

# **Project Name: Generation of Anime Faces**

## **Subject Name:** **Deep Learning and Neural Networks Subject Code:** **15CSE380**



**Project By:**

Bachu Krishna Lokesh : BL.EN.U4CSE18011

B Saketh Chandra : BL.EN.U4CSE18015

Dara Ekanth : BL.EN.U4CSE18024

I Hemanth : BL.EN.U4CSE18044

# **Project guide:** **Ms Jyotsna C**

**Abstract**

**Introduction to Generative Modeling**

Deep neural networks are used mainly for supervised learning: classification or regression. Generative Adversarial Networks or GANs, however, use neural networks for a very different purpose: Generative modelling

Generative modelling is an unsupervised learning task in machine learning that involves automatically discovering and learning the regularities or patterns in input data in such a way that the model can be used to generate or output new examples that plausibly could have been drawn from the original dataset.-

(https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/)

While there are many approaches used for generative modelling, a Generative Adversarial Network takes the following approach:

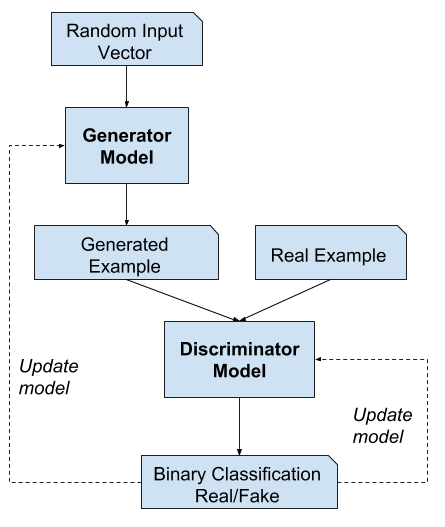


Fig-1: GAN model

There are two neural networks: a Generator and a Discriminator. The generator generates a "fake" sample given a random vector/matrix, and the discriminator attempts to detect whether a given sample is "real" (picked from the training data) or "fake" (generated by the generator). Training happens in tandem: we train the discriminator for a few epochs, then train the generator for a few epochs, and repeat. This way both the generator and the discriminator get better at doing their jobs.

GANs, however, can be notoriously difficult to train and are extremely sensitive to hyperparameters, activation functions and regularization. In this project, we trained a GAN to generate images of anime characters' faces.

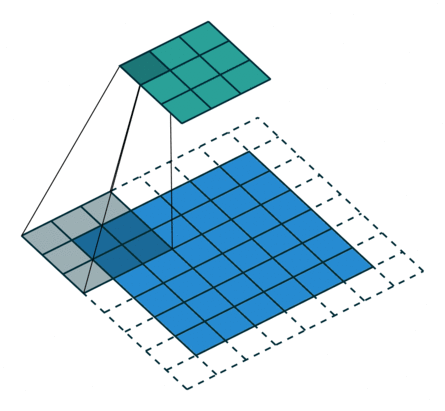
We used this [Anime Face Dataset](https://github.com/Mckinsey666/Anime-Face-Dataset), which consists of over 63,000 cropped anime faces. Note that generative modelling is an unsupervised learning task, so the images do not have any labels.



Fig-2: Fake Images

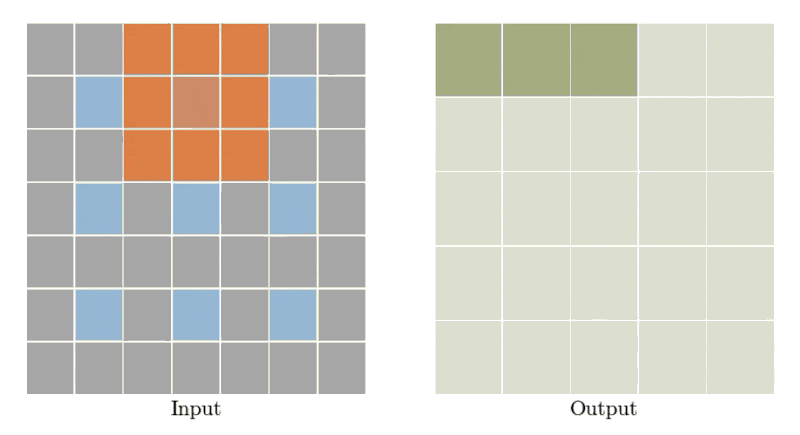
**Discriminator Network:**

The discriminator takes an image as input, and tries to classify it as "real" or "generated". In this sense, it's like any other neural network. We'll use a convolutional neural network (CNN) which outputs a single number output for every image. We'll use a stride of 2 to progressively reduce the size of the output feature map.



**Generator Network:**

The input to the generator is typically a vector or a matrix of random numbers (referred to as a latent tensor) which is used as a seed for generating an image. The generator will convert a latent tensor of shape (128, 1, 1) into an image tensor of shape (3x28x28). To achieve this we will use the ConvTranspose2d layer from PyTorch, which is performed as a transposed convolution (also referred to as a deconvolution).



**Tools**

* TensorFlow
* PyTorch

**Data Set**

* We are using [Anime Face Dataset](https://github.com/Mckinsey666/Anime-Face-Dataset)
* It contains more than 63,000 cropped images (Training and Testing and Validation)

## **Python Libraries used**

1. open datasets
2. Os
3. DataLoader
4. ImageFolder
5. Torch
6. Matplotlib
7. Make\_grid
8. CV2

### **Description of modules used**

###### ***show\_images(images, nmax)***

###### This module will print the nmax number of images.

###### ***2) od.download(dataset\_url)***

This module is used to download the dataset from the URL that is being passed as an argument.

###### ***3) os.listdir():***

This method will return a list containing the names of the entries in the directory given by path.

###### ***4) Resize():***

This module will resize the input image size to the size that we passed as an argument.

***5) normalize():***

Normalization is used to change the range of pixel intensity values. It is used to bring the image into a range that is more familiar and helps in faster processing.

###### ***6) centercrop():***

###### Remove the outer parts of an image but retain the central region of the image along each dimension.

###### ***7) ToTensor():***

###### This function converts Python objects of various types to Tensor objects.

###### ***8) Torch.device():***

###### With this module, we can assign the hardware on which we can run our code.

###### ***9) torch.cuda.is\_available():***

###### This module will check if there is any GPU hardware available to run the Cuda architecture.

###### ***10) zero\_grad():***

###### This module will clear the previous gradients.

###### ***11) save\_image():***

###### This module helps in saving an image passed as an argument in the path specified.

###### ***12) torch.cuda.empty\_cache():***

###### This module will empty the cache stored in the GPU if any.

### **Problems encountered and how it was solved**

1. **Problem**: Initially we did not include zero grads and it leads to the accumulation of gradients in subsequent back passes

**Solution**: We included the zero\_grad function, then the parameters update correctly.

1. **Problem**: Real Score was initially less

**Solution**: By increasing the number of epochs real score got increased

## 

## **Result**

### **Contribution of each team member**

Lokesh: Finding the Dataset, data preprocessing.

Saketh Chandra: Optimizing parameters, fitting the model.

Ekanth: Discriminator training and construction.

Hemanth: Generator training and construction.

### **Scope of future work**

* For making Game Characters
* Generation Fake fingerprint
* Generation fake RETINA image for RETINA SCANS FOR SECURITY PURPOSES
* Clone and making a modified version of Cars, Phone design

##### **Whether you will extend the project**

Yes, we are planning to implement a real-time analysis model.