EX.NO : 1
DATE : 20-08-2022

Write a python program to build a simple neural network with Keras

AIM:

To build a simple neural network with Keras using Python.

PROCEDURE:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

CODE:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
from sklearn.metrics import confusion_matrix

dataset = pd.read_csv('/content/data.csv')
X = dataset.iloc[:,2:32]
y = dataset.iloc[:,1]

labelencoder_Y = LabelEncoder()
y = labelencoder_Y.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
classifier = Sequential()
classifier.add(Dense(units = 16, kernel_initializer = 'uniform', activation = 'relu', input_dim = 30))
classifier.add(Dense(units = 8, kernel_initializer = 'uniform', activation = 'relu'))
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))
classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
classifier.compile(optimizer = 'rmsprop', loss = 'binary_crossentropy', metrics = ['accuracy'])

classifier.fit(X_train, y_train, batch_size = 128, epochs = 25, verbose = 2)
y_pred = classifier.predict(X_test)
y_pred = [ 1 if y>=0.5 else 0 for y in y_pred ]
y_pred = classifier.predict(X_test)
y_pred = [ 1 if y>=0.5 else 0 for y in y_pred ]

cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy = (cm[0][0]+cm[1][1])/(cm[0][0]+cm[0][1]+cm[1][0]+cm[1][1])
print("Accuracy: "+ str(accuracy*100)+"%")
```

OUTPUT:

```
[[64  3]
 [ 3 44]]
Accuracy: 94.73684210526315%
```

RESULT:

Thus, a simple neural network with Keras is built and compiled successfully using python.

EX.NO : 2
DATE : 23-09-2022

Write a python program to build a Convolutional Neural Network with Keras

AIM:

To build a convolutional neural network with Keras using Python.

PROCEDURE:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a convolutional neural network model using Keras.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

CODE:

```
import numpy as np
import pandas as pd
import cv2
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPool2D, BatchNormalization
from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense
from tensorflow.keras.losses import categorical_crossentropy
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.models import load_model

df = pd.read_csv('/content/data.csv')
df = df.loc[(df['category'] != 'OTHER')].reset_index(drop=True)

def image_processing(image_url):
```

```
  response = urllib.request.urlopen(image_url)
  image = np.asarray(bytearray(response.read()), dtype="uint8")
  image_bgr = cv2.imdecode(image, cv2.IMREAD_COLOR)
  image_hsv = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2HSV)
  image_gray = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2GRAY)
  mask = cv2.inRange(image_hsv, (0,255,255), (0,255,255))
  if len(np.where(mask != 0)[0]) != 0:
    y1 = min(np.where(mask != 0)[0])
    y2 = max(np.where(mask != 0)[0])
  else:
    y1 = 0
    y2 = len(mask)
  if len(np.where(mask != 0)[1]) != 0:
    x1 = min(np.where(mask != 0)[1])
    x2 = max(np.where(mask != 0)[1])
  else:
    x1 = 0
    x2 = len(mask[0])
  image_cropped = image_gray[y1:y2, x1:x2]
  image_100x100 = cv2.resize(image_cropped, (100, 100))
  image_arr = image_100x100.flatten()
  return image_arr

image_list = []
for url in df['image_url'] :
  image_list.append(image_processing(url))
X = np.array(image_list)
X = X/255
X = np.save('/kaggle/working/X.npy', X)

import gdown
url = 'https://drive.google.com/uc?id=1B6_rtcmGRy49hqpwoJT-_Ujnt6cYj5Ba'
output = 'X.npy'
gdown.download(url, output, quiet=False)
X = np.load('/kaggle/working/X.npy')

encoder = LabelEncoder()
Targets = encoder.fit_transform(df['category'])
Y = to_categorical(Targets, num_classes = n_classes)
X_test = X[14000:,]
```

```
Y_test = Y[14000:,]
X_train, X_val, Y_train, Y_val = train_test_split(X[:14000,], Y[:14000,], test_size=0.15, random_state=13)

img_rows, img_cols = 100, 100
input_shape = (img_rows, img_cols, 1)
X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols, 1)
X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols, 1)
X_val = X_val.reshape(X_val.shape[0], img_rows, img_cols, 1)

model = Sequential()
model.add(Conv2D(filters = 16, kernel_size = (3, 3), activation='relu', input_shape = input_shape))
model.add(BatchNormalization())
model.add(Conv2D(filters = 16, kernel_size = (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters = 32, kernel_size = (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(filters = 32, kernel_size = (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(n_classes, activation='softmax'))

learning_rate = 0.001
model.compile(loss = categorical_crossentropy,
        optimizer = Adam(learning_rate),
        metrics=['accuracy'])
history = model.fit( X_train, Y_train,
            epochs = 15, batch_size = 100,
```

```
            callbacks=[save_best], verbose=1,
            validation_data = (X_val, Y_val))
Y_pred = np.round(model.predict(X_test))
score = model.evaluate(X_test, Y_test, verbose=0)
print('Accuracy over the test set: \n ', round((score[1]*100), 2), '%')
```

OUTPUT:

```
Accuracy over the test set:
  69.22 %
```

RESULT:

Thus, a convolutional neural network with Keras was built and compiled successfully using python.

EX.NO : 3
DATE : 30-09-2022

Write a python program to create a Neural Network to recognize handwritten digits using MNIST dataset

AIM:

To create a neural network in python to recognize handwritten digits using the MNIST dataset.

PROCEDURE:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a neural network model for recognizing handwritten digits.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

CODE:

```
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense , Activation, Dropout
from keras.optimizers import Adam ,RMSprop
from keras import  backend as K
from keras.utils import to_categorical, plot_model
from keras.datasets import mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
image_size = x_train.shape[1]
input_size = image_size * image_size
x_train = np.reshape(x_train, [-1, input_size])
x_train = x_train.astype('float32') / 255
x_test = np.reshape(x_test, [-1, input_size])
x_test = x_test.astype('float32') / 255
```

```
batch_size = 128
hidden_units = 256
dropout = 0.45
model = Sequential()
model.add(Dense(hidden_units, input_dim=input_size))
model.add(Activation('relu'))
model.add(Dropout(dropout))
model.add(Dense(hidden_units))
model.add(Activation('relu'))
model.add(Dropout(dropout))
model.add(Dense(num_labels))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=20, batch_size=batch_size)
loss, acc = model.evaluate(x_test, y_test, batch_size=batch_size)
print("\nTest accuracy: %.1f%%" % (100.0 * acc))
```

OUTPUT:

```
10000/10000 [==============================] - 0s 22us/step

Test accuracy: 98.2%
```

RESULT:

Thus, a neural network for recognizing handwritten digits with MNIST dataset was built and compiled successfully in python.

EX.NO : 4
DATE : 07-10-2022

Write a python program to Visualize and design CNN with Transfer Learning

AIM:

To write a python program to visualize and design CNN with transfer learning.

PROCEDURE:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Prepare the already pre-trained model.
4. Compile and fit the pre-trained model with the dataset.
5. Perform visualization.
6. Calculate performance metrics.

CODE:

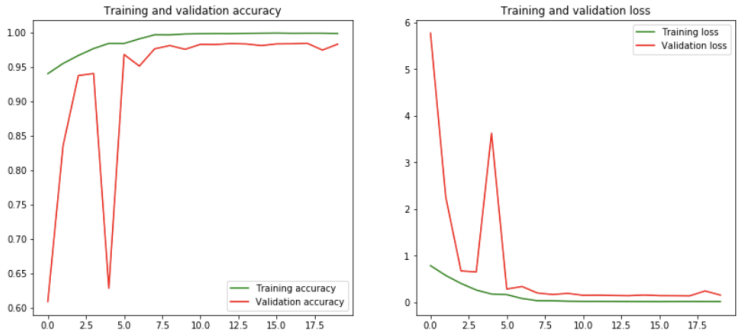
```
import os, cv2, random
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from tqdm import tqdm
from random import shuffle
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
from keras.utils import plot_model
from tensorflow.python.keras.applications import ResNet50
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense, Flatten, GlobalAveragePooling2D
%matplotlib inline

TEST_SIZE = 0.5
RANDOM_STATE = 2018
BATCH_SIZE = 64
```

```
NO_EPOCHS = 20
NUM_CLASSES = 2
SAMPLE_SIZE = 20000
PATH = '/kaggle/input/dogs-vs-cats-redux-kernels-edition/'
TRAIN_FOLDER = './train/'
TEST_FOLDER =  './test/'
IMG_SIZE = 224
RESNET_WEIGHTS_PATH=
'/kaggle/input/resnet50/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5'

train_image_path = os.path.join(PATH, "train.zip")
test_image_path = os.path.join(PATH, "test.zip")
import zipfile
with zipfile.ZipFile(train_image_path,"r") as z:
   z.extractall(".")
with zipfile.ZipFile(test_image_path,"r") as z:
   z.extractall(".")
train_image_list = os.listdir("./train/")[0:SAMPLE_SIZE]
test_image_list = os.listdir("./test/")

def label_pet_image_one_hot_encoder(img):
   pet = img.split('.')[-3]
   if pet == 'cat': return [1,0]
   elif pet == 'dog': return [0,1]

def process_data(data_image_list, DATA_FOLDER, isTrain=True):
   data_df = []
   for img in tqdm(data_image_list):
      path = os.path.join(DATA_FOLDER,img)
      if(isTrain):
         label = label_pet_image_one_hot_encoder(img)
      else:
         label = img.split('.')[0]
      img = cv2.imread(path,cv2.IMREAD_COLOR)
      img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))
      data_df.append([np.array(img),np.array(label)])
   shuffle(data_df)
   return data_df

train = process_data(train_image_list, TRAIN_FOLDER)
```

```
test = process_data(test_image_list, TEST_FOLDER, False)
X = np.array([i[0] for i in train]).reshape(-1,IMG_SIZE,IMG_SIZE,3)
y = np.array([i[1] for i in train])

model = Sequential()
model.add(ResNet50(include_top=False, pooling='max', weights=RESNET_WEIGHTS_PATH))
model.add(Dense(NUM_CLASSES, activation='softmax'))
model.layers[0].trainable = True

model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=TEST_SIZE, random_state=RANDOM_STATE)

train_model = model.fit(X_train, y_train,
              batch_size=BATCH_SIZE,
              epochs=NO_EPOCHS,
              verbose=1,
              validation_data=(X_val, y_val))

def plot_accuracy_and_loss(train_model):
    hist = train_model.history
    acc = hist['acc']
    val_acc = hist['val_acc']
    loss = hist['loss']
    val_loss = hist['val_loss']
    epochs = range(len(acc))
    f, ax = plt.subplots(1,2, figsize=(14,6))
    ax[0].plot(epochs, acc, 'g', label='Training accuracy')
    ax[0].plot(epochs, val_acc, 'r', label='Validation accuracy')
    ax[0].set_title('Training and validation accuracy')
    ax[0].legend()
    ax[1].plot(epochs, loss, 'g', label='Training loss')
    ax[1].plot(epochs, val_loss, 'r', label='Validation loss')
    ax[1].set_title('Training and validation loss')
    ax[1].legend()
    plt.show()
plot_accuracy_and_loss(train_model)

score = model.evaluate(X_val, y_val, verbose=0)
print('Validation loss:', score[0])
```

print('Validation accuracy:', score[1])

OUTPUT:




```
Validation loss: 0.15023201193418587
Validation accuracy: 0.9832
```

RESULT:

Thus, a python program to visualize and design CNN with transfer learning is done successfully.

EX.NO : 5
DATE : 14-10-2022

Write a python program to build an RNN with Keras

AIM:

To write a python program for building a recurrent neural network with Keras.

PROCEDURE:

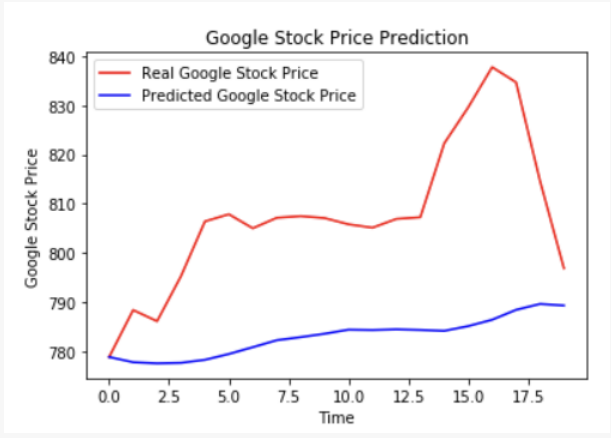
1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a recurrent neural network with Keras.
4. Compile and fit the model.
5. Perform prediction with the test dataset.

CODE:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import SimpleRNN
from keras.layers import Dropout

dataset_train=pd.read_csv('/content/Stock_Price_Train.csv')
train = dataset_train.loc[:, ['Open']].values
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range = (0, 1))
train_scaled = scaler.fit_transform(train)
X_train = []
y_train = []
timesteps = 50

for i in range(timesteps, 1250):
    X_train.append(train_scaled[i - timesteps:i, 0])
    y_train.append(train_scaled[i, 0])
```

```
X_train, y_train = np.array(X_train), np.array(y_train)
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))

regressor = Sequential()
regressor.add(SimpleRNN(units=50,activation='tanh',return_sequences=True,          input_shape=
(X_train.shape[1],1)))
regressor.add(Dropout(0.2))
regressor.add(SimpleRNN(units=50,activation='tanh',return_sequences=True))
regressor.add(Dropout(0.2))
regressor.add(SimpleRNN(units=50,activation='tanh',return_sequences=True))
regressor.add(Dropout(0.2))
regressor.add(SimpleRNN(units = 50))
regressor.add(Dropout(0.2))
regressor.add(Dense(units = 1))
regressor.compile(optimizer='adam', loss='mean_squared_error')
regressor.fit(X_train, y_train, epochs=100, batch_size=32)
dataset_test=pd.read_csv('/content/Stock_Price_Test.csv')
real_stock_price = dataset_test.loc[:, ['Open']].values
dataset_total=pd.concat((dataset_train['Open'],dataset_test['Open']),
axis=0)
inputs=dataset_total[len(dataset_total)-len(dataset_test)- timesteps:].values.reshape(-1,1)
inputs = scaler.transform(inputs)
X_test = []
for i in range(timesteps, 70):
    X_test.append(inputs[i-timesteps:i,0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
predicted_stock_price = regressor.predict(X_test)
predicted_stock_price = scaler.inverse_transform(predicted_stock_price)

plt.plot(real_stock_price, color='red', label='Real Google Stock Price')
plt.plot(predicted_stock_price, color='blue', label='Predicted Google Stock Price')
plt.title('Google Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Google Stock Price')
plt.legend()
plt.show()
```

OUTPUT:



RESULT:

Thus, a python program for building a recurrent neural network with Keras is done successfully.

EX.NO : 6
DATE : 21-10-2022

Write a python program to build autoencoders with Keras

AIM:

To write a python program for building autoencoders with Keras.

PROCEDURE:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build autoencoders with Keras.
4. Compile and fit the model.
5. Perform prediction with the test dataset.

CODE:

```
import numpy as np
import pandas as pd
from keras.layers import Input, Dense
from keras.models import Model
from sklearn.model_selection import train_test_split

encoding_dim = 32
input_img = Input(shape=(784,))
encoded = Dense(encoding_dim, activation='relu')(input_img)
decoded = Dense(784, activation='sigmoid')(encoded)
autoencoder = Model(input_img, decoded)
encoder = Model(input_img, encoded)
encoded_input = Input(shape=(encoding_dim,))
decoder_layer = autoencoder.layers[-1]
decoder = Model(encoded_input, decoder_layer(encoded_input))
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')

data = pd.read_csv("/content/train.csv",header=0)
x_data = data.values[:,1:]
y_data = data.label
x_train,x_test, y_train, y_test = train_test_split(x_data, y_data, test_size = 0.3)
```

```
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

autoencoder.fit(x_train, x_train,epochs=50,batch_size=256,shuffle=True,
                validation_data=(x_test, x_test))

encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)
import matplotlib.pyplot as plt
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

OUTPUT:

RESULT:

Thus, a python program for building autoencoders with Keras is done successfully.

EX.NO : 7
DATE : 28-10-2022

Write a python program to build GAN with Keras

AIM:

To write a python program for building GAN with Keras.

PROCEDURE:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Create a generator, discriminator, and GAN model using Keras.
4. Train the model.
5. Perform prediction with the test dataset.

CODE:

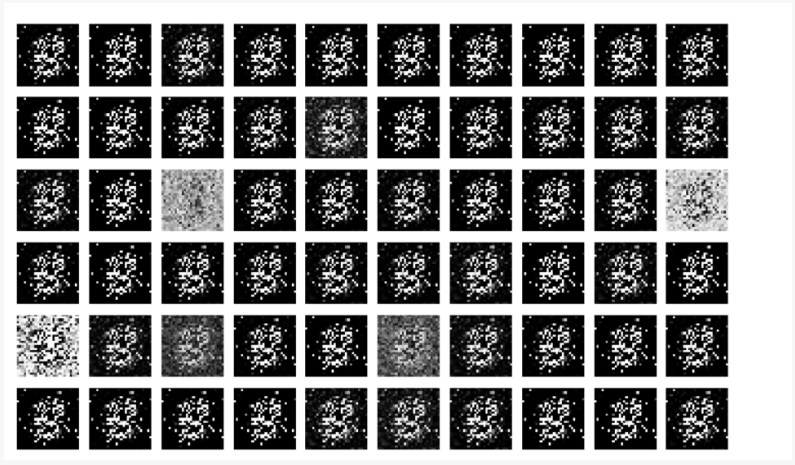
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tqdm import tqdm
%matplotlib inline
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Dense, Dropout, Input, LeakyReLU
from tensorflow.keras.models import Model,Sequential
from tensorflow.keras.datasets import mnist
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import plot_model, to_categorical

PATH_TO_DATA = '/content/digit-recognizer/'

def preprocessing(train, split_train_size = 1/7):

    X_train = train.drop(["label"],axis = 1)
    y_train = train["label"]
    X_train = X_train.values.reshape(-1,28,28)
    y_train = to_categorical(y_train, num_classes = 10)
```

```
    X_train, X_test, y_train, y_test = train_test_split(X_train, y_train,   test_size = split_train_size,
random_state=42)
    return X_train, X_test, y_train, y_test

def load_data(from_MNIST = True):
    if from_MNIST:
        (x_train, y_train), (x_test, y_test) = mnist.load_data()
    else:
        train = pd.read_csv(PATH_TO_DATA + 'train.csv')
        x_train, x_test, y_train, y_test = preprocessing(train)
    x_train = (x_train.astype(np.float32) - 127.5)/127.5
    nb_images_train = x_train.shape[0]
    x_train = x_train.reshape(nb_images_train, 784)
    return (x_train, y_train, x_test, y_test)

def adam_optimizer():
    return Adam(lr=0.0002, beta_1=0.5)

def create_generator():
    generator=Sequential()
    generator.add(Dense(units=256, input_dim=100))
    generator.add(LeakyReLU(0.2))
    generator.add(Dense(units=512))
    generator.add(LeakyReLU(0.2))
    generator.add(Dense(units=1024))
    generator.add(LeakyReLU(0.2))
    generator.add(Dense(units=784, activation='tanh'))
    generator.compile(loss = 'binary_crossentropy', optimizer = adam_optimizer())
    return generator
g = create_generator()

def create_discriminator():
    discriminator = Sequential()
    discriminator.add(Dense(units = 1024, input_dim = 784))
    discriminator.add(LeakyReLU(0.2))
    discriminator.add(Dropout(0.3))
    discriminator.add(Dense(units = 512))
    discriminator.add(LeakyReLU(0.2))
    discriminator.add(Dropout(0.3))
    discriminator.add(Dense(units=256))
```

```
    discriminator.add(LeakyReLU(0.2))
    discriminator.add(Dense(units=1, activation='sigmoid'))
    discriminator.compile(loss = 'binary_crossentropy', optimizer = adam_optimizer())
    return discriminator
d = create_discriminator()

def create_gan(discriminator, generator):
    discriminator.trainable=False
    gan_input = Input(shape=(100,))
    x = generator(gan_input)
    gan_output = discriminator(x)
    gan = Model(inputs = gan_input, outputs = gan_output)
    gan.compile(loss = 'binary_crossentropy', optimizer = 'adam')
    return gan
gan = create_gan(d,g)

def plot_generated_images(epoch, generator, examples=100, dim=(10,10), figsize=(10,10)):
    noise = np.random.normal(loc=0, scale=1, size=[examples, 100])
    generated_images = generator.predict(noise)
    generated_images = generated_images.reshape(100,28,28)
    plt.figure(figsize=figsize)
    for i in range(generated_images.shape[0]):
        plt.subplot(dim[0], dim[1], i+1)
        plt.imshow(generated_images[i], interpolation = 'nearest', cmap = 'gray')
        plt.axis('off')
    plt.tight_layout()

def training(epochs=1, batch_size=128):
    (X_train, y_train, X_test, y_test) = load_data(from_MNIST = False)
    batch_count = X_train.shape[0] / batch_size
    generator= create_generator()
    discriminator= create_discriminator()
    gan = create_gan(discriminator, generator)
    for e in range(1,epochs+1 ):
        for _ in range(batch_size):
            noise= np.random.normal(0,1, [batch_size, 100])
            generated_images = generator.predict(noise)
            image_batch=X_train[np.random.randint                    (low=0,high=X_train.shape[0],
size=batch_size)]
            X= np.concatenate([image_batch, generated_images])
```

```
        y_dis=np.zeros(2*batch_size)
        y_dis[:batch_size]=0.9
        discriminator.trainable=True
        discriminator.train_on_batch(X, y_dis)
        noise= np.random.normal(0,1, [batch_size, 100])
        y_gen = np.ones(batch_size)
        discriminator.trainable=False
        gan.train_on_batch(noise, y_gen)
    if e == 1 or e % 20 == 0:
        plot_generated_images(e, generator)
training(400,128)
```

OUTPUT:



RESULT:

Thus, a python program for building GAN with Keras is done successfully.

EX.NO : 8
DATE : 11-11-2022

Write a python program to perform Object detection with YOLO3

AIM:

To write a python program to perform Object detection with YOLO3.

PROCEDURE:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Perform object detection using YOLO3.
4. Train the model.
5. Perform testing with the test dataset.

CODE:

```
import os
import numpy as np
import pandas as pd
import struct
import scipy.io
import scipy.misc
import PIL
import cv2
from skimage.transform import resize
import tensorflow as tf
from keras import backend as K
from keras.layers import Input, Lambda, Conv2D, BatchNormalization, LeakyReLU, ZeroPadding2D, UpSampling2D
from keras.models import load_model, Model
from keras.layers.merge import add, concatenate
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
from matplotlib.patches import Rectangle
```

```
class Read_Weights:
    def __init__(self, file_name):
        with open(file_name, 'rb') as w_f:
            major, = struct.unpack('i', w_f.read(4))
            minor, = struct.unpack('i', w_f.read(4))
            revision, = struct.unpack('i', w_f.read(4))
            if (major*10 + minor)>= 2and major<1000 and minor<1000:
                w_f.read(8)
            else:
                w_f.read(4)
            transpose = (major > 1000) or (minor > 1000)
            binary = w_f.read()
        self.offset = 0
        self.all_weights = np.frombuffer(binary, dtype = 'float32')
    def read_bytes(self, size):
        self.offset = self.offset + size
        return self.all_weights[ self.offset-size : self.offset ]

    def load_weights(self, model):
        for i in range(106):
            try:
                conv_layer = model.get_layer('conv_' + str(i))
                print("loading weights of convolution #" + str(i))
                if i not in [81, 93, 105]:
                    norm_layer = model.get_layer('bnorm_' + str(i))
                    size = np.prod(norm_layer.get_weights()[0].shape)
                    beta  = self.read_bytes(size) # bias
                    gamma = self.read_bytes(size) # scale
                    mean  = self.read_bytes(size) # mean
                    var   = self.read_bytes(size) # variance
                 weights = norm_layer.set_weights([gamma,beta,mean, var])
                if len(conv_layer.get_weights()) > 1:
                        bias=self.read_bytes(np.prod(conv_layer.get_weights() [1].shape))
                         kernel=self.read_bytes(np.prod(conv_layer.get_weights() [0].shape))
                        kernel = kernel.reshape(list(reversed(conv_layer.get_weights()[0].shape)))
                        kernel = kernel.transpose([2,3,1,0])
                        conv_layer.set_weights([kernel, bias])
                else:
                    kernel = self.read_bytes(np.prod(conv_layer.get_weights()[0].shape))
                    kernel = kernel.reshape(list(reversed(conv_layer.get_weights()[0].shape)))
```

```
                kernel = kernel.transpose([2,3,1,0])
                conv_layer.set_weights([kernel])
            except ValueError:
                print("no convolution #" + str(i))

    def reset(self):
        self.offset = 0

def conv_block(inp, convs, skip=True):
    x = inp
    count = 0
    for conv in convs:
        if count == (len(convs) - 2) and skip:
            skip_connection = x
        count += 1
        if conv['stride'] > 1 :
            x = ZeroPadding2D(((1,0),(1,0)))(x)
            x=Conv2D(conv['filter'],conv['kernel'],
              strides = conv['stride'],
                padding = 'valid' if conv['stride'] > 1 else 'same',
                name = 'conv_' + str(conv['layer_idx']),
                use_bias = False if conv['bnorm'] else True)(x)
                if conv['bnorm']: x = BatchNormalization(epsilon = 0.001, name = 'bnorm_' +
str(conv['layer_idx']))(x)
        if conv['leaky']: x = LeakyReLU(alpha = 0.1, name = 'leaky_' + str(conv['layer_idx']))(x)
    return add([skip_connection, x]) if skip else x

def make_yolov3_model():
    input_image = Input(shape=(None, None, 3))
      x= conv_block(input_image, [{'filter': 32, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 0},{'filter': 64, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 1},
{'filter': 32, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 2},
 {'filter': 64, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 3}])
     x = conv_block(x, [{'filter': 128, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx':
5},
                    {'filter':  64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 6},
                    {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 7}])
    x = conv_block(x, [{'filter':  64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
9},
                    {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 10}])
```

```
    x = conv_block(x, [{'filter': 256, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 12},
                       {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 13},
                       {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 14}])
    for i in range(7):
        x = conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 16+i*3},
                           {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 17+i*3}])
    skip_36 = x
    x = conv_block(x, [{'filter': 512, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 37},
                       {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 38},
                       {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 39}])
    for i in range(7):
        x = conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 41+i*3},
                           {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 42+i*3}])
    skip_61 = x
    x = conv_block(x, [{'filter': 1024, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 62},
                       {'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 63},
                       {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 64}])
    for i in range(3):
        x = conv_block(x, [{'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 66+i*3},
                           {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 67+i*3}])
    x = conv_block(x, [{'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 75},
                       {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 76},
                       {'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 77},
                       {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 78},
                         {'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 79}], skip=False)
    yolo_82 = conv_block(x, [{'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True, 'layer_idx': 80},
                           {'filter':  255, 'kernel': 1, 'stride': 1, 'bnorm': False, 'leaky': False, 'layer_idx': 81}], skip=False)
```

```
    x = conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 84}], skip=False)
    x = UpSampling2D(2)(x)
    x = concatenate([x, skip_61])
    x = conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 87},
                       {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 88},
                       {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 89},
                       {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 90},
                       {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 91}], skip=False)
    yolo_94 = conv_block(x, [{'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True, 'layer_idx': 92},{'filter': 255, 'kernel': 1, 'stride': 1, 'bnorm': False, 'leaky': False, 'layer_idx': 93}], skip=False)
    x = conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,  'layer_idx': 96}], skip=False)
    x = UpSampling2D(2)(x)
    x = concatenate([x, skip_36])
    yolo_106 = conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,  'leaky': True, 'layer_idx': 99}, {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True,  'layer_idx': 100},
{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,  'leaky': True,  'layer_idx': 101}, 'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True,  'layer_idx': 102}, {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,  'leaky': True,  'layer_idx': 103}, {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True,  'layer_idx': 104}, {'filter': 255, 'kernel': 1, 'stride': 1, 'bnorm': False, 'leaky': False, 'layer_idx': 105}], skip=False)
    model = Model(input_image, [yolo_82, yolo_94, yolo_106])
    return model

yolov3 = make_yolov3_model()
weight_reader = Read_Weights("../input/data-for-yolo-v3-kernel/yolov3.weights")
weight_reader.load_weights(yolov3)
yolov3.save('yolo_model.h5')

def load_image_pixels(filename, shape):
  image = load_img(filename)
  width, height = image.size
  image = load_img(filename, target_size = shape)
  image = img_to_array(image)
  image = image.astype('float32')
```

```
  image /= 255.0
  image = np.expand_dims(image, 0)
  return image, width, height

class BoundBox:
    def __init__(self, xmin, ymin, xmax, ymax, objness = None, classes = None):
        self.xmin = xmin
        self.ymin = ymin
        self.xmax = xmax
        self.ymax = ymax
        self.objness = objness
        self.classes = classes
        self.label = -1
        self.score = -1

    def get_label(self):
        if self.label == -1:
            self.label = np.argmax(self.classes)
        return self.label

    def get_score(self):
        if self.score == -1:
            self.score = self.classes[self.get_label()]
        return self.get_score

def _sigmoid(x):
    return 1. /(1. + np.exp(-x))

def decode_netout(netout, anchors, obj_thresh, net_h, net_w):
    grid_h, grid_w = netout.shape[:2]
    nb_box = 3
    netout = netout.reshape((grid_h, grid_w, nb_box, -1))
    nb_class = netout.shape[-1] - 5
    boxes = []
    netout[..., :2]  = _sigmoid(netout[..., :2])
    netout[..., 4:]  = _sigmoid(netout[..., 4:])
    netout[..., 5:]  = netout[..., 4][..., np.newaxis] * netout[..., 5:]
    netout[..., 5:] *= netout[..., 5:] > obj_thresh
    for i in range(grid_h*grid_w):
        row = i / grid_w
```

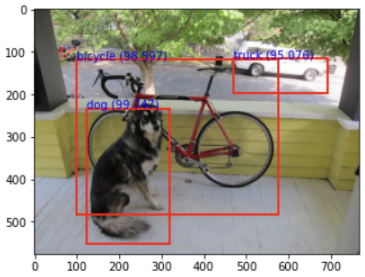
```
            col = i % grid_w
            for b in range(nb_box):
                objectness = netout[int(row)][int(col)][b][4]
                if(objectness.all() <= obj_thresh): continue
                x, y, w, h = netout[int(row)][int(col)][b][:4]
                x = (col + x) / grid_w
                y = (row + y) / grid_h
                w = anchors[2 * b + 0] * np.exp(w) / net_w
                h = anchors[2 * b + 1] * np.exp(h) / net_h
                classes = netout[int(row)][col][b][5:]
                box = BoundBox(x-w/2, y-h/2, x+w/2, y+h/2, objectness, classes)
                boxes.append(box)
    return boxes

def correct_yolo_boxes(boxes, image_h, image_w, net_h, net_w):
    new_w, new_h = net_w, net_h
    for i in range(len(boxes)):
        x_offset, x_scale = (net_w - new_w)/2./net_w, float(new_w)/net_w
        y_offset, y_scale = (net_h - new_h)/2./net_h, float(new_h)/net_h
        boxes[i].xmin = int((boxes[i].xmin - x_offset) / x_scale * image_w)
        boxes[i].xmax = int((boxes[i].xmax - x_offset) / x_scale * image_w)
        boxes[i].ymin = int((boxes[i].ymin - y_offset) / y_scale * image_h)
        boxes[i].ymax = int((boxes[i].ymax - y_offset) / y_scale * image_h)

def interval_overlap(interval_a, interval_b):
    x1, x2 = interval_a
    x3, x4 = interval_b
    if x3 < x1:
        if x4 < x1:
            return 0
        else:
            return min(x2,x4) - x1
    else:
        if x2 < x3:
             return 0
        else:
            return min(x2,x4) - x3

def bbox_iou(box1, box2):
    intersect_w = interval_overlap([box1.xmin, box1.xmax], [box2.xmin, box2.xmax])
```

```
    intersect_h = interval_overlap([box1.ymin, box1.ymax], [box2.ymin, box2.ymax])
    intersect = intersect_w * intersect_h
    w1, h1 = box1.xmax-box1.xmin, box1.ymax-box1.ymin
    w2, h2 = box2.xmax-box2.xmin, box2.ymax-box2.ymin
    union = w1*h1 + w2*h2 - intersect
    return float(intersect) / union

def nms(boxes, nms_thresh):
    if len(boxes) > 0:
        nb_class = len(boxes[0].classes)
    else:
        return
    for c in range(nb_class):
        sorted_indices = np.argsort([-box.classes[c] for box in boxes])
        for i in range(len(sorted_indices)):
            index_i = sorted_indices[i]

            if boxes[index_i].classes[c] == 0: continue
            for j in range(i+1, len(sorted_indices)):
                index_j = sorted_indices[j]
                if bbox_iou(boxes[index_i], boxes[index_j]) >= nms_thresh:
                    boxes[index_j].classes[c] = 0

def get_boxes(boxes, labels, thresh):
    v_boxes, v_labels, v_scores = list(), list(), list()
    for box in boxes:
        for i in range(len(labels)):
            if box.classes[i] > thresh:
                v_boxes.append(box)
                v_labels.append(labels[i])
                v_scores.append(box.classes[i]*100)
    return v_boxes, v_labels, v_scores

def draw_boxes(filename, v_boxes, v_labels, v_scores):
    data = plt.imread(filename)
    plt.imshow(data)
    ax = plt.gca()
    for i in range(len(v_boxes)):
        box = v_boxes[i]
        y1, x1, y2, x2 = box.ymin, box.xmin, box.ymax, box.xmax
```

```
        width, height = x2 - x1, y2 - y1
        rect = plt.Rectangle((x1, y1), width, height, fill = False, color = 'red', linewidth = '2')
        ax.add_patch(rect)
        label = "%s (%.3f)" % (v_labels[i], v_scores[i])
        plt.text(x1, y1, label, color = 'b')
    plt.show()
anchors = [[116,90, 156,198, 373,326], [30,61, 62,45, 59,119], [10,13, 16,30, 33,23]]
class_threshold = 0.6
labels = ["person", "bicycle", "car", "motorbike", "aeroplane", "bus", "train", "truck",
    "boat", "traffic light", "fire hydrant", "stop sign", "parking meter", "bench",
    "bird", "cat", "dog", "horse", "sheep", "cow", "elephant", "bear", "zebra", "giraffe",
    "backpack", "umbrella", "handbag", "tie", "suitcase", "frisbee", "skis", "snowboard",
    "sports ball", "kite", "baseball bat", "baseball glove", "skateboard", "surfboard",
    "tennis racket", "bottle", "wine glass", "cup", "fork", "knife", "spoon", "bowl", "banana",
    "apple", "sandwich", "orange", "broccoli", "carrot", "hot dog", "pizza", "donut", "cake",
    "chair", "sofa", "pottedplant", "bed", "diningtable", "toilet", "tvmonitor", "laptop", "mouse",
    "remote", "keyboard", "cell phone", "microwave", "oven", "toaster", "sink", "refrigerator",
    "book", "clock", "vase", "scissors", "teddy bear", "hair drier", "toothbrush"]
input_w, input_h = 416, 416
def predict_boxes(image_names):
    for image_name in image_names:
      image, image_w, image_h = load_image_pixels(image_name, (input_w, input_h))
      yhat = yolov3.predict(image)
      boxes = list()
      for i in range(len(yhat)):
        boxes += decode_netout(yhat[i][0], anchors[i], class_threshold, input_h, input_w)
      correct_yolo_boxes(boxes, image_h, image_w, input_h, input_w)
      nms(boxes, 0.5)
      v_boxes, v_labels, v_scores = get_boxes(boxes, labels, class_threshold)
      for i in range(len(v_boxes)):
        print(v_labels[i], v_scores[i])
      draw_boxes(image_name, v_boxes, v_labels, v_scores)
image_names = ["/content/data-for-yolo-v3-kernel/dog.jpg"]
predict_boxes(image_names)
```

OUTPUT:

```
[(1, 13, 13, 255), (1, 26, 26, 255), (1, 52, 52, 255)]
truck 95.07625102996826
bicycle 98.59656095504761
dog 99.74659085273743
```



RESULT:

Thus, a python program to perform Object detection with YOLO3 is done successfully.