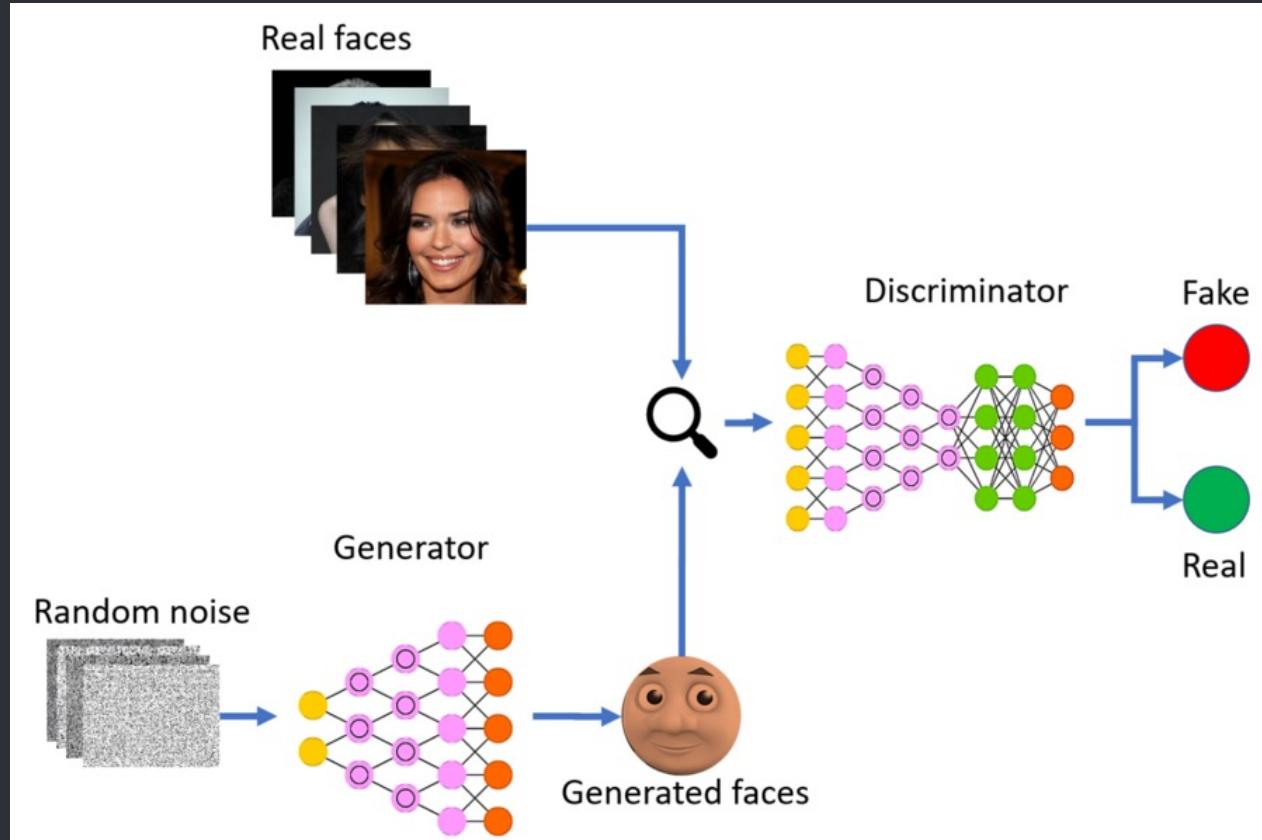




## **REALISTIC FACE IMAGE GENERATION BASED ON GENERATIVE ADVERSARIAL NETWORK**

## ● Introduction

- Using computer to generate images with realistic images is one of the trending topic in computer vision research.
- So mainly we need to generate realistic face in order to increase the face recognition dataset, which in return helps the face recognition algorithms to work better.
- In this Paper they had proposed a new methodology using GAN for generating artificial images based on a real image in the data set.
- The proposed model is obtained by using Deep Convolution Neural Network and Deep convolution Transpose network



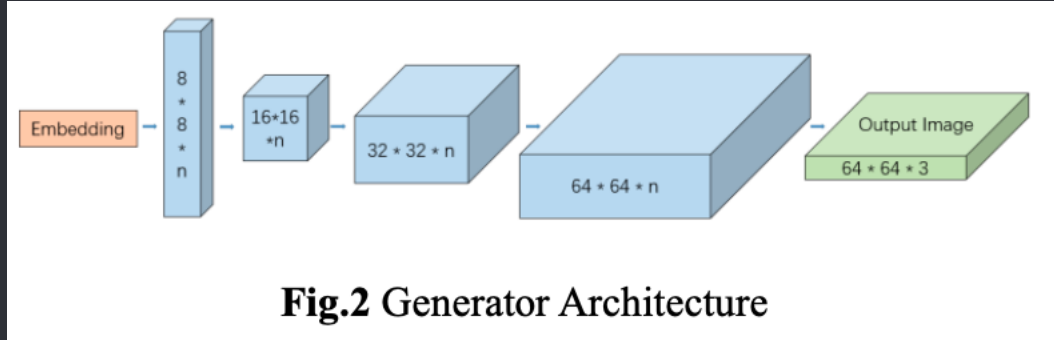
## ● Method Proposed

○ The proposed method have two different parts, Generator and a Discriminator

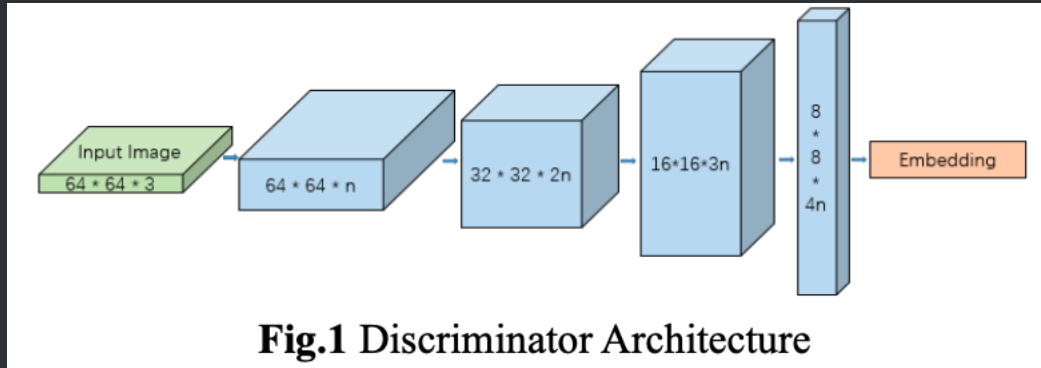
-> Generator in the model is responsible for creating a artificial images based on the given input image

-> And the Discriminator attempts to distinguish between the real sample and the output from the generator.

The goal is to generate samples as good as real samples; the discriminant model  $D$  is one the classifier that estimates the probability of deriving the samples from the training data (rather than the generated data). If the sample is from real training data,  $D$  outputs a large probability; otherwise,  $D$  outputs a small probability.



Generator G: We use the same architecture as the discriminator decoder, but with different weights. The first layer receives the random noise signal vector for the fully connected layer, and is multiplied by the weight matrix to be transformed into a three-dimensional tensor; the remaining layers are deconvolution operation layers, and  $2 \times 2$  strides are used in the deconvolution operation. The Relu activation function is used, the last deconvolution layer uses the tanh activation function to generate a three-channel image.



**Fig.1** Discriminator Architecture

Discriminator D: We use an auto-encoder with a depth encoder and decoder. We use two convolutions per layer. At each subsampled, the convolution filter is linearly increased, achieve subsampling with stride of two, and do up sampling by nearest neighbors. The discriminator receives the three- channel image, obtains the feature map after multiple convolution operations, maps it to a scalar value after deforming it into a vector, and uses the Sigmoid function to determine the loss function value of the discriminator

## ● Experiments and Setup

- Here we have performed our model on The CelebA Face Dataset is the open data, which contains 202,599 face images of 10,177 celebrity identities, all of which have been feature-marked, which is a very useful dataset for face-related experiments.
- We train and test our model on the CelebA dataset and compare them with the other existing models classic DCGAN and WGAN architecture
- The initial learning rate is given to 0.0001 and performed on 2000 iterations.

- Here we use Wasserstein Distance as a parameter to define our loss function
- We have two loss functions Generator loss function and Discriminator loss function.
- Goal of Discriminator would always increase the distance between two images, where as the Generator goal is to minimize it.



localhost

face\_GAN - Jupyter Notebook






















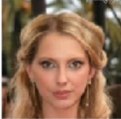



jupyter face\_GAN Last Checkpoint: 2 hours ago (autosaved)

File Edit View Insert Cell Kernel Help

Not Trusted Python 3 (ipykernel)

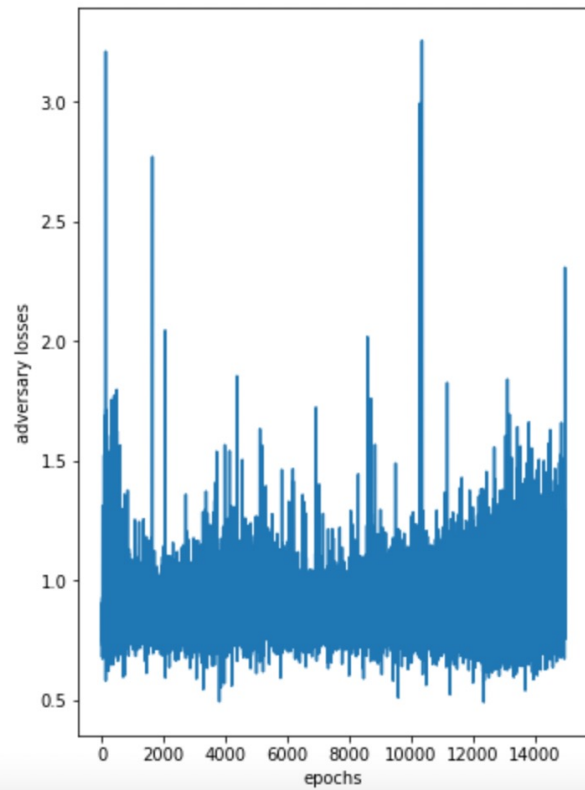
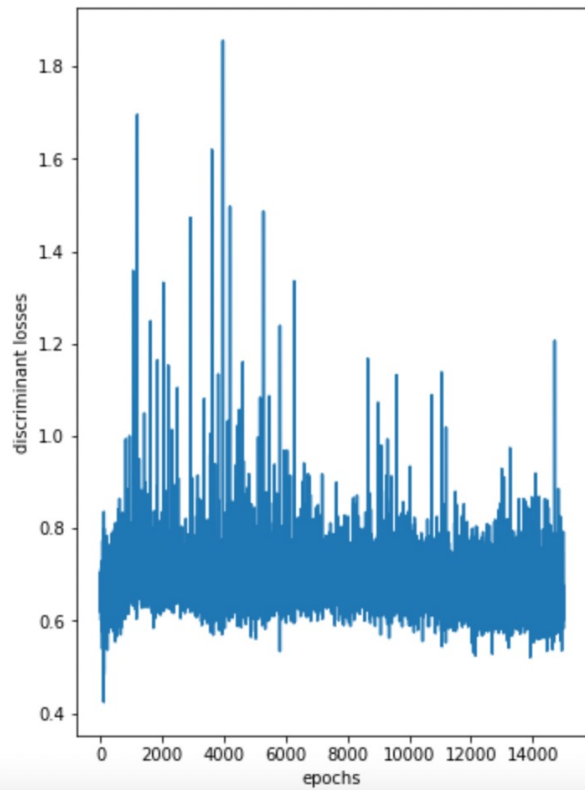
plt.axis('off')

plt.show()

**Table 1** The results of Face recognition

	Can't detect face	Can detect faces but can't identify who
DCGAN	281/1000	0/1000
WGAN	123/1000	0/1000
Our Model	19/1000	0/1000



- Results:



DCGAN



WGAN



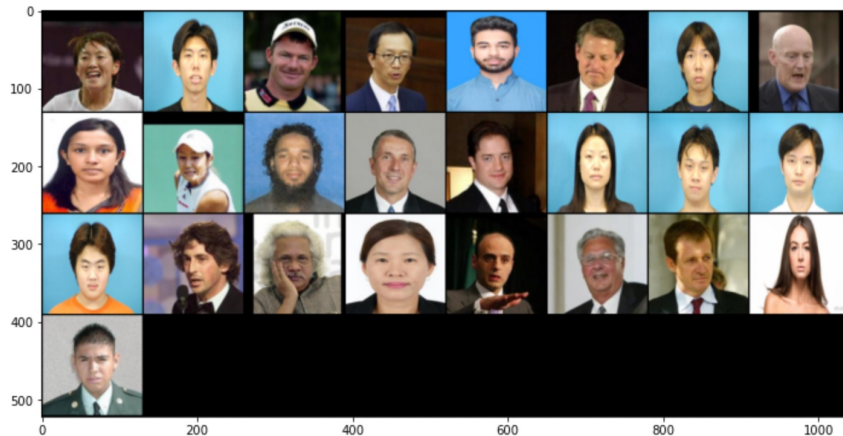
Our model

## Further Implements:

- We have used this model to and work it on Sketch data, and tried to convert sketch images to real images

```
# get some images from X
dataiter = iter(dataloader_X)
# the "_" is a placeholder for no labels
images, _ = dataiter.next()

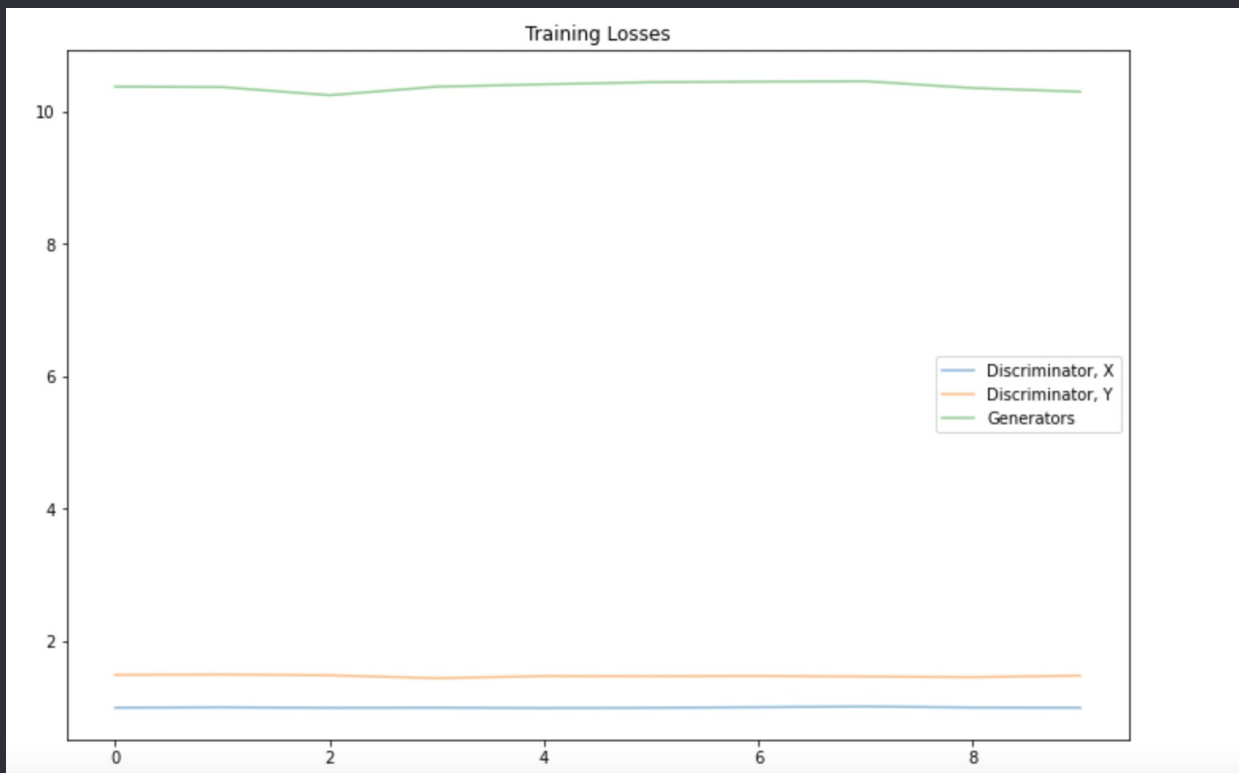
# show images
fig = plt.figure(figsize=(12, 8))
imshow(torchvision.utils.make_grid(images))
```



● Output:



## Loss Graph for our Implementation:



In [31]: `class Discriminator(nn.Module):`

```
    def __init__(self, conv_dim=64):
        super(Discriminator, self).__init__()
        self.conv1 = conv(3, conv_dim, 4, batch_norm=False) # x, y = 64, depth 64
        self.conv2 = conv(conv_dim, conv_dim*2, 4) # (32, 32, 128)
        self.conv3 = conv(conv_dim*2, conv_dim*4, 4) # (16, 16, 256)
        self.conv4 = conv(conv_dim*4, conv_dim*8, 4) # (8, 8, 512)

        # Classification layer
        self.conv5 = conv(conv_dim*8, 1, 4, stride=1, batch_norm=False)

    def forward(self, x):
        # relu applied to all conv layers but last
        out = F.relu(self.conv1(x))
        out = F.relu(self.conv2(out))
        out = F.relu(self.conv3(out))
        out = F.relu(self.conv4(out))
        # last, classification layer
        out = self.conv5(out)
        return out
```



```

In [34]: class CycleGenerator(nn.Module):

    def __init__(self, conv_dim=64, n_res_blocks=6):
        super(CycleGenerator, self).__init__()

        # 1. Define the encoder part of the generator

        # initial convolutional layer given, below
        self.conv1 = conv(3, conv_dim, 4)
        self.conv2 = conv(conv_dim, conv_dim*2, 4)
        self.conv3 = conv(conv_dim*2, conv_dim*4, 4)

        # 2. Define the resnet part of the generator
        # Residual blocks
        res_layers = []
        for layer in range(n_res_blocks):
            res_layers.append(ResidualBlock(conv_dim*4))
        # use sequential to create these layers
        self.res_blocks = nn.Sequential(*res_layers)

        # 3. Define the decoder part of the generator
        # two transpose convolutional layers and a third that looks a lot like the initial conv layer
        self.deconv1 = deconv(conv_dim*4, conv_dim*2, 4)
        self.deconv2 = deconv(conv_dim*2, conv_dim, 4)
        # no batch norm on last layer
        self.deconv3 = deconv(conv_dim, 3, 4, batch_norm=False)

    def forward(self, x):
        """Given an image x, returns a transformed image."""
        # define feedforward behavior, applying activations as necessary

        out = F.relu(self.conv1(x))
        out = F.relu(self.conv2(out))
        out = F.relu(self.conv3(out))

        out = self.res_blocks(out)

        out = F.relu(self.deconv1(out))
        out = F.relu(self.deconv2(out))
        # tanh applied to last layer
        out = F.tanh(self.deconv3(out))

        return out

```

**Thanks!**

**ANY QUESTIONS?**