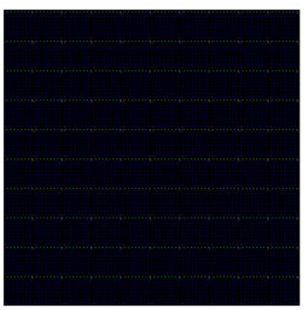
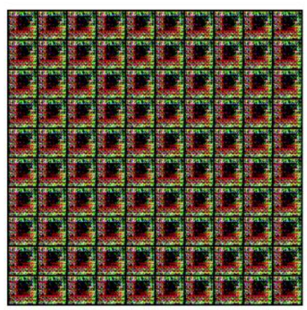
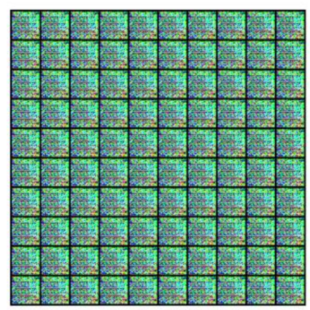
GNR 638 Assignment 3

Amruta Mahendra Parulekar (20d070009)
Hemant Hajare (20d070037)

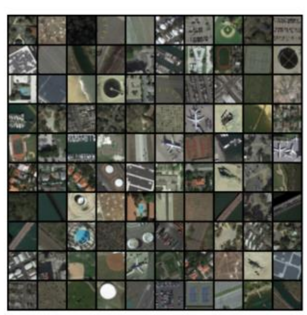
Question 1

Customizing and checking the code and the results of InfoGAN on the PatternNet dataset.

The code already had the facility to train the model for an external dataset. We replaced the path of the dataset with the downloaded PatternNet dataset. The learning rate was reduced by 10 times and the number of epochs to 10 instead of the former 100 so that training time is less. The generated images after 1^{st} , 5^{th} and 10^{th} epoch respectively are shown below. Due to less training, the images are not up to the mark. However, given a sufficient number of epochs we can get the good quality of generated images. Each sub-block is a different image in development

A typical training images looks like as shown below



The training log as generated using the code is given below.

```
Random Seed:  1123
cuda:0  will be used.
Generator(
  (tconv1): ConvTranspose2d(228, 448, kernel_size=(2, 2), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(448, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (tconv2): ConvTranspose2d(448, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (tconv3): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (tconv4): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (tconv5): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
)
Discriminator(
  (conv1): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
DHead(
  (conv): Conv2d(256, 1, kernel_size=(4, 4), stride=(1, 1))
)
QHead(
  (conv1): Conv2d(256, 128, kernel_size=(4, 4), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv_disc): Conv2d(128, 100, kernel_size=(1, 1), stride=(1, 1))
  (conv_mu): Conv2d(128, 1, kernel_size=(1, 1), stride=(1, 1))
  (conv_var): Conv2d(128, 1, kernel_size=(1, 1), stride=(1, 1))
)
------------------------
Starting Training Loop...
Epochs: 10
Dataset: CelebA
Batch Size: 128
Length of Data Loader: 238
------------------------
[1/10][100/238]	Loss_D: 0.0574	Loss_G: 26.5756
[1/10][200/238]	Loss_D: 0.0015	Loss_G: 27.7692
Time taken for Epoch 1: 178.95s
[2/10][100/238]	Loss_D: 0.0005	Loss_G: 26.0174
[2/10][200/238]	Loss_D: 0.0004	Loss_G: 24.0044
Time taken for Epoch 2: 175.24s
[3/10][100/238]	Loss_D: 0.0003	Loss_G: 20.3547
[3/10][200/238]	Loss_D: 0.0254	Loss_G: 19.6168
Time taken for Epoch 3: 173.52s
[4/10][100/238]	Loss_D: 0.0003	Loss_G: 16.0425
[4/10][200/238]	Loss_D: 0.0007	Loss_G: 16.3495
Time taken for Epoch 4: 172.93s
[5/10][100/238]	Loss_D: 0.0001	Loss_G: 13.3284
[5/10][200/238]	Loss_D: 0.0001	Loss_G: 12.8254
Time taken for Epoch 5: 172.43s
[6/10][100/238]	Loss_D: 0.0001	Loss_G: 12.3265
[6/10][200/238]	Loss_D: 0.0000	Loss_G: 12.3101
Time taken for Epoch 6: 173.72s
[7/10][100/238]	Loss_D: 0.0001	Loss_G: 12.3410
[7/10][200/238]	Loss_D: 0.0000	Loss_G: 12.3920
Time taken for Epoch 7: 172.05s
[8/10][100/238]	Loss_D: 0.0001	Loss_G: 10.4403
[8/10][200/238]	Loss_D: 0.0000	Loss_G: 11.9432
Time taken for Epoch 8: 172.10s
[9/10][100/238]	Loss_D: 0.0000	Loss_G: 12.2418
[9/10][200/238]	Loss_D: 0.0000	Loss_G: 12.8193
Time taken for Epoch 9: 174.70s
[10/10][100/238]	Loss_D: 0.0000	Loss_G: 11.5085
[10/10][200/238]	Loss_D: 0.0000	Loss_G: 15.6851
Time taken for Epoch 10: 178.54s
-------------------------------------------------
Training finished!
Total Time for Training: 29.09m
-------------------------------------------------
```

Question 2

The loss function if the relation between the output and input variables can be described through a Poisson distribution.

We implemented a Variational Autoencoder (VAE) combined with a classifier using TensorFlow and Keras. The goal was to explore how different regularization weights affect the performance of the VAE and its associated classifier on the MNIST dataset.

We utilized VAE, a type of generative model, which learns a low-dimensional representation of data while generating new data points. This is achieved by encoding input data into a latent space and decoding it back to the original data format. Additionally, we trained a classifier on top of the learned latent representations to classify digits.

Loading and Preprocessing Data

We loaded the MNIST dataset, which consists of 28x28 grayscale images of handwritten digits from 0 to 9. Before feeding the data into the models, we normalized pixel values to be within the range [0, 1] and reshaped the images to include a channel dimension.

VAE Architecture

The VAE architecture consists of two main components: an encoder and a decoder. The encoder takes input images and produces the mean and log variance of the latent distribution. The decoder generates reconstructed images from sampled latent variables. We implemented these components using Keras functional API.

Classifier Architecture

We designed a simple classifier that takes the latent representations from the VAE's encoder as input and predicts the digit class. This classifier is a fully connected neural network with ReLU activation functions and softmax output.

Training the VAE

We trained the VAE with different regularization weights (0.001, 0.01, and 0.1) to explore their effects on the learned latent representations. We used binary cross-entropy loss and the Adam optimizer for VAE training. Additionally, we added a regularization term to the VAE loss function to encourage the latent distribution to be close to a standard normal distribution.

Freezing Encoder and Training the Classifier

After training the VAE, we froze its encoder to prevent further training. We then trained the classifier on top of the frozen encoder's latent representations. The classifier was trained using sparse categorical cross-entropy loss and the Adam optimizer.

Testing and Visualization

We evaluated the trained classifier's performance on the test set. Furthermore, we visualized the reconstructed images generated by the VAE for a subset of test images to observe the quality of reconstructions.

Results

We analyzed the test loss and accuracy of the classifier for each regularization weight. Additionally, we observed the quality of reconstructed images. The performance metrics and visualizations helped us understand the impact of different regularization weights on the VAE and classifier performance.

Conclusion

We gained insights into how regularization affects VAEs and their associated classifiers. We observed that higher regularization weights could lead to better generalization but may also affect the quality of reconstructed images. This study contributes to understanding the trade-offs involved in training VAEs for generative modeling and classification tasks.