
**TECHNOLOGICAL INSTITUTE OF THE PHILIPPINES**
938 Aurora Blvd., Cubao, Quezon City

**COLLEGE OF ENGINEERING AND ARCHITECTURE**
**ELECTRONICS ENGINEERING DEPARTMENT**

**1ST SEMESTER SY 2022 - 2023**

**Prediction and Machine Learning**

COE 005
ECE41S11

# Generative Adversarial Network

## Generate Cartoon Characters

Submitted to:
**Engr. Christian Lian Paulo Rioflorido**

Submitted on:
**October 21, 2022**

Submitted by:
**So, Don Gabriel L.**

**INTRODUCTION:**

**GAN** is a deep learning technique used to generate some new data from scratch. It also flourishes the what so-called “This waifu does not exist” which becomes a popular topic amongst anime fans due to how you can generate your own “waifu” with just a click. GAN has two networks, the generator and the discriminator network wherein the generator will generate some fake data while the discriminator will be the one to identify a couple of data in the generated fake data as well as the real data. The generator will mimic the real data whereas the discriminator will train its neural network for detecting the fake and real data, both networks will try to improve one another's performance through backpropagation.

```
10000 [Discriminator loss: 0.481456, acc.: 76.31%] [Generator loss: 2.071867]
1/1 [==============================] - 0s 20ms/step
```

**Generated Anime Face at 10k epochs**

**PROCEDURE AND SCREENSHOTS OF THE PROGRAM:**

The editor used for this exercise is the Google Collab.

**1.** First, import the libraries needed for the program.

```
[ ]  import numpy as np
     import os
     import matplotlib.pyplot as plt
     import cv2

     import warnings
     warnings.filterwarnings('ignore')

     import keras
     from tensorflow.keras.optimizers import Adam
     from keras.models import Sequential, Model
     from keras.layers import Dense, LeakyReLU, Reshape, Flatten, Input
     from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Conv2DTranspose

     from tensorflow.compat.v1.keras.layers import BatchNormalization
```

**2.** This is followed by the loading of the data as well as data processing. Here, we created a function where only the set file type (.jpg, .jpeg, .png, .bmp) will be taken and used by the program. We also downloaded the dataset through the code, you just need to have the username and key of your Kaggle account since the url of our dataset is from Kaggle and after downloading, we will store in a variable, which is named images in this program.

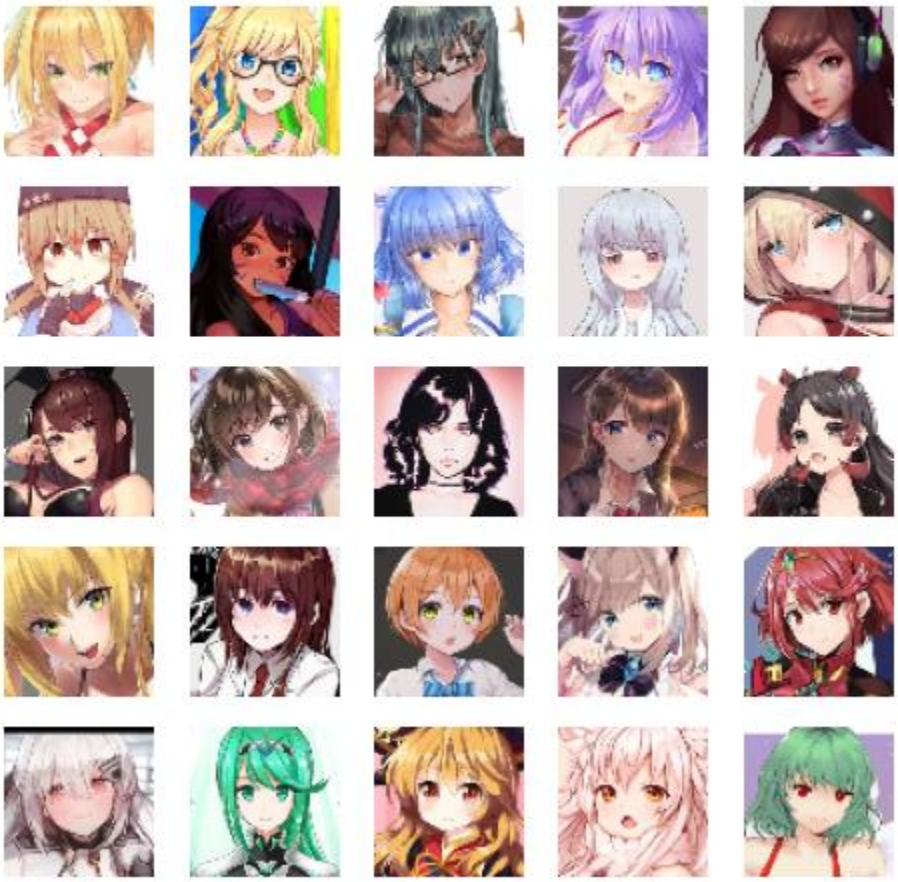
```
[ ]  def list_images(basePath, contains=None):
        # return the set of files that are valid
        return list_files(basePath, validExts=(".jpg", ".jpeg", ".png", ".bmp"), contains=contains)

    def list_files(basePath, validExts=(".jpg", ".jpeg", ".png", ".bmp"), contains=None):
        # loop over the directory structure
        for (rootDir, dirNames, filenames) in os.walk(basePath):
            # loop over the filenames in the current directory
            for filename in filenames:
                # if the contains string is not none and the filename does not contain
                # the supplied string, then ignore the file
                if contains is not None and filename.find(contains) == -1:
                    continue

                # determine the file extension of the current file
                ext = filename[filename.rfind("."):].lower()

                # check to see if the file is an image and should be processed
                if ext.endswith(validExts):
                    # construct the path to the image and yield it
                    imagePath = os.path.join(rootDir, filename).replace(" ", "\\ ")
                    yield imagePath

    def load_images(directory='', size=(64,64)):
        images = []
        labels = []  # Integers corresponding to the categories in alphabetical order
        label = 0

        imagePaths = list(list_images(directory))

        for path in imagePaths:

            if not('OSX' in path):

                path = path.replace('\\','/')

                image = cv2.imread(path) #Reading the image with OpenCV
                image = cv2.resize(image,size) #Resizing the image, in case some are not of the same size

                images.append(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

        return images
```

```
!pip install opendatasets --upgrade --quiet
import opendatasets as od

dataset_url = 'https://www.kaggle.com/datasets/scribbless/another-anime-face-dataset'
od.download(dataset_url) # {"username":"jjggggg","key":"dd2bf9a60631eb54931473f3e45347b4"}
```

```
[ ]  images = load_images('/content/another-anime-face-dataset')
```

**3.** We then set a dimension of 5x5 to display 25 images in the dataset.

```
[ ]  _,ax = plt.subplots(5,5, figsize = (8,8))
     for i in range(5):
         for j in range(5):
             ax[i,j].imshow(images[5*i+j])
             ax[i,j].axis('off')
```



**4.** We created a function for building our GAN model. Here, we build the generator network as well as the discriminator network. The generator network is the one responsible for generating the fake data/images while the discriminator acts as a classifier to identify or distinguish the real data part of the generated image.

```
[ ]  class GAN():
        def __init__(self):
            self.img_shape = (64, 64, 3)

            self.noise_size = 100

            optimizer = Adam(0.0002,0.5)

            self.discriminator = self.build_discriminator()
            self.discriminator.compile(loss='binary_crossentropy',
                                       optimizer=optimizer,
                                       metrics=['accuracy'])

            self.generator = self.build_generator()
            self.generator.compile(loss='binary_crossentropy', optimizer=optimizer)

            self.combined = Sequential()
            self.combined.add(self.generator)
            self.combined.add(self.discriminator)

            self.discriminator.trainable = False

            self.combined.compile(loss='binary_crossentropy', optimizer=optimizer)

            self.combined.summary()
```

```
# Creating the generator, the large kernels in the convolutional layers allow the network to create complex structures.
def build_generator(self):
    epsilon = 0.00001 # Small float added to variance to avoid dividing by zero in the BatchNorm layers.
    noise_shape = (self.noise_size,)

    model = Sequential()

    model.add(Dense(4*4*512, activation='linear', input_shape=noise_shape))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Reshape((4, 4, 512)))

    model.add(Conv2DTranspose(512, kernel_size=[4,4], strides=[2,2], padding="same",
                              kernel_initializer= keras.initializers.TruncatedNormal(stddev=0.02)))
    model.add(BatchNormalization(momentum=0.9, epsilon=epsilon))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Conv2DTranspose(256, kernel_size=[4,4], strides=[2,2], padding="same",
                              kernel_initializer= keras.initializers.TruncatedNormal(stddev=0.02)))
    model.add(BatchNormalization(momentum=0.9, epsilon=epsilon))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Conv2DTranspose(128, kernel_size=[4,4], strides=[2,2], padding="same",
                              kernel_initializer= keras.initializers.TruncatedNormal(stddev=0.02)))
    model.add(BatchNormalization(momentum=0.9, epsilon=epsilon))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Conv2DTranspose(64, kernel_size=[4,4], strides=[2,2], padding="same",
                              kernel_initializer= keras.initializers.TruncatedNormal(stddev=0.02)))
    model.add(BatchNormalization(momentum=0.9, epsilon=epsilon))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Conv2DTranspose(3, kernel_size=[4,4], strides=[1,1], padding="same",
                              kernel_initializer= keras.initializers.TruncatedNormal(stddev=0.02)))
```

```
        # Standard activation for the generator of a GAN
        model.add(Activation("tanh"))

        model.summary()

        noise = Input(shape=noise_shape)
        img = model(noise)

        return Model(noise, img)
```

```
def build_discriminator(self):

    model = Sequential()

    model.add(Conv2D(128, (3,3), padding='same', input_shape=self.img_shape))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization())
    model.add(Conv2D(128, (3,3), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(3,3)))
    model.add(Dropout(0.2))

    model.add(Conv2D(128, (3,3), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization())
    model.add(Conv2D(128, (3,3), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(3,3)))
    model.add(Dropout(0.3))

    model.add(Flatten())
    model.add(Dense(128))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(128))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(1, activation='sigmoid'))

    model.summary()

    img = Input(shape=self.img_shape)
    validity = model(img)

    return Model(img, validity)
```

**4.** We start to train the generator as well as the discriminator network for the last part of defining the GAN model. Here, the program defines the loss definition for the model.

```
def train(self, epochs, batch_size=128, metrics_update=50, save_images=100, save_model=2000):

    X_train = np.array(images)
    X_train = (X_train.astype(np.float32) - 127.5) / 127.5

    half_batch = int(batch_size / 2)

    mean_d_loss=[0,0]
    mean_g_loss=0

    for epoch in range(epochs):
        idx = np.random.randint(0, X_train.shape[0], half_batch)
        imgs = X_train[idx]

        noise = np.random.normal(0, 1, (half_batch, self.noise_size))
        gen_imgs = self.generator.predict(noise)

        # Training the discriminator

        # The loss of the discriminator is the mean of the losses while training on authentic and fake images
        d_loss = 0.5 * np.add(self.discriminator.train_on_batch(imgs, np.ones((half_batch, 1))),
                              self.discriminator.train_on_batch(gen_imgs, np.zeros((half_batch, 1))))

        # Training the generator
        for _ in range(2):
            noise = np.random.normal(0, 1, (batch_size, self.noise_size))

            valid_y = np.array([1] * batch_size)
            g_loss = self.combined.train_on_batch(noise, valid_y)

        mean_d_loss[0] += d_loss[0]
        mean_d_loss[1] += d_loss[1]
        mean_g_loss += g_loss
```

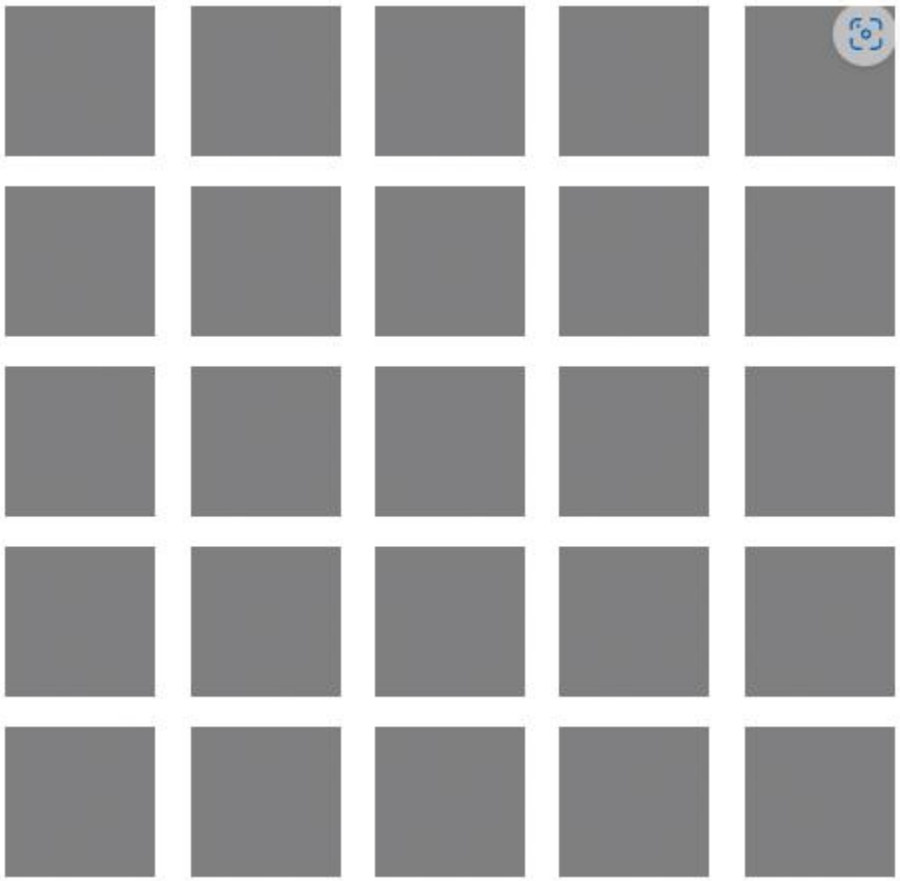
```
        # We print the losses and accuracy of the networks every 200 batches mainly to make sure the accuracy of the discriminator
        # is not stable at around 50% or 100% (which would mean the discriminator performs not well enough or too well)
        if epoch % metrics_update == 0:
            print ("%d [Discriminator loss: %f, acc.: %.2f%%] [Generator loss: %f]" % (epoch, mean_d_loss[0]/metrics_update, 100*mean_d_loss[1]/metrics_update, mean_g_loss/metrics_upd
            mean_d_loss=[0,0]
            mean_g_loss=0

        # Saving 25 images
        if epoch % save_images == 0:
            self.save_images(epoch)

        # We save the architecture of the model, the weights and the state of the optimizer
        # This way we can restart the training exactly where we stopped
        if epoch % save_model == 0:
            self.generator.save("generator_%d" % epoch)
            self.discriminator.save("discriminator_%d" % epoch)

# Saving 25 generated images to have a representation of the spectrum of images created by the generator
def save_images(self, epoch):
    noise = np.random.normal(0, 1, (25, self.noise_size))
    gen_imgs = self.generator.predict(noise)

    # Rescale from [-1,1] into [0,1]
    gen_imgs = 0.5 * gen_imgs + 0.5

    fig, axs = plt.subplots(5,5, figsize = (8,8))

    for i in range(5):
        for j in range(5):
            axs[i,j].imshow(gen_imgs[5*i+j])
            axs[i,j].axis('off')

    plt.show()

    fig.savefig("Generated_Anime/Faces_%d.png" % epoch)
    plt.close()
```

**5.** We started to train and let the model generate new anime faces. We set the epochs to 10,001 with images being generated and save every 1k epoch. There's a total of 11 pictures includes epochs at 1 wherein there is no anime faces generated. The generated images will then be saved in a file.

```
[7] #This folder will contain the images generated during the training
    !mkdir Generated_Anime
```

```
gan=GAN()
gan.train(epochs=10001, batch_size=256, metrics_update=200, save_images=1000, save_model=10000)
```

```
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 18ms/step
4/4 [==============================] - 0s 19ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 19ms/step
4/4 [==============================] - 0s 19ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 19ms/step
4/4 [==============================] - 0s 19ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 19ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 20ms/step
4/4 [==============================] - 0s 19ms/step
```



**Epoch at 1. No generated anime faces**

```
1000 [Discriminator loss: 0.600680, acc.: 67.45%] [Generator loss: 1.274487
1/1 [==============================] - 0s 18ms/step
```

**Epoch at 1k. First generated anime faces**

```
2000 [Discriminator loss: 0.556534, acc.: 71.15%] [Generator loss: 1.384334]
1/1 [==============================] - 0s 15ms/step
```

**Epoch at 2k. Generated anime faces**

```
3000 [Discriminator loss: 0.541438, acc.: 73.72%] [Generator loss: 1.150347]
1/1 [==============================] - 0s 16ms/step
```

**Epoch at 3k. Generated anime faces**

```
4000 [Discriminator loss: 0.577525, acc.: 69.32%] [Generator loss: 1.309778]
1/1 [==============================] - 0s 22ms/step
```

**Epoch at 4k. Generated anime faces**

```
5000 [Discriminator loss: 0.562326, acc.: 70.94%] [Generator loss: 1.424783
1/1 [==============================] - 0s 16ms/step
```

**Epoch at 5k. Generated anime faces**

```
6000 [Discriminator loss: 0.534986, acc.: 72.98%] [Generator loss: 1.687882]
1/1 [==============================] - 0s 17ms/step
```

**Epoch at 6k. Generated anime faces**

```
7000 [Discriminator loss: 0.524657, acc.: 73.37%] [Generator loss: 1.781694]
1/1 [==============================] - 0s 17ms/step
```

**Epoch at 7k. Generated anime faces**

```
8000 [Discriminator loss: 0.488257, acc.: 75.89%] [Generator loss: 2.071867]
1/1 [==============================] - 0s 18ms/step
```

**Epoch at 8k. Generated anime faces**

```
9000 [Discriminator loss: 0.481456, acc.: 76.31%] [Generator loss: 2.112323]
1/1 [==============================] - 0s 20ms/step
```

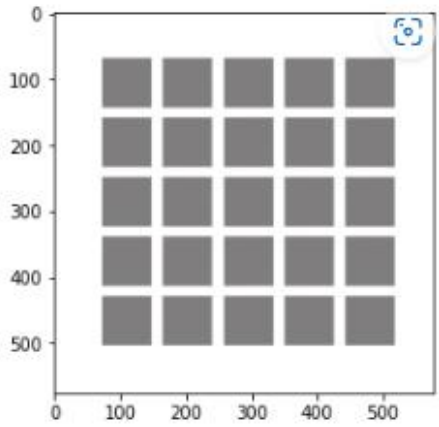
**Epoch at 9k. Generated anime faces**

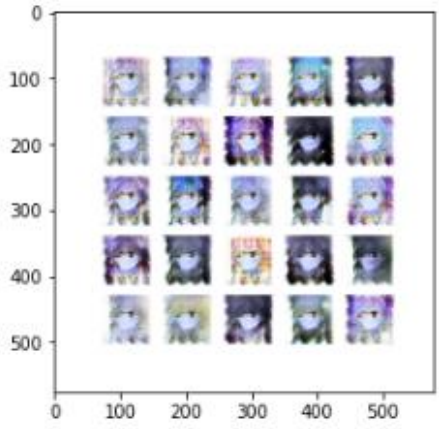
```
10000 [Discriminator loss: 0.481456, acc.: 76.31%] [Generator loss: 2.071867]
1/1 [==============================] - 0s 20ms/step
```

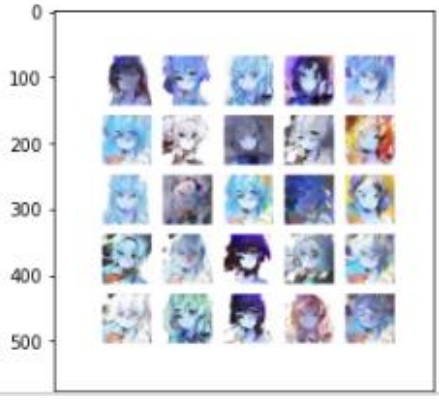
**Epoch at 10k. Final generated anime faces for the prediction**

**15.** We display all of the generated image and we can see that as epoch increases, the model is becoming better in generated anime faces.

```
imdir = '/content/Generated_Anime'
ext = ['png', 'jpg', 'jpeg']    # Add image formats here
images = os.listdir(imdir)

for img in images:
  img_arr = cv2.imread(os.path.join(imdir,img))
  plt.figure()
  plt.imshow(img_arr)
```



**DISCUSSION:**

This activity helped me to learn and understand how GAN is able to generate new examples from existing samples and one of its applications is what was done in this activity, generating anime faces. I was able to generate new anime faces just from the dataset that I have gathered. Here, we can see the step-by-step process of how the program improves its performance. This activity also emphasizes that the use of the graphics processing unit (GPU) was essential for the accomplishment of this activity. When I first run the program without any GPU, the program took 10 minutes just to do 50 epochs while when I activated the GPU as hardware accelerator, it only took 10 minutes to do 200 epochs. Due to limited free hardware accelerator of Google Colab, I was only able to do 10k epochs but we can see that the program starts to become better at generating anime faces at this epoch. The program is able to generate anime faces with good quality. Further epochs will make the program to generate a better and more realistic anime faces.

**References:**

Reference for the code: https://www.kaggle.com/code/nassimyagoub/gan-anime-faces/notebook

Reference for the dataset: https://www.kaggle.com/datasets/scribbless/another-anime-face-dataset

Brownlee, J. (2019, July 12). *18 impressive applications of generative adversarial networks (Gans)*. Machine Learning Mastery. Retrieved October 21, 2022, from https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks/

Ihalapathirana, A. (2022, January 13). *Generative adversarial networks for anime face generation - pytorch*. Medium. Retrieved October 21, 2022, from https://aihalapathirana.medium.com/generative-adversarial-networks-for-anime-face-generation-pytorch-1b4037930e21