

**Gernating Anime Character  
Using Gan's**

**In**

**Computer science and Engineering**

**By**

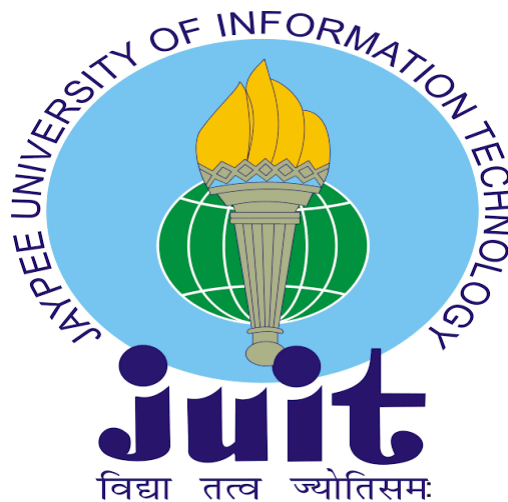
Nikhil Nagpal

171224

Under supervision of

(Professor Dr Vivek seghal)

To



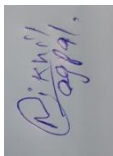
Department of computer science & Engineering and information Technology

**Jaypee University of Information Technology Wagnaghat, Solan 173234,  
Himachal Pradesh**

## Declaration

We hereby declare that the work presented in this report entitled "**ANIME GERNATION**" in partial fulfilment of requirements for the degree of "**Bachelor of Technology in computer science and Engineering**" submitted in the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology waknaghat, is an authentic report of our work arried out over a period from August 2020 under supervision of **Dr. Vivek Sehgal** ( Associate Professor, CSE/IT Department).

The matter embedded in the report has not been submitted for the award of any degree or diploma.



Nikhil Nagpal

171224

This is to certify that the above statement is made by candidate is true to the best of my knowledge.



Dr. Vivek Sehgal

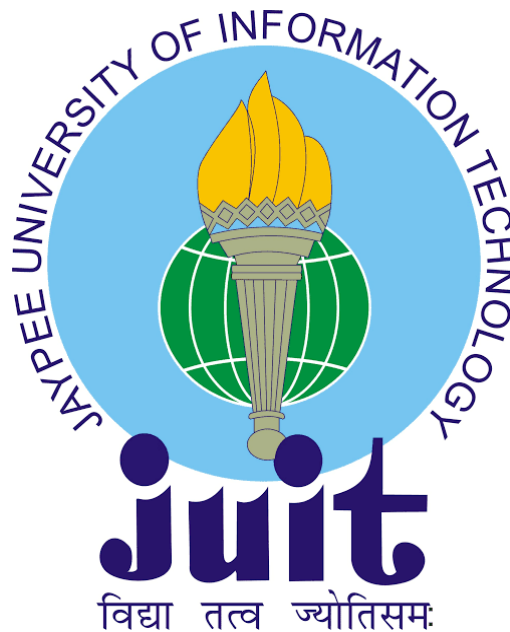
Associate Professor

CSE/IT Department

Dated

(II)

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING  
&  
INFORMATION TECHNOLOGY



**CERTIFICATE**

This is to certify that the work in this project title as “**Anime Face Generation**” is entirely written ,successfully completed and demonstrated by the following student themselves as a fulfillment of requirement for Bachelor’s of Engineering in Computer Science.

Nikhil Nagpal

(171224)

(III)

## **ACKNOWLEDGEMENTS**

We are highly indebted to all the members of **Computer Science Department, Jaypee University of Information Technology** for their guidance and constant supervision as well as providing necessary information regarding the project and for their support in completing the project.

We would like to express our gratitude towards **Dr. Vivek Sehgal**, Associate Professor, for his kind cooperation and encouragement which helped us in completion of this project and for giving us such attention and time.

(IV)

**TABLE OF CONTENTS**

Chapter	Page
1. Introduction.....	1
1.1 Motivation.....	1
1.2 Objectives.....	2
1.3 Dataset.....	2
1.4 Problem Statement.....	2
1.5 Methodology.....	2
1.6 Scope.....	2
<b>Chapter 2. Generative Adversarial Networks</b>	<b>3</b>
2.1 GANs.....	3
2.2 Architecture of gan's.....	4
2.3 Loss Function.....	5
2.4 Mode Collapse.....	8
2.5 Nash Equilibrium.....	9
2.6 Latent space.....	11
3. System Development	15
3.1 Working of GAN's.....	15
3.2 Working of DCGAN'S.....	16
3.3 Implementation of DCGAN'S.....	20
3.4 Training of DCGAN'S.....	24
4. Result.....	25
5 Conclusion.....	38

## List of Figures

<b>1 Figure 1.0</b>	<b>2</b>
<b>2 Figure 2.1</b>	<b>3</b>
<b>3 Figure 2.2</b>	<b>4</b>
<b>4 Figure 2.3</b>	<b>5</b>
<b>5 Figure 2.4</b>	<b>5</b>
<b>6 Figure 2.5</b>	<b>10</b>
<b>7 Figure 2.6</b>	<b>12</b>
<b>8 Figure 2.7</b>	<b>13</b>
<b>9 Figure 3.1</b>	<b>16</b>
<b>10 Figure 4.1</b>	<b>31</b>
<b>11 Figure 4.2</b>	<b>34</b>
<b>12 Figure 4.3</b>	<b>34</b>
<b>13 Figure 4.4</b>	<b>34</b>
<b>15 Figure 4.5</b>	<b>35</b>
<b>16 Figure 4.6</b>	<b>36</b>
<b>17 Figure 4.7</b>	<b>36</b>
<b>18 Figure 4.8</b>	<b>37</b>
<b>19 Figure 5.1</b>	<b>38</b>
<b>20 Figure 5.2</b>	<b>38</b>
<b>21 Figure 5.3</b>	<b>39</b>
<b>22 Figure 5.4</b>	<b>39</b>
<b>23 Figure 5.5</b>	<b>40</b>
<b>24 Figure 5.6</b>	<b>40</b>

# CHAPTER 1

## Introduction

### 1.1 Motivation

With the current digital climate, data is the most valuable resource there is. And corporations routinely collect incomprehensible amounts of data regularly from the users which goes towards targeted marketing or fodder for data sets to run models upon. Real data has many drawbacks such as the need to find, gather and organise it. Additionally, real data is often limited in quantity and might not be sufficient for the requirements of the task.

On top of that the real data is often strictly regulated due to privacy concerns as it may contain personal information about the users. Synthetic data can provide a much safer alternative in situations like these. The crux of synthetic data is generating new data which can be used in the form of completely separate dataset for model evaluation and training or as test data. Since, Synthetic datasets do not contain any of the real data, it does not allow users to access the real data, which solves our privacy problem to begin with. Additionally, this data can be used to add to the existing data. The present deep learning models more often than not require great amount of data for generalization.

In generating anime character, we will explore special case when the generative model generates samples by passing a random noise vector and on the other hand a discriminative model will be used to discriminate whether the image is real or fake. Through multiple cycles of generation and discrimination, both the networks train each other, while simultaneously trying to outwit each other, where a generative data uses existing data to generate new data and discriminator network tries to differentiate.

However, Synthesis data is an ever evolving field of research. At first, synthetic data was usually created by modeling some joint multivariate probability distribution, which was then sampled. Example models include Bayesian networks and Gaussian Distribution Networks, but most of these methods have one or more restrictions related to data size or complexity. Recent developments in work on Generative Adversarial Networks (GANs) have shown promising results with tabular data

Synthetic data is algorithmically generated information that imitates real-time information. This type of data is a substitute for datasets that are used for testing and training. Since the very beginning, synthetic data has been helping corporations from different domains to validate and train machine-learning models.

## 1.2 Objective

Objective of the project is to overcome one of the major problem of getting tons of data to deal with the overfitting problem of deep learning and artificial intelligence problem here we create a lots of data from noise vector and some of the data for example with some 50's or 100's images we can generate large dataset which help to avoid overfitting problem.

## 1.3 Dataset

The dataset is created by crawling anime database websites using `curl`. The script `anime_dataset_gen.py` crawls and processes the images into 64x64 PNG images with only the faces cropped.

## 1.4 Problem statement

Generate dataset using some of the input images

## 1.5 Methodology

- i. Image collection
- ii. Visualization
- iii. Generate random noise and make it input for generator neural network. It will output generated fake data.
- iv. Combining of fake generated data and real data and consider them as input to discriminator
- v. Discriminator will try to learn by predicting whether the data is fake or real

## 1.6 Scope

This will give a framework to stimulate generative modeling of anime style images and eventually help both amateurs and professional designers to create new anime characters. We explored automatic creation of anime character. In future, we can try to improve the GAN model when class labels in training dataset are not evenly distributed. Also improvement in final resolution of generated images can be worked upon.



## CHAPTER 2

### 2.1 Generative Adversarial Networks

Generative adversarial networks are a deep-learning-based architecture which is used for training a generative model..

GANs model architecture has two sub-models-

1. Discriminator model : it's role is of classification whether or not input fed into it is real or generated by the Generator
2. Generator model : it's role is to generate input data from latent random noise.It is used to generate new data which can plausibly be part of the problem domain

Both ,the generator and discriminator, are trained together.

The generator generates a batch of samples, and these along with real examples from the domain are provided to the discriminator and classified as real or fake. The generator a multivariate gaussian distribution to a random distribution

The discriminator is updated so that it can get better at differentiating between the real and fake batch data in the next iteration, the generator is also updated based on how good the generated data was good at fooling the discriminator.. The discriminator is supposed to tell whether the distribution parameterized by the generator is comparable to real training data.

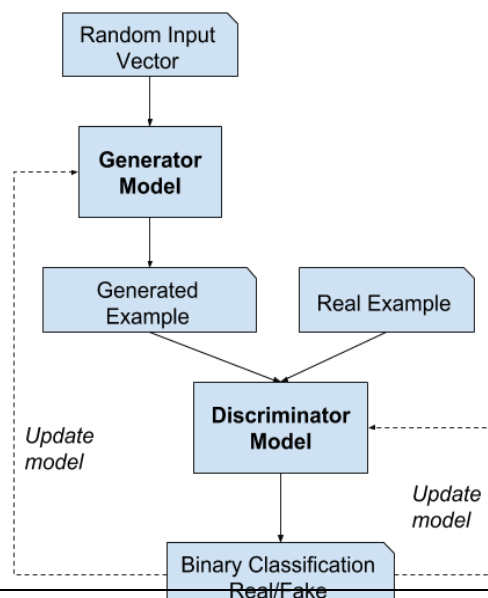


Figure 2.1

### Some parameters and variables

- $\theta_d$ = Parameter of discriminator
- $\theta_g$ = Parameter of generator
- $P_z(z)$ = Input noise distribution
- $P_{data}(x)$ =Original data distribution
- $P_g(x)$ = Generated distribution

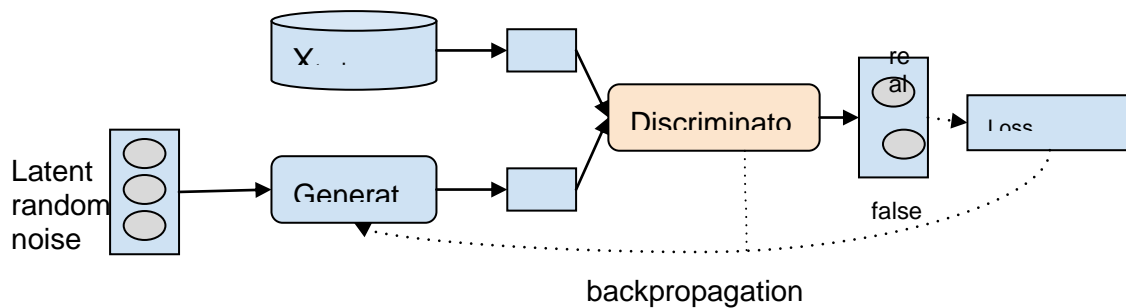


Figure 2.2: GAN architecture

## 2.2 Architecture

The architecture of a GAN has two basic elements: the generator network and the discriminator network. Each network can be any neural network, such as an Artificial Neural Network (ANN), a Convolutional Neural Network (CNN), etc.

### Generator

The generator model takes a fixed-length random vector as input and generates a sample in the domain. The vector is drawn from randomly from a Gaussian distribution, and the vector is used to seed the generative process. After training, points in this multidimensional vector space will correspond to points in the problem domain, forming a compressed representation of the data distribution. This vector space is referred to as a latent space, or a vector space comprised of latent variables. Latent variables, or hidden variables, are those variables that are important for a

domain but are not directly observable. It has five layers: an input layer, three hidden layers, and an output layer.

4

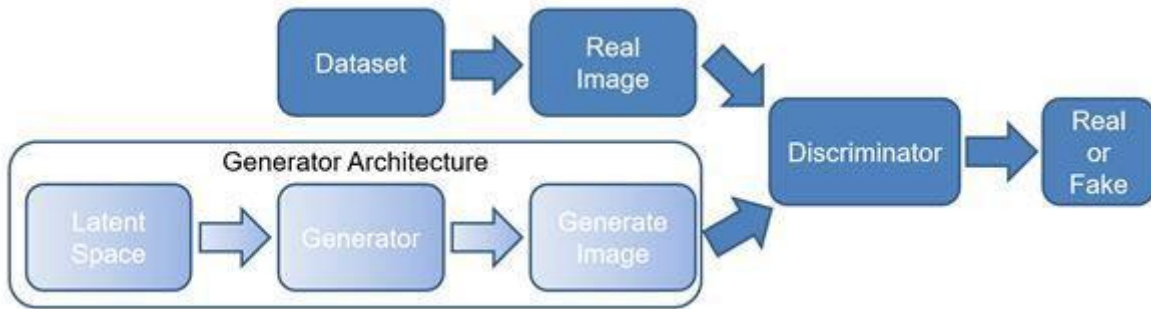


Figure 2.3:

### Discriminator

The discriminator model takes an example from the domain as input (real or generated) and predicts a binary class label of real or fake (generated). The real example comes from the training dataset. The generated examples are output by the generator model.

The discriminator is a normal (and well understood) classification model. After the training process, the discriminator model is discarded as we are interested in the generator.

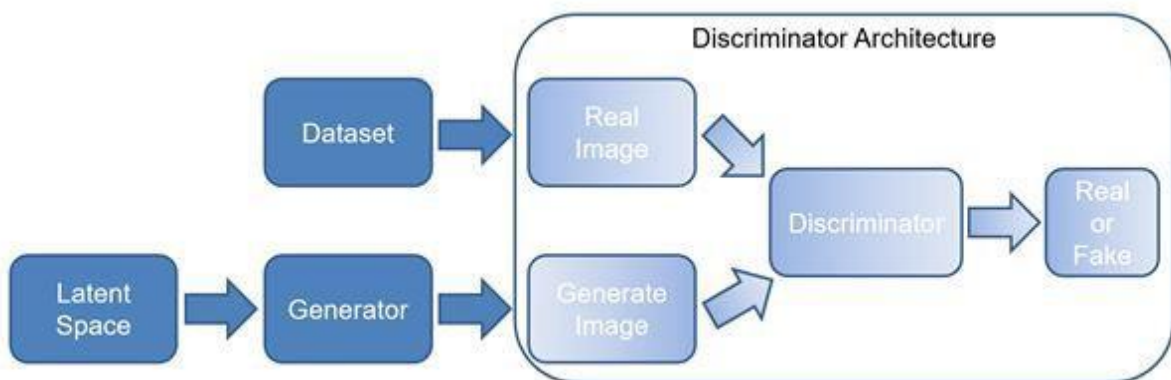


Figure 2.4

## 2.3 Loss Functions

The loss function described in the original paper written by Ian Goodfellow. can be derived from the formula of binary cross-entropy loss. The binary cross-entropy loss can be written as,

$$L(y^{hat}, y) = [y \cdot \log(y^{hat}) + (1-y) \cdot \log(1-y^{hat})]$$

### Discriminator Loss Function

While training the discriminator, the label of data coming from  $P_{data}(x)$  is  $y = 1$  for real data and  $\hat{y} = D(x)$ . Putting these values in the above loss function we get,

$$L(D(x), 1) = \log(D(x))$$

And for data coming from the generator, the label is  $y = 0$  for fake data and  $\hat{y} = D(G(z))$ . So in this case, we get

$$L(D(G(z)), 0) = \log(1-D(G(z)))$$

the role of the discriminator is to correctly classify the fake and real dataset. For this above equations should be maximized and the final loss function for the discriminator can be given as

$$L^{(D)} = \max[\log D(x) + \log(1 - D(G(x)))]$$

### Generator Loss Function

Since the generator is competing against the discriminator. So, it will try to minimize the above equation and loss function is given as,

$$L^{(G)} = \min [\log D(x) + \log(1 - D(G(x)))]$$

6

Thus the combined loss function

$$L = \min_G \max_D [\log D(x) + \log(1 - D(G(x)))]$$

Remember that the above loss function is valid only for a single data point, to consider the entire dataset we need to take the expectation of the above equation as

$$\min_G \max_D V(D, G) = \min_G \max_D (E_{x \sim p(x)} [\log D(x)] + E_{x \sim p(x)} [\log(1 - D(G(x)))])$$

The algorithm as per **Ian Goodfellow** original paper on GAN:

Mini Batch stochastic gradient descent training of generative adversarial nets.

For a number of training iterations we do.

For k steps do

**Part 1**

- Sample mini-batch of m noise sample  $\{Z^{(1)} \dots Z^{(m)}\}$  form noise prior to  $P_g(z)$
- Sample of mini-batch of m noise sample  $\{x^{(1)} \dots x^{(m)}\}$  from data generating distribution  $P_{data}(x)$
- Update the discriminator by ascending its stochastic gradient

$$1/m \sum_{i=1}^m [\log D(x^{(i)}) + \log(1 - D(G(Z^{(i)})))]$$

End for

**Part 2**

- Sample mini-batch of m noise samples  $\{Z^{(1)} \dots Z^{(m)}\}$  form noise prior to  $P_g(z)$
- Update the generator by descending its stochastic gradient

$$\partial \theta / \partial \theta \sum_{i=1}^m \log(1 - D(G(Z^{(i)})))$$

End for

The gradient-based updates can use any standard gradient-based learning rule. We use momentum in our experiments

---

As we can see from the above algorithm that both the generator and the discriminator are trained separately. In part 1 real data, fake data are input into the discriminator with correct labels and then we train the discriminator. Gradients are propagated through the network keeping the generator fixed. We update the discriminator by increasing the stochastic gradient as we want the discriminator to maximize its loss function.

In part 2, update the values in the generator model by keeping the discriminator static and inputting fake data with fake labels in order to trick the discriminator. We update generator model by reducing its stochastic gradient as we want to minimize the generator loss function

## 2.4 Mode Collapse

In a perfect world, GAN will produce a wide variety of outputs, but if a generator outputs a very realistic, plausible output it may learn to produce only that one output. In essence, the role of the generator is to produce that one output to trick the discriminator.

However, if this happens over and over again, the discriminator learns to always reject this output. This may make it stall into local minima and doesn't track down the best strategy then it is excessively simple for generator iteration to trick the discriminator.

Every iteration of the generator over-optimizes for a specific discriminator, and the discriminator never figures out how to get familiar on how to break out of the cycle. Consequently the generators rotate through a small set of output values which brings about mode collapse..

The objective of the gan generator is to create output that can fool the discriminator  $D$  the most as possible.

$$\frac{\partial \theta l}{\partial \theta} = \sum_{i=1}^n \lambda_i \frac{\partial l}{\partial \theta} (1 - \sigma(\sigma(\theta(\sigma(\theta))))))$$

One verge case is when the Generator trains extensively without any updates to the Discriminator. The generated (output of generator ) output will converge to find the optimal output  $x^*$  which will fool the Discriminator the most and the most realistic output from the discriminator point of view. In this extreme,  $x^*$  will be independent of  $z$ .

$$x^* = \operatorname{argmax}_x D(x)$$

The mode collapses to a single point. The gradient associated with  $z$  tends to zero. When we restart the training in the discriminator, the most effective way to detect generated images is to detect this single mode. Since the generator reduces the impact of  $z$ , the gradient from the discriminator will probably push the single point around for the next vulnerable mode

The generator produces such an imbalance of modes in training that it degrades its ability to detect others. Now, both networks i.e generator and discriminator are overfitted to exploit short-term opponent weakness. As a result, the model fails to converge

## 2.5 Nash equilibrium

GANs are based on the zero-sum game concept i.e. if one wins the other loses as fixed. In a zero-sum game also called minimax your opponent wants to maximize their actions and your action to minimize them.

The GANs model converges when the discriminator and the generator reach a point known as Nash equilibrium. This optimal point is represented as -

$$\min_G \max_D V(D, G) = E_{x \sim p(x)} [\log D(x)] + E_{x \sim p(x)} [\log (1 - D(x))]$$

Since both sides want to undermine the others, a Nash equilibrium happens when one player will not change its action regardless of what the opponent may do. Consider two players A and B which manipulate and survey the value of “x and y” respectively. Player A wants to maximize the value of “xy” while B wants to minimize it.

$$\min_B \max_A V(D, G) = xy$$

The Nash equilibrium happens when  $x=y=0$ . This is the only state where the action of your opponent does not matter. It is the only state that any opponent’s actions will not change the game decision.

Let's see whether we can find the point of Nash equilibrium easily using gradient descent. We update the parameters  $x$  and  $y$  based on the gradient of the value function  $V$ .

$$\Delta x = \alpha (\partial V / \partial x)$$

$$\Delta y = -\alpha (\partial V / \partial y)$$

where  $\alpha$  (*alpha*) is the learning rate of problems. When we plot  $x$  and  $y$ , and  $xy$  against the training iterations, we realize our solution does not converge.

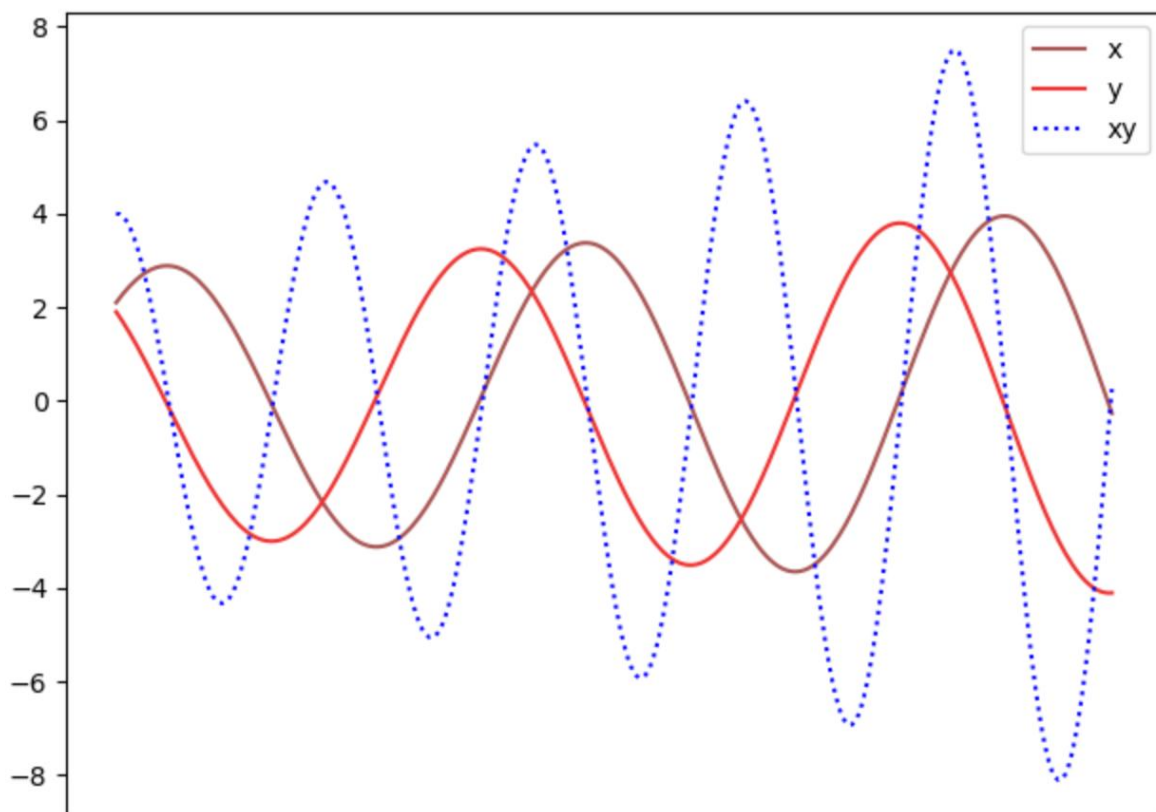


Figure 2.5 : Representation of when Nash Equilibrium isn't reached

If we increase the learning rate or train the model longer duration of time, we can see the parameters  $x$ ,  $y$  is unstable with big swings

How do you find Nash Equilibrium?

To find the Nash equilibrium in a game, one would have to model out each of the possible scenarios to determine the results and then choose what the optimal strategy would be. In a two-person game, this would take into consideration the possible strategies that both players could choose. If neither player changes their strategy knowing all of the information, a Nash equilibrium has occurred



## 2.6 Latent space

It simply means a presentation of compressed data. The concept of “latent space” is important because its utility is at the core of ‘deep learning’ — learning the features of data and simplifying data representations for the purpose of finding patterns. The latent space representation of our data contains all the important information needed to represent our original data point. This representation must then represent the features of the original data. In other words, the model learns the data features and simplifies its representation to make it easier to analyze. This is at the core of a concept called Representation Learning, defined as a set of techniques that allow a system to discover the representations needed for feature detection or classification from raw data. In this use case, our latent space representations are used to transform more complex forms of raw data (i.e. images, video), into simpler representations which are ‘more convenient to process’ and analyze.

## 2.6 Pre-defined Networks

### A. Residual Networks(ResNet)

A residual network is an artificial neural network particularly known as ANN. It is used for computer vision tasks. ResNet makes it possible to train up to hundreds or even thousands of layers and still achieves compelling performance. Taking advantage of its powerful representational ability, the performance of many computer vision applications other than image classification has been boosted, such as object detection and face recognition. The core idea of ResNet is introducing a so-called “identity shortcut connection”. For face detection we will be using the pre-trained Inception-ResNet-2 model without fully connected layers.

Resnet architecture was evaluated on ImageNet 2012 classification dataset consisting of 1000 classes. The model was trained on the 1.28 million training images and evaluated on the 50k validation images. Moreover, 100k images were used for testing the model accuracy

## B. ImageNet

ImageNet is a dataset of over 15 million high resolution images belonging to roughly 22,000 categories. These images are collected from the web. It started in 2010, as a part of an annual competition called ILSVRC. It uses a subset of ImageNet with roughly 1000 images in each 1000 categories. In all there are roughly 1.2 million training images, 50,000 validation images, and 150000 testing images.

ImageNet consists of variable-resolution images. Therefore, the images have been down-sampled to a fixed resolution of  $256 \times 256$ . Given a rectangular image, the image is rescaled and cropped out the central  $256 \times 256$  patch from the resulting image.

## C. VGG16-Convolutional Network for Classification and Detection

VGG16 is a convolutional neural network model, proposed in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”. The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It makes the improvement over AlexNet by replacing large kernel-sized filters with multiple  $3 \times 3$  kernel-sized filters one after another.

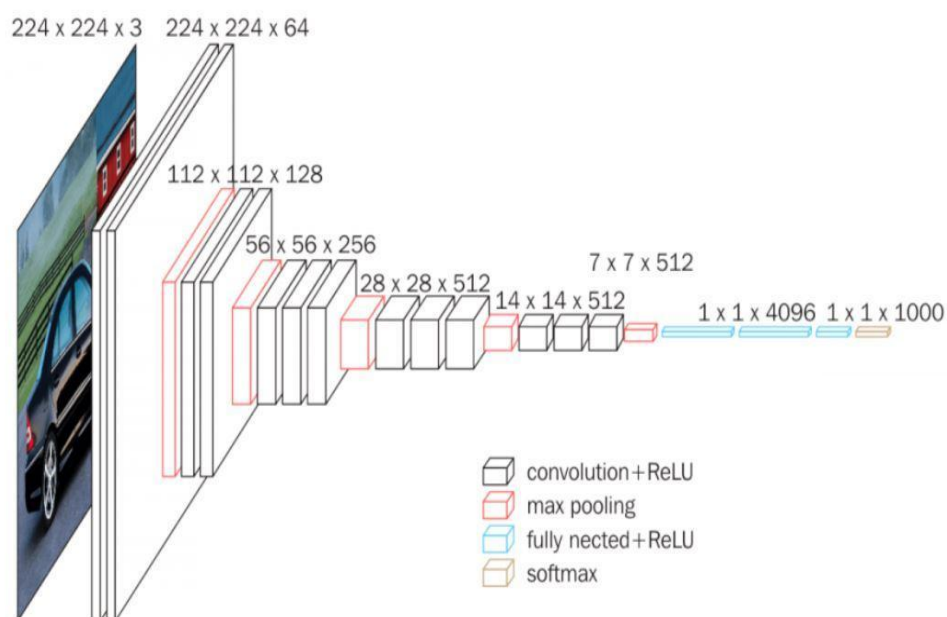


Figure 2.6 : Architecture of VGG16

The input to the conv1 layer is a 224 x 224 RGB image with a fixed size. The image is passed through a series of convolutional layers, each with a very small receptive field: 3x3 (the smallest size that captures the concepts of left/right, up/down, and centre). In one of the configurations, it also utilizes 1x1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3x3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers. Max-pooling is performed over a 2x2 pixel window, with stride 2.

Following a stack of convolutional layers, three Fully-Connected layers are added; the first two have 4096 channels each, while the third performs 1000-way ILSVRC classification and thus has 1000 channels. The soft-max layer is the final layer. In all networks, the completely connected layers are configured in the same way.

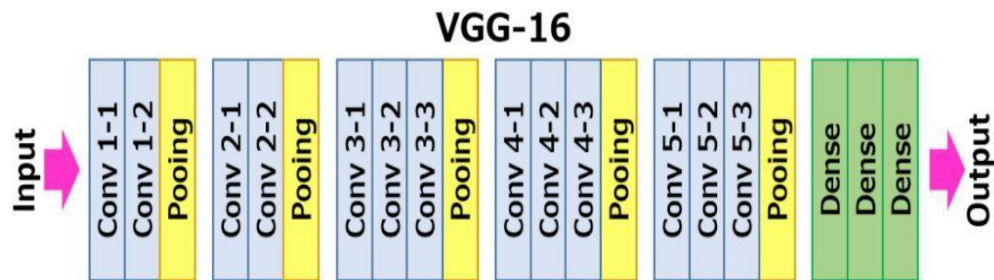


Figure 2.7 : VGG-16

Figure 2 illustrates the ConvNet configurations. The nets are known by their names (A-E). All of the configurations are based on a common design found in architecture, with the only difference being the depth: from the network's 11 weight layers A to 19 weight layers in the network E (8 conv. and 3 FC layers) (16 conv. and 3 FC layers). The number of channels in the conv. layers is minimal, starting at 64 in the first layer and rising by a factor of two after each max-pooling layer., until it reaches 512.

## D.Cascade Classifier

Object Detection using feature-based cascade classifiers is an effective object detection method. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. An OpenCV based system that uses a Haar Cascade Classifier to detect facial features of Japanese anime characters in a given image file. The **cascade classifier** consists of a collection of stages, where each stage is an ensemble of weak learners. If the label is positive, the **classifier** passes the region to the next stage. The detector reports an object found at the current window location when the final stage classifies the region as positive.

## **CHAPTER 3**

### **(SYSTEM DEVELOPMENT)**

#### **3.1 Working Of GANs**

The Discriminator and Generator are the two neural networks that make up GAN. GAN would pit these two networks against one another in a Zero-Sum game (Game Theory). This is a game that these agents are playing (the networks). This is where the adversarial name in GAN comes from.

Generator will produce some fake data, and the Discriminator will define a few data sets that contain both fake data produced by Generator and real data samples. The Generator's main goal is to create fake data that looks like real data in order to trick the Discriminator into thinking it's real..

The Discriminator's goal is to improve its ability to distinguish between real and fake data. Each agent will travel in a clockwise direction. We hope that by duelling these agents, particularly the Generator, they will become stronger. That is, the Generator will sample a distribution that has been studied and is supposed to be the same distribution as the real data, simulating the real data. It will develop a neural network capable of generating it. The Discriminator, on the other hand, will use a supervised technique to train its neural network to detect fake and real data. Each network will train its network in a different order each time.

There are 3 major steps in the training of a GAN:

1. Using the generator to create fake inputs based on random noise or in our case, random normal noise.
2. Training the discriminator with both real and fake inputs (either simultaneously by concatenating real and fake inputs, or one after the other, the latter being preferred).
3. Train the whole model: the model is built with the discriminator combined with the generator.

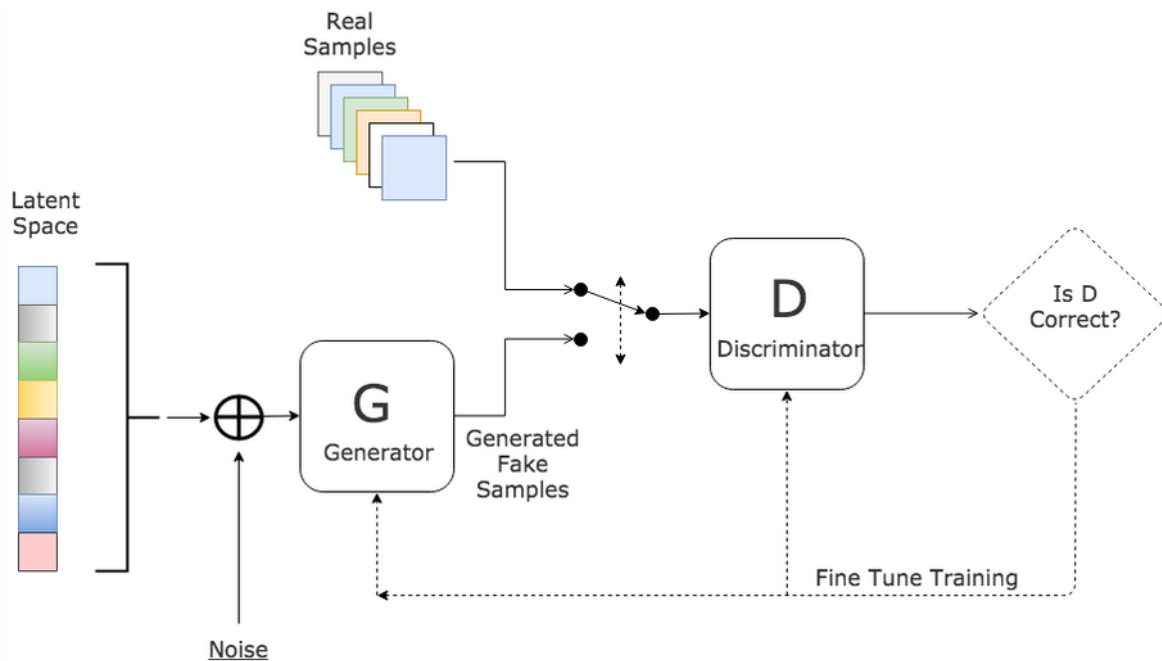


Figure 3.1: How GANs work

### 3.2 Working of DCGANs

- Replace all max pooling with convolutional strides
- Use transposed convolution for upsampling.
- Eliminate fully connected layers.
- Use batch normalization except for the output layer for the generator and the input layer of the discriminator.
- Use ReLU in the generator, except for the output, which uses tanh.
- Use LeakyReLU in the discriminator.

Layers in generator network:

Layer Number	Layer name	Configuration
1	Input Layer	Input shape=(batch size, 100), output shape= (batch size, 100)
2	Dense Layer	neurons=2048, input shape=(batch size, 100), output_shape=(batch size, 2048), activation='Relu'
3	Dense Layer	neurons=16384, input shape=(batch size, 100), output_shape=(batch size, 2048), batch normalization=Yes, activation='Relu'
4	Reshape Layer	Inputshape=(batch_size=16384), output_shape= (batch_size, 8, 8, 256)
5	Up sampling Layer	size=(2, 2), input shape=(batch_size, 8, 8, 256), output_shape=(batch size, 16, 16, 256)
6	2D convolution Layer	filters=128, kernel size=(5, 5), strides=(1, 1), padding='same', input_shape=(batch_size, 16, 16, 256), output_shape=(batch_size, 16, 16, 128), activation='relu'
7	Upsampling Layer	size=(2, 2), input_shape=(batch_size, 16, 16, 128), output_shape=(batch_size, 32, 32, 128)

8	2D convolution Layer	filters=64, kernel_size=(5, 5), strides=(1, 1), padding='same', activation=ReLU, input_shape= (batch_size, 32, 32, 128), output_shape=(batch_size, 32, 32, 64), activation='relu'
9	Upsampling Layer	size=(2, 2), input_shape=(batch_size, 32, 32, 64), output_shape=(batch_size, 64, 64, 64)
10	2D convolution Layer	filters=3, kernel_size=(5, 5), strides=(1, 1), padding='same', activation=ReLU, input_shape= (batch_size, 64, 64, 64), output_shape=(batch_size, 64, 64, 3), activation='tanh'



Layers in discriminator network:

Layer Number	Layer Name	Configuration
1	Input Layer	input_shape=(batch_size, 64, 64, 3), output_shape=(batch_size, 64, 64, 3)
2	2D convolutional Layer	filters=128, kernel_size=(5, 5), strides=(1, 1), padding='valid', input_shape=(batch_size, 64, 64, 3), output_shape=(batch_size, 64, 64, 128), activation='leakyrelu', leaky_relu_alpha=0.2
3	MaxPooling2D	pool_size=(2, 2), input_shape=(batch_size, 64, 64, 128), output_shape=(batch_size, 32, 32, 128)
4	2D convolutional Layer	filters=256, kernel_size=(3, 3), strides=(1, 1), padding='valid', input_shape=(batch_size, 32, 32, 128), output_shape=(batch_size, 30, 30, 256), activation='leakyrelu', leaky_relu_alpha=0.2
5	MaxPooling2D	pool_size=(2, 2), input_shape=(batch_size, 30, 30, 256), output_shape=(batch_size, 15, 15, 256)

6	2D convolutional Layer	filters=512, kernel_size=(3, 3), strides=(1, 1), padding='valid', input_shape=(batch_size, 15, 15, 256), output_shape=(batch_size, 13, 13, 512), activation='leakyrelu', leaky_relu_alpha=0.2
7	MaxPooling2D	pool_size=(2, 2), input_shape=(batch_size, 13, 13, 512), output_shape=(batch_size, 6, 6, 512)
8	Flatten Layer	input_shape=(batch_size, 6, 6, 512), output_shape=(batch_size, 18432)
9	Dense Layer	neurons=1, input_shape=(batch_size, 1024), output_shape=(batch_size, 1), activation='sigmoid'
10	Dense Layer	neurons=1, input_shape=(batch_size, 1024), output_shape=(batch_size, 1), activation='sigmoid'

### 3.3 Implementing DCGANs using Keras

Keras is a meta-framework that uses TensorFlow or Teano as a backend. It provides high-level APIs for working with neural networks. It also has pre-built neural network layers, optimizers, regularizers, initializers, and data-preprocessing layers for easy prototyping compared to low-level frameworks, such as TensorFlow.

For Generator Network: the generator network consists of some 2D convolutional layers, upsampling layers, a reshape layer, and a batch normalization layer. In Keras, every operation can be specified as a layer. Even activation functions are

layers in Keras and can be added to a model just like a normal dense layer.

For discriminator Network: All convolutional layers have LeakyReLU as the activation function with an alpha value of 0.2 the convolutional layers have 128, 256, and 512 filters, respectively. Their kernel sizes are (5, 5), (3, 3), and (3, 3), respectively. After the convolutional layers, we have a flatten layer, which flattens the input to a one-dimensional tensor. Following this, the network has two dense layers with 1,024 neurons and one neuron, respectively. The first dense layer has LeakyReLU as the activation function, while the second layer has sigmoid as the activation function. Sigmoid activation is used for binary classification. We are training the discriminator network to classify between real or fake images.

Steps to create generator network involves:

1. creating a Sequential Keras model: `gen_model = sequential()`

2. add a dense layer that has 2,048 nodes, followed by an activation layer, tanh

```
gen_model.add(Dense(units=2048))
```

```
gen_model.add(Activation('tanh'))
```

3. add the second layer, which is also a dense layer that has 16,384 neurons. This is followed by a batch normalization layer with default hyperparameters and tanh as the activation function:

```
gen_model.add(Dense(256*8*8))
```

```
gen_model.add(BatchNormalization())
```

```
gen_model.add(Activation('tanh'))
```

4. Next, add a reshape layer to the network to reshape the tensor from the last layer to a tensor of a shape of (batch\_size, 8, 8, 256)

```
gen_model.add(Reshape((8, 8, 256),
```

```
input_shape=(256*8*8,)))
```

5. Next, add a 2D upsampling layer to alter the shape from (8, 8, 256) to (16, 16, 256). The upsampling size is (2, 2), which increases the size of the tensor to double its original size. Here, we have 256 tensors of a dimension of 16 x 16

```
gen_model.add(UpSampling2D(size=(2, 2)))
```

6. Next, add a 2D convolutional layer. This applies 2D convolutions on the tensor using a specified number of filters. Here, we are using 64 filters and a kernel of a shape of (5, 5)

```
gen_model.add(Conv2D(128, (5, 5),  
padding='same')) gen_model.add(Activation('tanh'))
```

7. Next, add a 2D upsampling layer to change the shape of the tensor from (batch\_size, 16, 16, 64) to (batch\_size, 32, 32, 64)
8. Next, add a second 2D convolutional layer with 64 filters and a kernel size of (5, 5) followed by tanh as the activation function.
9. Next, add a 2D upsampling layer to change the shape from (batch\_size, 32, 32, 64) to (batch\_size, 64, 64, 64)
10. Finally, add the third 2D convolutional layer with three filters and a kernel size of (5, 5) followed by tanh as the activation function.

Steps to create discriminator network involves:

1. Start by creating a Sequential Keras mode : *dis\_model = Sequential()*
2. Add a 2D convolutional layer that takes an input image of a shape of (64, 64, 3). The hyperparameters for this layer are the following. Also, add LeakyReLU with an alpha value of 0.2 as the activation function  
Filters: 128  
Kernel Size: (5, 5)  
Padding: Same
3. Next, add a 2D max pooling layer with a pool size of (2, 2). Max pooling is used to downsample an image representation and it is applied by using a max-filter over non-overlapping sub-regions of the representation.

4. Next, add another 2D convolutional layer with the following configurations:  
Filters: 256  
Kernel size: (3, 3)  
Activation function: LeakyReLU with alpha 0.2  
Pool size in 2D max pooling: (2, 2)
5. Next, add the third 2D convolutional layer with the following configurations:  
Filters: 512  
Kernel size: (3, 3)  
Activation function: LeakyReLU with alpha 0.2  
Pool size in 2D Max Pooling: (2, 2)
6. Next, add a flatten layer. This flattens the input without affecting the batch size. It produces a two-dimensional tensor
7. Next, add a dense layer with 1024 neurons and LeakyReLU with alpha 0.2 as the activation function.
8. Finally, add a dense layer with one neuron for binary classification. The sigmoid function is the best for binary classification, as it gives the probability of the classes.  
Finally, add a dense layer with one neuron for binary classification. The sigmoid function is the best for binary classification, as it gives the probability of the classes.

### 3.4 Training the DCGANs

Training a DCGAN is similar to training a Vanilla GAN network. It is a four-step process:

1. Load the dataset.
2. Build and compile the networks.
3. Train the discriminator network.
4. Train the generator network

Initially, both of the networks are naive and have random weights. The standard process to train a DCGAN network is to first train the discriminator on the batch of samples. To do this, we need fake samples as well as real samples. We already have the real samples, so we now need to generate the fake samples. To generate fake samples, create a latent vector of a shape of (100,) over a uniform distribution. Feed this latent vector to the untrained generator network. The generator network will generate fake samples that we use to train our discriminator network. Concatenate the real images and the fake images to create a new set of sample images. We also need to create an array of labels: label 1 for real images and label 0 for fake images

To train the generator network, we have to train the adversarial model. When we train the adversarial model, it trains the generator network only but freezes the discriminator network.

## CHAPTER 4

### (PERFORMANCE ANALYSIS)

Synthetic data is a very powerful tool that can overcome many barriers in data science. High-quality synthetic data can substitute for real data to alleviate privacy concerns. The quality of synthetic data is determined by whether the synthetic data correctly captures the correlations between different columns. We define the learning task so that the quality of synthetic data generated can be measured quantitatively.

#### 4.1 Synthetic data generation

Learning task definition:

The synthetic data generation task is to train  $G$  i.e. a data synthesizer, which takes a table as input and generates an engineered form of this input. We require the input table to contain independent rows and only continuous and discrete columns. A table  $T$  contains  $N_c$  continuous columns  $\{C_1, C_2, \dots, C_{N_c}\}$ , and  $N_d$  discrete columns  $\{D_1, D_2, \dots, D_{N_d}\}$ . Each and every segment/column is thought of as an arbitrary variable. These arbitrary variables trail an unidentified joint distribution  $P(C_1:N_c, D_1:N_d)$ . One row  $r_j = \{c_{1,j}, \dots, c_{N_c,j}, d_{1,j}, \dots, d_{N_d,j}\}$  is one sample from the joint distribution.  $T$  is then apportioned into a test set  $T_{test}$  and training set  $T_{train}$ . After training  $G$  on  $T_{train}$ ,  $T_{syn}$  is generated by autonomously sampling rows from data synthesizer  $G$ .

Evaluation metrics: Direct evaluation of  $T_{syn}$  either tests whether  $T_{train}$ ,  $T_{syn}$  are sampled from the same distribution or calculates the distance between two underlying distributions. Existing methods for this test make some strong assumptions on the underlying distribution. For example, Z-test takes into consideration, the data trails a Gaussian distribution. By using these strong assumptions, the methods don't apply to tabular data with complicated distributions. As direct evaluation is intractable, we, therefore, use two alternative methods described below:

Sample likelihood: In this method, the distribution of  $T_{train}$ , denoted as  $P_{train}()$ , is known and the distribution of  $T_{syn}$ , denoted as  $P_{syn}()$ , can be approximated. The likelihood of  $T_{test}$  on  $P_{syn}()$ , and the likelihood of  $T_{syn}$  on  $P_{train}()$  can reveal the distance between two distributions.

Machine learning efficacy: In general cases, finding the underlying distributions is hard. Alternatively, the quality of Tsyn can be evaluated by machine learning applications such as the classification or regression. For instance, an individual can train a regressor and/or classifier or envisage one column using other columns as features. We can then measure efficacy by evaluating whether a classifier or regressor learned from Tsyn can achieve equivalent or higher performance on Ttest as a model learned on Ttrain would.

## 4.2 Existing techniques for generation of synthetic data

The possibility of generating fully synthetic data appeals to different research communities, including statistics, database management, and machine learning. PrivBayes uses traditional Bayesian networks but adds a differentially private learning algorithm. GANs are better because of their performance and the flexibility they exhibit in representing data.

### 4.2.1 PrivBayes

PrivBayes create the superior grade, differentially private engineered data utilizing Bayesian networks.

Motivation: Bayesian networks represent a joint distribution of discrete variables. To generate differentially private synthetic data, we use three steps as shown below:

Learn a Bayesian network.

Inject Laplace noise to each parameter in the network.

Sample from the noisy network.

Usually, this process leads to low-quality synthetic data due to a large amount of noise that is injected in step 2. To acquire a certain privacy level, each network structure needs a different amount of noise.

Preprocessing: Bayesian networks cannot model continuous variables. In PrivBayes all continuous variables are discretized into 16 equal-sized bins in such a way that the modeling algorithm only deals with columns having discrete values.



Model details: There is a compromise between the Bayesian network's original quality and the quality decrease after adding noise. For example, in a table with  $N_d$  discrete columns, and  $(N_d-1)$ -way Bayesian network can fit the distribution, but some weights in the network will have low value and high sensitivity. This means that noise can play a huge role in a noisy model. PrivBayes uses a greedy algorithm to find a graph that maximizes the mutual information.

Datasets and evaluation metrics: PrivBayes is rigorously evaluated on four real datasets. Machine learning efficacy is evaluated, so is the distance of marginal distribution.

#### 4.2.2. MedGAN

As health records are valued for the purpose of research, however, they are heavily guarded due to privacy concerns, healthcare is a domain that needs synthetic data technology more than other fields primarily because of the significant expense of data retrieval. One solution to that is MedGAN which uses a GAN framework to generate completely synthetic health records.

##### Motivation

In health records, each column usually is dependent upon others, which makes the learning of the GAN model very hard. Direct demonstration can't be dependent upon as it creates incorrect results. To rectify that, in MedGAN usually then, an autoencoder is deployed so that we can project raw data in a lower-dimensional representation.

##### Preprocessing

MedGAN supports a table where all columns are binary in addition to continuous columns. A binary column is either 0 or 1 whereas a continuous variable is normalized between 0-1 using min-max normalization:

$$c_{i,j} = \frac{\max(C_i) - \min(C_i)}{\max(C_i) - \min(C_i)}$$

## Datasets and evaluation metrics

Experiments are performed on three different healthcare datasets. The machine learning efficacy is calculated.

**Preprocessing:** All continuous columns are normalized. Discrete columns are also converted to a floating-point number. Each category included in the discrete column is initially represented by a unique integer in  $0, 1, \dots, D_i - 1$  and then divided by  $D_i - 1$ . Since DCGAN is created for images, the input is a matrix instead of a vector, and a row in the table is converted to a square matrix. On the off chance that the quantity of columns is anything but a square number, zeros are cushioned to the row to build the number of columns to the following square number.

**Model Details:** In MedGAN, the generator and discriminator work on different domains. The generator's role is to create a hidden representation. The discriminator checks for raw data. In this manner, the yield of the generator first goes through a decoder before sent down the pipeline to a discriminator. During training initially, the autoencoder is trained. It remains fixed when we are training the GAN. The loss function for it varies depending on the variable type of the column. The loss function for it differs relying upon the variable kind of the column.

### Loss Function

Continuous variables

Mean squared error

Binary variables

Cross entropy loss

Figure 4.1: The MedGAN architecture comprises encoder, decoder, generator, and discriminator. The decoder and the encoder are to be trained on real data and then fixed in later steps. During the process of training, the output of the generator is gone through the decoder prior to feeding into the discriminator. The discriminator determines whether the information/data is genuine or counterfeit.

### TableGAN

Motivation TableGAN is fundamentally used to generate synthetic information/data pointed toward settling privacy comprehensions.

### Model Details

It uses Convolutional networks for both the discriminator as well as the generator, it is trained as a basic GAN. When tabular data contains a label column, a prediction loss is added to the generator to especially improve the correlation between the label column and the other columns

### Datasets and evaluation metrics

This model is evaluated on 4 datasets. The evaluation metric includes machine learning efficacy.

### 4.3 Challenges of Modelling Synthetic Data Using GAN

Countless properties which are unique to tabular data make designing a GAN-based model very difficult. In this section, first we highlight these challenges as they relate to single-table non-time series data, which we try to address in our model.

#### Challenges on tabular data generation

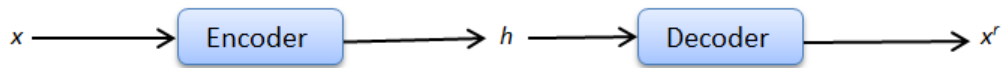
Modeling and synthetically making single-table non-time series data is the simplest difficulty in synthetic data. Each row in the table is sampled independently from the distribution of all possible rows. One could argue that if a row of data is symbolized as a vector, specifically using min-max normalization on continuous values and one-hot depiction for discrete values, then GAN models intended for images could easily be adapted to tabular data. However, here we list numerous special properties of single-table non-time-series data that can break this naive adaptation.

(1. Mixed data types: Real-life tabular data consists of diverse data types (continuous, ordinal, categorical, etc.). Each column has a complex relationship with other columns. For tabular data, alterations to GANs must apply both softmax and tanh on the output to at the same time generate a mix of discrete and continuous columns. In the meantime, the modeling technique should be able to model the probability density of mixed discrete-continuous distribution.

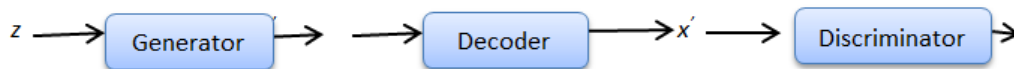
(2. Multimodal distributions: We observe that multiple continuous columns in our real-world datasets have numerous modes. It showed that a traditional GAN can't model all modes on a two-dimensional dataset, thus it also wouldn't be able to model the multimodal distribution which is present in continuous valued columns.

This is a recognized issue of GAN. GANs make their real/fake conclusion on only one example, so if the generator figures out one accurate example and tries to duplicate that example every time, the discriminator does not have enough data to figure out the issue.

The encoder and the decoder are trained on real data



The output of the generator passes through the decoder before being fed into the discriminator



Real data directly fed into the discriminator

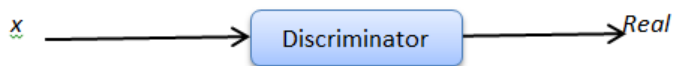


Figure 4-1: Shows that a traditional GAN cannot model a simple 2-dimensional Gaussian mixture.

(A) shows the probability density of 25 Gaussian distributions

(B) shows the distribution learned by GAN

(C) shows the original distribution

(D) shows distribution learned by GAN

(3. Learning from encoded vectors: To allow learning from categorical columns, which has been changed into a one-hot vector. While producing synthetic data samples, a generative model is trained to produce the probability distribution across all the categories using softmax function.

This is the issue in GANs because an insignificant discriminator can simply differentiate real and fake data by checking the distribution's scarceness instead of considering the general realness of a row.

(4. Highly imbalanced categorical columns: In practical datasets, most categorical columns have a highly imbalanced distribution. In our datasets, we discover that half of the categorical columns are highly imbalanced – the major category appears in more than 80% of the all data entries, causing mode collapse.

Not including a small category only results in minute changes to the distribution of data, but imbalanced data leads to inadequate training chances for these smaller classes. The Discriminator network cannot detect such problems unless mode-collapse-preventing mechanisms are implemented. These methods can help prevent GANs from producing only the most noticeable classes. Synthetic data for the smaller categories are anticipated to be of lower quality, demanding to resample.

(5. High dimensionality: The high dimensionality of tabular data escalates the complexity exponentially. For example,  $n$  binary variables have  $2^n$  likelihoods. Precisely representing the probability distribution using a small neural network is impossible because there are not enough parameters, and there is typically not sufficient training data. In this case, any modeling technique introduces bias to the estimate. For example, when modeling with a GAN, bias could come in while selecting a specific network structure or learning objective. Compared to statistical models, the bias introduced in neural network models is difficult to analyze.

(6. Lack of training data: Learning with small training data is a perplexing problem. Similar difficulties have been registered as few-shot learning or meta-learning. Such tasks are easier with images because the content in different images shares similar filters. Yet, tabular data is radically different. It is perplexing to efficiently transfer knowledge learned from one table to another.

(7. Missing values: Tabular data has missing values. To directly train a GAN model on tabular data with missing values, one should modify the data representation to appropriately differentiate missing values from identified values, and guise the model to make it robust towards missing values. A substitute approach is to assign the missing values before training the model. However, the data assertion also necessitates modeling of the table. Mistakes in data imputation would be propagated to learned GAN models.

<b>Problems</b>	<b>MedGAN</b>	<b>TableGAN</b>	<b>Mod-GAN</b>
<i>Mixed data types</i>	<b>Yes*</b>	<b>Yes*</b>	<b>Yes</b>
<i>Multimodal Distribution</i>	<b>No</b>	<b>Yes</b>	<b>Yes</b>
<i>Learning from encoded vectors</i>	<b>No</b>	<b>No</b>	<b>Yes</b>
<i>Imbalanced Categorical columns</i>	<b>No</b>	<b>Yes</b>	<b>Yes</b>
<i>High Dimensionality</i>	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>
<i>Lack of Training Data</i>	<b>No</b>	<b>No</b>	<b>No</b>
<i>Missing Values</i>	<b>No</b>	<b>No</b>	<b>No</b>

```

import numpy as np
import os
import matplotlib.pyplot as plt
import cv2

import warnings
warnings.filterwarnings('ignore')

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

```

Figure 4.2: Loading the libraries

```

def list_images(basePath, contains=None):
    return list_files(basePath, validExts=(".jpg", ".jpeg", ".png", ".bmp"), contains=contains)

def list_files(basePath, validExts=(".jpg", ".jpeg", ".png", ".bmp"), contains=None):
    for (rootDir, dirNames, filenames) in os.walk(basePath):
        for filename in filenames:
            if contains is not None and filename.find(contains) == -1:
                continue

            ext = filename[filename.rfind("."):].lower()

            if ext.endswith(validExts):
                imagePath = os.path.join(rootDir, filename).replace(" ", "\\ ")
                yield imagePath

def load_images(directory='', size=(64,64)):
    images = []
    labels = []
    label = 0

    imagePaths = list(list_images(directory))

    for path in imagePaths:
        if not('OSX' in path):
            path = path.replace('\\', '/')

            image = cv2.imread(path)
            image = cv2.resize(image, size)

```

Figure 4.3: reading images from the dataset

32

```

images=load_images('../input/data')

```

```

_,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')

```

Figure 4.4 : Displaying images

34





Figure 4.5 : Output of the image dataset

```

def build_generator(self):
    epsilon = 0.00001
    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)))

```

Figure 4.6 : Building the generator

```

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)

```

Figure 4.7 : Building of Discriminator

```
mkdir animeGenerated
```

```
gan=GAN()  
gan.train(epochs=15001, batch_size=256, metrics_update=200, save_images=1000, save_model=15000)
```

Figure 4.8 : Folder containing trained images

## CHAPTER 5

### (RESULT)

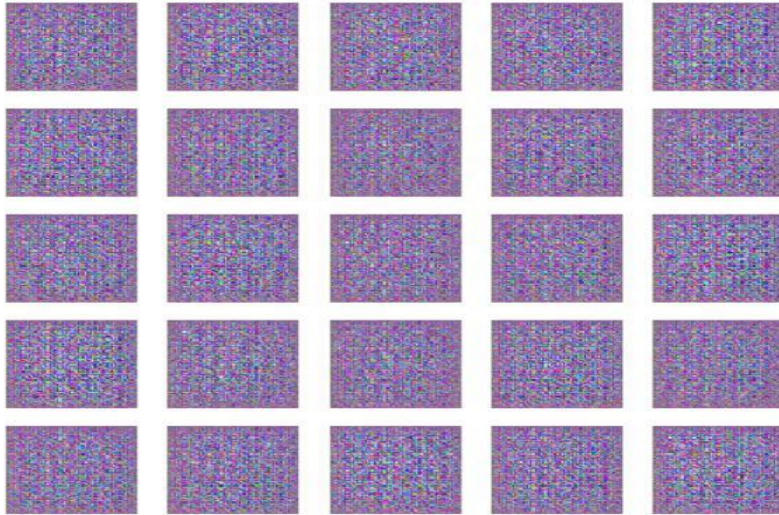


Figure 5.1 : Output at 100 epochs



Figure 5.2 : Output at 200 epochs



Figure 5.3 : Output at 500 epochs



```
11200 [Discriminator loss: 0.196949, acc.: 92.13%] [Genera
```

Figure 5.4: Output at 1000 epochs

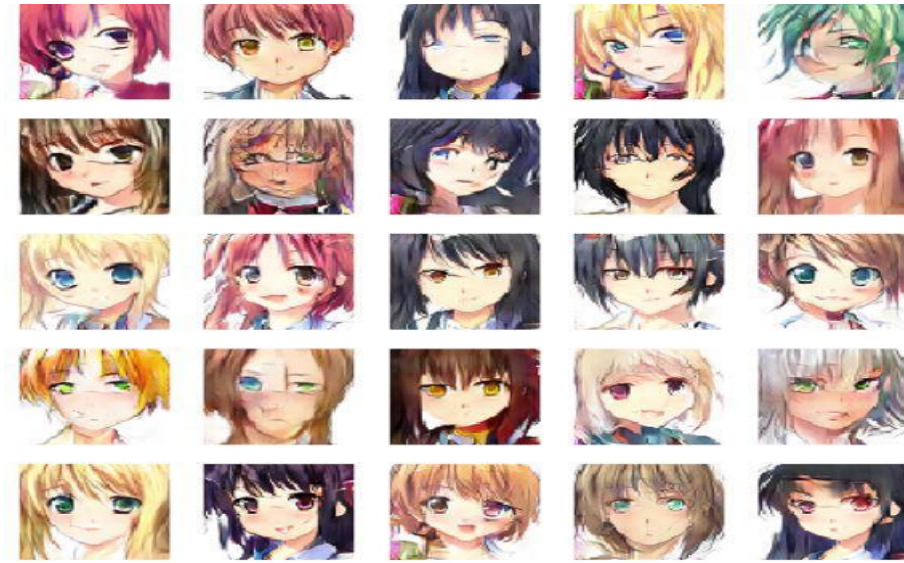


Figure 5.5: Output at 2000 epochs

Python notebook using data from [Anime Faces](#) · 8 views · 4d ago · [Edit tags](#)

Input  
217.24 MB

- Data Sources
  - Anime Faces
    - data
      - 1.png
      - 10.png
      - 100.png
      - 1000.png
      - 10000.png
      - 10001.png
      - 10002.png
      - 10003.png
      - 10004.png
      - 10005.png
      - 10006.png
      - 10007.png
      - 10008.png
      - 10009.png
      - 1001.png
      - 10010.png
      - 10011.png
      - 10012.png
      - 10013.png

data (21.6k files)

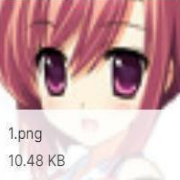
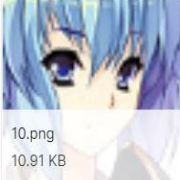
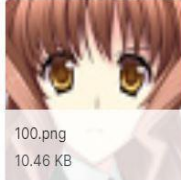
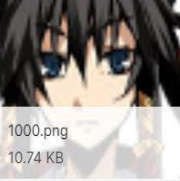
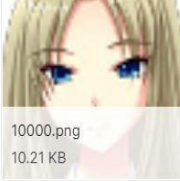
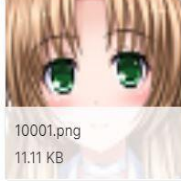
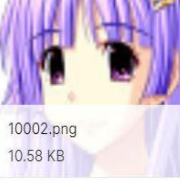
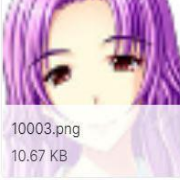
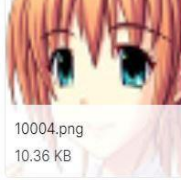
 1.png 10.48 KB	 10.png 10.91 KB	 100.png 10.46 KB
 1000.png 10.74 KB	 10000.png 10.21 KB	 10001.png 11.11 KB
 10002.png 10.58 KB	 10003.png 10.67 KB	 10004.png 10.36 KB

Figure 5.6: Final Output

## Bibliography

- Generative Adversarial Networks - 3 Course Specialization on [Coursera](#)
- Generative Adversarial Networks with Python- Deep Learning Generative Models for Image synthesis and Image Translation - [Jason Brownlee](#)
- Generative Adversarial Networks Projects: Build Next-generation Generative Models Using TensorFlow and Keras - [Kailash Ahirwar](#)
- Adult,census,coverttype and intrusion dataset -[UCI machine learning repository](#)
- Credit risk modelling dataset -[Kaggle](#)
- MNIST dataset -[Kaggle](#)
- Synthesizing Tabular Data using Conditional GAN by Lei Xu B.E., Tsinghua University (2017)
- Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In Advances in neural information processing systems, pages 153–160, 2007.
- P. Berka and M. Sochorova. 1999 czech financial dataset - real anonymized transactions, 2019.
- Mihaela van der Schaar Jinsung Yoon, James Jordon. Gain: Missing data imputation using generative adversarial nets. arXiv, 2018.
- Joan Serrà Santiago Pascual, Antonio Bonafonte. SEGAN: Speech enhancement generative adversarial network. arXiv, 2017.
- Samuel A. Barnett. Convergence problems with generative adversarial networks (GANs), 2018.
- Maayan Frid-Adar, Eyal Klang, Michal Amitai, Jacob Goldberger, and Hayit Greenspan. Synthetic data augmentation using gan for improved liver lesion classification,2017
- F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. Acm transactions on interactive intelligent systems , 2015.

- Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks.



# Gernating\_Anime\_Character\_1. docx

*by*

---

**Submission date:** 24-Jun-2021 10:06PM (UTC+0530)

**Submission ID:** 1611634293

**File name:** Gernating\_Anime\_Character\_1.docx (1.59M)

**Word count:** 6583

**Character count:** 43826

# Gernating\_Anime\_Character\_1.docx

## ORIGINALITY REPORT

**28** %  
SIMILARITY INDEX

**27** %  
INTERNET SOURCES

**8** %  
PUBLICATIONS

**17** %  
STUDENT PAPERS

## PRIMARY SOURCES

<b>1</b>	<b>hub.packtpub.com</b> Internet Source	<b>6</b> %
<b>2</b>	<b>Submitted to University of Oklahoma</b> Student Paper	<b>4</b> %
<b>3</b>	<b>towardsdatascience.com</b> Internet Source	<b>3</b> %
<b>4</b>	<b>medium.com</b> Internet Source	<b>3</b> %
<b>5</b>	<b>dspace.mit.edu</b> Internet Source	<b>2</b> %
<b>6</b>	<b>neurohive.io</b> Internet Source	<b>1</b> %
<b>7</b>	<b>www.ir.juit.ac.in:8080</b> Internet Source	<b>1</b> %
<b>8</b>	<b>Submitted to Jaypee University of Information Technology</b> Student Paper	<b>1</b> %
<b>9</b>	<b>www.ijert.org</b> Internet Source	<b>1</b> %

10	Submitted to Nepal College of Information Technology Student Paper	1 %
11	developers.google.com Internet Source	1 %
12	Submitted to Yonsei University Student Paper	1 %
13	Submitted to Coventry University Student Paper	1 %
14	jonathan-hui.medium.com Internet Source	1 %
15	www.willberger.org Internet Source	<1 %
16	www.slideshare.net Internet Source	<1 %
17	Submitted to City University of Hong Kong Student Paper	<1 %
18	Submitted to Fachhochschule Salzburg GmbH Student Paper	<1 %
19	Submitted to Liverpool John Moores University Student Paper	<1 %
20	Submitted to University of Glasgow Student Paper	<1 %

21	Submitted to Republic of the Maldives Student Paper	<1 %
22	dai.lids.mit.edu Internet Source	<1 %
23	scholar.uwindsor.ca Internet Source	<1 %
24	www.packtpub.com Internet Source	<1 %
25	Submitted to B.V. B College of Engineering and Technology, Hubli Student Paper	<1 %
26	machinelearningmastery.com Internet Source	<1 %
27	P. Kuhry. "The palaeoecology of a treed bog in western boreal Canada: a study based on microfossils, macrofossils and physico- chemical properties", Review of Palaeobotany and Palynology, 1997 Publication	<1 %
28	Submitted to Erasmus University of Rotterdam Student Paper	<1 %

Exclude bibliography  On

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: ...16.06.2021.....

Type of Document (Tick):  PhD Thesis  M.Tech Dissertation/ Report  B.Tech Project Report  Paper

Name: Nikhil Nagpal Department: Computer science

Enrolment No 171224 Contact No. 9636610300\_\_

E-mail. Nikhil.nagpal98@gmail.com

Name of the Supervisor: Dr. vivek seghal

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): **Gernating Anime Character Using Gan's**

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/ revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

- Total No. of Pages =48
- Total No. of Preliminary pages =6
- Total No. of pages accommodate bibliography/references =2

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at 28 ..... (%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor)

Signature of HOD

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	Excluded	Similarity Index (%)	Generated Plagiarism Report Details (Title, Abstract & Chapters)	
	<ul style="list-style-type: none"> <li>• All Preliminary Pages</li> <li>• Bibliography/Images/Quotes</li> <li>• 14 Words String</li> </ul>	28	Word Counts	6583
<b>Report Generated on</b>			Character Counts	43826
		<b>Submission ID</b>	Total Pages Scanned	
			File Size	

Checked by  
Name & Signature

Librarian

**Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at [plagcheck,juit@gmail.com](mailto:plagcheck,juit@gmail.com)**