SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

COURSE CODE: DJ19DSC501 **DATE:**

COURSE NAME: Machine Learning - II **CLASS: AY 2023-24**

Jhanvi Parekh

60009210033

D11

LAB EXPERIMENT NO. 09

AIM :

Build Generative adversarial model for fake news/image/video prediction.

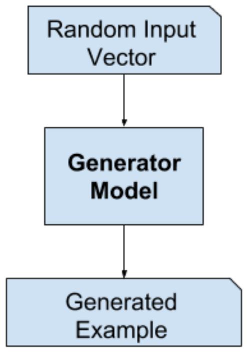
THEORY:

GAN: Generative adversarial network

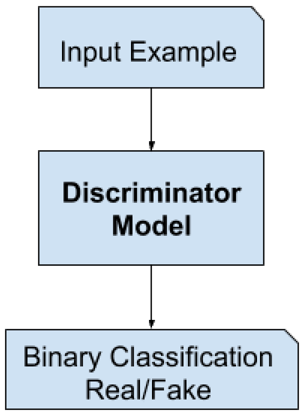
GAN is an approach to generative modeling using deep learning methods, such as convolutional neural networks. Generative modeling is an unsupervised learning task in machine learning that involves automatically discovering and learning the regularities or patterns in input data in such a way that the model can be used to generate or output new examples that plausibly could have been drawn from the original dataset. GANs are used to produce synthetic data.

GANs algorithmic architectures use two neural networks -

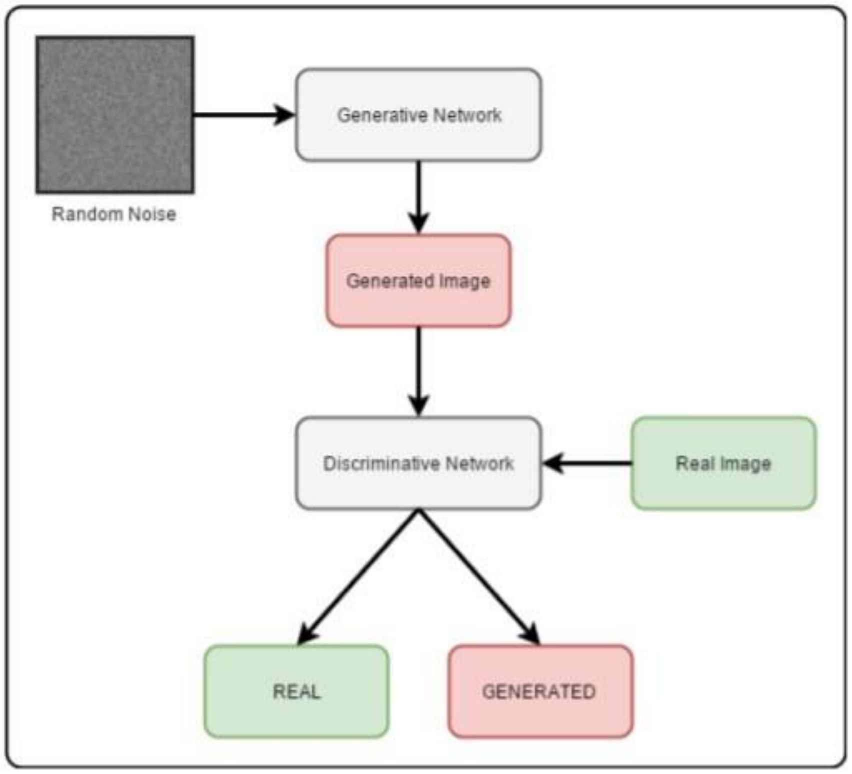
1. Generator - Model that is used to generate new plausible examples from the problem domain - The Generator Model generates new images by taking a fixed size random noise as an input. Generated images are then fed to the Discriminator Model. The main goal of the Generator is to fool the Discriminator by generating images that look like real images and thus makes it harder for the Discriminator to classify images as real or fake.



2. Discriminator - Model that is used to classify examples as real (from the domain) or fake (generated) - Used to classify images as real or fake. Discriminator Model takes an image as an input (generated and real) and classifies it as real or fake. Generated images come from the Generator and the real images come from the training data. The discriminator model is the simple binary classification model.



Generator and a Discriminator “compete” against one another to create the desired result. If both are functioning at high levels, the result is images that are seemingly identical real-life photos.



Hyperparameters to tune -

1. Layers - Explore additional hierarchical learning capacity by adding more layers and varied numbers of neurons in each layer
2. Number of inputs in dense layer - Dense layers improve overall accuracy and 5–10 units or nodes per layer is a good base

3. Dropout - Slow down learning with regularization methods like dropout on the recurrent LSTM connections. A good starting point is 20% but the dropout value should be kept small (up to 50%). The 20% value is widely accepted as the best compromise between preventing model overfitting and retaining model accuracy.
4. Learning Rate - This hyperparameter defines how quickly the network updates its parameters.
5. Number of epochs

Tasks to be performed:

1. **Input Fashion-MNIST dataset**
2. **Use GAN to generate and classify fake images.**
3. **Perform GAN hyperparameter tuning to improve accuracy score.**
4. **Plot generated and original input images.**

Link :
https://colab.research.google.com/drive/1fUbRt1-uerV2uE8MkfQhuKAfA5kB2uvn?usp=sharing

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, LeakyReLU, BatchNormalization, Reshape, Flatten, Input
from tensorflow.keras.optimizers import Adam


(x_train, _), (_, _) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.0


latent_dim = 100


# Define generator
generator = Sequential([
    Dense(128, input_dim=latent_dim),
    LeakyReLU(alpha=0.2),
    BatchNormalization(momentum=0.8),
    Dense(256),
    LeakyReLU(alpha=0.2),
    BatchNormalization(momentum=0.8),
    Dense(512),
    LeakyReLU(alpha=0.2),
    BatchNormalization(momentum=0.8),
    Dense(784, activation='sigmoid'),  # Output layer with 28*28=784 neurons
    Reshape((28, 28, 1))  # Reshape to the size of the images in Fashion-MNIST
])


# Define discriminator
discriminator = Sequential([
    Flatten(input_shape=(28, 28, 1)),  # Flatten the input image
    Dense(512),
    LeakyReLU(alpha=0.2),
    Dense(256),
    LeakyReLU(alpha=0.2),
    Dense(1, activation='sigmoid')  # Output layer with 1 neuron for binary classification
])


# Define GAN model
discriminator.trainable = False  # This freezes the discriminator during GAN training
gan_input = Input(shape=(latent_dim,))
fake_image = generator(gan_input)
gan_output = discriminator(fake_image)
gan = Model(gan_input, gan_output)


discriminator.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.0002, beta_1=0.5), metrics=['accuracy'])


gan.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.0002, beta_1=0.5))
```

```
epochs = 5000
batch_size = 64

for epoch in range(epochs):
    idx = np.random.randint(0, x_train.shape[0], batch_size)
    real_images = x_train[idx]

    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    generated_images = generator.predict(noise)

    labels_real = np.ones((batch_size, 1))
    labels_fake = np.zeros((batch_size, 1))

    d_loss_real = discriminator.train_on_batch(real_images, labels_real)
    d_loss_fake = discriminator.train_on_batch(generated_images, labels_fake)
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    labels_gan = np.ones((batch_size, 1))

    g_loss = gan.train_on_batch(noise, labels_gan)

    if epoch % 100 == 0:
        print(f"Epoch {epoch}/{epochs} [D loss: {d_loss[0]} | D accuracy: {100 * d_loss[1]}] [G loss: {g_loss}]")
```

```
     2/2 [==============================] - 0s 7ms/step
     2/2 [==============================] - 0s 9ms/step
```

```
num_samples = 10
random_latent_vectors = np.random.normal(size=(num_samples, latent_dim))
generated_images = generator.predict(random_latent_vectors)

fig, axes = plt.subplots(1, num_samples, figsize=(num_samples, 1))
for i in range(num_samples):
    axes[i].imshow(generated_images[i].reshape(28, 28), cmap='gray')
    axes[i].axis('off')

plt.show()
```

```
     1/1 [==============================] - 0s 127ms/step
```