**PROJECT**
Sudipta Paul; RIN- 662026666
email: pauls5@rpi.edu
PhD Student in ECSE Department

# 1 Introduction

In this project, I have implemented conditional generative adversarial network (cGAN) for image generation on CelebA dataset. The primary application of GAN is to generate fake data that looks realistic enough. The main components of any variants of GANs are "Generator" and "Discriminator" which competes with each other to generate and identify fake data; thus, the adversarial part in "GAN" comes from. Although GAN models are able to generate satisfactory synthetic samples, it lacks controllability on the generated samples. For example, if an user wants to generate a human face with some face attributes such as the human face has to be male, he needs to have black hair, he is smiling and young, these conditions can not be fulfilled by using a simple GAN model. To solve this issue, M. Mirza *et al* proposed cGAN for the first time [1]. In cGAN, the generator generates synthetic samples based on a given condition from the user.

A general architecture for cGAN is represented in Fig. 1. The generator takes noise vector $z$ and a conditional vector $y$ as it's input, and then outputs a synthetic images, $x'$. Later, the discriminator takes the input as real image, $x$ or synthetic image $x'$ and their associated condition vector $y$. Discriminates outputs the probability of the images being real or fake.

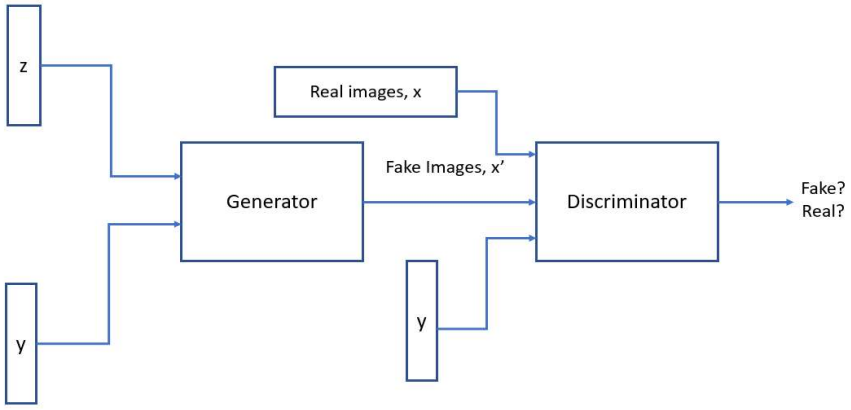


Figure 1: A general overview of conditional generative adversarial network (cGAN)

# 2 Method

## 2.1 Problem setting

Given, noise vector $z \in R^D$ and it's associated condition vector $y$, the goal is to generate synthetic images $x'$ by finding the optimized generator parameter $\theta_G$ by solving the following equation

$$\theta_G^{*,r+1} = arg \ \ min_{\theta_G} V_{cGAN}(\theta_G, \theta_D^{*,r+1})$$
$$= arg \ \ min_{\theta_G} \frac{1}{n} \sum_{i=1}^{n} log(1 - D(G(z_i, y_i; \theta_G^{*,r})), y_i; \theta_D^{*,r+1})$$

where, $\theta_D^{*,r+1}$ can be found by solving

$$\theta_D^{*,r+1} = arg \ \ max_{\theta_D} V_{cGAN}(\theta_G^{*,r}, \theta_D)$$
$$= arg \ \ max_{\theta_D} \frac{1}{2n} \sum_{i=1}^{n} log(D(x_i, y_i; \theta_D)) + arg \ \ max_{\theta_D} \frac{1}{2n} \sum_{i=1}^{n} log(1 - D(G(z_i, y_i; \theta_G^{*,r})), y_i; \theta_D)$$

*The details about this optimization proceduces is described in section 2.3 (Model training objectives*

## 2.2 Model description

The two key components of implementing cGAN are the designing of generator and discriminator. In this section, the architecture of these two key components will be discussed. In this assignment, I followed the architecture given on the Project Instructions manual.

Generator takes gaussian noise, $z \in R^{100}$ and conditional information, $y \in R^5$ as the input and provides sinthetic images of size $64 \times 64 \times 3$ as the output. First, $z$ and $y$ and converted into $1 \times 1 \times 5$ and $1 \times 1 \times 100$ tensors. Then $z$ and $y$ tensors are deconvoluted separately to obtain similar size embeddings. After that, these two separate tensors are concatenated and fed into the third deconvolution layer. This layer is then followed by another three deconvolution layers, and the output of the last deconvolution layers provides an image of size $64 \times 64 \times 3$. The details architecture of generator and the correcponding input output mappings are shown in the following figure.

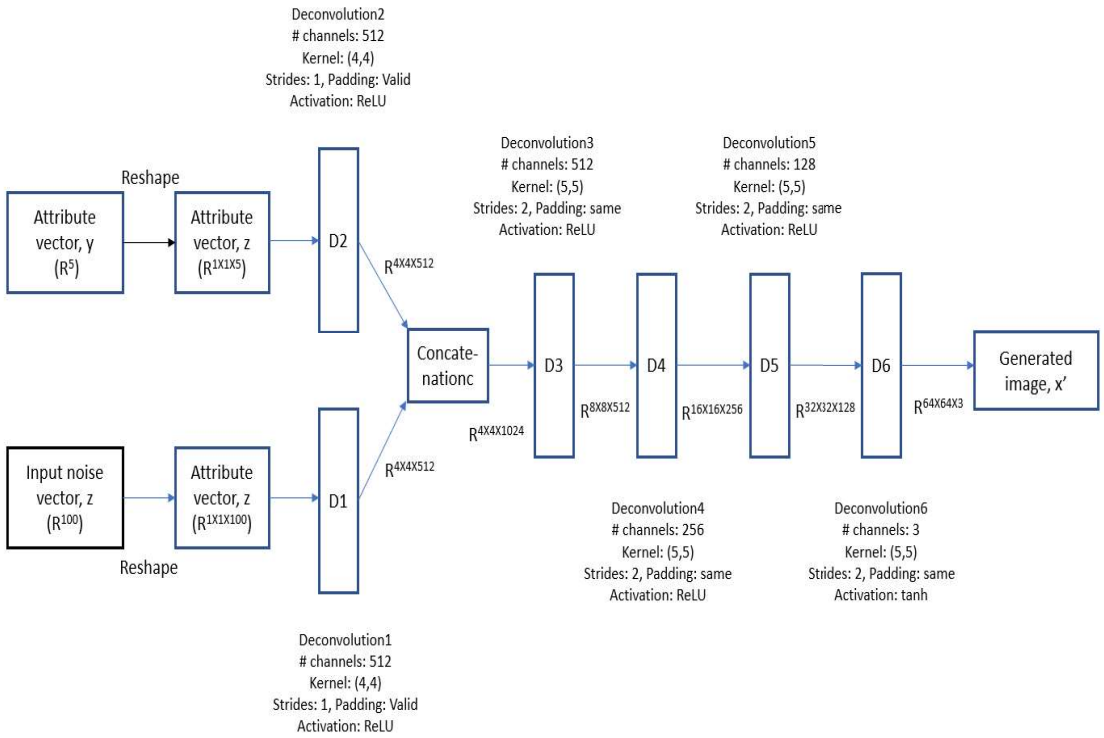


Figure 2: Implemented generator architecture of cGAN

On the other hand, the discriminator takes images of shapes $64 \times 64 \times 3$ and the condition vector $y \in R^5$ as the input. First, the $y$ vector is converted to $y'$ of shape $64 \times 64 \times 5$ tensor.

Then two separate convolution operations are applied to both image and $y'$. Later, The outputs of these convolution layers are concatenated, which then followed by three convolutional layers. The output of the final convolution layer is connected to a fully connected layer which predicts the probability of the image being fake or real. The overall architecture of the discriminator is presented in Fig. 3.

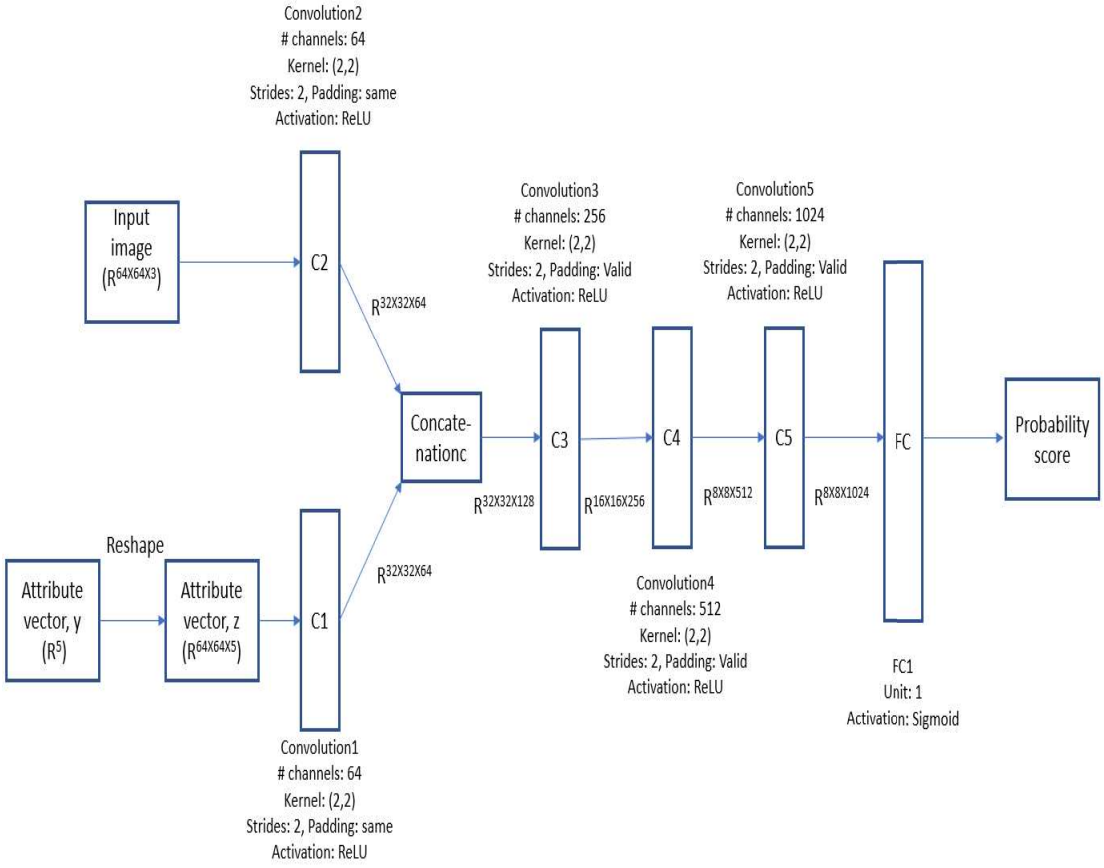


Figure 3: Implemented discriminator architecture of cGAN

## 2.3 Model training objectives

For each minibatch, the first step is to train the discriminator D, and the second step is to train the generator G. We continue this process for the full dataset and for a predefined specific number of epochs. The goal is to obtain the optimized generator and discriminator weights. The generator $G$ and the discriminator $D$ plays a max-min optimization game with the training objective $V_{cGAN}$ defined as

$$V_{cGAN} = E_{x,y-p_{data}(x,y)}[log(D(x,y;\theta_D))] + E_{y-p_y(y),y-p_y(y)}[log(1 - D(G(z,y;\theta_G),y;\theta_D))]$$

Here, the parameters to be optimized are $\theta_D$ and $\theta_G$. The optimization can be written as

$$\theta_G^*, \theta_D^* = arg \;\; min_{\theta_G} \;\; min_{\theta_D} V_{cGAN}(\theta_G, \theta_D)$$

Also, the loss function can be defined as

$$V_{cGAN}(\{(x_i, y_i)\}_{i=1}^n, \theta_G, \theta_D) = \frac{1}{2n}\sum_{i=1}^{n} log(D(x_i, y_i; \theta_D)) + \frac{1}{2n}\sum_{i=1}^{n} log(1 - D(G(z_i, y_i; \theta_G)), y_i; \theta_D)$$

Then, we update the parameter $\theta_D$ by maximizing the following function

$$\theta_D^{*,r+1} = arg \;\; max_{\theta_D} V_{cGAN}(\theta_G^{*,r}, \theta_D)$$

$$= arg \;\; max_{\theta_D} \frac{1}{2n}\sum_{i=1}^{n} log(D(x_i, y_i; \theta_D)) + arg \;\; max_{\theta_D} \frac{1}{2n}\sum_{i=1}^{n} log(1 - D(G(z_i, y_i; \theta_G^{*,r})), y_i; \theta_D)$$

The next step is to update the parameter of the generators as stated below

$$\theta_G^{*,r+1} = arg \;\; min_{\theta_G} V_{cGAN}(\theta_G, \theta_D^{*,r+1})$$

$$= arg \;\; min_{\theta_G} \frac{1}{n}\sum_{i=1}^{n} log(1 - D(G(z_i, y_i; \theta_G^{*,r})), y_i; \theta_D^{*,r+1})$$

# 3 Experiment

## 3.1 Dataset description

In this project, we have utilized the CelebA dataset for implementing cGAN. The dataset has 202, 559 celebrity images, each with 40 attributes. Although the original image was $218 \times 178 \times 3$, for this project the cGAN will be implemented into smaller images of size $64 \times 64 \times 3$. Also, out of 40 attributes, this projects uses only 5 of them. These attricutes are: [Black hair, Male, Oval Face, Smiling, and Young]. A conditional vector fulfilling all these conditions will have vector elements as [1,1,1,1,1]. If anyone of the conditions are missing, then the vector element of these particular position will be replaced by 0. AN outlook of the CelebA dataset is shown in the following below.

## 3.2 Experimental settings

This project is written in Python 3 with tensorflow version 2.8.0. Due to the unavailability of the GPU in my personal computer, I used Kaggle notebook to train my model. It provided 13 GB of RAM memory and 15.6 GB of GPU memory. Due to the memory issue, the training images were first converted into float16. All the images were used for the training purposes. Each epoch in the Kaggle GPU took around 6 minutes to run. The hyperparameters used in this projects are listed below:

Batch size: 50
Noise dimension: 100
Generator optimizer: Adam (learning rate = 0.0005)
Discriminator optimizer: Adam (learning rate = 0.0001)
Number of epochs: 50

## 3.3 Model evaluation

I have written my generator and discriminator using a class based approach. I realized it is very hard to develop my model with Sequential approach that I have worked previously. I had hard time implementing the cGAN with Sequential methods. I designed my own classes, however for model training and evaluations, I took help from a Tensorflow official tutorial on GAN [1]. It was developed for MNIST dataset. I modified the model training and evaluation for CelebA dataset.


Figure 4: An outlook of CelebA dataset [3]

### 3.3.1 Generated images and associated conditions

I evaluated my model on randomly generated $z$ and specified conditional attribute $y$. The image set shown in Fig. 6 is the 100 generated images with my model. **Here, it is to be noted that, the images generated from my models are low resolution images. Due to the memory issues, I converted the CelebA dataset into float16 and then trained on these dataset. I was having memory error with float32 or float64 datasets. After converting my dataset into float16 I was able to resolve the memory error. However, the output images generated are suffering from poor resolutions.**

Figure 5: Sample generated image to match condition

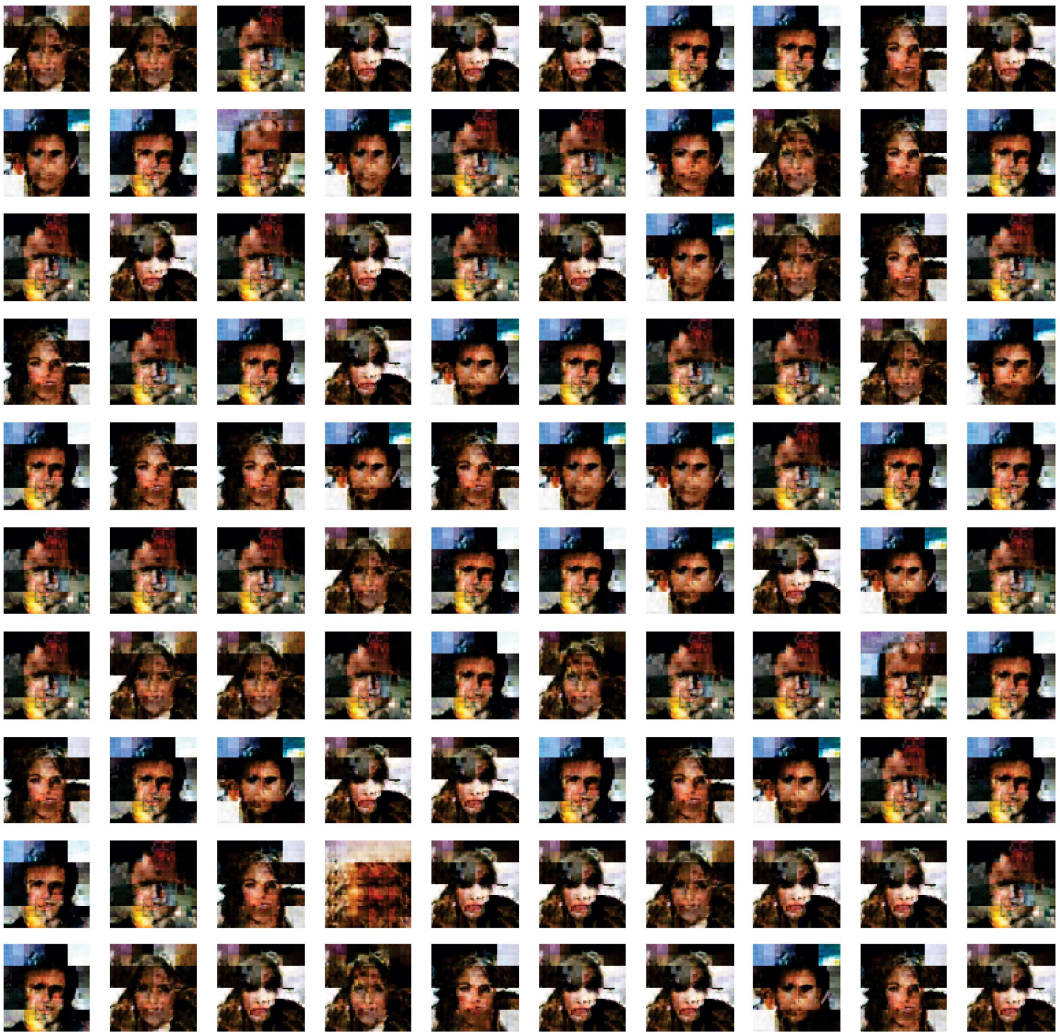

Figure 6: Generated images from cGAN model

Now, let's see an individual image whether it aligns with the given condition. I am taking the first image to show my condition aligning. For this image, the given condition vector was [0,0,0,1,1]. It means the generated image should NOT have the criteria of Black hair, Male or Oval face. However, it should have the criteria of Smiling and Young. From the image we can see, the hair color is brown, this is NOT a Male face or the face is NOT oval. On the other hand, the generated face is Young and Smiling. From here, we can see the generated image aligns with the given condition.

### 3.3.2 FID and IS scores for generated images

The FID and IS score of my generated images with seed value 13 is reported below:

FID Score: 16.69
IS Score: 2.13

```
fid = calculate_fid(model, images1, images2)
print('FID Score: %.5f' % fid)
```

```
FID Score: 16.69520
```

```
print('IS score', is_avg)
```

```
IS score 2.1265368
```

The Frechet Inception Distance score, or FID for short, is one of the comparison metrics for synthetically generated images that calculate the distance between the feature vectors of real and generated images. Lower FID score represents that the groups of images have more similar statistics. In my experiment, I found the FID score as 16.69 which is a little higher than the expected result according to the Project Manual.

On the other hand, the inception score is another popular metrices to evaluate the GAN models. Similar to FID, it measures how realistic the synthetic data is. A higher IS score is better, it means the GAN models can generate many distinct output images. In my model, the IS score becomes 2.13 which is better than the expected result according to the Project Manual.

### 3.3.3 FID and IS score with variation of conditional attribute

After varying the conditional attribute 3 times, I found the following results:
**Trial 1**
FID Score: 16.69
IS Score: 2.12
**Trial 2**
FID Score: 15.69
IS Score: 2.17
**Trial 3**
FID Score: 16.49
IS Score: 2.07

We can see, in all of those trials, my IS score was above the expected value, however, the FID score become a little lower than the expected score.

### 3.4 Ablation study

I varied my model several times with different model parameters. Some parameters seems to have greater effect on model performance, on the other hand some parameters have minimal effect. For example, the model performance varies a lot with the number of epochs. I found that its hard to generate meaningful images with epochs lower than 10. Also, most of the images appears to be the same. I started to find meaning images after 10/12 epochs which continuously creates human faces. However, I realized the condition vector does not map properly. I found much better images after 50 epochs which maps to the condition vector quite accurately. Still there were some issues like appearing same faces again and again, however with proper condition mapping.

I have also experimented with generator and discriminator optimization learning rate. I have implemented Adam optimizer in both of these cases. The learning rate impacts the model performance quite heavily. I started getting better performance after setting the generator learning rate at least 5 times greater than the discriminator learning rate.

Model architecture also had a great impact on generating images. Initially, I started with some simple generator and discriminator model. However, I wasn't getting meaningful images even after 10/15 epochs. Instead of wasting time to vary that model's hyperparameters, I started making the model complex. Finally, I ended up adopting the model architecture provided with the Project Instruction Manual.

Apart from that, I have also experimented with varying batch size, buffer size, etc. I varied all of these hyperparameters within a reasonable range, however, I did not experienced noticeable change in model performance.

### 3.5 Explanation and analysis

In this project, I implemented conditional GAN for CelebA dataset. Two things regarding the results require explanations. First, the generated images are having poor quality. The reason behind this is, I converted the CelebA dataset into float16 datatype, which results in poor image resolution. However, by looking into the images closely we can find that the model is working perfectly. It is able to map the conditional vector in most of the images.

Another thing that requires explanation is lower FID score. While running the code I realized that my expected images are generating with a little bit higher epoch. I got my first human face image on around 12 epochs. I ran the code for about 50 epochs. I should have run the code for 100 epochs. There might be good chance that with higher epochs I could get a FID score which is well below 15.

## 4 Conclusion

In this assignment, I have implemented conditional generative adversarial network (cGAN) on CelebA dataset to generate synthetic human faces given some conditions. cGAN composed of two key components- generator and discriminator. The generator acts as a sampler to generate images from random gaussian distribution with some conditions. Here, first we train the discriminator to distinguish real vs fake data. In the next step, we train the generator to make fake data which can fool the discriminator. We then continue this discriminator and generator training for multiple epochs.

I obtained a satisfactory IS score from my model. However, the FID score resulted from this project is little higher than the expected value. The reason behind this could be the lack of running

epoch. Although I ran my code for about 50 epochs, I used the dataset with float16 datatype. It could create an issue while generating good FID score. I noticed that with float16 data type, I started getting reasonable generated images (with human face) with a higher epoch value (around 15). I believe running for around 100 epochs could have solved this issue. However, I did not have enough time to run for another 100 epochs.

I faced several difficulties in implementing cGAN for this project. First, I faced memory issues. It was very difficult to implement cGAN with a float64 datatype of 200K images. Even though I converted the dataset to float32, it did not solve the issue. I finally solved it by converting the dataset to float16. However, it created another issues of low resolution images. Although my generated images are having low resolution, they are able to map the condition vector properly. I also faced issues with high run time in my PC, and then I realized cGAN will be extremely difficult to implement without GPU. Finally, I was able to solve it with Kaggle notebook with a run time of around 6 minutes per epochs on full dataset of type float16.

# 5 Reference

[1] Mirza, M. and Osindero, S. (2014) Conditional Generative Adversarial Nets.

[2] https://www.tensorflow.org/tutorials/generative/dcgan?fbclid=IwAR0BXuBXItiPjsoRg0dwpzkrT51T1rcpQLcuJTDyboWaz0ueo-nCsMPaV7g

[3] Ziwei Liu, Ping Luo, Xiaogang Wang, Xiaoou Tang."Deep Learning Face Attributes in the Wild", 2015 IEEE International Conference on Computer Vision (ICCV)