Mythological Figures Face Prediction Using Generative Adversarial Networks

Vinil Gupta[1], Devesh C Singhal[2], Aby Anthony[3], Aishwarya Funaguskar[4], Rishav Tiwari[5], Sparsh Gaba[6], Patel Abdullah Zuber[7]
[1, 2, 3, 4, 5, 6, 7] *U.G Scholars, Department of Computer Science and Engineering, Amity University Maharashtra, Mumbai, Maharashtra, India*

Abstract: **A potent style of neural network called generative adversarial networks is used to generate new data from a collection of training data that maintains the same characteristics as the training data. It has a wide variety of applications, (with some of them not being as noble as others) for animation ranging from animation model generation to deepfake generation. In this paper, we propose the application of GANs to generate realistic images of a mythological being depending on the existing depictions from various artists in the same period.**

***Index Terms*—Deep Learning, Generative Adversarial Networks(GANs), Neural Networks, Unsupervised Machine Learning**

I. INTRODUCTION

It's a known fact that different people have wildly different interpretations of how a certain mythological figure would have looked like. Through the use of *GANs*, we are going to explore a methodology to accurately predict the depiction of a particular mythological figure by borrowing the features from most existing depictions. With the acute use of academy-industry collaboration, this method can help us resolve several long-standing conflicts in religion. For instance, we can get a decent idea of how God would have looked like.

II. LITERATURE REVIEW

Generative Adversarial Networks use two neural networks contesting with each other in a game in the form of a zero-sum game, where one agent's gain is another agent's loss. Given a training set, this technique learns to generate new data with the same statistics as the training set. For example, a GAN trained in photographs can generate new photographs that look at least superficially authentic to human observers, having many realistic characteristics.

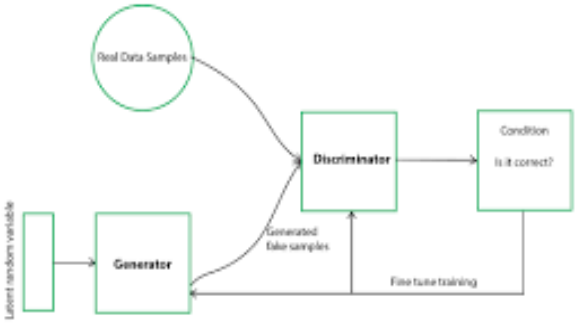

Figure 1: Working of GANs

Blended learning: Blended learning is the term given to the educational practice of combining digital learning tools with more traditional classroom face-to-face teaching(7).

III. PROPOSED METHODOLOGY

GAN, when unabridged, is also known as Generative Adversarial Networks, starting to show the new face and capabilities of modern developments in the field of Artificial Intelligence (A.I). Nowadays we see a lot of apps and websites being able to swap faces, generate realistic human voices and be able to smartly reply to any task given. These challenges are largely facilitated by highly developed AI conceptions known as GAN, which now has morphed over time to emerge as the most interesting factor of deep learning. From being able to autocorrect to being able to make smart decisions, GAN has taken a different way of approaching and learning than the other types of neural networks. GAN's algorithmic architecture is made up of two different neural networks known as Generator and Discriminator.

The algorithm makes both of them (Generator and Discriminator) compete against each other to form the desired approach. The Generator generates realistic-looking fake images, while the discriminator puts a separation in searching into action to differentiate between real and fake images. If both neural networks will function at a high level, the output can be surprisingly realistic.

Generative Adversarial Networks' concepts were introduced by Ian J in 2014 and people have been interested in its progress ever since. Ever since its introduction, it has mesmerised people with its outputs and the capability to create such vast and surprising outputs.

Here is how the GAN works:

1. The generator picks up the random numbers and returns a created image.
2. This created image is now given to the discriminator with a set of images that are taken from the actual dataset side by side.
3. Now the discriminator picks both real and fake images and gives probabilities, which is a number between 0 and 1, where 1 represents a guess of originality while 0 guesses it is fake.

So, now we have a double feedback loop where the discriminator gives the ground truth of the images, only of which is the truth we know, and while the generator is in a feedback loop with the discriminator where it is trying its best to make the fake images pass the test conducted by the discriminator.

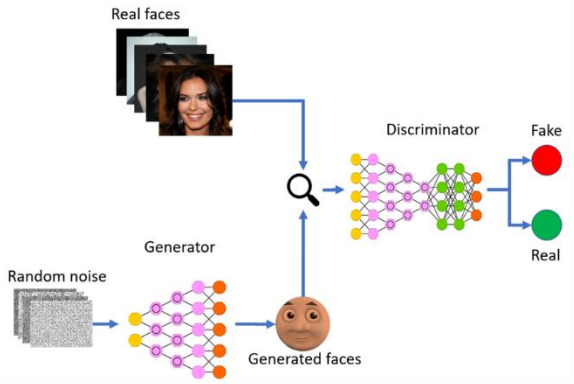

Figure 2: Working of GANs

Now let's have a look at the working mathematically. Discriminator basically takes the help of Binary Classification to figure out the difference between real and fake so its loss function is Binary Cross Entropy. The work that the generator does is Density Estimation, that is to form the noise into real data and then provide it to the Discriminator and try to fool it somehow.

The cost functions basically look like:

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(\boldsymbol{z})\right)\right)$$

$$J^{(G)} = -J^{(D)}$$

The first term in J(D) asks to be fed the actual data to the discriminator, and the discriminator would very much like the real data, resulting in the discriminator trying to increase the log function part of the equation so that it can get a higher probability of providing 1 as an output, which signifies that the data provided is real. All while the second term tells the number of samples that are generated by G. In this second part the discriminator would like to decrease the log function so that it can catch the fake images generated by the generator, and tries to give the output as 0, implying that the output is fake, so that it can remove the fake images and it does it by increasing the log part to provide the output as 0 in the second part of the equation. But in the second part only D is not giving its contribution in the output, there the effect of generator is also present. Here, the generator wants to decrease the probability of the discriminator being correct so that it can pass in as many of the fake images generated by it as it can. So, the output is a sweet spot between both the generator and discriminator being correct in one or the other manner.

Dataset

The dataset provides a great source of test objects on which the training and testing can be done to compare how the output is coming out, as it finds the facial attributes of those objects provided, such as hair colour, facial expression. Images can cover vast varieties of content starting from the object pose, background object, diverse colour which are present in various images and are rich with variations. But our objective is to create a model which looks more real than the objects sourced. Our dataset consists of a collection of images from a particular historical figure from the same time period.

So, loading the dataset:

```
from tqdm import tqdm
import numpy as np
import pandas as pd
import os
from matplotlib import pyplot as plt
PIC_DIR = './drive/img_align_celeba/'
IMAGES_COUNT = 10000
ORIG_WIDTH = 178
ORIG_HEIGHT = 208
diff = (ORIG_HEIGHT - ORIG_WIDTH) // 2
WIDTH = 128
HEIGHT = 128
crop_rect = (0, diff, ORIG_WIDTH, ORIG_HEIGHT - diff)
images = []
for pic_file in tqdm(os.listdir(PIC_DIR)[:IMAGES_COUNT]):
    pic = Image.open(PIC_DIR + pic_file).crop(crop_rect)
    pic.thumbnail((WIDTH, HEIGHT), Image.ANTIALIAS)
    images.append(np.uint8(pic)) #Normalize the images
images = np.array(images) / 255
images.shape #print first 25 images
plt.figure(1, figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i+1)
    plt.imshow(images[i])
    plt.axis('off')
plt.show()
```

Creating the Generator:

We can take the generator as a con artist who is trying to get the fake images approved as real by the discriminator. This algorithm is made up of 8 convolutional layers. First, we take the input and name it as "gen_input" and provide it to our convolution layer. Each convolution layer performs a convolution and then performs a batch normalisation and a Leaky ReLu as well. Then as an output we provide the tanh activation function.

```
LATENT_DIM = 32
CHANNELS = 3
def create_generator():
    gen_input = Input(shape=(LATENT_DIM, ))

    x = Dense(128 * 16 * 16)(gen_input)
    x = LeakyReLU()(x)
    x = Reshape((16, 16, 128))(x)

    x = Conv2D(256, 5, padding='same')(x)
    x = LeakyReLU()(x)

    x = Conv2DTranspose(256, 4, strides=2, padding='same')(x)
    x = LeakyReLU()(x)

    x = Conv2DTranspose(256, 4, strides=2, padding='same')(x)
    x = LeakyReLU()(x)

    x = Conv2DTranspose(256, 4, strides=2, padding='same')(x)
    x = LeakyReLU()(x)

    x = Conv2D(512, 5, padding='same')(x)
    x = LeakyReLU()(x)
    x = Conv2D(512, 5, padding='same')(x)
    x = LeakyReLU()(x)
    x = Conv2D(CHANNELS, 7, activation='tanh', padding='same')(x)

    generator = Model(gen_input, x)
    return generator
```

Creating a discriminator:

The algorithm in the discriminator is composed of the same type of convolutional layer as the generator. For each layer of the algorithm, we are performing convolution, then there is another layer of batch normalisation to make the algorithm (or we can refer it as code here) faster and more realistic, and finally we perform a Leaky ReLu.

```
def create_discriminator():
    disc_input = Input(shape=(HEIGHT, WIDTH, CHANNELS))

    x = Conv2D(256, 3)(disc_input)
    x = LeakyReLU()(x)

    x = Conv2D(256, 4, strides=2)(x)
    x = LeakyReLU()(x)

    x = Conv2D(256, 4, strides=2)(x)
    x = LeakyReLU()(x)

    x = Conv2D(256, 4, strides=2)(x)
    x = LeakyReLU()(x)

    x = Conv2D(256, 4, strides=2)(x)
    x = LeakyReLU()(x)

    x = Flatten()(x)
    x = Dropout(0.4)(x)

    x = Dense(1, activation='sigmoid')(x)
    discriminator = Model(disc_input, x)

    optimizer = RMSprop(
        lr=.0001,
        clipvalue=1.0,
        decay=1e-8
    )

    discriminator.compile(
        optimizer=optimizer,
        loss='binary_crossentropy'
    )

    return discriminator
```

Defining a GAN model:

A GAN model can be said to be a combination of both the generator model and the discriminator model into one large model. This big model is now going to be used to train the model weights in the generator with the help of the output and the errors found by the discriminator model. While the discriminator is trained in a different manner, where the model weights earlier marked as not trainable in this large GAN model are used to cross verify that only the weights provided by the generator model are putted updates on. This kind of modification to the training ability of the discriminator weights only shows its effects when training the whole and combined GAN model, and not while training the discriminator alone on itself.

Here we take the output of the discriminator as sigmoid, so we use binary cross entropy for the loss. RMSProp is taken as an optimiser that generates more realistic fake object images as compared to the Adam

in this case. Right now, the learning rate is 0.0001. Weight decay and clip value stabilise the learning process during the earlier part of the training. If we want to adjust the learning rate of the algorithm we have to deal with the adjustment of the decay.

The GAN tries to recreate an equally distributed probability of features in the objects. So, we can use the loss function that is used to show the distance between the distribution of the data that is generated by the GAN and the distribution of the real data of the objects that was provided.

```
generator = create_generator()
discriminator = create_discriminator()
discriminator.trainable = False
gan_input = Input(shape=(LATENT_DIM, ))
gan_output = discriminator(generator(gan_input))
gan = Model(gan_input, gan_output)#Adversarial Model
optimizer = RMSprop(lr=.0001, clipvalue=1.0, decay=1e-8)
gan.compile(optimizer=optimizer, loss='binary_crossentropy')
```

Rather than using only one single loss function, we need to define three functions of the algorithm that are the loss of the generator, the loss of the discriminator while we are using the real data as an input and the loss of the discriminator when we are using the fake objects as an input. The sum of the real and the fake objects will give us the total discriminator loss.

The training of the GAN model

Training is the most difficult part in GAN since it has two separate parts that need training and those are separate pieces of algorithms. In its training we have to address two major issues:

1. GAN has to work with and in between two different kinds of training. One for generator and the other one for the discriminator.
2. The convergence in the GAN is very hard to identify.

Here the best part to observe is that as with training the generator is improving, the performance of the discriminator gets decreasing. This is because the output from the generator keeps getting better and better while the discriminator is struggling to keep up as the images from the generator keep getting difficult to be distinguishable from fake and real. If an output from the generator gets accepted perfectly, then the discriminator is working at 50% accuracy. As an analogy we can say that the discriminator is now flipping coin to make its predictions.

The code is as follows:-

```
iters = 20000
batch_size = 16RES_DIR = 'res2'
FILE_PATH = '%s/generated_%d.png'
if not os.path.isdir(RES_DIR):
    os.mkdir(RES_DIR)
CONTROL_SIZE_SQRT = 6
control_vectors = np.random.normal(size=(CONTROL_SIZE_SQRT**2, LATENT_DIM)) / 2
start = 0
d_losses = []
a_losses = []
images_saved = 0
for step in range(iters):
    start_time = time.time()
    latent_vectors = np.random.normal(size=(batch_size, LATENT_DIM))
    generated = generator.predict(latent_vectors)

    real = images[start:start + batch_size]
    combined_images = np.concatenate([generated, real])

    labels = np.concatenate([np.ones((batch_size, 1)), np.zeros((batch_size, 1))])
    labels += .05 * np.random.random(labels.shape)

    d_loss = discriminator.train_on_batch(combined_images, labels)
    d_losses.append(d_loss)

    latent_vectors = np.random.normal(size=(batch_size, LATENT_DIM))
    misleading_targets = np.zeros((batch_size, 1))

    a_loss = gan.train_on_batch(latent_vectors, misleading_targets)
    a_losses.append(a_loss)

    start += batch_size
    if start > images.shape[0] - batch_size:
        start = 0

    if step % 50 == 49:
        gan.save_weights('gan.h5')

        print('%d/%d: d_loss: %.4f,  a_loss: %.4f.  (%.1f sec)' % (step + 1, iters, d_loss, a_loss,
time.time() - start_time))

        control_image = np.zeros((WIDTH * CONTROL_SIZE_SQRT, HEIGHT * CONTROL_SIZE_SQRT, CHANNELS))
        control_generated = generator.predict(control_vectors)
        for i in range(CONTROL_SIZE_SQRT ** 2):
            x_off = i % CONTROL_SIZE_SQRT
            y_off = i // CONTROL_SIZE_SQRT
            control_image[x_off * WIDTH:(x_off + 1) * WIDTH, y_off * HEIGHT:(y_off + 1) * HEIGHT, :] =
control_generated[i, :, :, :]
        im = Image.fromarray(np.uint8(control_image * 255))
        im.save(FILE_PATH % (RES_DIR, images_saved))
        images_saved += 1
```

End of code.

Finally, our algorithm is complete and that is how it processes the input.

IV. RESULT AND DISCUSSION

We have seen a new approach to predict how some mythological figures from different periods and religions would have looked like. It is a known fact that different artists from the same period might have wildly different interpretations of the same mythological being, each being correct to some extent in their own right. Using this approach, we will be able to borrow the features from all these interpretations and accurately depict how the mythological figure would have looked like.

V. CONCLUSION

We have developed an effective methodology to predict, compare and contrast how a mythological figure would have looked like according to different interpretations.

Although there is nothing wrong with the idea, there is still huge scope for improvement in the form of industry-academia collaboration to make our implementation more optimized for large scale usage. For starters, we need comprehensive datasets with more than 20,000 epochs to generate the desired output. The algorithm to generate the images can be optimized to deliver more accurate results. Improvements of these scale can't be achieved on a personal level and an active industry-academia collaboration is required to make the idea reach its full potential. With proper execution, the ideas discussed here can disrupt the way we look at these figures and significantly influence the fields of mythology and history.

ACKNOWLEDGMENT

We would like to extend our thanks to Amity University Maharashtra for providing research-oriented infrastructure and Department of Computer Science and Engineering for providing resources for research work.

REFERENCE

[1] Wikipedia, Generative Adversarial Networks, available at: https://en.wikipedia.org/wiki/ Generative_adversarial_network
[2] Medium, available at : A brief introduction to GANs, available at: https://medium.com/ sigmoid /a-brief-introduction-to-gans-and-how-to-code-them-2620ee465c30
[3] KDNuggets, Generate Realistic Human Face Using GAN, available at: https://www.kdnuggets. com/2020/03/generate-realistic-human-face-using-gan.html#