4,2,1,bias=False) - upsampling network which converts 8x8x256 into 16x16x128. This is followed by Batch Normalization and ReLU activation function.

4. ConvTranspose2d(128,64,4,2,1,bias=False) - upsampling network which converts 16x16x128 into 32x32x64. This is followed by Batch Normalization and ReLU activation.

5. ConvTranspose2d(64,3,4,2,1.bias=False) - upsampling network which converts 32x32x64 input into 64x64x3 image. This is followed by sigmoid function to make the pixel values between [0,1].

The Generator network converts an input 1d noise vector into a 64x64x3 image by upsampling it, in other words, Transpose convolving it. Thus to achieve this we use Transpose convolution layers. The Bstch norm is to normalize all the channels of the output.

4.c

The Discriminator Network: We use the convolution layers in discriminator as the network should learn the important features from the input image and predict if it's a real or a fake image. The network used in the project is as follows:

1. Conv2d(3,64,4,2,1,bias=False) - This is a convolution layer which converts the input 64x64x3 image into 32x32x64 output.

2. Conv2d(64,128,4,2,1,bias=False) - This is a convolution layer that converts 32x32x64 input to 16x16x128. This is follow by batch norm and thern LeakyRELU(0.2) activation function.

3. Conv2d(128,256,4,2,1,bias=False) - This is a convolution layer that converts 16x16x128 input to 8x8x256. This is follow by batch norm and thern LeakyRELU(0.2) activation function.

4. Conv2d(256,512,4,2,1,bias=False) - This is a convolution layer that converts 8x8x256 input to 4x4x512. This is follow by batch norm and then LeakyRELU(0.2). activation function. final conv layer that covers 4x4x512 input into single value. This is followed by sigmoid function to map this to probability.

4.d

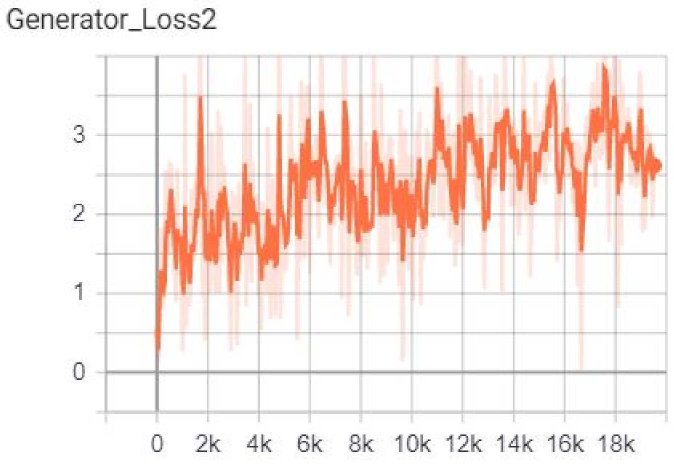
The Ground truth value for Fake data is 1 here since the Generator should produce images such that it makes the discriminator think that these generated images are real.

4.e

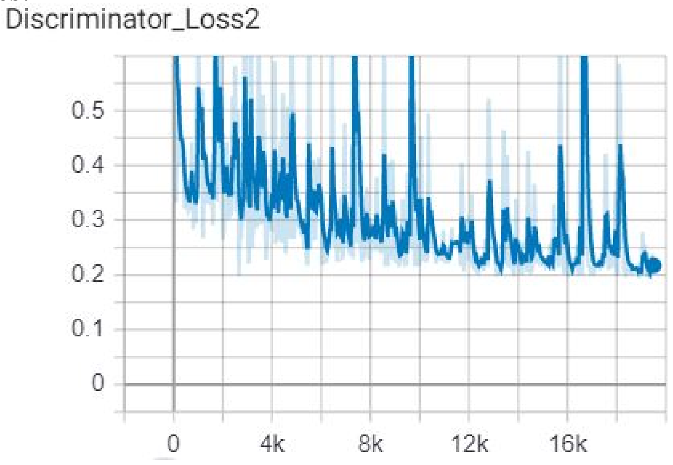
The detach function on generator detaches it from the computational graph of the discriminator thus preventing the backpropagation of gradients. If we don't detach the the generator then it gets trained on both the discriminator loss and it's own loss function.

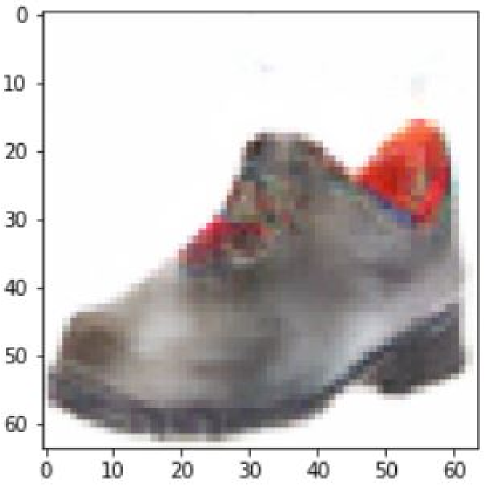
4.f

We are using Adam optimizer with learning rate=0.0002 to optimize both Discriminator and Generator. *Generator Loss:*

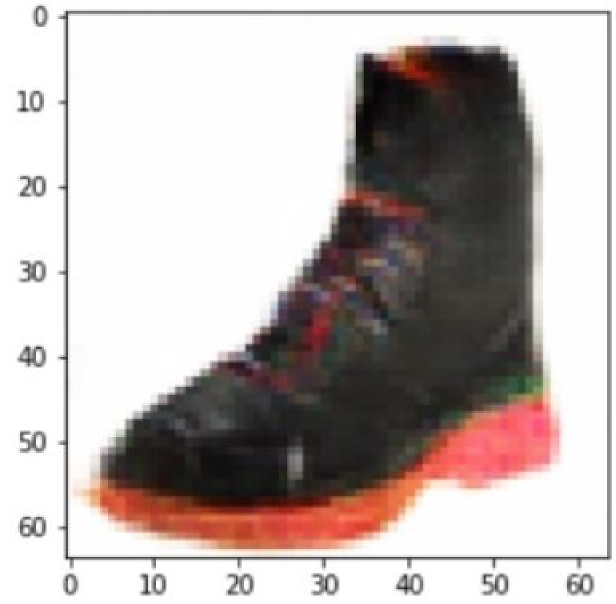
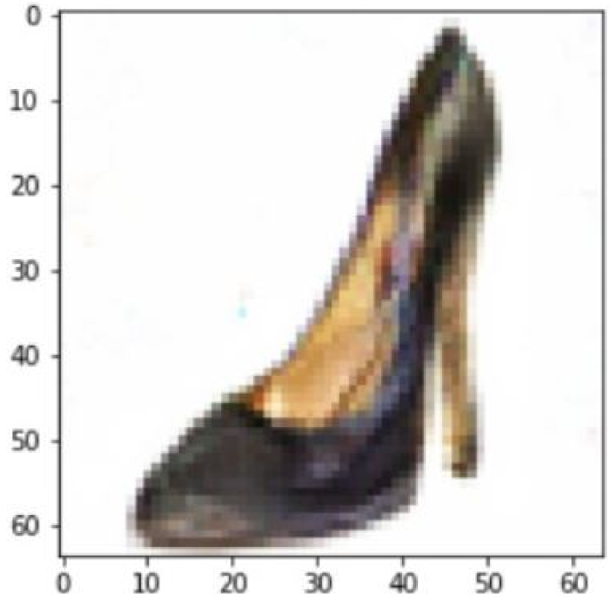
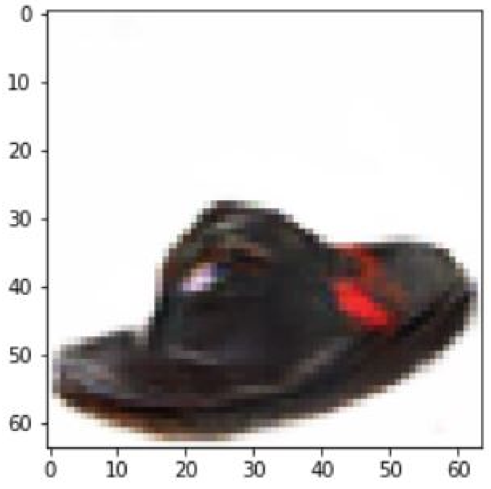
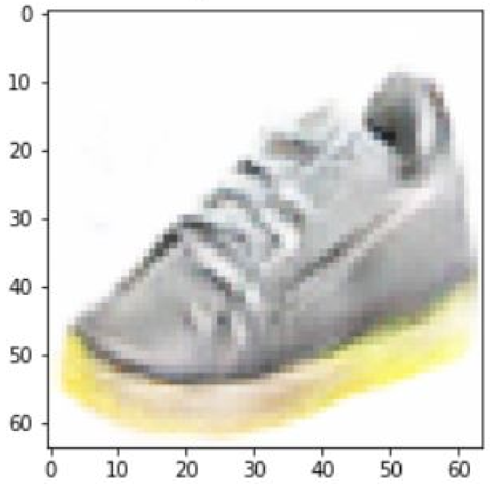


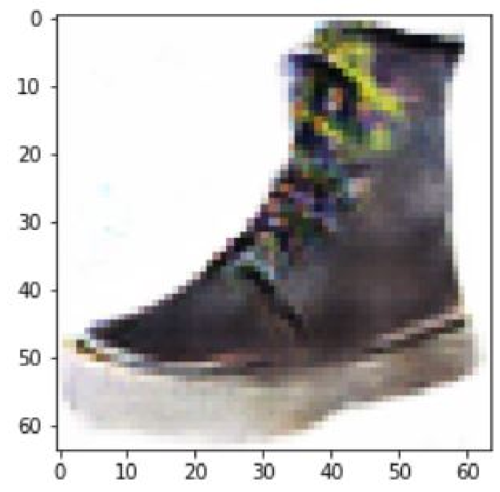
Discriminator Loss:





Outputs of GANS:


0
10
20
30
40
50
60
0
10
20
30
40
50
60