Final Project Report

Name: Mohaiminul Al Nahian

RIN: 662026703

Course No.: ECSE 6850

1. Introduction:

In this final project our goal is to implement a conditional Generative Adversial Network (cGAN) that can generate human faces with certain conditions applied to the appearance of the generated images. The traditional GANs have the limitation that the user does not have any control over the appearance of the generated images once the model has been trained. Whereas, in cGAN we define certain attributes about how the images should look like while training the network. The training real images have some labels, i.e. certain attributes associated with it. While training the network, we generate our random inputs and add the attribute features with it so that our trained model can generate images with given attributes set by the user once the training has been completed. For this task we have used a Conditional GAN network. cGANs were first introduced by Dosovitskiy et al. [1]. The main idea is to train a generative network and a discriminative network at the same time. During the training, the task of the generator is to create fake images that look like real in order to fool the discriminator to assume at as a real image, while the task of the discriminator is to learn how to differentiate between a real image and a fake image. Over time, both the generator and the discriminator should try to become better than the other. With careful construction of the network the setting of different hyperparameters, there comes a point when the discriminator can no longer differentiate between real and fake images and we conclude that the model has converged. Then we use the trained generator network to generate fake images. In our case we have used the cGAN to generate human faces having different attributes. The attributes selected was Black hair, Male, Oval Shaped Face, Smiling, Young. In the generated images, one or more of the attributes are present based on the user input during the generation process. We have used a modified version of the CelebA dataset [2] which contains 202,559 images with 40 attributes. In our case, the dataset image size was cropped to 64x64x3 sizes and we had to consider 5 out of 40 attributes for conditional information. We design a discriminator and a generator network and train these using Stochastic Gradient Descent. We calculate the performance of the model with respect to the metrics called Frechet Inception Distance (FID) [3] and Inception Score (IS) [4] and get fairly good results as compared to the minimum threshold set by instructions of this project.

The rest of the report is organized as follows: We describe the method used to implement the cGAN with mathematics, then we show the experiments and the results that we get from our trained network and we analyze different aspects of our experimentation and the reported result. Finally we conclude the report with a conclusion.

2. Method:

2.1 Problem Setting:

The cGAN consists of two networks, a generator G and a discriminator D. The generator G takes a noise vector generated from a Gaussian distribution p(z) = N(O^{zx1}, I^{zxz}) as an input and also a set of attributes as another input. The output of the generator is an image with same dimension as the real images. The discriminator D takes an image as an input whether it being a real or fake, generated by the generator. The output of the discriminator is a probability denoting the probability of the input image as being real or fake. The ultimate goal of the discriminator is to reliably distinguish between real and fake image, whereas the goal of our generator is to create a realistic image that is close to a real image and in turns, tricking the discriminator to assume it to be real. Discriminator D and generator G play a max-min game. we assume the joint distribution of data to be p(x,y) we can define the training objective function as

$$V(\Theta_G, \Theta_D) = E_{x,y\ p(x,y)}[log(D(x, y; \theta_D)] + E[log(1 - D(G(z, y; \Theta_G), y; \Theta_D)] \quad (1)$$

$$= \frac{1}{2n}[\sum_{i=1}^{n} log(D(x_i, y_i; \Theta_D)) + \sum_{i=1}^{n} log(1 - D(G(z_i, y_{i;G}), y_{i;D}))] \quad (2)$$

We have to optimize the parameters of the discriminator and generator. The optimization is a minmax game defined as the following

$$\Theta_G^*, \Theta_D^* = argmin(\Theta_G)max(\Theta_D)V(\Theta_G, \Theta_D)$$

The training or optimization of the objective function is carried out in 2 steps. First, update the discriminator parameters by gradient ascend while not changing the generator parameter. Then the generator parameters are updated based by doing a gradient descend on them subject to new discriminator parameters. Mathematically we can write,

$$\Theta_D^{r+1} = argmax(\Theta_D^r)V(\Theta_G, \Theta_D)$$
$$= argmax(\Theta_D)\frac{1}{2n}[\sum_{i=1}^{n} log(D(x_i, y_i; \Theta_D)) + \sum_{i=1}^{n} log(1 - D(G(z_i, y_i; \Theta_G), y_i; \Theta_D))]$$
$$= \Theta_D^r + \lambda^D \nabla_{\Theta_D} \frac{1}{2n}[\sum_{i=1}^{n} log(D(x_i, y_i; \Theta_D)) + \sum_{i=1}^{n} log(1 - D(G(z_i, y_i; \Theta_G), y_i; \Theta_D))]$$

$$\Theta_G^{r+1} = argmin(\Theta_G^r)V(\Theta_G, \Theta_D^{r+1})$$
$$= argmin(\Theta_G)\frac{1}{n}[\sum_{i=1}^{n} log(1 - D(G(z_i, y_i;_G), y_i; \Theta_{D^{r+1}}))]$$
$$= \Theta_G^r - \lambda^G \nabla_{\Theta_G} \frac{1}{n}[\sum_{i=1}^{n} log(1 - D(G(z_i, y_i; \Theta_G), y_i; \Theta_{D^{r+1}}))]$$

These two steps are repeated until convergence.

2.2 Model Architecture:

The architectures used in our experiment has a discriminator and a generator network. The generator we have used is almost similar to the example architecture given in the instructions. The only difference is that we have used batch normalization after the convolution layers. The input shape in deconv1 is 1x1x100 noise vector and deonv2 is 1x1x5 attribute vector. Deconv1 and deconv2 output is 4x4x512 where last dimension represents the number of filters. Then we concatenate these two outputs to give us a vector of 4x4x1024 which is fed to the deconv3 layer. Its output has a shape of 8x8x512. This gives an output of 16x16x256 from deconv4. This output is fed to deonv5 where it gives a feature set of 32x32x128. Finally it is fed to deconv6 layer giving an output of 64x64x3 which is basically our generated fake image. All the layers have ReLu activation except final layer which has a tanh activation in order to scale the output between -1 and 1. The following table summarizes it.

Layer	Input	filter	stride	padding	activation	output
Deconv1	1x1x100	4x4x100x512	1	3	Relu	4x4x512
Deconv2	1x1x5	4x4x5x512	1	3	relu	4x4x512
combine	4x4x512, 4x4x512					4x4x1024
Deconv3	4x4x1024	5x5x1024x512	2	3	relu	8x8x512
Deconv4	8x8x512	5x5x512x256	2	3	Relu	16x16x256
Deconv5	16x16x256	5x5x256x128	2	3	Relu	32x32x128
Deconv6	32x32x128	5x5x128x3	2	3	tanh	64x64x3

Table: Architecture of cGAN Generator

The discriminator is basically a Convolutional Neural Network. Here we have 5 convolution layer, 1 reshaping layer, 1 concatenation layer and 1 final layer in total. Architecture is summarized in the table below

Layer	Input	Filter	Stride	Activation	Output
Conv1	64x64x3	3x3x3x64	2	LeakyRelu	32x32x64
y-reshape	5				64x64x5
Conv2	64x64x5	3x3x5x64	2	LeakyRelu	32x32x64
Concatenate	32x32x64, 32x32x64				32x32x128
Conv3	32x32x128	3x3x128x256	2	LeakyRelu	16x16x256
Conv4	16x16x256	3x3x256x512	2	LeakyRelu	8x8x512
Conv5	8x8x512	3x3x512x256	2	LeakyRelu	4x4x256
FC	(4x4x256)x1			Sigmoid	1

Table: Architecture of cGAN Discriminator

Here all the convolution layers have LeakyReLu activation with alpha parameter=0.3. The final layer has sigmoid activation in order to predict the probability of the image being real or fake. The input image (64x64x3) is fed to the conv1 layer having 64 filters. The output shape is 32x32x64. Another input is the conditional attributes. We make 64 times 64 copies of the attribute and reshape it to give shape of 64x64x5. This one goes to conv2 layer having 64 filters of shape 3x3x5, which gives feature map size of 32x32x64. This one is concatenated with the output of conv2 to give feature of size 32x32x128. This goes into conv3 layer that has 256 filters of size 3x3x128, giving an output map of 16x16x256. This output is passed to conv4 layer having 512 filters of size 3x3x256. The 8x8x512 output from this layer is passed to conv5 layer which has 256 filters of size 3x3x512. So we end up having 4x4x256 feature maps which is connected to the fully connected layer, that outputs a single value which is the probability of an image being real or fake.

2.3 Model Training:

We have employed gradient update method for training our model. The core building block of loss function is the cross-entropy loss. Mathematically is written as

$$
\begin{aligned}
l(y[m], \hat{y}[m]) &= -\log p(y[m] \mid x[m]) \\
&= -\log \prod_{k=1}^{K} p(y[m][k] = 1 \mid x[m])^{y[m][k]} \\
&= -\sum_{k=1}^{K} y[m][k] * \log(p(y[m][k] = 1 \mid x[m])) \\
&= -\sum_{k=1}^{K} y[m][k] * \log(\hat{y}[m][k])
\end{aligned}
$$

We have defined 2 loss functions for Generator and Discriminator. The description is given below:

Discriminator Loss:

The discriminator has 2 tasks; declaring real images as real and fake images as fake. So, we have divided the discriminator loss into 2 parts. $Loss_{real}$ and L_{fake}

The idea is that if we assume discriminator output=1 means real image and 0 means fake image, we can construct our loss function as follows. If we get the predictions from the discriminator on real images in each iteration, we expect that all of them should be 1. But at first some of the predictions will be wrong. So, we can write the loss as

$$L_{Real} = crossentropy(\ ones[batchsize], prediction[batchsize]_{real})$$

Again, for the fake images, we expect that all the predictions of the discriminator on a given batch should be 0. But at first, many of the predictions from the discriminator will be falsely 1. So, we can write the loss as

$$L_{fake} = crossentropy(\ zeors[batchsize], prediction[batchsize]_{fake})$$

So, the total discriminator loss will be

$$L_{Discriminator} = ½\ \ (L_{Real} + L_{Fake})$$

Generator Loss:

The target of the generator is to fool the discriminator so that it predicts the generated images as real. So, the loss of the generator should be such that it tries to make the discriminator predict every fake example as 1. But it is evident that some of the predictions will be 0 by the discriminator. So, we define a cross entropy loss for the generator like following:

$$L_{Generator} = crossentropy\ (ones[batchsize], predictions[batchsize]_{Fake})$$

So, in other words, generator will act to make all the predictions of discriminator on fake images as 1.

Parameter update:

In each iteration, we first update the discriminator parameters based on the total discriminator loss defined above using the equation below:

$$\Theta_D^{r+1} = \Theta_D^r - \lambda^D \nabla L_{Discriminator, \Theta_D^r, \Theta_G^r}$$

After the discriminator parameters have been updated in an iteration, we update the generator parameters by minimizing the Generator loss using the following equation

$$\Theta_G^{r+1} = \Theta_G^r - \lambda^G \nabla L_{Generator, \Theta_D^{r+1}, \Theta_G}$$

The process continues until neither the generator nor the discriminator loss can be reduced further and we assume that the discriminator is maximally confused between the real and fake images and we declare that the model has converged.

3. Experiment

3.1 Dataset:

The dataset we have used is called CelebA dataset containing 202,559 images of human faces with 40 attributes. In our case, the dataset image size was cropped to 64x64x3 sizes and we had to consider 5 out of 40 attributes for conditional information. The conditions are Black hair, Male, Oval Shaped Face, Smiling and Young. The attributes are 5x1 vectors. An attribute y=[1,1,0,1,0] means that the face should have black hair, should be a male, not oval shaped face, smiling and not young. We have trained our described model using the mentioned method to generate fake human faces containing the set of 5 attributes described.

3.2 Experimental Settings:

For this project we have used python as the main language and tensorflow2.6 as for training of our designed cGAN network. In order to be familiarized with GANs and tensorflow, we have gained knowledge from tensorflow tutorials online [5]. The hyperparameter setting for this project was a cumbersome job. The first issue is to choose an appropriate batch size. We have experimented with different batch sizes, starting from small values. The training becomes really unstable with small batch sizes so we tried to increase it, but faced Memory Error problem, as the training device goes out of memory. The final chosen batch size was 64 which gave more than decent result. Another hyper parameter is the appropriate

learning rates. We started the experiment with same learning rate for the discriminator and generator loss but found that the training process was not converging, rather our discriminator was becoming stronger after each iterations. After a lot of trial and error, we set our Generator learning rate 5 times more that the Discriminator. The final learning rate chosen for the Generator was 0.0005 and for Discriminator, it was set to 0.0001. Any larger learning rates was making the training very unstable even after many epochs. We have trained our network for 50 epochs. We got the best visual results from epoch 23, so we have used the model parameters from that epoch. After that the visual quality of the generated images start to degrade. We have also employed batch normalization layers in the generator as discussed before, which made our training extremely stable.

3.3 Model Evaluation:

During training, we generate 30 images after each epoch using a predefined noise vector and predefined set of attributes. Some images after epoch 5 and epoch 20 is given below:

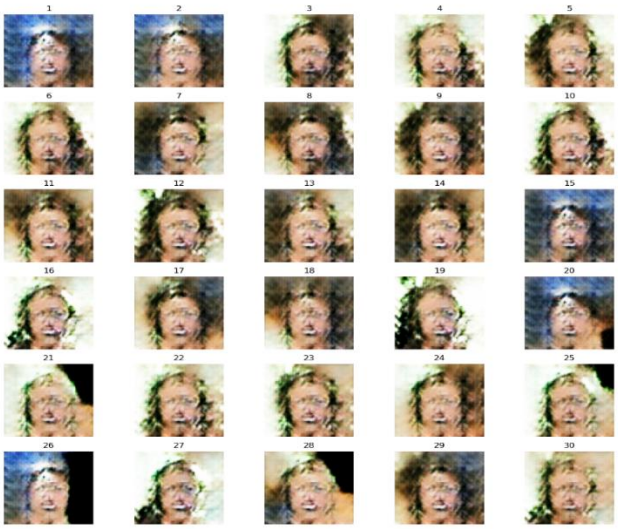

Figure: 30 generated images after epoch 5

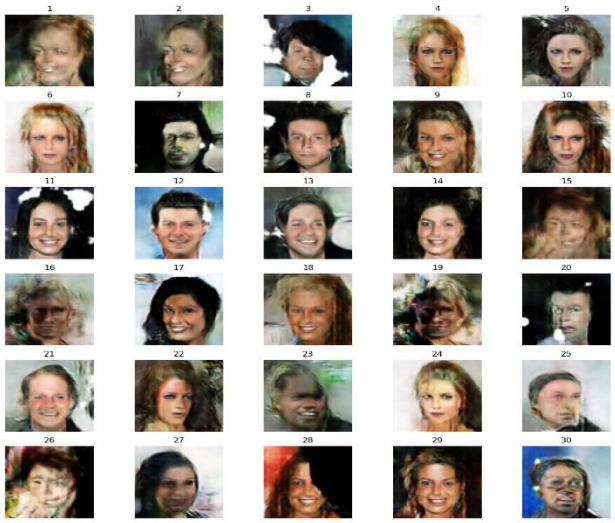

Figure: 30 generated images after epoch 20 using same noise vector and

Attributes as the previous image

After the training is completed and the weights have been saved, we evaluate our model on the best visually performing epoch. We load the weights into our model and generate 1000 fake images using 1000 noise vectors of size 100. We then calculate the FID and IS score using a pretrained Inception Network the quantitatively evaluate our model. For image generation we have used three types of attributes. y1=[10011], y2=[11001], y3=[11011]. From the 1000 images, 100 images are shown below:

For y= [10011] (Black hair, Female, Not Oval, Smiling, Young):

Figure: Generated Images with 10011 (Black hair, female, not oval shape, Smiling, Young)

For y=[11001] (Black Hair, Male, not oval, not smiling, Young):

Figure: Generated Images with 11001 (Black hair, male, not oval shape, not smiling, Young)

For y=[11011] (Black Hair, Male, Not Oval, Smiling, Young)

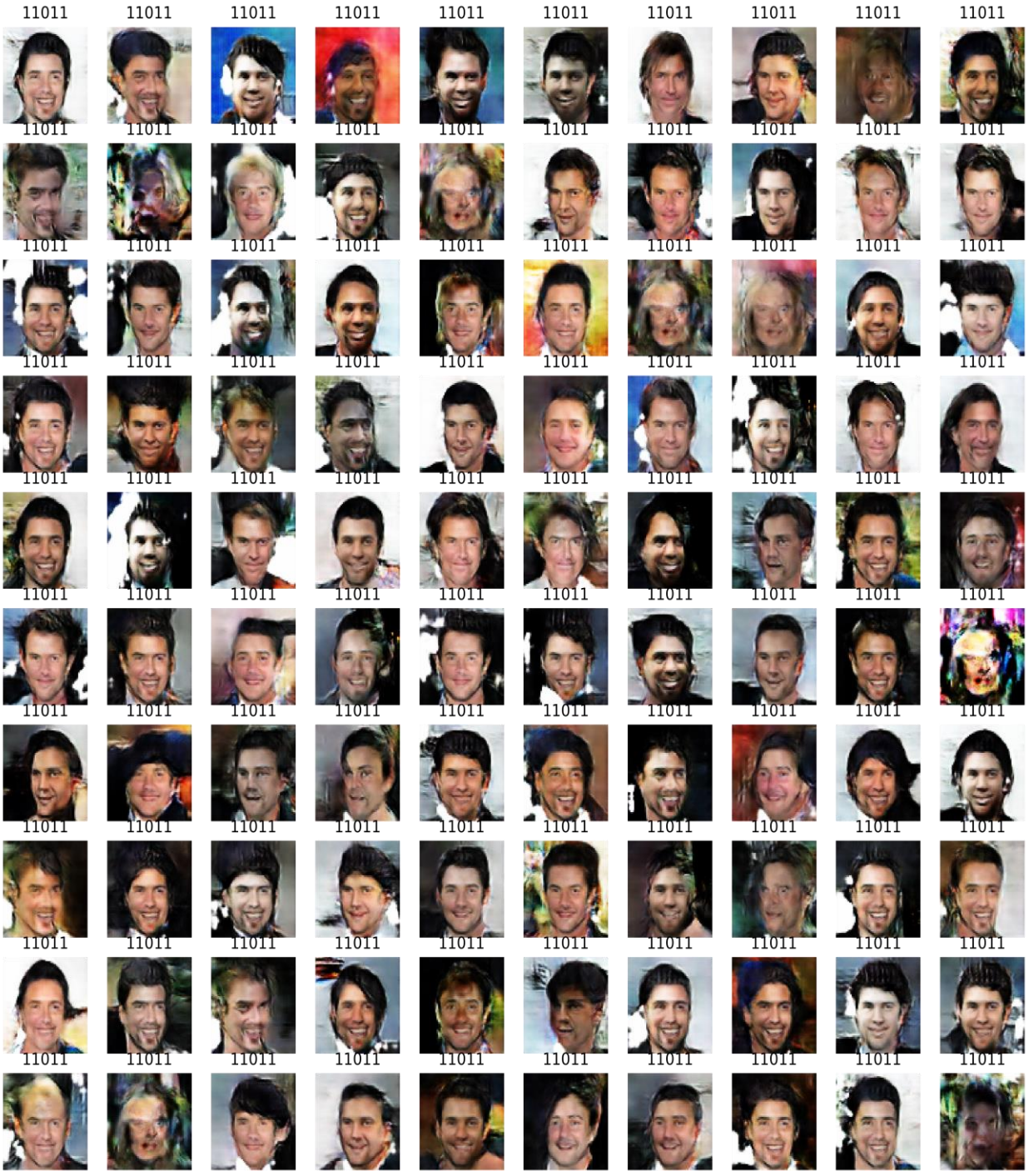

Figure: Generated Images with 11011 (Black hair, male, not oval shape, Smiling, Young)

We have also generated 1000 images using the first 1000 attributes of the given dataset. The results are shown below. Each image has its corresponding attributes set mentioned on top of that image.

For Different types of Attributes:

Figure: Generated images with a mixture of attributes. Each image's attributes are mentioned on top of it.

FID and IS score:

We have calculated the IS score of the generated images using pretrained inception network. In order to calculate the FID score, we had used a subsample of 20,000 images from the real image set and calculated the FID with the Generated image set using the pretrained inception network. The reason for taking subsample is that the calculation process takes a huge amount of time for FID for larger data and also, we run into memory error problem. But we have verified that using 1000, 5000 ,10000 or 20000 real images, the FID score does not change that much. So, it is assumed that the sample mean and variances are invariant with respect to size of the taken samples. The results are shown in the following table.

Metric Name / **Attributes**	**FID**	**IS**
Y1=11001 (Black hair, male, not oval shape, not smiling, Young)	11.331	2.535
Y2=10011 (Black hair, female, not oval shape, Smiling, Young)	12.969	1.958
Y3=11011 (Black hair, male, not oval shape, Smiling, Young)	11.797	2.363
Mixed Attributes	12.772	2.453
Average Score	**12.217**	**2.327**

Table: Results of FID and IS for Different Conditional Attributes on the Generated Images

Here it is seen that the average FID and IS scores are above par values mentioned in the task description (less than 15 for FID and more than 2 for IS). For a single type of attributes, we have a little less IS score when the images are only of females but still close to 2.

Ablation Study and Explanations & analyses on experimental results:

As discussed in section 3.2 The hyperparameter setting for this project was very crucial. We had to make a balance between performance and hardware limitations. For example,

smaller batch sizes was giving a very bad results in our designed architecture after several epochs. But we could not increase the batch size after a certain value, so we settled for a batch size of 64 in the end before crashing the program due to memory issue. Also choosing appropriate learning rate was very important. Below is a picture showing the model's generated image training after 50 epochs when we chose same learning rate for the Generator and Discriminator (0.001 for both)

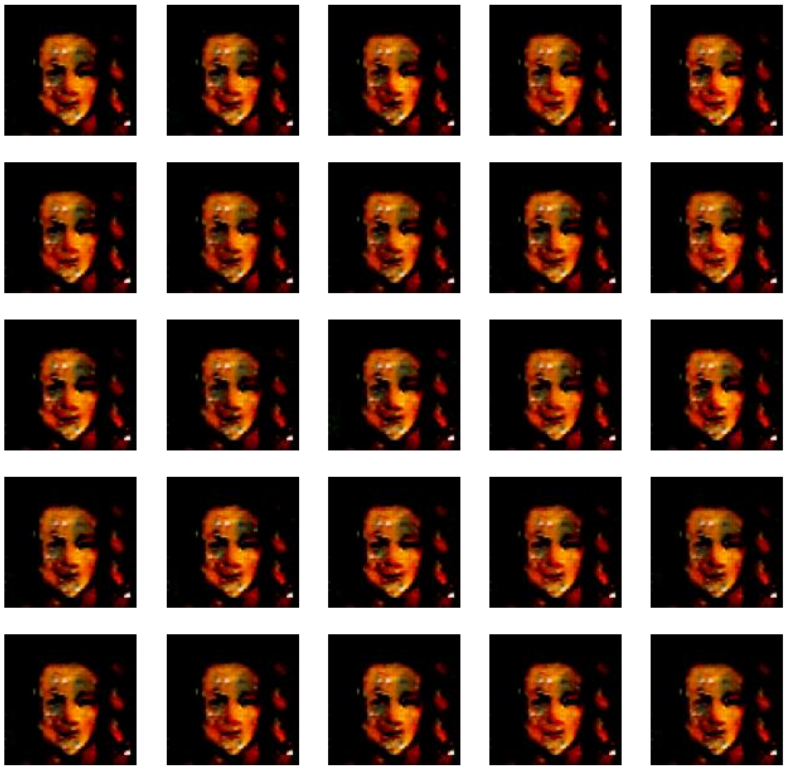

Figure: Generated image after 50 epochs when using same learning rate (0.001)

For Discriminator and Generator (A Failed Attempt)

Also, with the trained network, we have seen that changing the random seed value to generate images gives visually different images but quality of the new images does not change much and the IS and FID scores are also consistent with images generated from different noise vectors. As for the attributes, we have seen that in the generated images our model can differentiate between male and female attributes almost all the time. It also does

extremely well when generating smiling face and faces without a smile based on the attributes given. Black hair generation is also very good. We can also see that oval shape generation is fairly well as well. Only condition that is not visually very clear is the young/ old criteria where sometimes the images which should be old, looks a bit younger. In terms of FID and IS results, the model's performance is almost similar for any attributes but we have seen that the performance is higher in male images than in female ones having black hair (when the condition is such that all the generated images are of females with black hair). This could be due to the fact that long black hair covers a large portion of the generated image, where the pixel values are very low due to being nearly black, so when calculating performance scores, we get below par results. Overall, the method has done a very decent job of generating visually attractive images based on the conditions specified in each case with fairly good values of FID and IS. It should also be mentioned that in one experiment, generating more image than 1000 somewhat increased the IS score by a little.

4. Conclusion

In this final project, we have successfully implemented a Conditional Generative Adversial Network trained on a modified CelebA dataset. We have used a small cGAN network with a Discriminator and a Generator and trained the model using Gradient Optimization approach. We have seen that training a GAN is a very difficult job, when a small change in the hyperparameters may bring about large change in the outcome. Also, incorporating conditional attributes to the generated images needs more careful selection of architecture and hyperparameters. Another problem faced during this project is to make a balance between computational resources and model performance. Training this cGAN requires a very powerful hardware, and even so, the training process takes a lot of time. And even after training for several hours the model might fail due to improper selection of model architecture and hyperparameters. We have tweaked our architecture multiple times, applied regularizations such as batch normalizations, tweaked the learning rates many times to create a stable and well performing training of the network. Applying the knowledge gained from the coursework on conditional GANs and the experience gained throughout different programming assignment done on this course, we have successfully implemented a cGAN that can generate human faces with 5 different conditional attributes. We have also achieved good FID and IS scores on the generated images for different noise inputs and conditional attributes.

References:

1. A. Dosovitskiy, J. Tobias Springenberg, and T. Brox. Learning to generate chairs with
   convolutional neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1538–1546, 2015
2. Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In

Proceedings of International Conference on Computer Vision (ICCV), December 2015.
3. M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. Advances in neural information processing systems, 30, 2017.
4. S. Barratt and R. Sharma. A note on the inception score. arXiv preprint arXiv:1801.01973, 2018.
5. https://www.tensorflow.org/tutorials/generative/dcgan