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**

LAB EXPERIMENT NO. 09

60009210105

Amitesh Sawarkar

D 12

AIM :

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

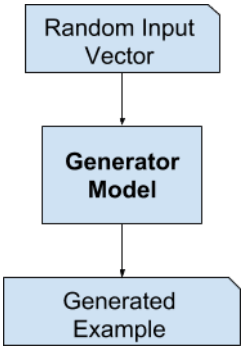
THEORY:

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.

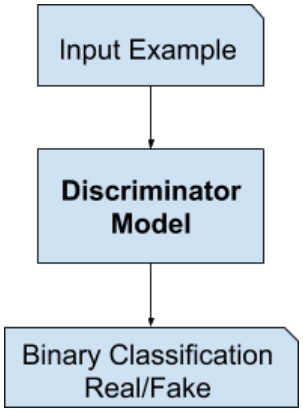
GANs algorithmic architectures use two neural networks -

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.

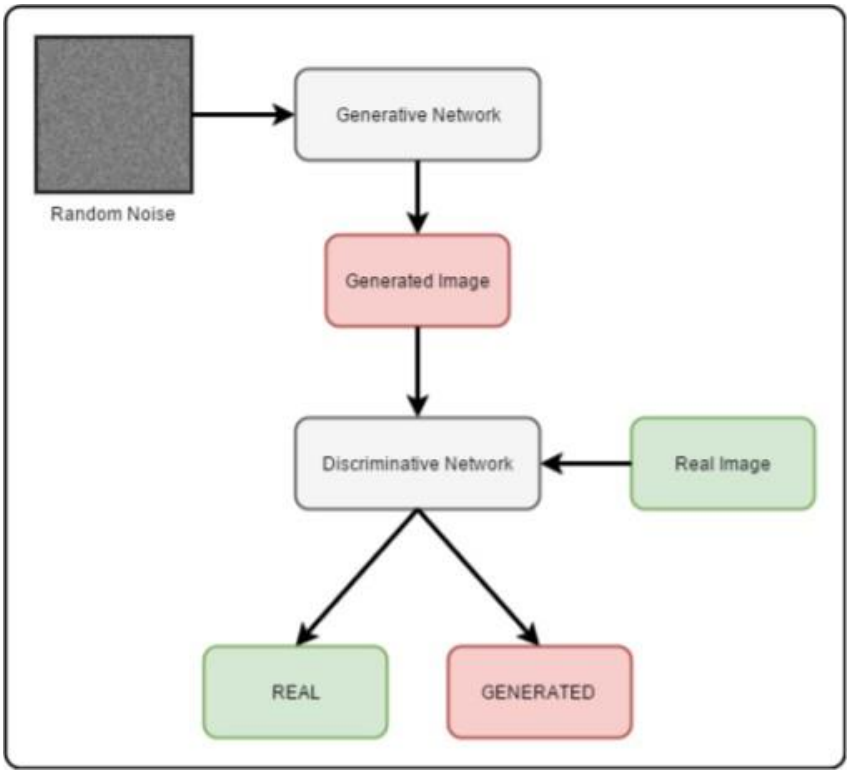


2. Discriminator - Model that is used to classify examples as real (from the domain) or fake (generated) - Used to classify images as real or fake. Discriminator Model takes an image as an input (generated and real) and classifies it as real or fake. Generated images come from the Generator and the real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

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)



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.**

Import necessary libraries

```
from numpy import zeros, ones, expand_dims, asarray
from numpy.random import randn, randint
from keras.datasets import fashion_mnist
from keras.optimizers import Adam
from keras.models import Model, load_model
from keras.layers import Input, Dense, Reshape, Flatten
from keras.layers import Conv2D, Conv2DTranspose, Concatenate
from keras.layers import LeakyReLU, Dropout, Embedding
from keras.layers import BatchNormalization, Activation
from keras import initializers
from keras.initializers import RandomNormal
```

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)

```
from keras.optimizers import Adam, RMSprop, SGD

from matplotlib import pyplot

import numpy as np

from math import sqrt
```

```
(X_train, _), (_, _) = fashion_mnist.load_data()

X_train = X_train.astype(np.float32) / 127.5 - 1

X_train = np.expand_dims(X_train, axis=3)

print(X_train.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 11s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0s/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 2s 0us/step
(60000, 28, 28, 1)
```

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)

Generator

```
def define_generator(latent_dim):
    init  = RandomNormal(stddev=0.02)
    in_lat = Input(shape=(latent_dim,))
    gen = Dense(256, kernel_initializer=init)(in_lat)
    gen = LeakyReLU(alpha=0.2)(gen)
    gen = Dense(512, kernel_initializer=init)(gen)
    gen = LeakyReLU(alpha=0.2)(gen)
    gen = Dense(1024, kernel_initializer=init)(gen)
    gen = LeakyReLU(alpha=0.2)(gen)
    gen = Dense(28 * 28 * 1, kernel_initializer=init)(gen)
    out_layer = Activation('tanh')(gen)
    out_layer = Reshape((28, 28, 1))(gen)
    model = Model(in_lat, out_layer)
    return model
generator = define_generator(100)
```

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)

Discriminator

```
def define_discriminator(in_shape=(28, 28, 1)):
    #init = RandomNormal(stddev=0.02)
    in_image = Input(shape=in_shape)
    fe = Flatten()(in_image)
    fe = Dense(1024)(fe)
    fe = LeakyReLU(alpha=0.2)(fe)
    fe = Dropout(0.3)(fe)
    fe = Dense(512)(fe)
    fe = LeakyReLU(alpha=0.2)(fe)
    fe = Dropout(0.3)(fe)
    fe = Dense(256)(fe)
    fe = LeakyReLU(alpha=0.2)(fe)
    fe = Dropout(0.3)(fe)
    out = Dense(1, activation='sigmoid')(fe)
    model = Model(in_image, out)
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
    return model

discriminator = define_discriminator()
```

```
C:\Users\shah\anaconda3\lib\site-packages\keras\optimizers\optimizer_v2\adam.py:117: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
```

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)

GAN architecture

```
def define_gan(g_model, d_model):
    d_model.trainable = False
    gan_output = d_model(g_model.output)
    model = Model(g_model.input, gan_output)
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
    return model
gan_model = define_gan(generator, discriminator)
```

```
def generate_latent_points(latent_dim, n_samples):
    x_input = randn(latent_dim * n_samples)
    z_input = x_input.reshape(n_samples, latent_dim)
    return z_input
def generate_real_samples(X_train, n_samples):
    ix = randint(0, X_train.shape[0], n_samples)  # returns an integer number selected element from the specified ra
    X = X_train[ix]
    y = ones((n_samples, 1))
    return X, y
```

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)

```
def generate_fake_samples(generator, latent_dim, n_samples):
    z_input = generate_latent_points(latent_dim, n_samples)
    images = generator.predict(z_input)
    y = zeros((n_samples, 1))
    return images, y
def summarize_performance(step, g_model, latent_dim, n_samples=100):
    X, _ = generate_fake_samples(g_model, latent_dim, n_samples)
    X = (X + 1) / 2.0
    for i in range(100):
        pyplot.subplot(10, 10, 1 + i)
        pyplot.axis('off')
        pyplot.imshow(X[i, :, :, 0], cmap='gray_r')
```

```
    filename2 = 'model_%04d.h5' % (step+1)
    g_model.save(filename2)
    print('>Saved: %s' % (filename2))
def save_plot(examples, n_examples):
    for i in range(n_examples):
        pyplot.subplot(sqrt(n_examples), sqrt(n_examples), 1 + i)
        pyplot.axis('off')
        pyplot.imshow(examples[i, :, :, 0], cmap='gray_r')
    pyplot.show()
```

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)

```
def train(g_model, d_model, gan_model, X_train, latent_dim, n_epochs=100, n_batch=1000):
    bat_per_epo = int(X_train.shape[0] / n_batch)
    n_steps = bat_per_epo * n_epochs
    for i in range(n_steps):
        X_real, y_real = generate_real_samples(X_train, n_batch)
        d_loss_r, d_acc_r = d_model.train_on_batch(X_real, y_real)
        X_fake, y_fake = generate_fake_samples(g_model, latent_dim, n_batch)
        d_loss_f, d_acc_f = d_model.train_on_batch(X_fake, y_fake)
        z_input = generate_latent_points(latent_dim, n_batch)
        y_gan = ones((n_batch, 1))
        g_loss, g_acc = gan_model.train_on_batch(z_input, y_gan)
        print('>%d, dr[%.3f,%.3f], df[%.3f,%.3f], g[%.3f,%.3f]' % (i+1, d_loss_r,d_acc_r, d_loss_f,d_acc_f, g_loss,g
        if (i+1) % (bat_per_epo * 1) == 0:
            summarize_performance(i, g_model, latent_dim)
```

+ Code + Text

```
latent_dim = 100
train(generator, discriminator, gan_model, X_train, latent_dim, n_epochs=20, n_batch=64)
model = load_model('model_18740.h5')
```

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)

```
2/2 [==============================] - 9s 17ms/step
>1, dr[1.057,0.234], df[0.696,0.188], g[0.691,0.781]
2/2 [==============================] - 0s 7ms/step
>2, dr[0.171,1.000], df[0.706,0.000], g[0.681,1.000]
2/2 [==============================] - 0s 9ms/step
>3, dr[0.045,1.000], df[0.715,0.000], g[0.673,1.000]
2/2 [==============================] - 0s 9ms/step
>4, dr[0.031,1.000], df[0.729,0.000], g[0.663,1.000]
2/2 [==============================] - 0s 9ms/step
>5, dr[0.022,1.000], df[0.744,0.000], g[0.650,1.000]
2/2 [==============================] - 0s 9ms/step
>6, dr[0.018,1.000], df[0.770,0.000], g[0.629,1.000]
2/2 [==============================] - 0s 10ms/step
>7, dr[0.018,1.000], df[0.797,0.000], g[0.611,1.000]
2/2 [==============================] - 0s 8ms/step
>8, dr[0.016,1.000], df[0.833,0.000], g[0.588,1.000]
2/2 [==============================] - 0s 9ms/step
>9, dr[0.017,1.000], df[0.855,0.000], g[0.580,1.000]
2/2 [==============================] - 0s 7ms/step
>10, dr[0.015,1.000], df[0.885,0.000], g[0.570,1.000]
2/2 [==============================] - 0s 8ms/step
>11, dr[0.021,1.000], df[0.866,0.000], g[0.594,1.000]
2/2 [==============================] - 0s 8ms/step
>12, dr[0.034,1.000], df[0.848,0.000], g[0.649,0.844]
2/2 [==============================] - 0s 9ms/step
>13, dr[0.037,1.000], df[0.779,0.031], g[0.721,0.234]
2/2 [==============================] - 0s 9ms/step
>14, dr[0.056,1.000], df[0.723,0.281], g[0.790,0.016]
2/2 [==============================] - 0s 9ms/step
>15, dr[0.032,1.000], df[0.656,0.797], g[0.857,0.000]
2/2 [==============================] - 0s 11ms/step
>16, dr[0.045,1.000], df[0.649,0.844], g[0.878,0.000]
2/2 [==============================] - 0s 9ms/step
>17, dr[0.040,1.000], df[0.647,0.906], g[0.855,0.000]
2/2 [==============================] - 0s 11ms/step
>18, dr[0.033,1.000], df[0.658,0.812], g[0.851,0.000]
2/2 [==============================] - 0s 9ms/step
>19, dr[0.038,1.000], df[0.675,0.703], g[0.817,0.000]
2/2 [==============================] - 0s 12ms/step
>20, dr[0.038,1.000], df[0.702,0.375], g[0.802,0.000]
2/2 [==============================] - 0s 12ms/step
>21, dr[0.046,1.000], df[0.732,0.203], g[0.776,0.031]
2/2 [==============================] - 0s 9ms/step
```

```python
latent_dim = 100
n_examples = 100
latent_points = generate_latent_points(latent_dim, n_examples)
X  = model.predict(latent_points)
X = (X + 1) / 2.0
save_plot(X, n_examples)
```