Assignment no :4

Name : CHOVATIYA JAY MAHESHBHAI
Student ID : 1161980
Course: COMP5421 Deep Learning
Topic:GAN network for data augmentation and LSTM for sequence problem recognition.

Notes : Here in the second try i have tried to implement using Labelbinarizer to one-hot encode the target variables.

Requirement no: 1

Using GAN network to boost the performance in Assignment 2.

IMPLEMENTATION :

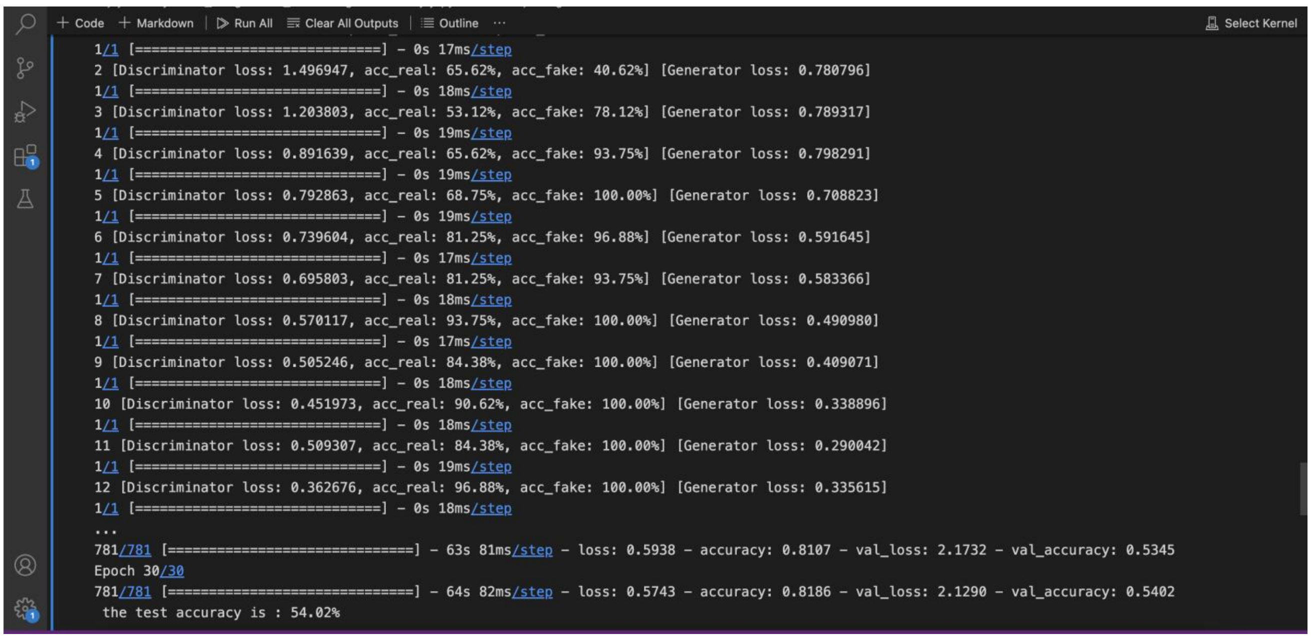
```
#importing the libraries
import numpy as np
import sklearn.metrics as metrics
from keras.applications import densenet
from keras.datasets import cifar100
from keras.utils import np_utils
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
from keras.layers import Input, Dense, Reshape, Flatten, Dropout
from keras.layers import BatchNormalization, Activation, ZeroPadding2D
from keras.layers import LeakyReLU
from keras.layers.convolutional import UpSampling2D, Conv2D
from keras.models import Sequential, Model
from keras import initializers
# create the model from keras and set weights=None for training from scratch
model = densenet.DenseNet121(weights=None, input_shape=(32,32,3),
pooling=None, classes=100)
# printing the model summary
model.summary()
# Splitting training and testing set
(cifarx_train, cifary_train), (cifarx_test, cifary_test) = cifar100.load_data()
```

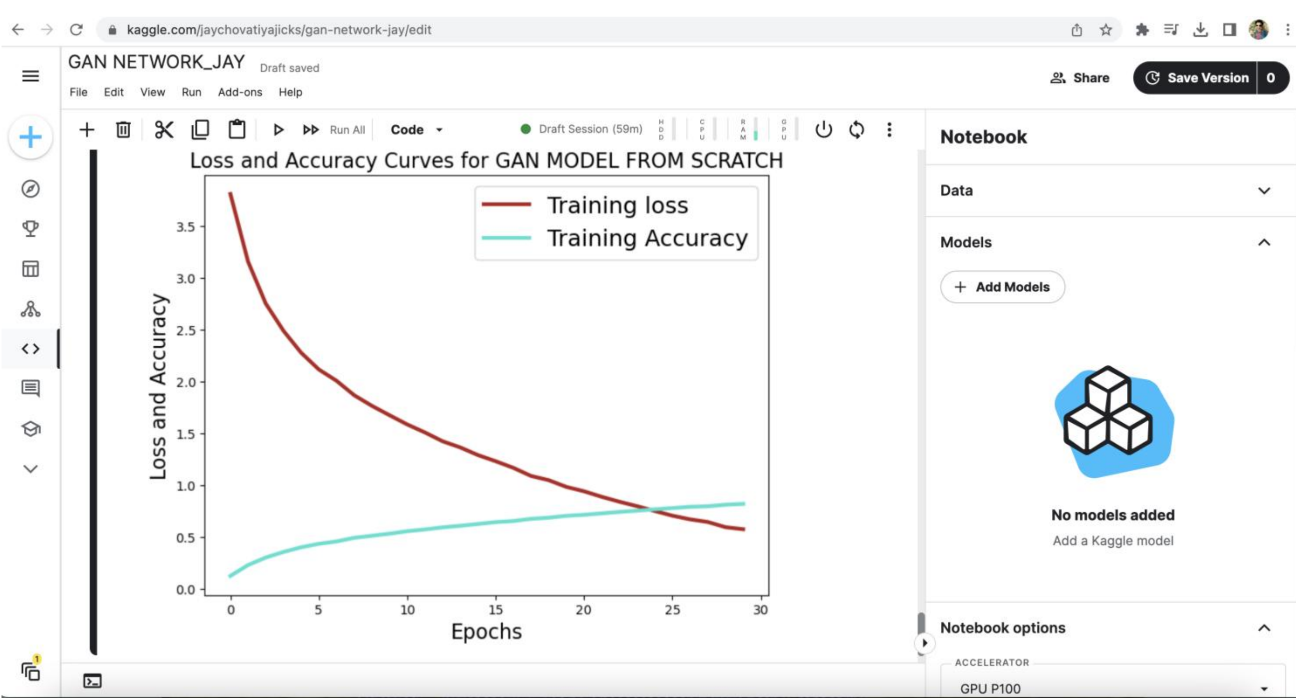
```
# Converting to float
cifarx_train = cifarx_train.astype('float32')
cifarx_test = cifarx_test.astype('float32')
# converting data into normalize form
cifarx_train = densenet.preprocess_input(cifarx_train)
cifarx_test = densenet.preprocess_input(cifarx_test)
# one-hot encoding
cifarY_train = np_utils.to_categorical(cifary_train, 100)
cifarY_test = np_utils.to_categorical(cifary_test, 100)
# Defining the generator model for GAN
def Gan_build_generator():
    model = Sequential()
    model.add(Dense(128 * 8 * 8, activation="relu", input_dim=100))
    model.add(Reshape((8, 8, 128)))
    model.add(BatchNormalization(momentum=0.8))
    model.add(UpSampling2D())
    model.add(Conv2D(128, kernel_size=3, padding="same"))
    model.add(Activation("relu"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(UpSampling2D())
    model.add(Conv2D(64, kernel_size=3, padding="same"))
    model.add(Activation("relu"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Conv2D(3, kernel_size=3, padding="same"))
    model.add(Activation("tanh"))
    noise = Input(shape=(100,))
    img = model(noise)
    return Model(noise, img)
# Defining the discriminator model for GAN
def Gan_build_discriminator():
    model = Sequential()
    model.add(Conv2D(32, kernel_size=3, strides=2, input_shape=(32, 32, 3),
padding="same"))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(0.25))
    model.add(Conv2D(64, kernel_size=3, strides=2, padding="same"))
    model.add(ZeroPadding2D(padding=((0,1),(0,1))))
    model.add(BatchNormalization(momentum=0.8))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(0.25))
    model.add(Conv2D(128, kernel_size=3, strides=2, padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(LeakyReLU(alpha=0.2))
```

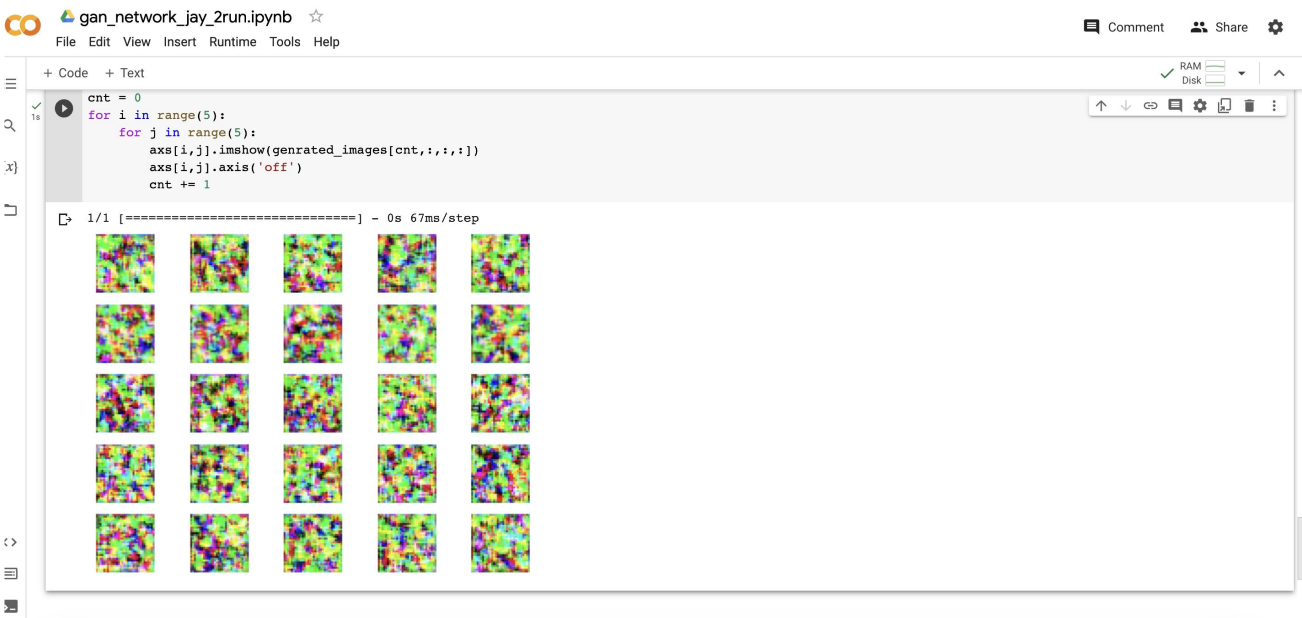
```
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))
    img = Input(shape=(32, 32, 3))
    validity = model(img)
    return Model(img, validity)
# Compiling the discriminator model
discriminator = Gan_build_discriminator()
discriminator.compile(loss='binary_crossentropy',
                      optimizer=Adam(0.0002, 0.5),
                      metrics=['accuracy'])
# Compiling the generator model
generator = Gan_build_generator()
# The generator takes noise as input and generates images
z = Input(shape=(100,))
img = generator(z)
# Training the generator
discriminator.trainable = False
# Discriminator taking generated images as input and checking validity
valid = discriminator(img)
combined = Model(z, valid)
combined.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5))
# declaring hyperparameters
epochs =50
batch_size = 32
save_interval = 1000
# Training the GAN model
for epoch in range(epochs):
    # Training discriminator with real images
    idx = np.random.randint(0, cifarx_train.shape[0], batch_size)
    real_imgs = cifarx_train[idx]
    real_labels = np.ones((batch_size, 1))
    discriminator_loss_real = discriminator.train_on_batch(real_imgs, real_labels)
# Training discriminator with fake images
    noise = np.random.normal(0, 1, (batch_size, 100))
    fake_imgs = generator.predict(noise)
    fake_labels = np.zeros((batch_size, 1))
    discriminator_loss_fake = discriminator.train_on_batch(fake_imgs, fake_labels)
# Training generator
    noise = np.random.normal(0, 1, (batch_size, 100))
    valid_y = np.array([1] * batch_size)
    generator_loss = combined.train_on_batch(noise, valid_y)
# describing the progress
```

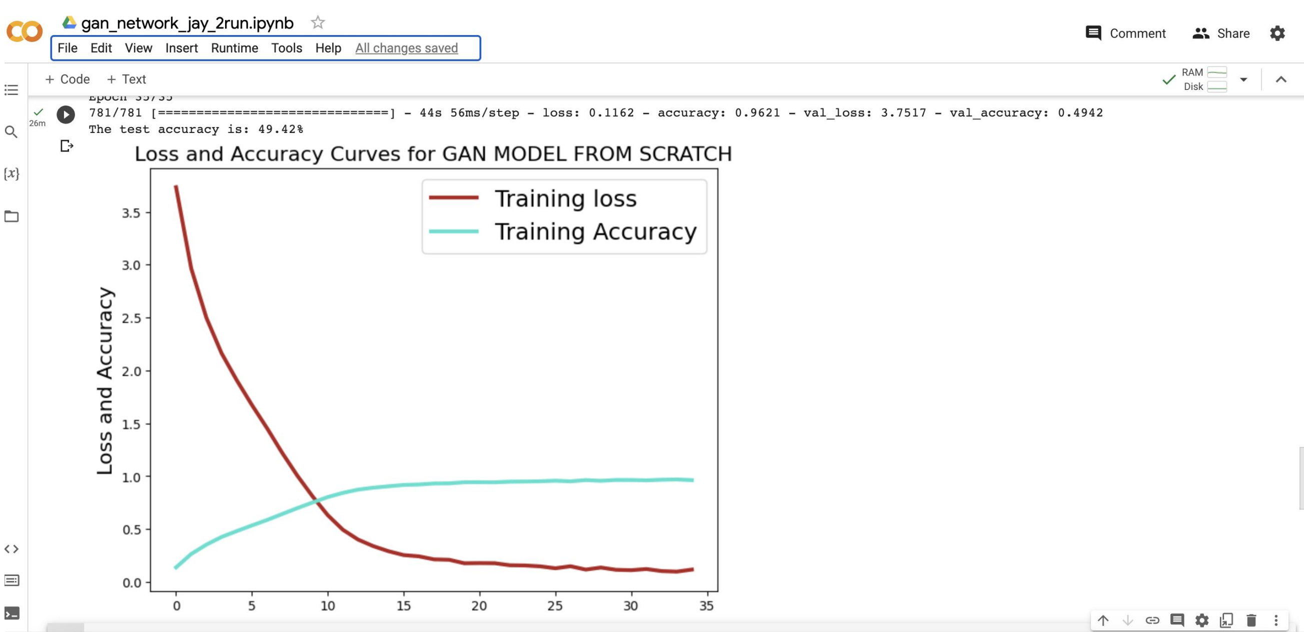
```
    print ("%d [Discriminator loss: %f, acc_real: %.2f%%, acc_fake: %.2f%%]
[Generator loss: %f]" % (epoch,
discriminator_loss_real[0]+discriminator_loss_fake[0], 100*discriminator_loss_real[1],
100*discriminator_loss_fake[1], generator_loss))
    if epoch % save_interval == 0:
        # show images from the generator
        noise = np.random.normal(0, 1, (25, 100))
        gen_imgs = generator.predict(noise)
        gen_imgs = 0.5 * gen_imgs + 0.5
        # Plot the generated images
        fig, axs = plt.subplots(5, 5)
        counter = 0
        for i in range(5):
            for j in range(5):
                axs[i,j].imshow(gen_imgs[counter, :,:,:])
                axs[i,j].axis('off')
                counter += 1
        plt.show()
# Using Adam and set learning rate 0.001
optimizer = Adam(lr=0.001)
model.compile(loss='categorical_crossentropy', optimizer=optimizer,
metrics=["accuracy"])
history = model.fit(datagen_train.flow(cifarx_train, cifarY_train, batch_size=64,
shuffle=True),
                    steps_per_epoch=len(cifarx_train)/64, epochs=30,
validation_data=(cifarx_test, cifarY_test))
# Evaluate the model
scores = model.evaluate(cifarx_test, cifarY_test, verbose=0)
print(" the test accuracy is : %.2f%%" % (scores[1]*100))
# Define plotchart function
def plotchart(history, value):
    plt.figure(figsize=[8,6])
    plt.plot(history.history['loss'], 'firebrick', linewidth=3.0)
    plt.plot(history.history['accuracy'], 'turquoise', linewidth=3.0)
    plt.legend(['Training loss', 'Training Accuracy'], fontsize=18)
    plt.xlabel('Epochs', fontsize=16)
    plt.ylabel('Loss and Accuracy', fontsize=16)
    plt.title('Loss and Accuracy Curves for {}'.format(value), fontsize=16)
    plt.show()
# Plot the training history
plotchart(history, 'GAN MODEL FROM SCRATCH')
```

<u>Outputs:1</u> Here i have attached the screenshot of GAN Model on CIFAR100 datasets output whichyou can see below.









Requirement no: 2

Using LSTM for time serious problem recognition.

IMPLEMENTATION :

```
import pandas as pd
import tensorflow as tf
from math import sqrt
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
```

```
from keras.layers import Dense, LSTM
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import matplotlib.pyplot as plt
# Function to create LSTM model
def Lstmcreate_model(input_shape):
   model = Sequential()
   model.add(LSTM(50, input_shape=input_shape))
   model.add(Dense(1))
   model.compile(loss='mse', optimizer='adam')
   return model
# Function to train LSTM model
def Lstmtrain_model(model, train_X, train_y, epochs):
   history = model.fit(train_X, train_y, epochs=epochs, batch_size=72, verbose=1,
shuffle=False)
   return model
# Function to test LSTM model
def Lstmtest_model(model, test_X, test_y):
   y_pred = model.predict(test_X)
   rmse = sqrt(mean_squared_error(test_y, y_pred))
   return rmse, y_pred
# Function to create time series datasets
def create_dataset(data, time_steps=1):
   X, Y = [], []
   for i in range(len(data)-time_steps):
      X.append(data[i:i+time_steps])
      Y.append(data[i+time_steps])
   return np.array(X), np.array(Y)
# Function to load and preprocess data
def load_data(file_path):
   # Load time series dataset
   df = pd.read_csv(file_path, header=0, parse_dates=[0], index_col=0,
squeeze=True)
   # Split dataset into train and test sets
   train_size = int(len(df) * 0.8)
   train, test = df[0:train_size], df[train_size:len(df)]
   # Normalize data
   scaler = MinMaxScaler()
   train = scaler.fit_transform(train.values.reshape(-1,1))
   test = scaler.transform(test.values.reshape(-1,1))
   # Create time series datasets
   x_train, y_train = create_dataset(train, time_steps=1)
   x_test, y_test = create_dataset(test, time_steps=1)
```

```
    return x_train, y_train, x_test, y_test, scaler
# Load and preprocess data
file_path = '/content/soil_data.csv'
x_train, y_train, x_test, y_test, scaler = load_data(file_path)
# Creating LSTM model
input_shape = (x_train.shape[1], x_train.shape[2])
model = Lstmcreate_model(input_shape)
model = Lstmtrain_model(model, x_train, y_train, epochs=200)
rmse, y_pred = Lstmtest_model(model, x_test, y_test)
print("Testing loss for time series model on soil data is:", rmse)
# Load and preprocess new data
new_data = pd.read_csv('/content/soil_data.csv')
new_data = scaler.transform(new_data.values.reshape(-1, 1))
new_data = np.reshape(new_data, (1, new_data.shape[0], 1))
# Make predictions using the trained model
prediction = model.predict(new_data)
# Inverse transform the predicted value to get the original scale
prediction = scaler.inverse_transform(prediction)
print("The predicted Soil moisture value is:", prediction)
# Plot actual vs predicted values
plt.plot(y_pred, label='Predicted')
plt.plot(y_test, label='Actual')
plt.xlabel('Time')
plt.ylabel('Soil Moisture')
plt.legend()
plt.show()
```

<u>Outputs:2</u> Here i have attached the screenshot of LSTM Model output along with the testing loss and prediction value which you can see below.