DEEP LEARNING - LAB

(COURSE CODE: 23UPCSC4L03)

A Laboratory record Submitted to Periyar University, Salem.
In partial fulfillment of the Requirements for the
Degree of

**MASTER OF SCIENCE
IN
DATA SCIENCE**

BY

**ARCHANA M
REG NO: U23PG507DTS005**

DEPARTMENT OF COMPUTER SCIENCE

PERIYAR UNIVERSITY

PERIYAR PALKALAI NAGAR,
SALEM – 636011

CERTIFICATE

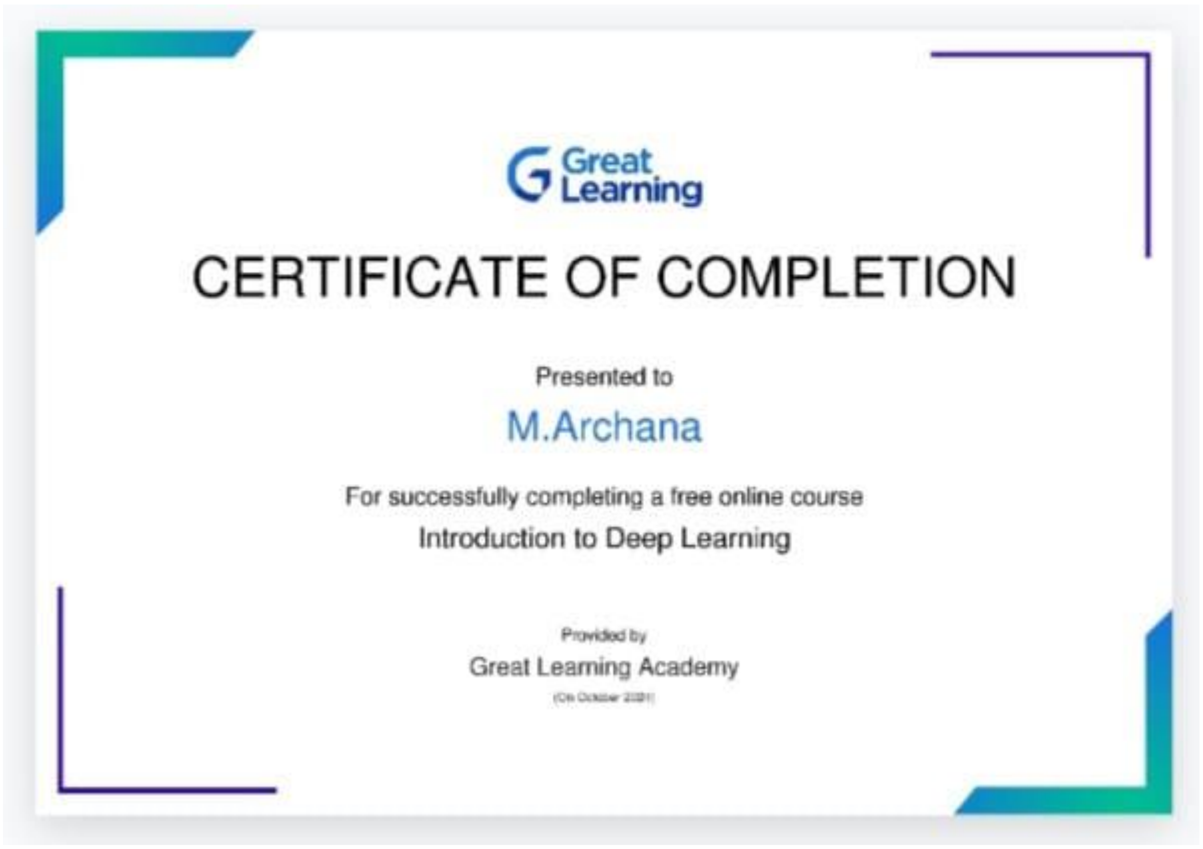
This is to certify that the Programming Laboratory entitled **“DEEP LEARNING LAB”** (Course code : 23UPCSC4L03) is a bonafide record work done by **ARCHANA .M** Register No. U23PG507DTS005 as partial fulfillment of the requirements degree of MASTER OF SCIENCE IN DATA SCIENCE in the Department of Computer Science, Periyar University, Salem, during the year 2023 – 2025.

Faculty In-Charge   Head of the Department

Submitted for the practical examination held on …… /…… /………………

Marks Obtained

Internal Examiner   External Examiner



Great Learning

CERTIFICATE OF COMPLETION

Presented to

M.Archana

For successfully completing a free online course

Introduction to Deep Learning

Provided by

Great Learning Academy

(On October 2021)

INDEX

Ex.No: 1	INTRODUCTION TO DEEP LEARNING AND FRAMEWORK
Date: 15.07.2024	

AIM:

Introduction to deep learning and framework.

THEORY:

Deep Learning:

Deep learning is a subset of machine learning, which is essentially a neural network with three or more layers. These neural networks attempt to simulate the behavior of the human brain—albeit far from matching its ability—allowing it to “learn” from large amounts of data. Some deep learning frameworks are

- TensorFlow
- PyTorch
- Keras
- MXNet
- Caffe
- Theano

Tensorflow:

TensorFlow is an open-source framework developed by Google researchers to run machine learning, deep learning, and other statistical and predictive analytics workloads. Likesimilar platforms, it's designed to streamline the process of developing and executing advanced analytics applications for users such as data scientists, statisticians, and predictive modelers.

PyTorch:

PyTorch is highly favored in academic research because of its flexibility and dynamic computation graph, which allows for more intuitive debugging. It’s known for being more "Pythonic" and user-friendly compared to TensorFlow.

Keras:

Keras is a high-level API that runs on top of TensorFlow, Theano, or CNTK. It allows for rapid prototyping and model experimentation with a simple, user-friendly interface.

MXNet:

MXNet is a flexible and efficient deep learning framework that supports both symbolic and imperative programming. It's well-suited for distributed computing and can scale across multiple GPUs and servers.

Caffe:

Caffe is known for its speed and efficiency, particularly in image processing tasks. It is commonly used for computer vision applications but has fewer updates and community support compared to TensorFlow or PyTorch.

Theano:

Theano is one of the first deep learning libraries and is still used for research and smaller projects. It offers efficient computations and is optimized for both CPU and GPU execution, although it is now considered less popular with the rise of TensorFlow and PyTorch.

RESULT:

Thus, the Introduction of Deep learning and Framework were completed successfully.

Ex.No: 2	# FEED FORWARD NETWORK
Date:24.07.2024	

AIM:

To implement the process of Simple Feed Forward Network.

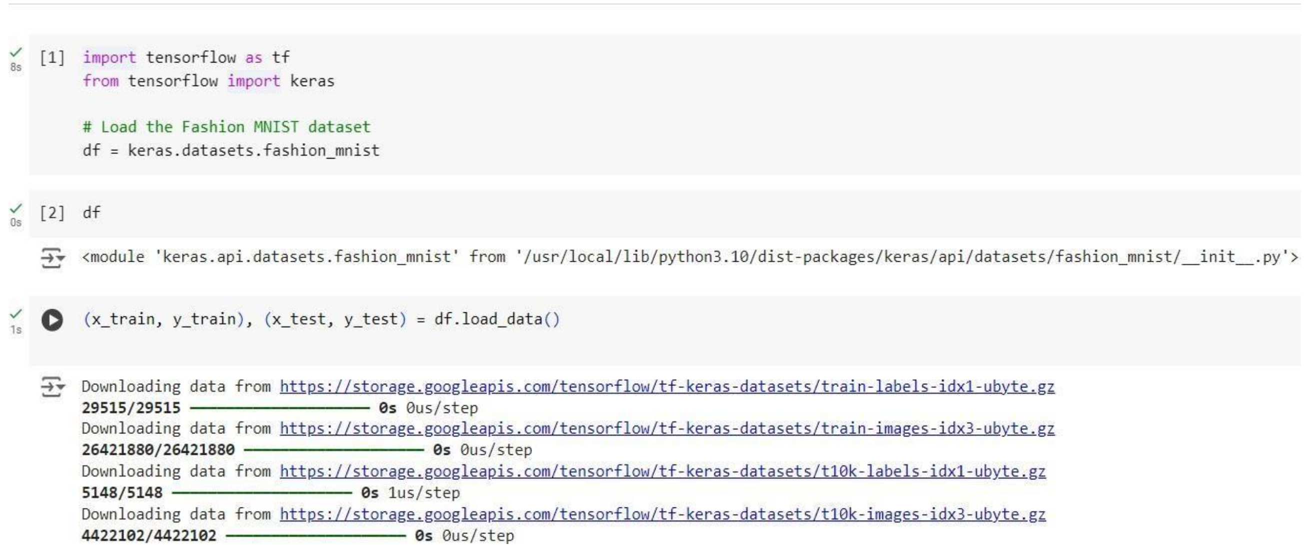
THEORY:

A feed-forward neural network is a biologically inspired classification algorithm. It consists of several simple neuron-like processing units, organized in layers, and every unit in a layer is connected with all the units in the previous layer.

PROCEDURE:

- Prepare the data
- Create s baseline
- Non-determinacy
- Beat the baseline

PROGRAM:



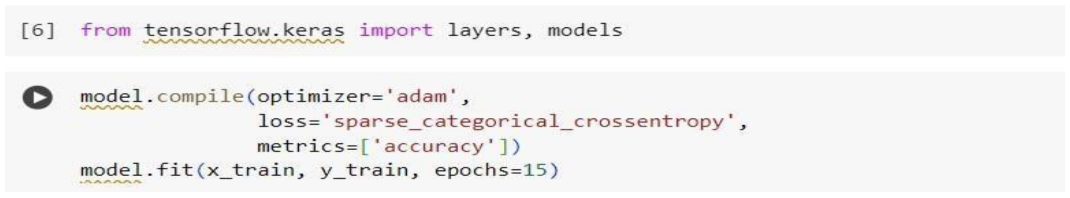
```
[1]  import tensorflow as tf
     from tensorflow import keras

     # Load the Fashion MNIST dataset
     df = keras.datasets.fashion_mnist
```

```
[2]  df
```

```
<module 'keras.api.datasets.fashion_mnist' from '/usr/local/lib/python3.10/dist-packages/keras/api/datasets/fashion_mnist/__init__.py'>
```

```
(x_train, y_train), (x_test, y_test) = df.load_data()
```

```
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 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 ━━━━━━━━━━━━━━━━━━━━ 0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```



```
[6]  from tensorflow.keras import layers, models
```

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=15)
```

RESULT:

In the above program feedforward neural network was Completed successfully.

Ex.No: 3	**MULTILAYER PERCEPTRON**
Date: 31.07.2024	

AIM:

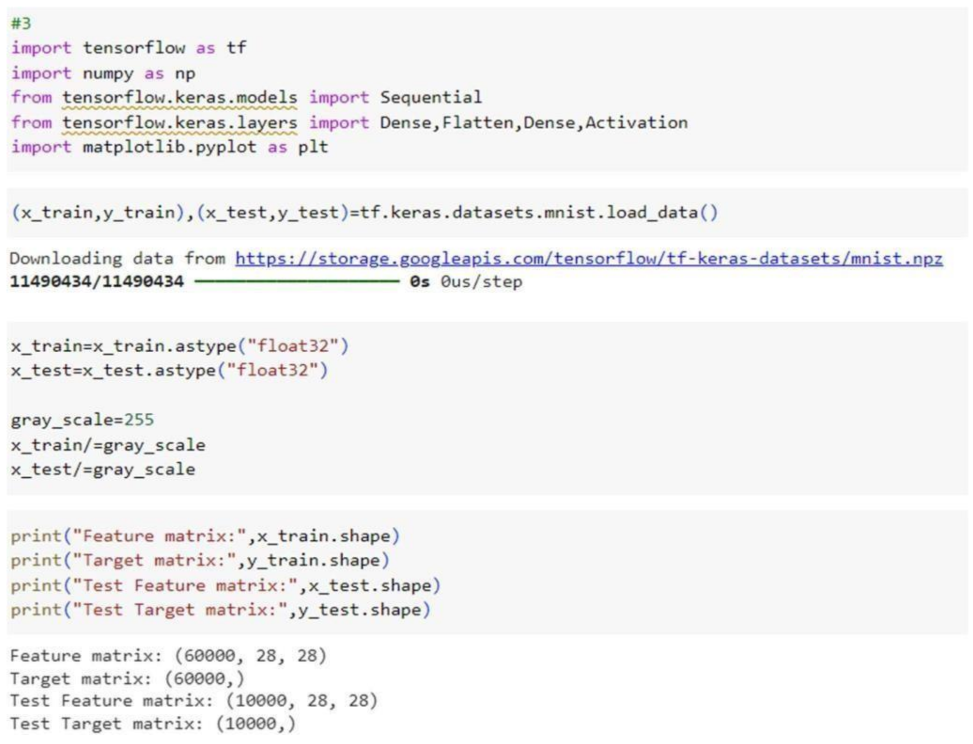
To Create Multilayer Perceptron Program.

THEORY:

Multi-layer Perceptron (MLP) is a supervised learning algorithm that learns a functionf (.): R m → R o by training on a dataset, where is the number of dimensions for input and is the number of dimensions for output.

MLPs are suitable for classification prediction problems where inputs are assigned a class or label. They are also suitable for regression prediction problems where a real- valued quantity is predicted given a set of inputs.

PROGRAM:



```
#3
import tensorflow as tf
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Flatten,Dense,Activation
import matplotlib.pyplot as plt
```

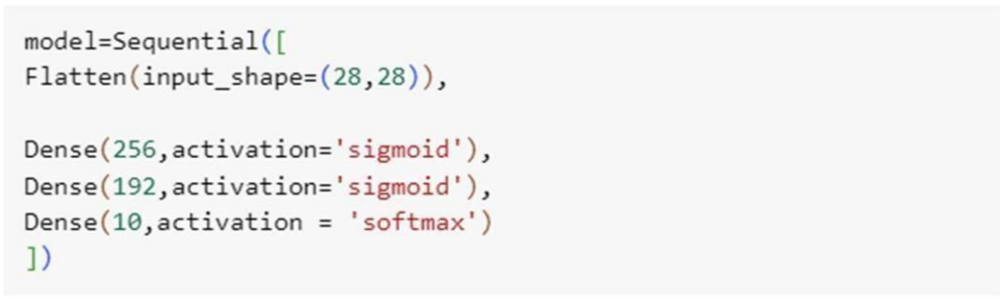
```
(x_train,y_train),(x_test,y_test)=tf.keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

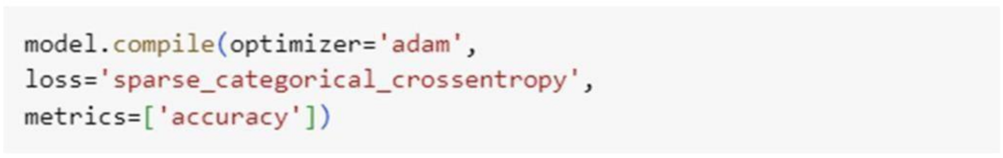
```
x_train=x_train.astype("float32")
x_test=x_test.astype("float32")

gray_scale=255
x_train/=gray_scale
x_test/=gray_scale
```

```
print("Feature matrix:",x_train.shape)
print("Target matrix:",y_train.shape)
print("Test Feature matrix:",x_test.shape)
print("Test Target matrix:",y_test.shape)
```

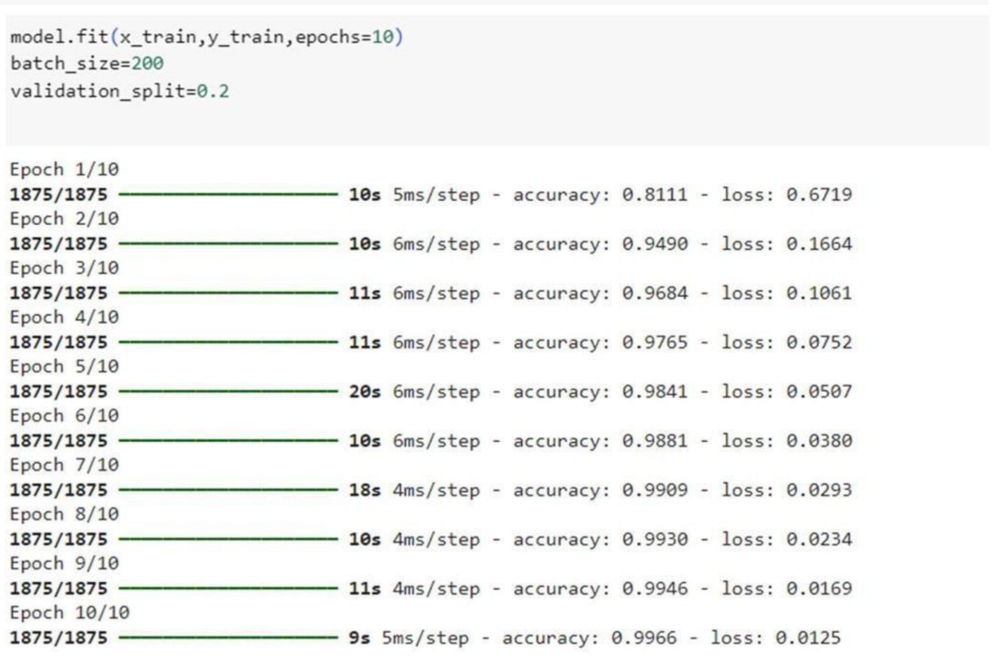
```
Feature matrix: (60000, 28, 28)
Target matrix: (60000,)
Test Feature matrix: (10000, 28, 28)
Test Target matrix: (10000,)
```



```
model=Sequential([
Flatten(input_shape=(28,28)),

Dense(256,activation='sigmoid'),
Dense(192,activation='sigmoid'),
Dense(10,activation = 'softmax')
])
```

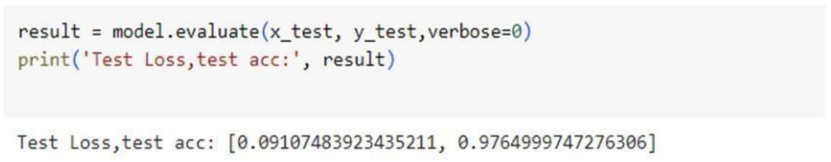


```
model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
```



```
model.fit(x_train,y_train,epochs=10)
batch_size=200
validation_split=0.2
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.8111 - loss: 0.6719
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - accuracy: 0.9490 - loss: 0.1664
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - accuracy: 0.9684 - loss: 0.1061
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - accuracy: 0.9765 - loss: 0.0752
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 20s 6ms/step - accuracy: 0.9841 - loss: 0.0507
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - accuracy: 0.9881 - loss: 0.0380
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 18s 4ms/step - accuracy: 0.9909 - loss: 0.0293
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9930 - loss: 0.0234
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.9946 - loss: 0.0169
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9966 - loss: 0.0125
```



```
result = model.evaluate(x_test, y_test,verbose=0)
print('Test Loss,test acc:', result)
```

```
Test Loss,test acc: [0.09107483923435211, 0.9764999747276306]
```

RESULT:

In this program multilayer perceptron was build successfully.

Ex.No: 4	**CNN – ON BINARY CLASSIFICATION TASK APPLE AND TOMATO DATASET**
Date:05.08.2024	

AIM:

To Program for Convolution neural network on binary classification task: Apple and Tomato dataset.

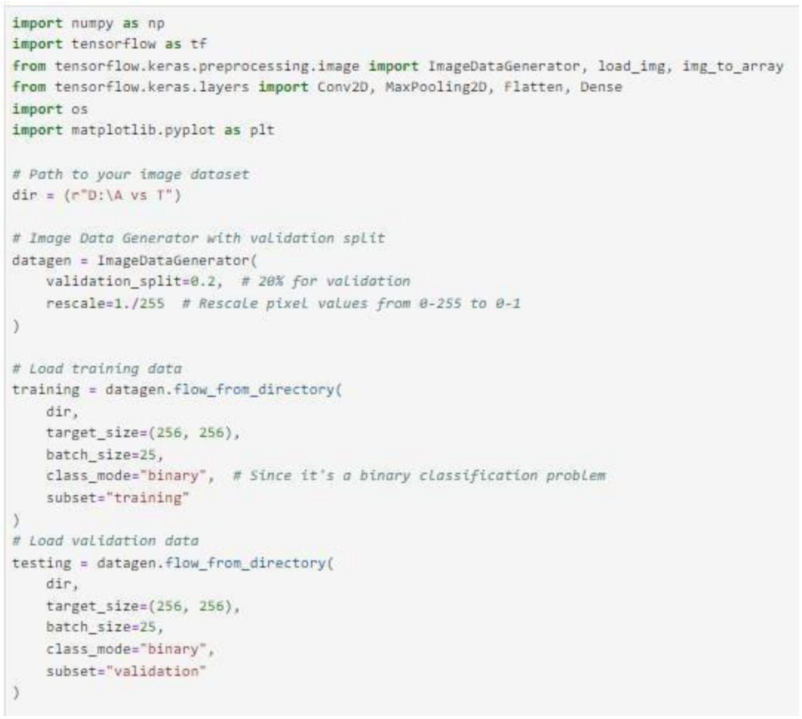
THEORY:

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of artificial neural network, most commonly applied to analyse visual imagery. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on the shared-weight architecture of the convolution kernels or filters that slide along input features and provide translation equivariant responses known as feature maps. Counter- intuitively, most convolutional neural networks are only equivariant, as opposed to invariant, to translation.

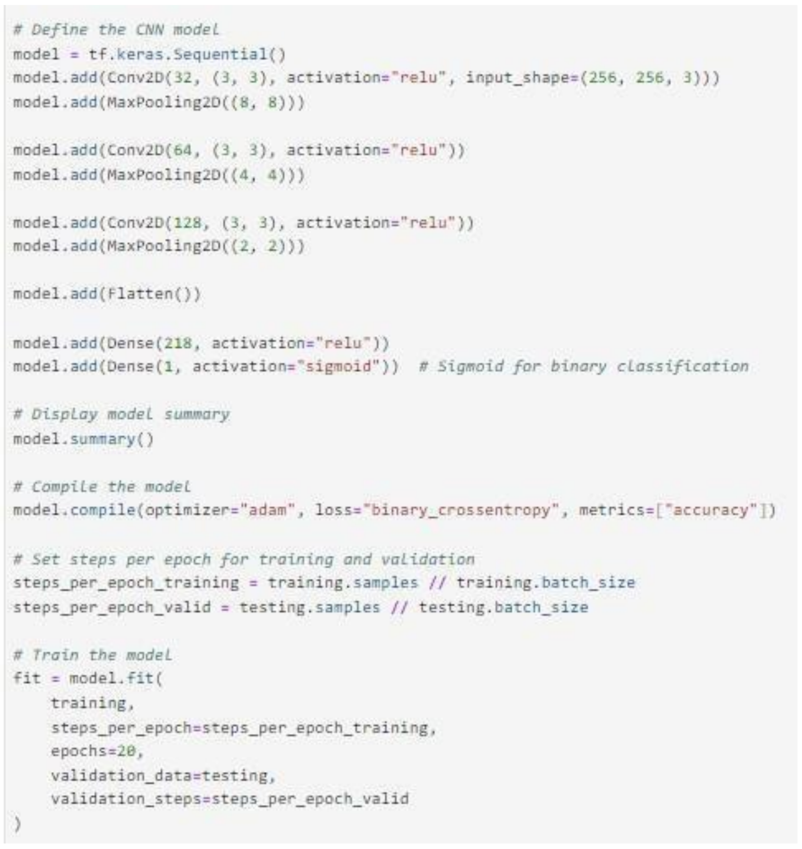
PROCEDURE:

1. Convolution
2. Pooling
3. Flattening
4. Fullconversio
5. Output layer

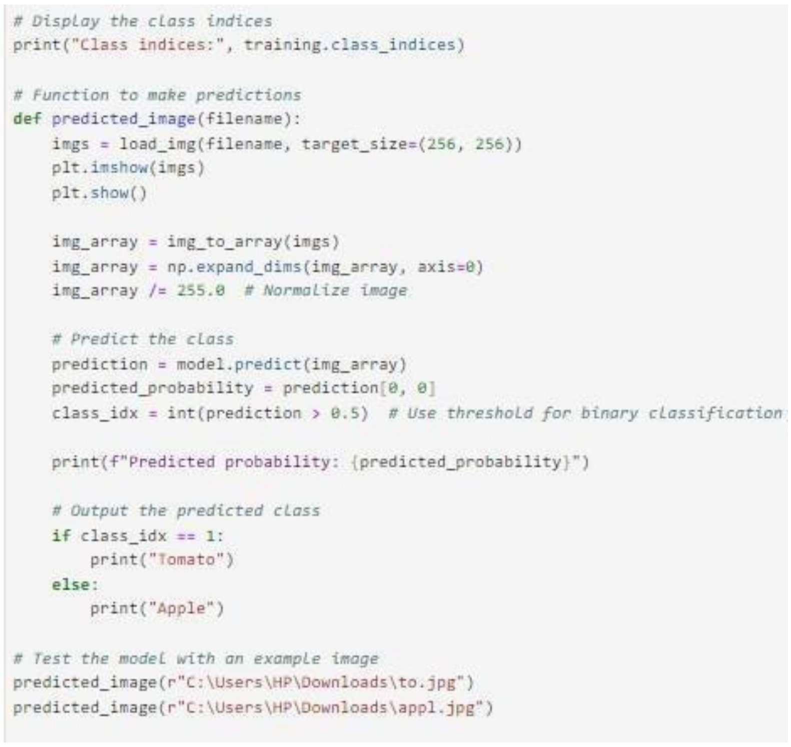
PROGRAM:



```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
import os
import matplotlib.pyplot as plt

# Path to your image dataset
dir = (r"D:\A vs T")

# Image Data Generator with validation split
datagen = ImageDataGenerator(
    validation_split=0.2,  # 20% for validation
    rescale=1./255  # Rescale pixel values from 0-255 to 0-1
)

# Load training data
training = datagen.flow_from_directory(
    dir,
    target_size=(256, 256),
    batch_size=25,
    class_mode="binary",  # Since it's a binary classification problem
    subset="training"
)
# Load validation data
testing = datagen.flow_from_directory(
    dir,
    target_size=(256, 256),
    batch_size=25,
    class_mode="binary",
    subset="validation"
)
```

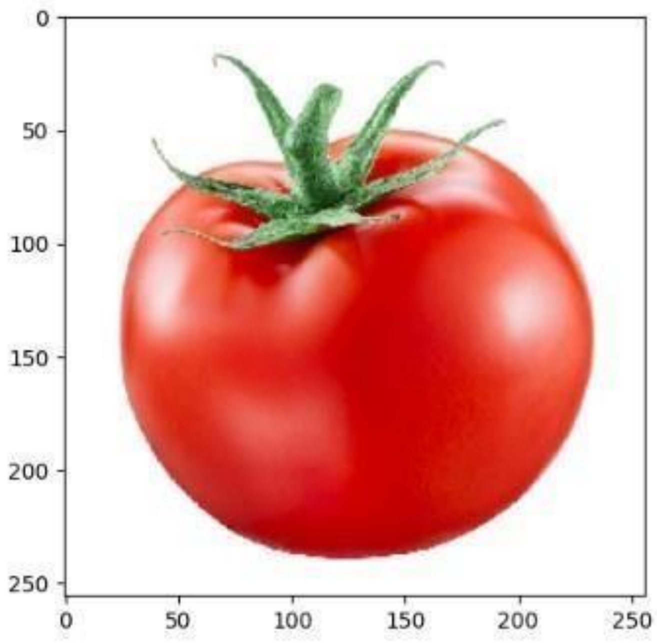


```
# Define the CNN model
model = tf.keras.Sequential()
model.add(Conv2D(32, (3, 3), activation="relu", input_shape=(256, 256, 3)))
model.add(MaxPooling2D((8, 8)))

model.add(Conv2D(64, (3, 3), activation="relu"))
model.add(MaxPooling2D((4, 4)))

model.add(Conv2D(128, (3, 3), activation="relu"))
model.add(MaxPooling2D((2, 2)))

model.add(Flatten())

model.add(Dense(218, activation="relu"))
model.add(Dense(1, activation="sigmoid"))  # Sigmoid for binary classification

# Display model summary
model.summary()

# Compile the model
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])

# Set steps per epoch for training and validation
steps_per_epoch_training = training.samples // training.batch_size
steps_per_epoch_valid = testing.samples // testing.batch_size

# Train the model
fit = model.fit(
    training,
    steps_per_epoch=steps_per_epoch_training,
    epochs=20,
    validation_data=testing,
    validation_steps=steps_per_epoch_valid
)
```

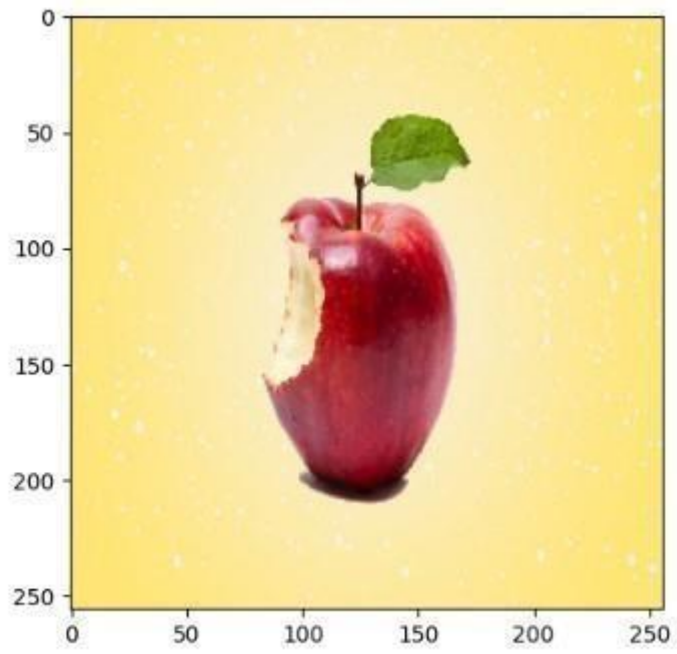


```python
# Display the class indices
print("Class indices:", training.class_indices)

# Function to make predictions
def predicted_image(filename):
    imgs = load_img(filename, target_size=(256, 256))
    plt.imshow(imgs)
    plt.show()

    img_array = img_to_array(imgs)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0  # Normalize image

    # Predict the class
    prediction = model.predict(img_array)
    predicted_probability = prediction[0, 0]
    class_idx = int(prediction > 0.5)  # Use threshold for binary classification

    print(f"Predicted probability: {predicted_probability}")

    # Output the predicted class
    if class_idx == 1:
        print("Tomato")
    else:
        print("Apple")

# Test the model with an example image
predicted_image(r"C:\Users\HP\Downloads\to.jpg")
predicted_image(r"C:\Users\HP\Downloads\appl.jpg")
```

```
Class indices: {'apple': 0, 'tomato': 1}
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 109ms/step
Predicted probability: 0.9908272624015808
Tomato
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
Predicted probability: 0.020304648205637932
Apple
```

RESULT:

Thus, the CNN network was bulid and the image is detected successfully.

Ex.No: 5	CNN – ON MULTI CLASSIFICATION TASK: DOG BREED CLASSIFICATION
Date:12.08.2024	

AIM:

To Program for CNN on Multi-classification task.

THEORY:

CNN (Convolutional Neural Network) is the common type of artificial neural networkused for image classification, that is designed to process pixel data. CNN architecture composed of multiple layers of artificial neurons, which will take large data set of labeled images, process the data through hidden layers, and output the image class. This article will walkthrough an example of CNN application to identify dog breed from the input images.

PROCEDURE:

Step 1: Import Datasets

Step 2: Detect Humans

Step 3: Detect Dogs

Step 4: Create a CNN to classify dog breed

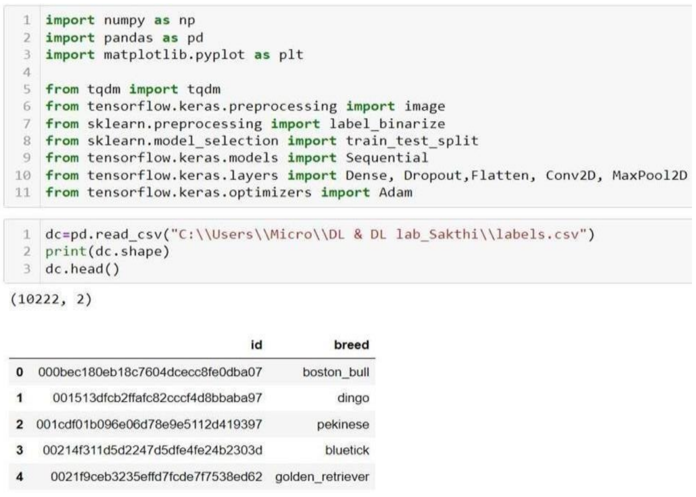
Step 5: Use a CNN to Classify Dog Breeds(using transfer learning)

Step 6: Create a CNN to classify dog breed (using transfer learning)

Step 7: write your algorithm

Step 8: Test your algorithm

PROGRAM:

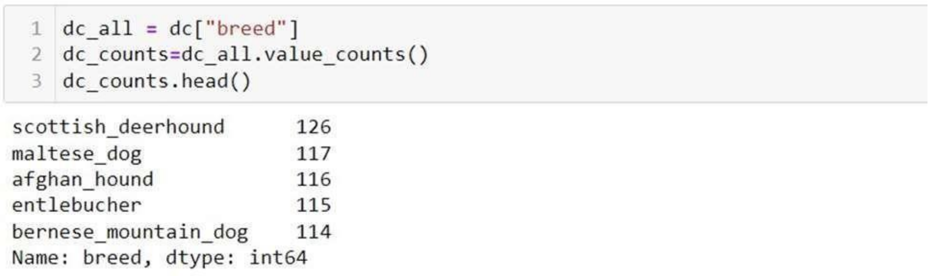


```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from tqdm import tqdm
from tensorflow.keras.preprocessing import image
from sklearn.preprocessing import label_binarize
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout,Flatten, Conv2D, MaxPool2D
from tensorflow.keras.optimizers import Adam
```

```python
dc=pd.read_csv("C:\\Users\\Micro\\DL & DL lab_Sakthi\\labels.csv")
print(dc.shape)
dc.head()
```

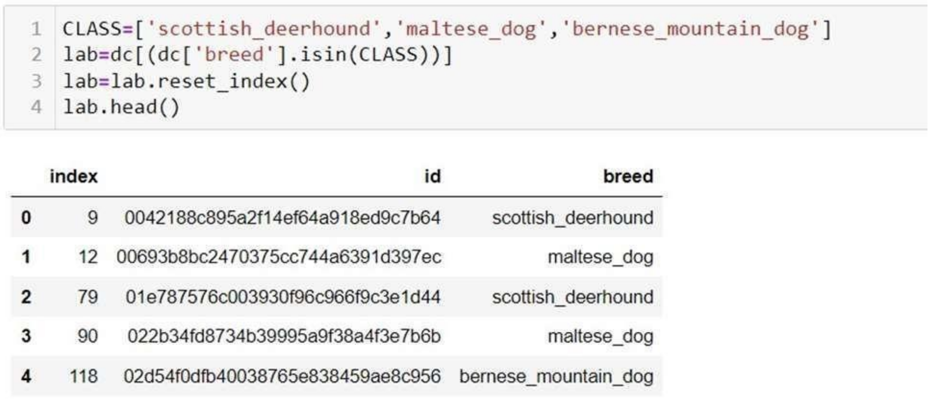
```
(10222, 2)
```

	id	breed
0	000bec180eb18c7604dcecc8fe0dba07	boston_bull
1	001513dfcb2ffafc82cccf4d8bbaba97	dingo
2	001cdf01b096e06d78e9e5112d419397	pekinese
3	00214f311d5d2247d5dfe4fe24b2303d	bluetick
4	0021f9ceb3235effd7fcde7f7538ed62	golden_retriever



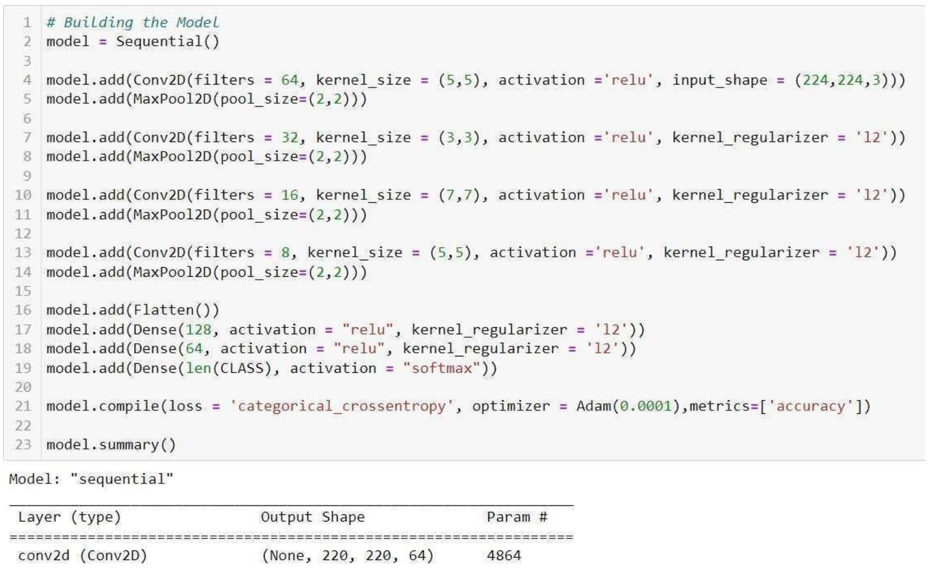
```python
dc_all = dc["breed"]
dc_counts=dc_all.value_counts()
dc_counts.head()
```

```
scottish_deerhound      126
maltese_dog             117
afghan_hound            116
entlebucher             115
bernese_mountain_dog    114
Name: breed, dtype: int64
```



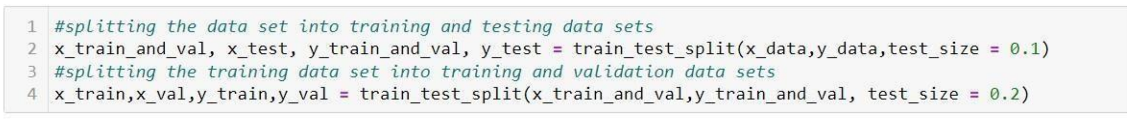
```python
CLASS=['scottish_deerhound','maltese_dog','bernese_mountain_dog']
lab=dc[(dc['breed'].isin(CLASS))]
lab=lab.reset_index()
lab.head()
```

	index	id	breed
0	9	0042188c895a2f14ef64a918ed9c7b64	scottish_deerhound
1	12	00693b8bc2470375cc744a6391d397ec	maltese_dog
2	79	01e787576c003930f96c966f9c3e1d44	scottish_deerhound
3	90	022b34fd8734b39995a9f38a4f3e7b6b	maltese_dog
4	118	02d54f0dfb40038765e838459ae8c956	bernese_mountain_dog

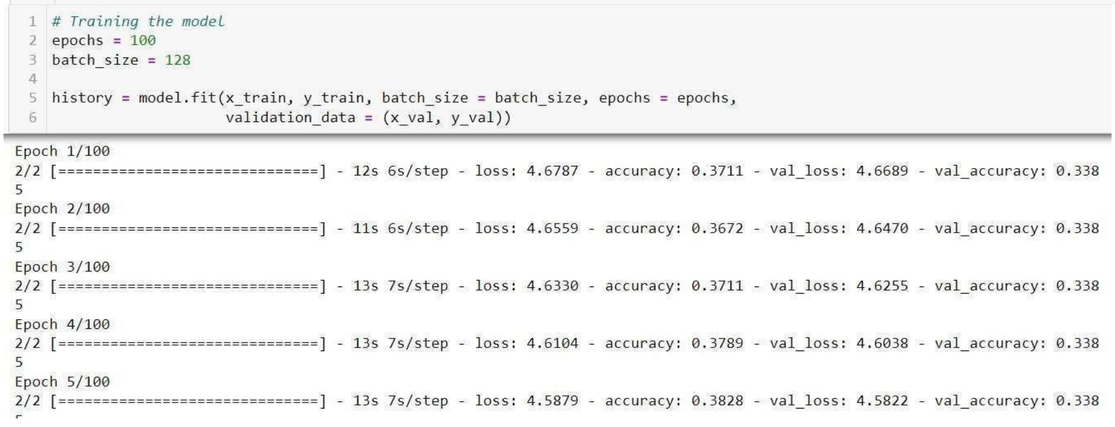


```python
# Building the Model
model = Sequential()

model.add(Conv2D(filters = 64, kernel_size = (5,5), activation ='relu', input_shape = (224,224,3)))
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Conv2D(filters = 32, kernel_size = (3,3), activation ='relu', kernel_regularizer = 'l2'))
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Conv2D(filters = 16, kernel_size = (7,7), activation ='relu', kernel_regularizer = 'l2'))
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Conv2D(filters = 8, kernel_size = (5,5), activation ='relu', kernel_regularizer = 'l2'))
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(128, activation = "relu", kernel_regularizer = 'l2'))
model.add(Dense(64, activation = "relu", kernel_regularizer = 'l2'))
model.add(Dense(len(CLASS), activation = "softmax"))

model.compile(loss = 'categorical_crossentropy', optimizer = Adam(0.0001),metrics=['accuracy'])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 220, 220, 64)      4864
```

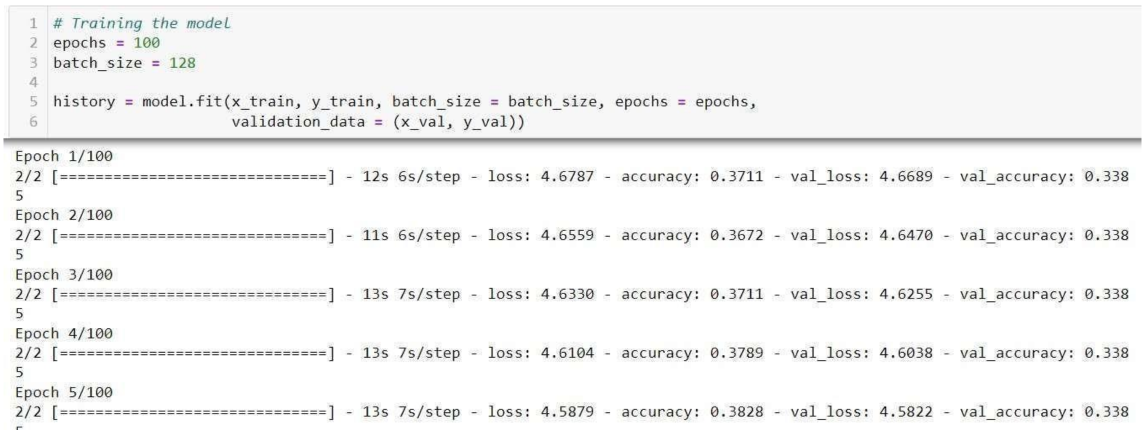


```python
#splitting the data set into training and testing data sets
x_train_and_val, x_test, y_train_and_val, y_test = train_test_split(x_data,y_data,test_size = 0.1)
#splitting the training data set into training and validation data sets
x_train,x_val,y_train,y_val = train_test_split(x_train_and_val,y_train_and_val, test_size = 0.2)
```



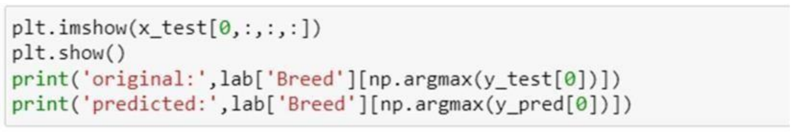
```python
# Training the model
epochs = 100
batch_size = 128

history = model.fit(x_train, y_train, batch_size = batch_size, epochs = epochs,
                    validation_data = (x_val, y_val))
```

```
Epoch 1/100
2/2 [==============================] - 12s 6s/step - loss: 4.6787 - accuracy: 0.3711 - val_loss: 4.6689 - val_accuracy: 0.338
5
Epoch 2/100
2/2 [==============================] - 11s 6s/step - loss: 4.6559 - accuracy: 0.3672 - val_loss: 4.6470 - val_accuracy: 0.338
5
Epoch 3/100
2/2 [==============================] - 13s 7s/step - loss: 4.6330 - accuracy: 0.3711 - val_loss: 4.6255 - val_accuracy: 0.338
5
Epoch 4/100
2/2 [==============================] - 13s 7s/step - loss: 4.6104 - accuracy: 0.3789 - val_loss: 4.6038 - val_accuracy: 0.338
5
Epoch 5/100
2/2 [==============================] - 13s 7s/step - loss: 4.5879 - accuracy: 0.3828 - val_loss: 4.5822 - val_accuracy: 0.338
```

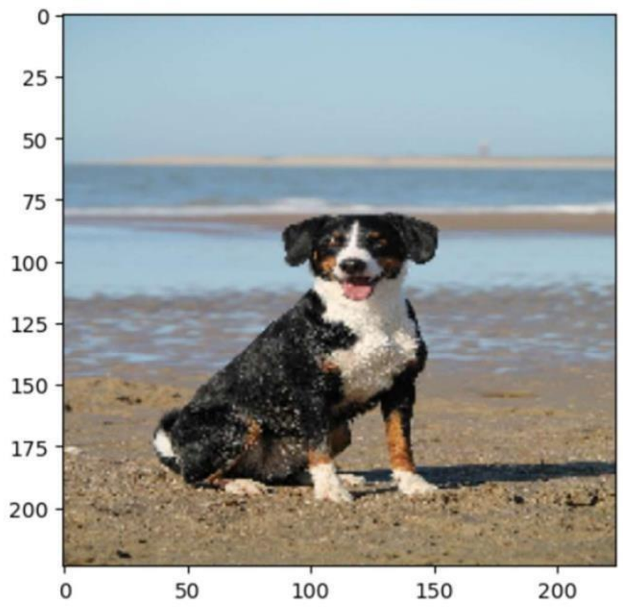


```
# Training the model
epochs = 100
batch_size = 128

history = model.fit(x_train, y_train, batch_size = batch_size, epochs = epochs,
                    validation_data = (x_val, y_val))
```

```
Epoch 1/100
2/2 [==============================] - 12s 6s/step - loss: 4.6787 - accuracy: 0.3711 - val_loss: 4.6689 - val_accuracy: 0.338
5
Epoch 2/100
2/2 [==============================] - 11s 6s/step - loss: 4.6559 - accuracy: 0.3672 - val_loss: 4.6470 - val_accuracy: 0.338
5
Epoch 3/100
2/2 [==============================] - 13s 7s/step - loss: 4.6330 - accuracy: 0.3711 - val_loss: 4.6255 - val_accuracy: 0.338
5
Epoch 4/100
2/2 [==============================] - 13s 7s/step - loss: 4.6104 - accuracy: 0.3789 - val_loss: 4.6038 - val_accuracy: 0.338
5
Epoch 5/100
2/2 [==============================] - 13s 7s/step - loss: 4.5879 - accuracy: 0.3828 - val_loss: 4.5822 - val_accuracy: 0.338
```



```
plt.imshow(x_test[0,:,:,:])
plt.show()
print('original:',lab['Breed'][np.argmax(y_test[0])])
print('predicted:',lab['Breed'][np.argmax(y_pred[0])])
```



```
original: Pug
predicted: Pug
```

RESULT:

The four types of dog images was classified and the output is verified successfully

Ex.No: 6	**TRANSFER LEARNING USING PRE TRAINED ARCHITECTURE**
Date:21.08.2024	

AIM:

To write Program for Transfer learning using pre trained architectures.

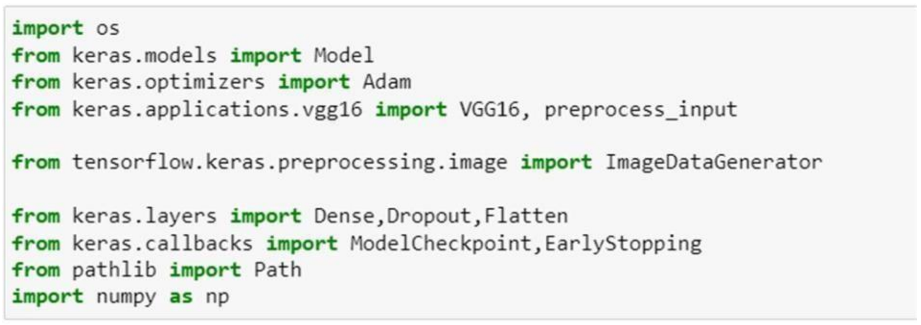
THEORY:

Transfer learning is a popular method in computer vision because it allows us to buildaccurate models in a timesaving way (Rawat & Wang 2017). With transfer learning, insteadof starting the learning process from scratch, you start from patterns that have been learned when solving a different problem. This way you leverage previous learnings and avoid starting from scratch. Take it as the deep learning version of Chartres' expression 'standingon the shoulder of giants.' Simply put, a pre-trained model is a model created by some one else to solve a similarproblem. Instead of building a model from scratch to solve a similar problem, you use the model trained on other problem as a starting point.

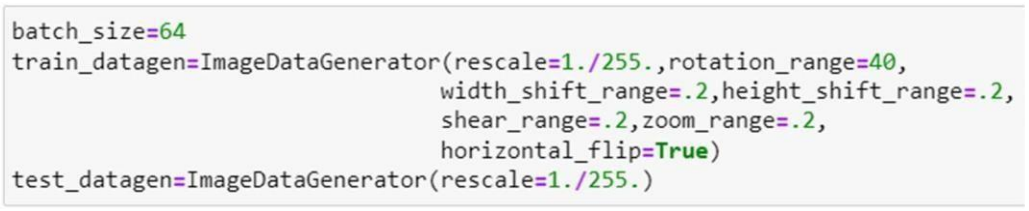
PROCEDURE:

1. Transfer learning
2. Convolutional neural networks
3. Repurposing a pre-trained model
4. Transfer learning process.

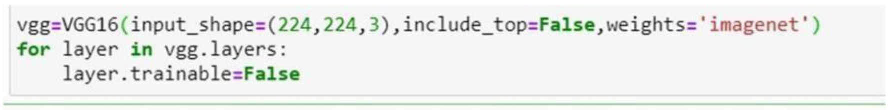
PROGRAM:



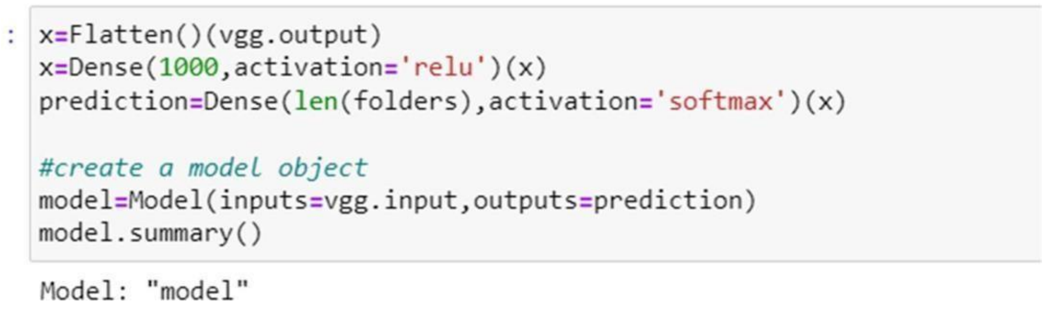
```
import os
from keras.models import Model
from keras.optimizers import Adam
from keras.applications.vgg16 import VGG16, preprocess_input

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from keras.layers import Dense,Dropout,Flatten
from keras.callbacks import ModelCheckpoint,EarlyStopping
from pathlib import Path
import numpy as np
```



```
batch_size=64
train_datagen=ImageDataGenerator(rescale=1./255.,rotation_range=40,
                                 width_shift_range=.2,height_shift_range=.2,
                                 shear_range=.2,zoom_range=.2,
                                 horizontal_flip=True)
test_datagen=ImageDataGenerator(rescale=1./255.)
```

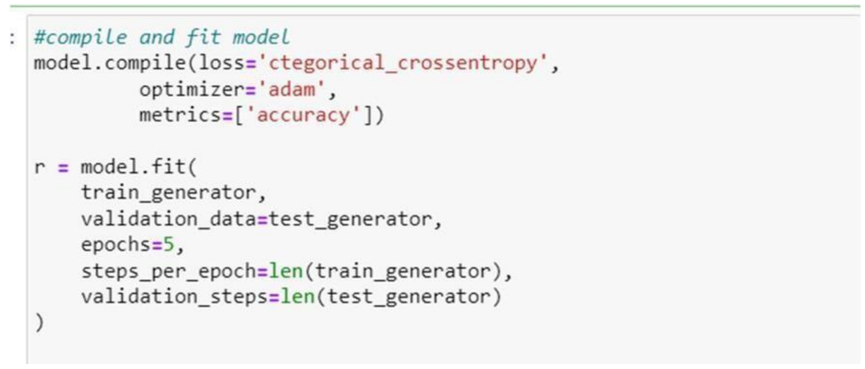


```
vgg=VGG16(input_shape=(224,224,3),include_top=False,weights='imagenet')
for layer in vgg.layers:
    layer.trainable=False
```



```
x=Flatten()(vgg.output)
x=Dense(1000,activation='relu')(x)
prediction=Dense(len(folders),activation='softmax')(x)

#create a model object
model=Model(inputs=vgg.input,outputs=prediction)
model.summary()
```

```
Model: "model"
```



```
#compile and fit model
model.compile(loss='ctegorical_crossentropy',
         optimizer='adam',
         metrics=['accuracy'])

r = model.fit(
    train_generator,
    validation_data=test_generator,
    epochs=5,
    steps_per_epoch=len(train_generator),
    validation_steps=len(test_generator)
)
```

RESULT:

Thus, the transfer learning method is used to build accurate model and the outputs verified.

Ex.No: 7	HYPER PARAMETER OPTIMIZATION ON CNN MODELS
Date:02.09.2024	

AIM:

To write a Program for Hyper parameter optimization on CNN models.

THEORY:

Hyperparameter tuning :

Tuning hyperparameters for deep neural network is difficult as it is slow to traina deep neural network and there are numerous parameters to configure. In this part, we briefly survey the hyper parameters for convnet.

Learning rate:

Learning rate controls how much to update the weight in the optimization algorithm. We can use fixed learning rate, gradually decreasing learning rate, momentum based methods or adaptive learning rates, depending on our choiceof optimizer such as SGD, Adam, Adagrad, AdaDelta or RMSProp.

Number of epochs :

Number of epochs is the the number of times the entire training set pass throughthe neural network. We should increase the number of epochs until we see a small gap between the test error and the training error

Batch size :

Mini-batch is usually preferable in the learning process of convnet. A range of 16 to 128 is a good choice to test with. We should note that convnet is sensitiveto batch size.

Activation function :

Activation funtion introduces non-linearity to the model. Usually, rectifierworks well with convnet. Other alternatives are sigmoid, tanh and other activation functions depening on the task.

Number of hidden layers and units :

It is usually good to add more layers until the test error no longer improves. Thetrade off is that it is computationally expensive to train the network.

Grid search or randomized search :

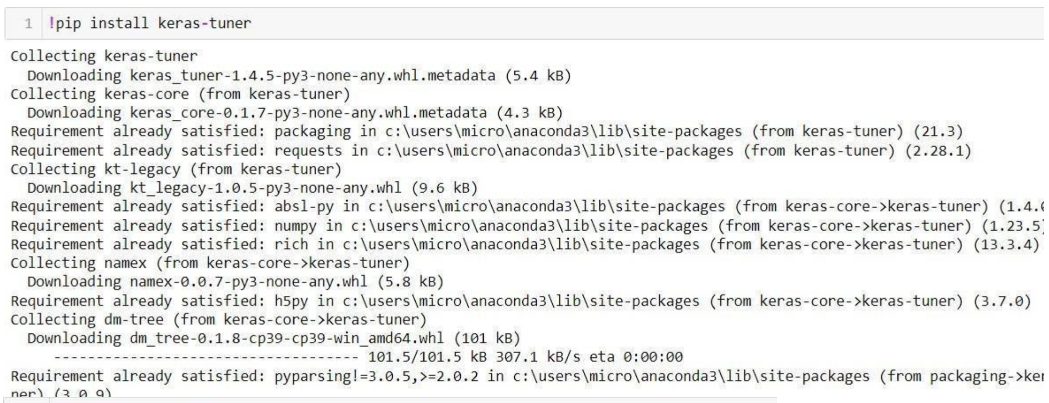
Manually tuning hyperparameter is painful and also impractical. There are two generic approaches to sampling search candidates. Grid search exhaustively search all parameter combinations for given values. Random search sample a given number of candidates from a parameter space with a specified distribution.

PROCEDURE:

Step 1: Decouple search parameters from code. Take the parameters that you want to tune and put them in a dictionary at the top of your script
Step 2: Wrap training and evaluation into a function.

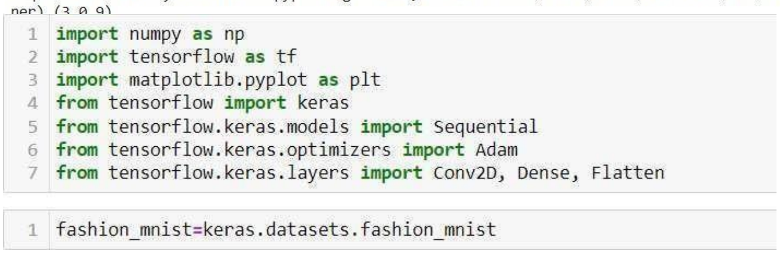
Step 3: Run Hyper parameter tuning script.

PROGRAM:



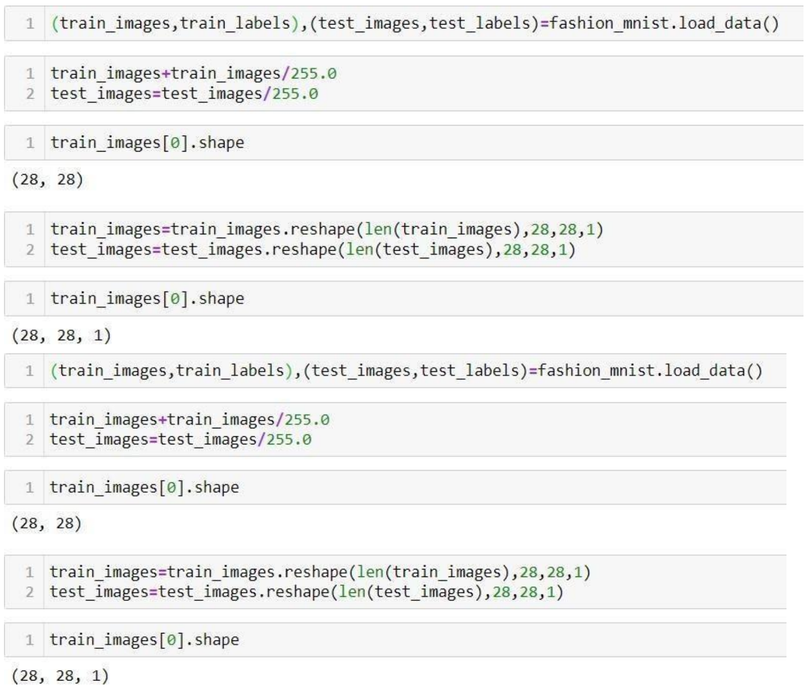
```
!pip install keras-tuner
```

```
Collecting keras-tuner
  Downloading keras_tuner-1.4.5-py3-none-any.whl.metadata (5.4 kB)
Collecting keras-core (from keras-tuner)
  Downloading keras_core-0.1.7-py3-none-any.whl.metadata (4.3 kB)
Requirement already satisfied: packaging in c:\users\micro\anaconda3\lib\site-packages (from keras-tuner) (21.3)
Requirement already satisfied: requests in c:\users\micro\anaconda3\lib\site-packages (from keras-tuner) (2.28.1)
Collecting kt-legacy (from keras-tuner)
  Downloading kt_legacy-1.0.5-py3-none-any.whl (9.6 kB)
Requirement already satisfied: absl-py in c:\users\micro\anaconda3\lib\site-packages (from keras-core->keras-tuner) (1.4.
Requirement already satisfied: numpy in c:\users\micro\anaconda3\lib\site-packages (from keras-core->keras-tuner) (1.23.5
Requirement already satisfied: rich in c:\users\micro\anaconda3\lib\site-packages (from keras-core->keras-tuner) (13.3.4)
Collecting namex (from keras-core->keras-tuner)
  Downloading namex-0.0.7-py3-none-any.whl (5.8 kB)
Requirement already satisfied: h5py in c:\users\micro\anaconda3\lib\site-packages (from keras-core->keras-tuner) (3.7.0)
Collecting dm-tree (from keras-core->keras-tuner)
  Downloading dm_tree-0.1.8-cp39-cp39-win_amd64.whl (101 kB)
     ------------------------------------ 101.5/101.5 kB 307.1 kB/s eta 0:00:00
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\micro\anaconda3\lib\site-packages (from packaging->ke
ner) (3.0.9)
```



```python
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Conv2D, Dense, Flatten
```

```python
fashion_mnist=keras.datasets.fashion_mnist
```



```python
(train_images,train_labels),(test_images,test_labels)=fashion_mnist.load_data()
```

```python
train_images+train_images/255.0
test_images=test_images/255.0
```

```python
train_images[0].shape
```

```
(28, 28)
```

```python
train_images=train_images.reshape(len(train_images),28,28,1)
test_images=test_images.reshape(len(test_images),28,28,1)
```

```python
train_images[0].shape
```

```
(28, 28, 1)
```

```python
(train_images,train_labels),(test_images,test_labels)=fashion_mnist.load_data()
```

```python
train_images+train_images/255.0
test_images=test_images/255.0
```

```python
train_images[0].shape
```

```
(28, 28)
```

```python
train_images=train_images.reshape(len(train_images),28,28,1)
test_images=test_images.reshape(len(test_images),28,28,1)
```

```python
train_images[0].shape
```

```
(28, 28, 1)
```

```python
def build_model(hp):
    model = Sequential([
        Conv2D(
        filters = hp.Int('conv_1_filter',min_value=32,max_value=128,step=16),
        kernel_size=hp.Choice('conv_1_kernel',values = [3,5]),
        activation='relu',
        input_shape = (28,28,1)
        ),

        Conv2D(
        filters = hp.Int('conv_2_filter',min_value=32,max_value=64,step=16),
        kernel_size=hp.Choice('conv_2_kernel',values = [3,5]),
        activation='relu'
        ),

        Flatten(),
        Dense(
        units = hp.Int('dense_1_units',min_value=32,max_value=128,step=16),
        activation='relu'
        ),
        Dense(10,activation = 'softmax')
    ])

    model.compile(optimizer=Adam(hp.Choice('learning_rate',values=[1e-2,1e-3])),
                loss = 'sparse_categorical_crossentropy',
                metrics=['accuracy'])

    return model
```

```python
from kerastuner import RandomSearch
from kerastuner import HyperParameters
```

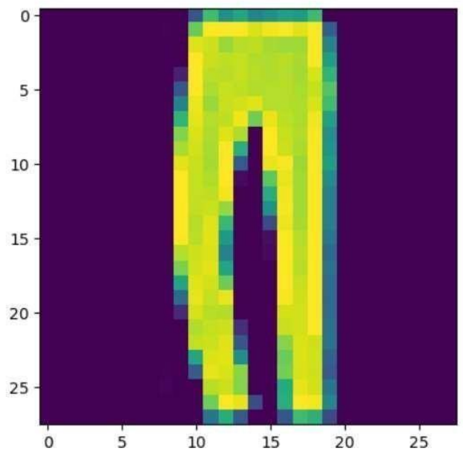
```python
tuner_search=RandomSearch(build_model,objective="val_accuracy",max_trials=5,directory='output',project_name="mnist_fashinor"
```

```
Reloading Tuner from output\mnist_fashinor\tuner0.json
```

```python
tuner_search.search(train_images,train_labels,epochs=2,validation_split=0.1)
```

```python
class_names=["T_shirt/top","Trouser","Pullover","Dress","Coat","Sandel","shirt","sneaker","Bag","Ankle Boot"]
predicted=class_names[np.argmax(prediction[2])]
print("Prediction Label: {}".format(predicted))
actual = class_names[test_labels[2]]
print("Prediction Label: {}".format(actual))

plt.imshow(test_images[2])
plt.show()
```

```
Prediction Label: Sandel
Prediction Label: Trouser
```



RESULT:

Thus the Keras tuner was successfully implemented on CNN model for hyper parameter.

Ex.No: 8	RECURRENT NEURAL NETWORK ON IMAGE SEGMENTATION TASK
Date:16.09.2024	

AIM:

To write a Program for Recurrent neural network for stock price prediction.

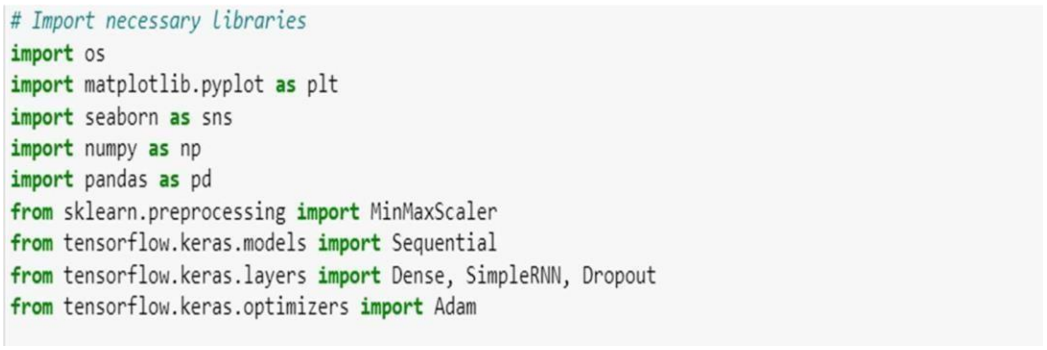
THEORY:

A recurrent neural network (RNN) is a class of artificial neural network where connections between nodes form a direct graph along a sequence. This allows it to exhibit temporal dynamic behavior for a time sequence. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to price prediction but also handwriting and speech recognition.

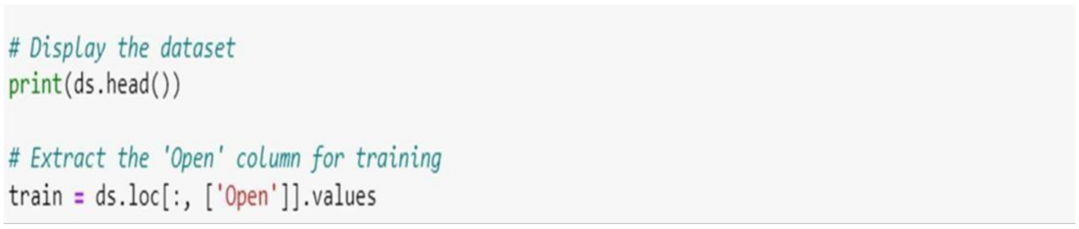
PROCEDURE:

1. Loading Data
2. Spliting Data as Train and Validation
3. Creating Train Dataset from Train split
4. Normalization/Feature Scaling
5. Creating X_train and y_train from Train data
6. Creating simple RNN model
7. Evaluating model

PROGRAM:

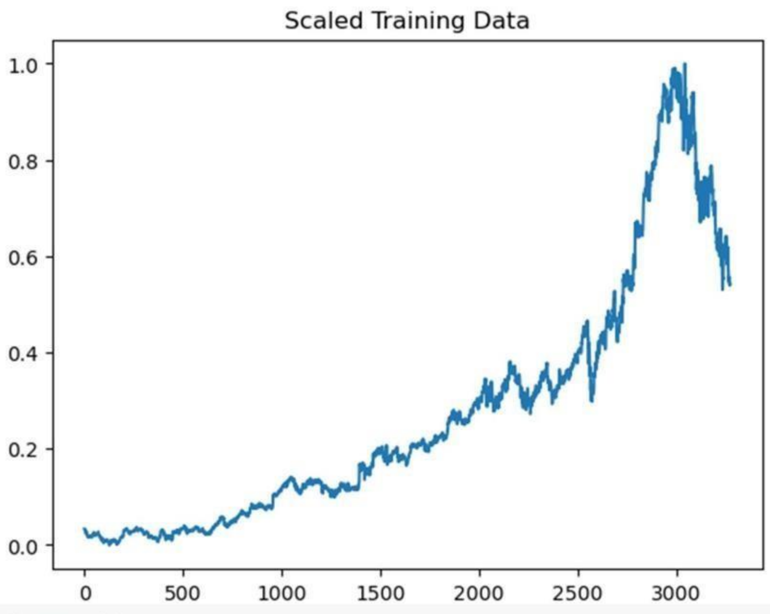


```
# Import necessary libraries
import os
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, SimpleRNN, Dropout
from tensorflow.keras.optimizers import Adam
```



```
# Display the dataset
print(ds.head())

# Extract the 'Open' column for training
train = ds.loc[:, ['Open']].values
```

```python
# Plot the scaled data
plt.plot(train_scaled)
plt.title('Scaled Training Data')
plt.show()
```

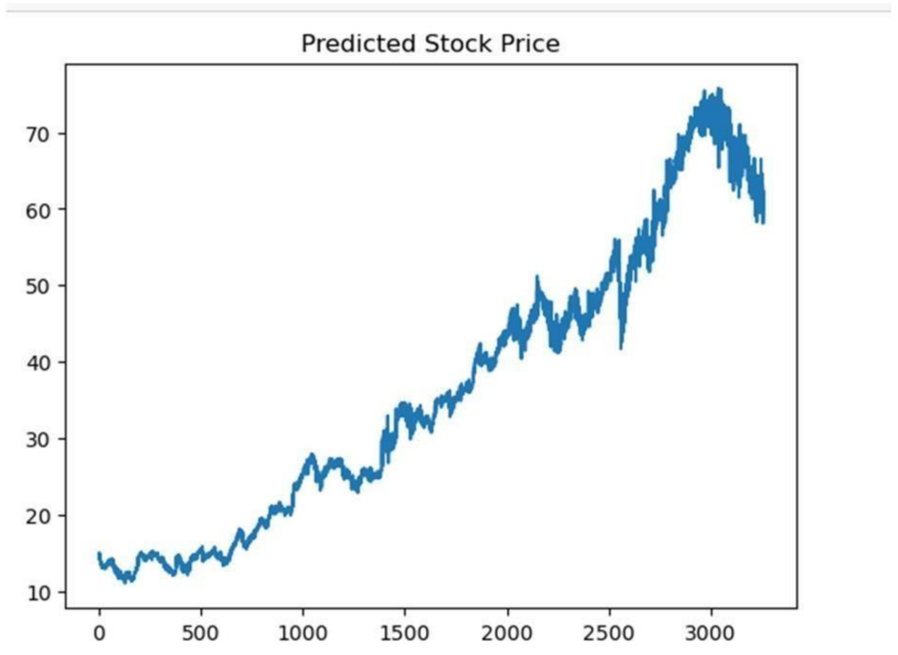


```python
# Building the RNN model
regressor = Sequential()

# First RNN layer with Dropout
regressor.add(SimpleRNN(units=50, activation='tanh', return_sequences=True, input_shape=(x_train.shape[1], 1)))
regressor.add(Dropout(0.2))

# Second RNN layer with Dropout
regressor.add(SimpleRNN(units=50, activation='tanh', return_sequences=True))
regressor.add(Dropout(0.2))

# Third RNN layer without return_sequences (since it's the last RNN)
regressor.add(SimpleRNN(units=30))
regressor.add(Dropout(0.2))

# Output layer for regression
regressor.add(Dense(units=1))

# Compile the model
regressor.compile(optimizer='adam', loss='mean_squared_error', metrics=['mean_absolute_error'])

# Print model summary
regressor.summary()

# Train the model
regressor.fit(x_train, y_train, epochs=10, batch_size=20, verbose=True)

# Prepare the test data for predictions
inputs = np.concatenate((train_data, test_data), axis=0)
timestep = 10
```

```python
x_test = []
for i in range(timesteps,200):
    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)
```

```
4/4 [==============================] - 0s 11ms/step
```

```python
# Inverse transform the predicted stock prices to the original scale
predicted_stock_price = scaler.inverse_transform(predicted_stock_price)

# Plot the predicted stock price
plt.plot(predicted_stock_price, label='Predicted Stock Price')
plt.title('Predicted Stock Price')
plt.show()
```



RESULT:

Thus the model is build using recurrent neural network and the Result is predicted.

Ex.No: 9	GATED RECURRENT NEURAL NETWORK ON IMAGESEGMENTATION TASK
Date:23.09.2024	

AIM:

To write a program for Gated Recurrent neural network to perform Image segmentation.

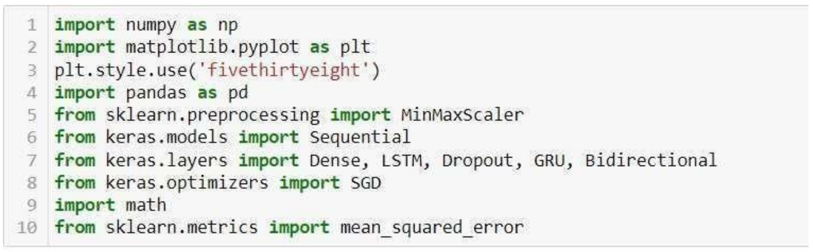
THEORY:

Image segmentation is a key task in computer vision and image processing with important applications such as scene understanding, medical image analysis, robotic perception, video surveillance, augmented reality, and image compression, among others, and numerous segmentation algorithms are found in the literature.Semantic and instance segmentation, including convolutional pixel-labelling networks, encoder-decoder architectures, multiscale and pyramid-based approaches, recurrent networks, visual attention models, and generative models in adversarial settings.

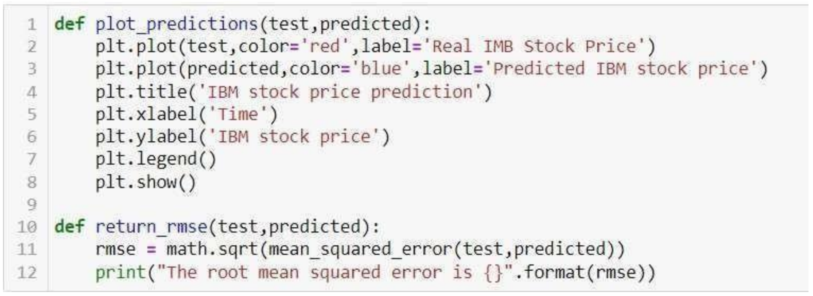
PROCEDURE:

1. Import necessary libraries.
2. Load the dataset.
3. Train the model.
4. predict the model.
5. Evaluate the model.

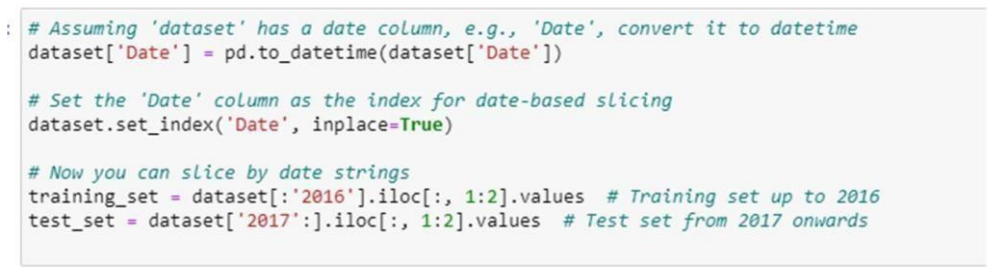
PROGRAM:



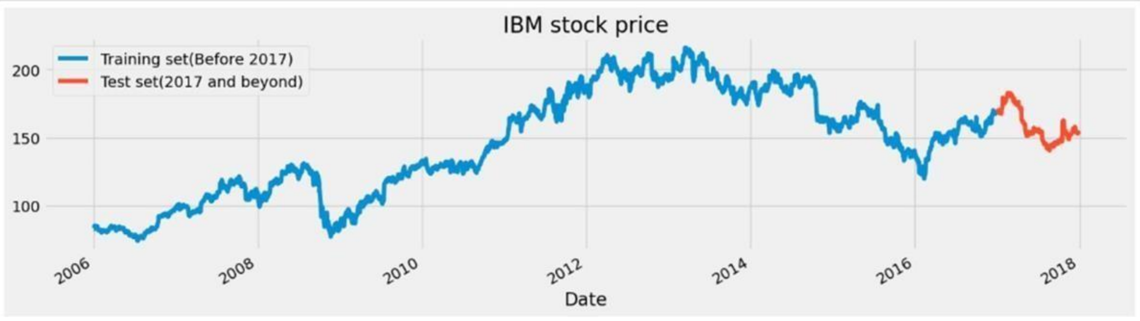
```python
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, GRU, Bidirectional
from keras.optimizers import SGD
import math
from sklearn.metrics import mean_squared_error
```

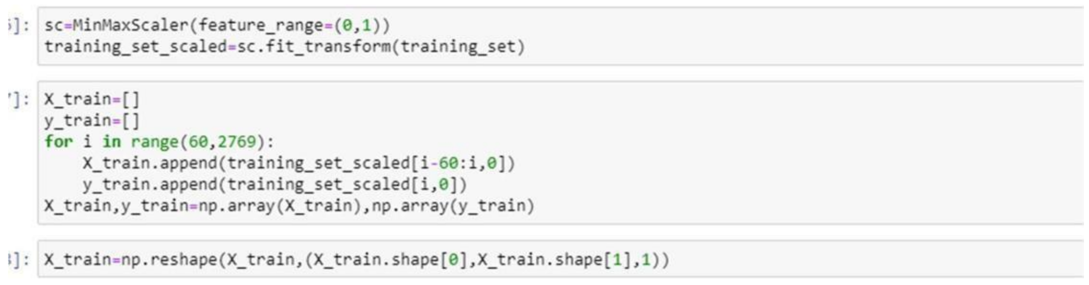


```python
def plot_predictions(test,predicted):
    plt.plot(test,color='red',label='Real IMB Stock Price')
    plt.plot(predicted,color='blue',label='Predicted IBM stock price')
    plt.title('IBM stock price prediction')
    plt.xlabel('Time')
    plt.ylabel('IBM stock price')
    plt.legend()
    plt.show()

def return_rmse(test,predicted):
    rmse = math.sqrt(mean_squared_error(test,predicted))
    print("The root mean squared error is {}".format(rmse))
```



```python
# Assuming 'dataset' has a date column, e.g., 'Date', convert it to datetime
dataset['Date'] = pd.to_datetime(dataset['Date'])

# Set the 'Date' column as the index for date-based slicing
dataset.set_index('Date', inplace=True)

# Now you can slice by date strings
training_set = dataset[:'2016'].iloc[:, 1:2].values  # Training set up to 2016
test_set = dataset['2017':].iloc[:, 1:2].values  # Test set from 2017 onwards
```

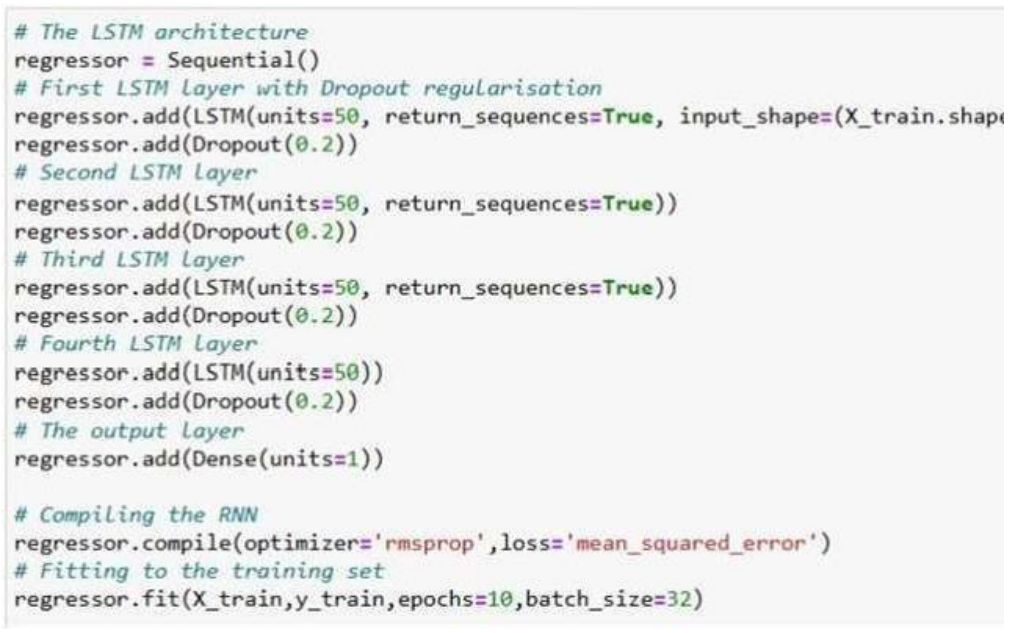




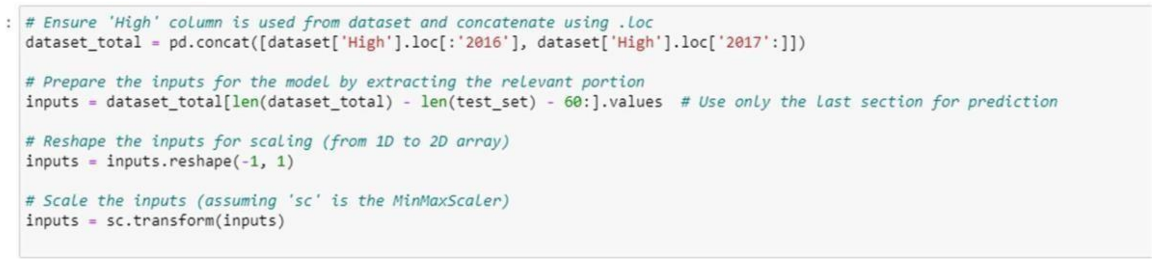
```python
sc=MinMaxScaler(feature_range=(0,1))
training_set_scaled=sc.fit_transform(training_set)
```

```python
X_train=[]
y_train=[]
for i in range(60,2769):
    X_train.append(training_set_scaled[i-60:i,0])
    y_train.append(training_set_scaled[i,0])
X_train,y_train=np.array(X_train),np.array(y_train)
```

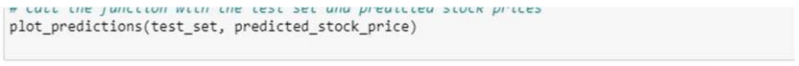
```python
X_train=np.reshape(X_train,(X_train.shape[0],X_train.shape[1],1))
```



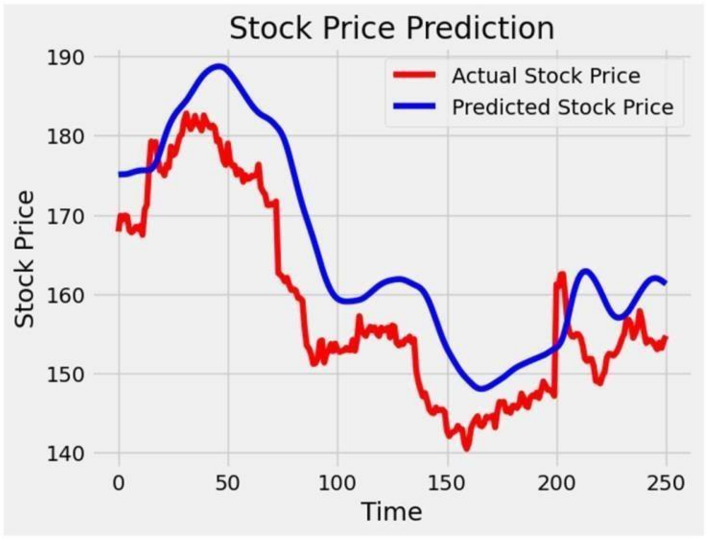
```python
# The LSTM architecture
regressor = Sequential()
# First LSTM layer with Dropout regularisation
regressor.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape
regressor.add(Dropout(0.2))
# Second LSTM layer
regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
# Third LSTM layer
regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
# Fourth LSTM layer
regressor.add(LSTM(units=50))
regressor.add(Dropout(0.2))
# The output layer
regressor.add(Dense(units=1))

# Compiling the RNN
regressor.compile(optimizer='rmsprop',loss='mean_squared_error')
# Fitting to the training set
regressor.fit(X_train,y_train,epochs=10,batch_size=32)
```

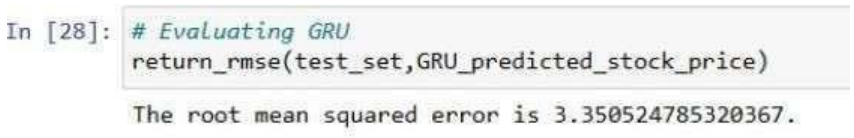


```
# Ensure 'High' column is used from dataset and concatenate using .loc
dataset_total = pd.concat([dataset['High'].loc[:'2016'], dataset['High'].loc['2017':]])

# Prepare the inputs for the model by extracting the relevant portion
inputs = dataset_total[len(dataset_total) - len(test_set) - 60:].values  # Use only the last section for prediction

# Reshape the inputs for scaling (from 1D to 2D array)
inputs = inputs.reshape(-1, 1)

# Scale the inputs (assuming 'sc' is the MinMaxScaler)
inputs = sc.transform(inputs)
```



```
# call the function with the test set and predicted stock prices
plot_predictions(test_set, predicted_stock_price)
```





```
In [28]: # Evaluating GRU
         return_rmse(test_set,GRU_predicted_stock_price)

         The root mean squared error is 3.350524785320367.
```

RESULT:

The image segmentation is done using recurrent neural network.

Ex.No: 10	LSTM ON STOCK PRICE PREDICTION
Date:30.09.2024	

AIM:

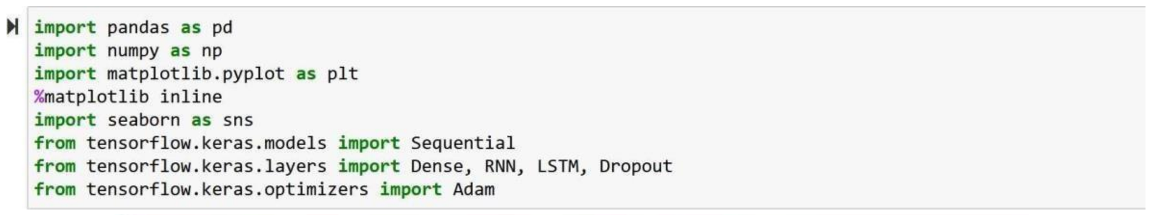
Use the LSTM on Stock Price Prediction.

DEFINITION OF LSTM:

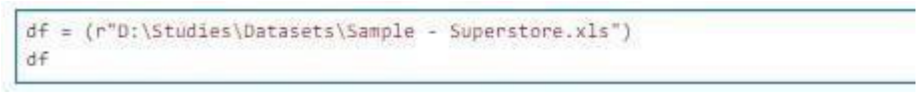
Long Short Term Memory is a kind of recurrent neural network. In RNN output from the last step is fed as input in the current step. LSTM was designed by Hochreiter & Schmidhuber. It tackled the problem of long-term dependencies of RNN in which the RNN cannot predict the word stored in the long-term memory but can give more accurate predictions from the recent information. As the gap length increases RNN does not give an efficient performance. LSTM can by default retain the information for a long period of time. It is used for processing, predicting, and classifying on the basis of time-series data.

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) that is specifically designed to handle sequential data, such as time series, speech, and text. LSTM networks are capable of learning long-term dependencies in sequential data, which makes them well suited for tasks such as language translation, speech recognition, and time series forecasting.

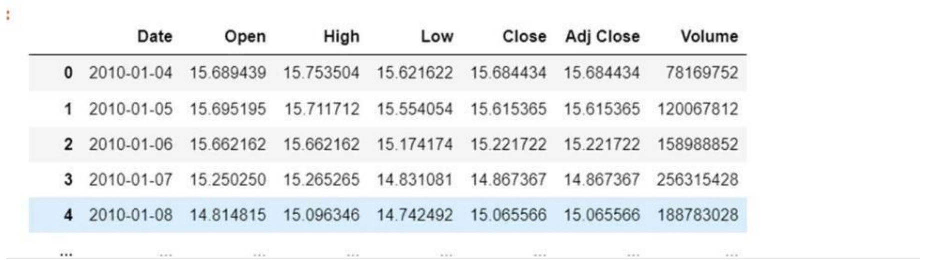
PROGRAM:



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, RNN, LSTM, Dropout
from tensorflow.keras.optimizers import Adam
```



```
df = (r"D:\Studies\Datasets\Sample - Superstore.xls")
df
```



```
:
         Date       Open       High        Low      Close  Adj Close     Volume
0  2010-01-04  15.689439  15.753504  15.621622  15.684434  15.684434   78169752
1  2010-01-05  15.695195  15.711712  15.554054  15.615365  15.615365  120067812
2  2010-01-06  15.662162  15.662162  15.174174  15.221722  15.221722  158988852
3  2010-01-07  15.250250  15.265265  14.831081  14.867367  14.867367  256315428
4  2010-01-08  14.814815  15.096346  14.742492  15.065566  15.065566  188783028
...       ...        ...        ...        ...        ...        ...        ...
```

```
def  create_dataset(dataset, time_step=1):
    datax, datay = [], []
    for i in range(len(dataset)- time_step-1):
        datax.append(dataset[i:(i+time_step), 0])
        datay.append(dataset[i + time_step, 0])
    return np.array(datax), np.array(datay)
```

```
time_step = 100
x_train, y_train = create_dataset(train_data, time_step)
x_test, y_test = create_dataset(test_data, time_step)
```

```
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1], 1)
x_test = x_test.reshape(x_test.shape[0], x_test.shape[1], 1)
```

```
model = Sequential()

model.add(LSTM(50, input_shape=(x_train.shape[1], 1), return_sequences = True))
model.add(Dropout(0.2))

model.add(LSTM(50, return_sequences = True))
model.add(Dropout(0.2))

model.add(LSTM(30))
```

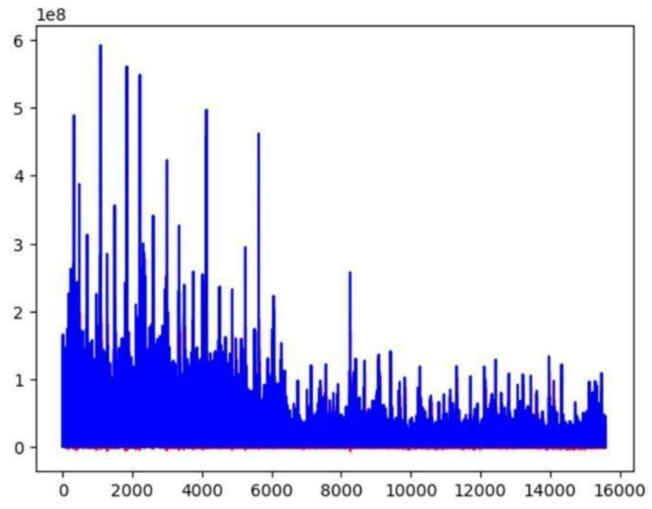
```
model.fit(x_train, y_train,
         validation_data = (x_test, y_test),
         epochs=10,
         batch_size=20,
         verbose=True)
```

```
Epoch 1/10
WARNING:tensorflow:From C:\Users\Vignesh\anaconda3\lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.Ra
TensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

781/781 [==============================] - 150s 178ms/step - loss: 0.0025 - val_loss: 4.1401e-04
Epoch 2/10
781/781 [==============================] - 135s 173ms/step - loss: 0.0013 - val_loss: 1.7914e-04
Epoch 3/10
781/781 [==============================] - 150s 192ms/step - loss: 7.5860e-04 - val_loss: 2.1776e-04
Epoch 4/10
```

```
x_train_predict = model.predict(x_train)
x_train_predict = mm.inverse_transform(x_train_predict)
```

```
488/488 [==============================] - 33s 64ms/step
```

```
y_train = mm.inverse_transform(y_train.reshape(-1, 1))
```



```
import math
from sklearn.metrics import mean_squared_error
```

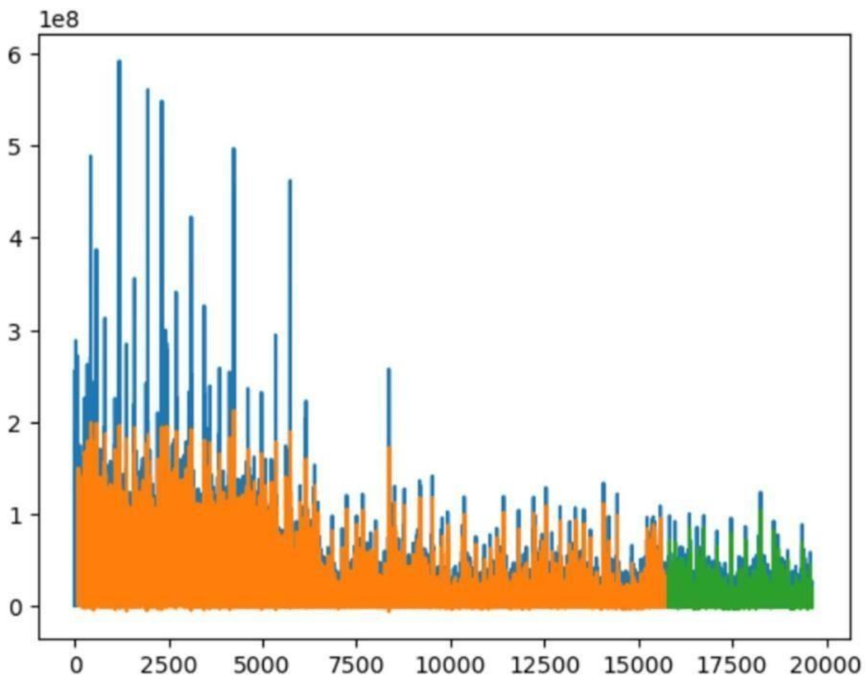
```
math.sqrt(mean_squared_error(y_train, train_predict))
```

```
13971566.983567407
```

```
math.sqrt(mean_squared_error(y_test, test_predict))
```

```
6391499.812839582
```

```
look_back  = 100

trainpredictplot = np.empty_like(data)
trainpredictplot[:, :] = np.nan
trainpredictplot[look_back:len(train_predict)+look_back, :] = train_predict

testpredictplot = np.empty_like(data)
testpredictplot[:, :] = np.nan
testpredictplot[len(train_predict)+(look_back*2)+1:len(data)-1, :] = test_predict

plt.plot(mm.inverse_transform(data))
plt.plot(trainpredictplot)
plt.plot(testpredictplot)
plt.show()
```



RESULT:

The LSTM model was successfully implemented on stock price prediction.

Ex.No: 11	LSTM ON IMAGE SEGMENTATION
Date:07.10.2024	

AIM:

To write a Program for Image segmentation using LSTM.

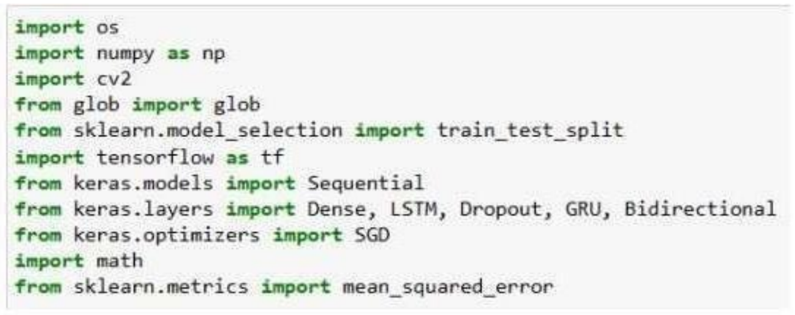
THEORY:

LSTM is well-suited to classify, process and predict time series given time lags of unknown duration. It trains the model by using back-propagation. In an LSTM network, three gates are present: Input gate — discover which value from input should be used to modify the memory.

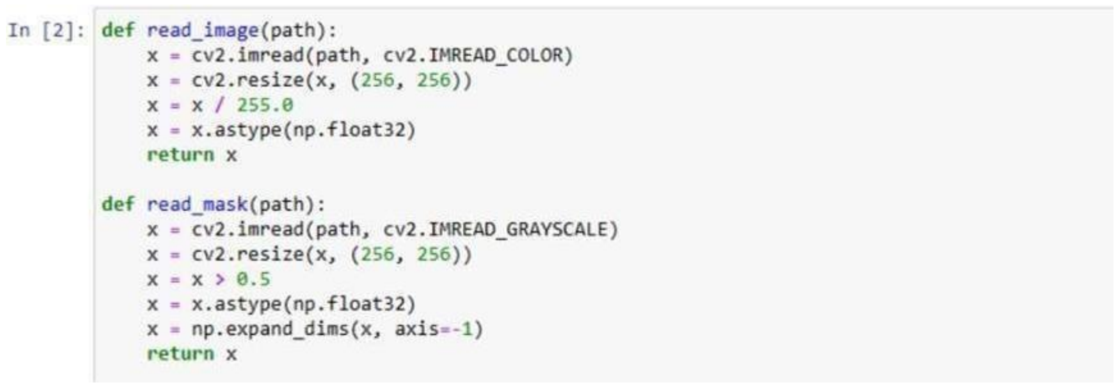
PROCEDURE:

- Import Libraries .
- Loading Image .
- Converting to Grayscale.
- Converting to binary inverted image.
- Segmenting image.

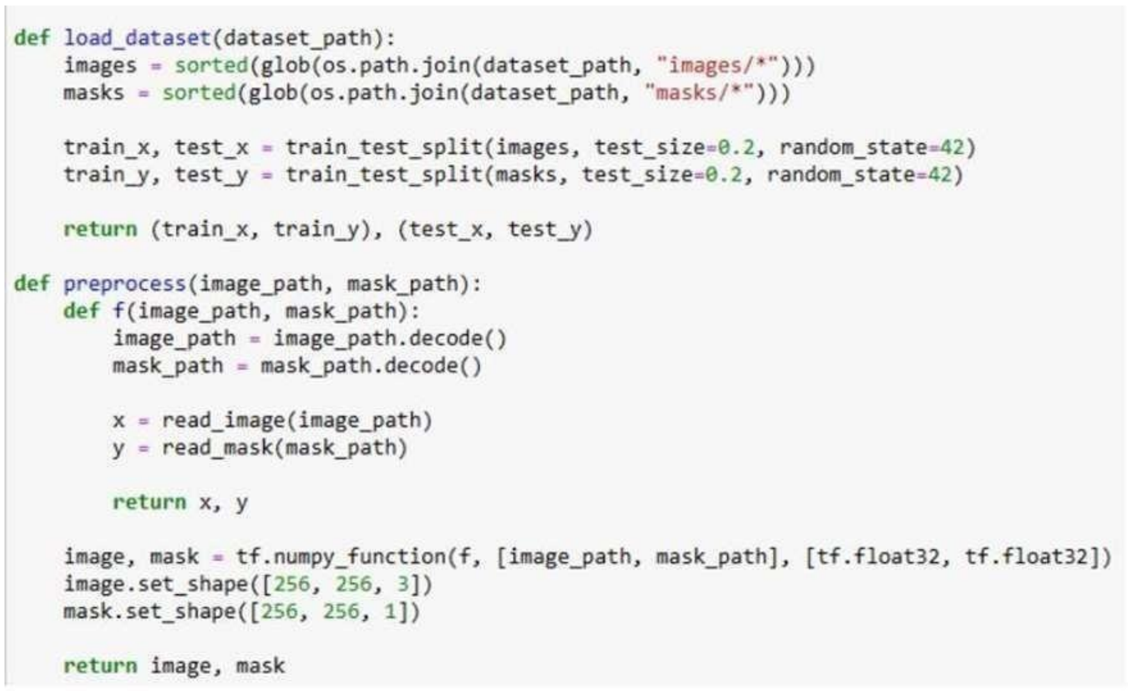
PROGRAM:



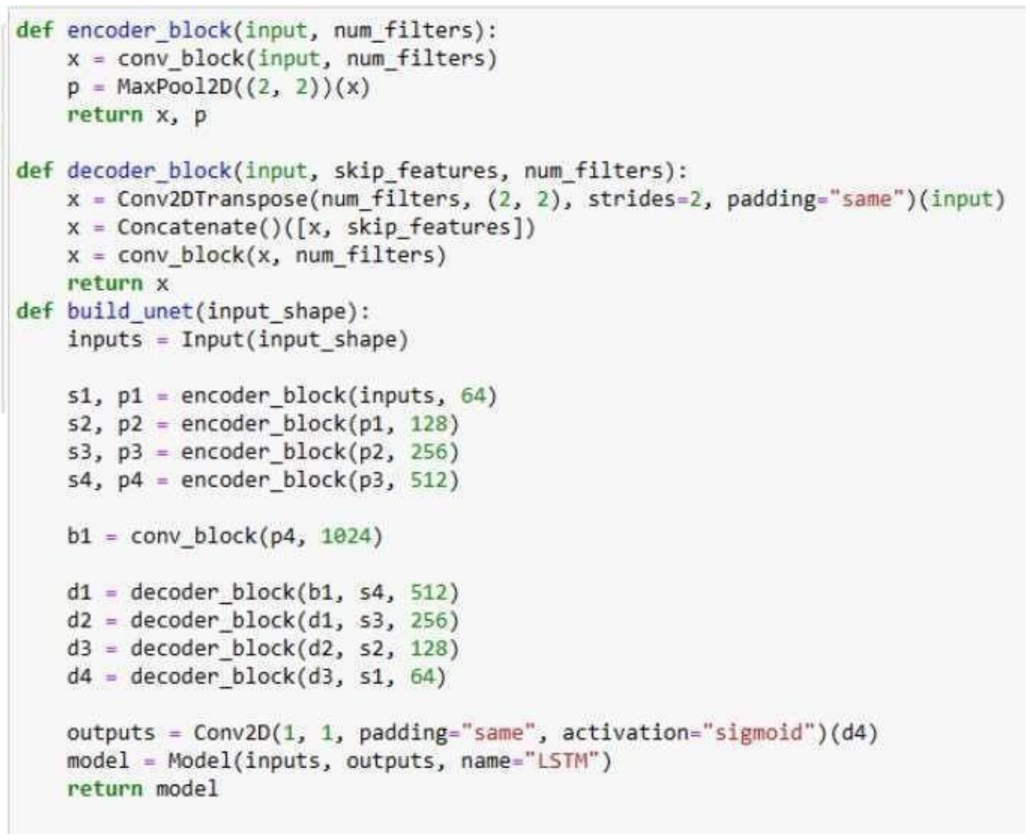
```
import os
import numpy as np
import cv2
from glob import glob
from sklearn.model_selection import train_test_split
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, GRU, Bidirectional
from keras.optimizers import SGD
import math
from sklearn.metrics import mean_squared_error
```



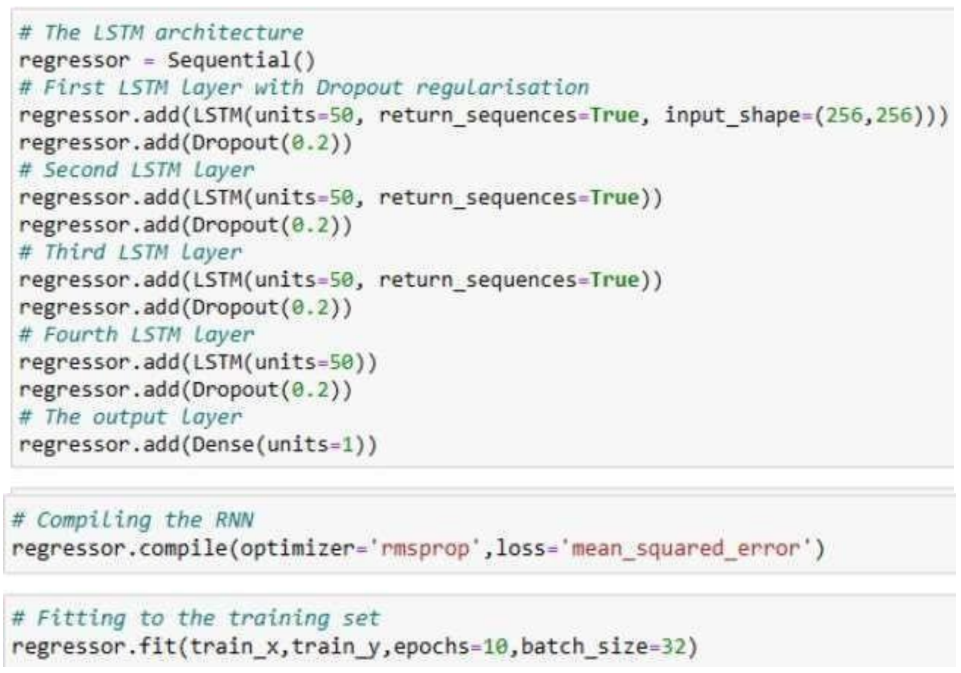
```
In [2]: def read_image(path):
            x = cv2.imread(path, cv2.IMREAD_COLOR)
            x = cv2.resize(x, (256, 256))
            x = x / 255.0
            x = x.astype(np.float32)
            return x

        def read_mask(path):
            x = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
            x = cv2.resize(x, (256, 256))
            x = x > 0.5
            x = x.astype(np.float32)
            x = np.expand_dims(x, axis=-1)
            return x
```



```
def load_dataset(dataset_path):
    images = sorted(glob(os.path.join(dataset_path, "images/*")))
    masks = sorted(glob(os.path.join(dataset_path, "masks/*")))

    train_x, test_x = train_test_split(images, test_size=0.2, random_state=42)
    train_y, test_y = train_test_split(masks, test_size=0.2, random_state=42)

    return (train_x, train_y), (test_x, test_y)

def preprocess(image_path, mask_path):
    def f(image_path, mask_path):
        image_path = image_path.decode()
        mask_path = mask_path.decode()

        x = read_image(image_path)
        y = read_mask(mask_path)

        return x, y

    image, mask = tf.numpy_function(f, [image_path, mask_path], [tf.float32, tf.float32])
    image.set_shape([256, 256, 3])
    mask.set_shape([256, 256, 1])

    return image, mask
```



```
def encoder_block(input, num_filters):
    x = conv_block(input, num_filters)
    p = MaxPool2D((2, 2))(x)
    return x, p

def decoder_block(input, skip_features, num_filters):
    x = Conv2DTranspose(num_filters, (2, 2), strides=2, padding="same")(input)
    x = Concatenate()([x, skip_features])
    x = conv_block(x, num_filters)
    return x
def build_unet(input_shape):
    inputs = Input(input_shape)

    s1, p1 = encoder_block(inputs, 64)
    s2, p2 = encoder_block(p1, 128)
    s3, p3 = encoder_block(p2, 256)
    s4, p4 = encoder_block(p3, 512)

    b1 = conv_block(p4, 1024)

    d1 = decoder_block(b1, s4, 512)
    d2 = decoder_block(d1, s3, 256)
    d3 = decoder_block(d2, s2, 128)
    d4 = decoder_block(d3, s1, 64)

    outputs = Conv2D(1, 1, padding="same", activation="sigmoid")(d4)
    model = Model(inputs, outputs, name="LSTM")
    return model
```


```
# The LSTM architecture
regressor = Sequential()
# First LSTM layer with Dropout regularisation
regressor.add(LSTM(units=50, return_sequences=True, input_shape=(256,256)))
regressor.add(Dropout(0.2))
# Second LSTM layer
regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
# Third LSTM layer
regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
# Fourth LSTM layer
regressor.add(LSTM(units=50))
regressor.add(Dropout(0.2))
# The output layer
regressor.add(Dense(units=1))
```

```
# Compiling the RNN
regressor.compile(optimizer='rmsprop',loss='mean_squared_error')
```

```
# Fitting to the training set
regressor.fit(train_x,train_y,epochs=10,batch_size=32)
```

RESULT:

Thus, the model predicted successfully.

Ex.No: 12	**ANOMALY DETECTION USING AUTOENCODERS**
Date:09.10.2024	

AIM:

To write a program For Anomaly detection using Auto- encoders.

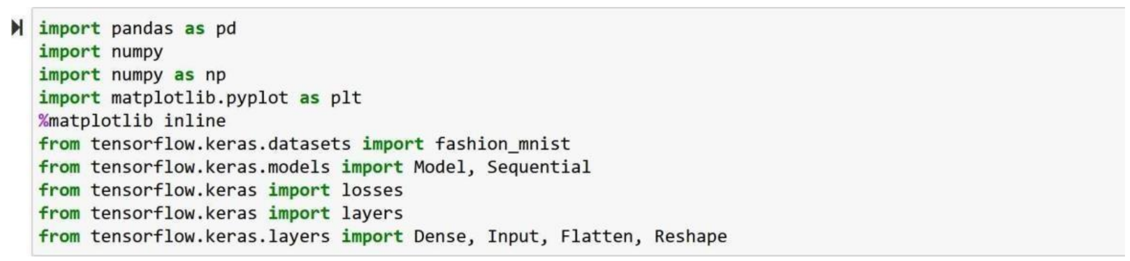
THEORY:

An autoencoder is a type of artificial neural network used to learn efficient coding's of unlabelled data (unsupervised learning). The encoding is validated and refined by attempting to regenerate the input from the encoding. The autoencoder learns a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore in significant data.

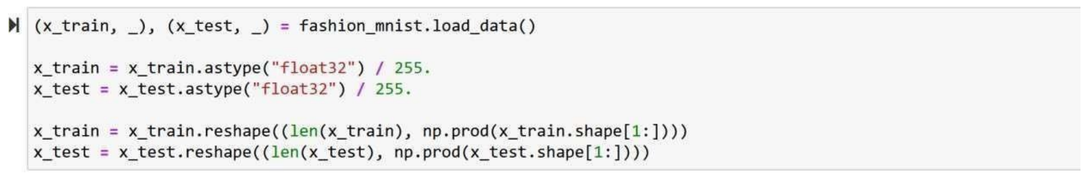
PROCEDURE:

1. import the necessary library
2. Train/Validate/Test Split
3. Training the auto-encoders
4. Model fitting
5. Predicting the model

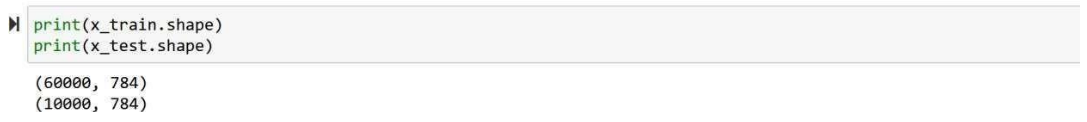
PROGRAM:



```
import pandas as pd
import numpy
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras import losses
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense, Input, Flatten, Reshape
```

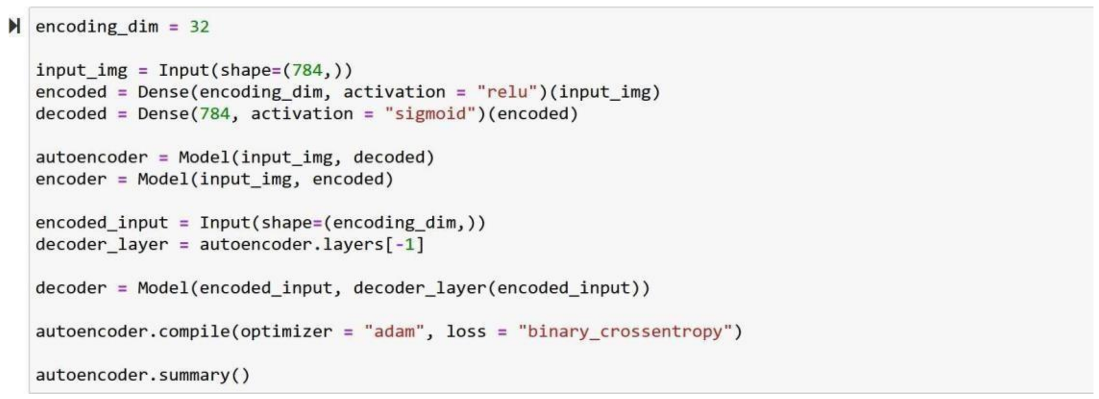


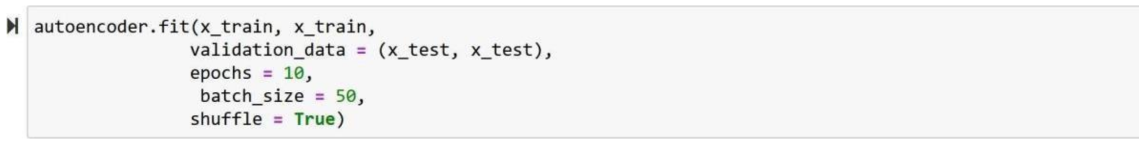
```
(x_train, _), (x_test, _) = fashion_mnist.load_data()

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:])))
```

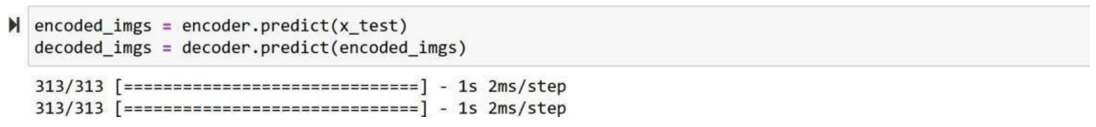


```
print(x_train.shape)
print(x_test.shape)
```

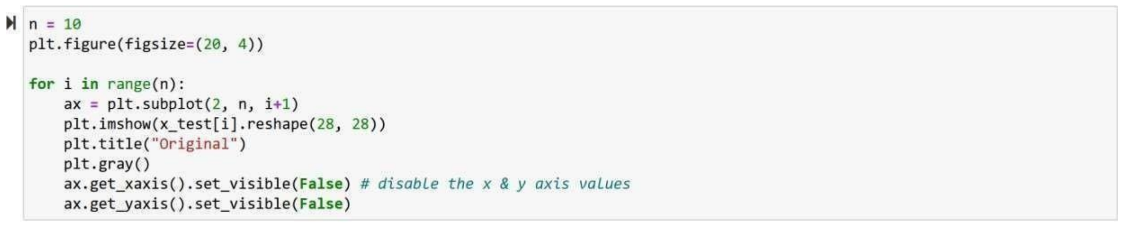
```
(60000, 784)
(10000, 784)
```

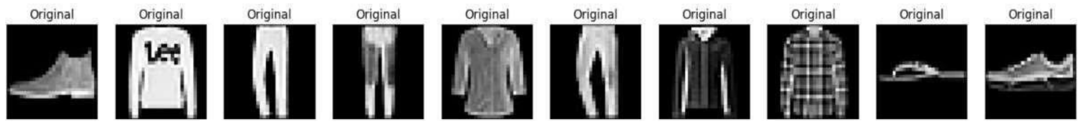

```
encoding_dim = 32

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 = "adam", loss = "binary_crossentropy")

autoencoder.summary()
```

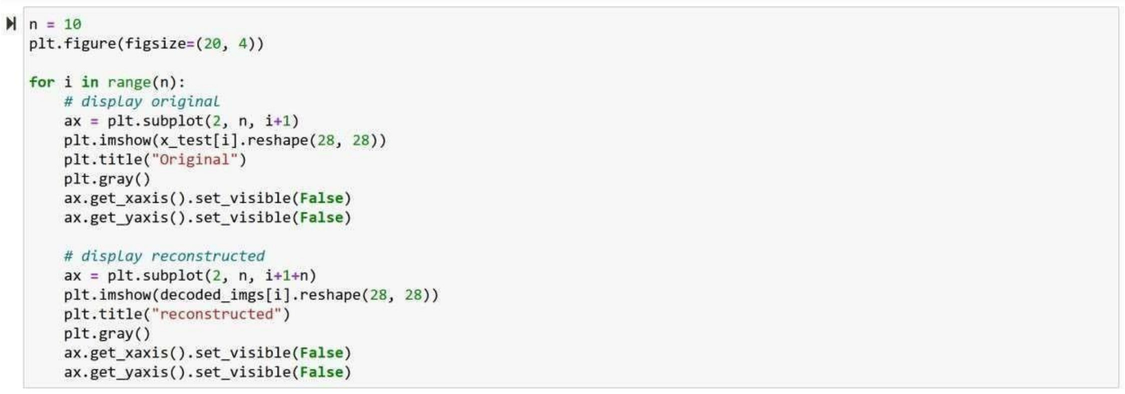

```
autoencoder.fit(x_train, x_train,
                validation_data = (x_test, x_test),
                epochs = 10,
                 batch_size = 50,
                shuffle = True)
```


```
encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)
```

```
313/313 [==============================] - 1s 2ms/step
313/313 [==============================] - 1s 2ms/step
```


```
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.title("Original")
    plt.gray()
    ax.get_xaxis().set_visible(False) # disable the x & y axis values
    ax.get_yaxis().set_visible(False)
```




```
n = 10
plt.figure(figsize=(20, 4))

for i in range(n):
    # display original
    ax = plt.subplot(2, n, i+1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.title("Original")
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display reconstructed
    ax = plt.subplot(2, n, i+1+n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.title("reconstructed")
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
```

RESULT:

The input image was reconstructed successfully by Autoencoders.