EX.NO : 01	INTRODUCTION TO DL AND FRAMEWORK
DATE : 10/07/2023	

AIM :

To Create an Introduction to Deep Learning and its Framework .

DESCRIBE :

Deep learning (DL) frameworks are building blocks for designing, training, and validating deep neural networks through a high-level programming interface. Popular open source DL frame works are Tensorflow, Keras, Pytorch.

PROCEDURE :

- Install Tensorflow DL framework.
- Install Keras DL framework.
- Install Pytorch DL framework.

CODES :

1) TENSORFLOW :

Tensorflow is an open-source software library. Tensorflow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well! Let us first try to understand what the word tensorflow actually mean! Tensorflow is basically a software library for numerical computation using data flow graphs.

INSTALL TENSORFLOW :

```
In [1]: !pip install tensorflow

Requirement already satisfied: tensorflow in c:\users\abinashkumar\anaconda3\lib\site-packages (2.13.0)
Requirement already satisfied: tensorflow-intel==2.13.0 in c:\users\abinashkumar\anaconda3\lib\site-packages (from tensorflow)
(2.13.0)
Requirement already satisfied: h5py>=2.9.0 in c:\users\abinashkumar\anaconda3\lib\site-packages (from tensorflow-intel==2.13.0-
>tensorflow) (3.7.0)
Requirement already satisfied: tensorflow-estimator<2.14,>=2.13.0 in c:\users\abinashkumar\anaconda3\lib\site-packages (from te
nsorflow-intel==2.13.0->tensorflow) (2.13.0)
Requirement already satisfied: packaging in c:\users\abinashkumar\anaconda3\lib\site-packages (from tensorflow-intel==2.13.0->t
ensorflow) (21.3)
Requirement already satisfied: google-pasta>=0.1.1 in c:\users\abinashkumar\anaconda3\lib\site-packages (from tensorflow-intel=
=2.13.0->tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in c:\users\abinashkumar\anaconda3\lib\site-packages (from tensorflow-intel==2.
13.0->tensorflow) (16.0.6)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in c:\users\abinashkumar\anaconda3\lib\site-packages (from
tensorflow-intel==2.13.0->tensorflow) (0.31.0)
Requirement already satisfied: numpy<=1.24.3,>=1.22 in c:\users\abinashkumar\anaconda3\lib\site-packages (from tensorflow-intel
==2.13.0->tensorflow) (1.24.3)
Requirement already satisfied: astunparse>=1.6.0 in c:\users\abinashkumar\anaconda3\lib\site-packages (from tensorflow-intel==
2.13.0->tensorflow) (1.6.3)
```

IMPORT TENSORFLOW :

```
In [2]: import tensorflow as tf
```

2) KERAS :

Keras is an open-source high-level Neural Network library, which is written in Python is capable enough to run on Theano, TensorFlow, or CNTK. It was developed by one of the Google engineers, Francois Chollet. It is made user-friendly, extensible, and modular for facilitating faster experimentation with deep neural networks. It not only supports Convolutional Networks and Recurrent Networks individually but also their combination.

It cannot handle low-level computations, so it makes use of the Backend library to resolve it. The backend library act as a high-level API wrapper for the low-level API, which lets it run on TensorFlow, CNTK, or Theano.

IMPORT KERAS AND IT'S PAKEGES :

```
In [1]: !pip install keras

        Requirement already satisfied: keras in c:\users\abinashkumar\anaconda3\lib\site-packages (2.13.1)

In [2]: import keras as ks
        from keras.layers import Dense
        from keras.models import Sequential
```

3) PYTORCH :

PyTorch is an open-source Deep Learning framework developed by Facebook. It is based on the Torch library and was designed with one primary aim – to expedite the entire process from research prototyping to production deployment. What's interesting about PyTorch is that it has a C++ frontend atop a Python interface.

While the frontend serves as the core ground for model development, the torch.distributed" backend promotes scalable distributed training and performance optimization in both research and production. This is one of the best deep learning frameworks you can use.

RESULT :

Thus the introduction to Deep learning and it's framework successfully created.

EX.NO : 02	**FEED FORWARD NETWORK ON SAMPLE DATASET**
DATE : 17/07/2023	

AIM :

To using the feed-forward neural network to work with a sample dataset.

DESCRIBE ABOUT FEED FORWARD NETWORK :

A Feed Forward Neural Network is an artificial neural network in which the connections between nodes does not form a cycle. The opposite of a feed forward neural network is a recurrent neural network, in which certain pathways are cycled. The feed forward model is the simplest form of neural network as information is only processed in one direction. While the data may pass through multiple hidden nodes, it always moves in one direction and never backwards.

ABOUT SAMPLE DATASET :

Here we using mnist inbuild dataset import from the keras deep learning framework. This dataset contain numeric values images from zero to nine around 60000 samples.

PROCEDURE :

- Import the needed libraries.
- Load the inbuild dataset from tensorflow keras.
- Split the dataset to tarin and test the model.
- Preprocess the dataset and build the model.
- Compile all the layer using compile function.
- Finally predict the model using the new data.

CODES :

Import the libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import random
import tensorflow as tf
import keras
```

Call the dataset from the kares framework

```
In [2]: mnist = tf.keras.datasets.mnist
```

Split the data

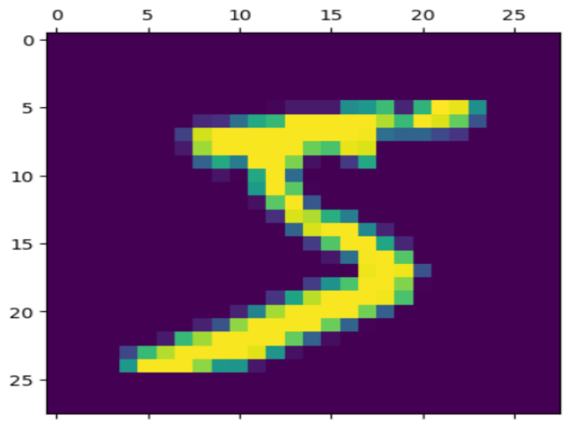
```
In [3]: (X_train,y_train),(X_test,y_test) = mnist.load_data()
```

```
In [4]: len(X_train)
        X_train.shape

Out[4]: (60000, 28, 28)
```

```
In [6]: X_train[0]

Out[6]: array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
                 18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170,
                253, 253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253, 253,
                253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,   0,
                  0,   0]
```

PLOTTING THE MNIST IMAGE :

```
In [7]: plt.matshow(X_train[0])

Out[7]: <matplotlib.image.AxesImage at 0x25392ced190>
```



Data preprocessing

In [8]:
```
x_train = X_train / 255
x_test = X_test / 255
```

In [9]:
```
x_train[0]
```

```
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
```

creating model

In [10]:
```
from keras.layers import Activation, Dense, Flatten

model = keras.Sequential([keras.layers.Flatten(input_shape = (28,28)),
                          keras.layers.Dense(128,activation = 'relu'),
                          keras.layers.Dense(10,activation = 'softmax')])
```

In [11]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 128)               100480

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 101770 (397.54 KB)
Trainable params: 101770 (397.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

compile the model

In [12]:
```
model.compile(optimizer = 'sgd',
              loss = 'sparse_categorical_crossentropy',
              metrics = ['accuracy'])
```

In [14]:
```
history = model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=4)
```

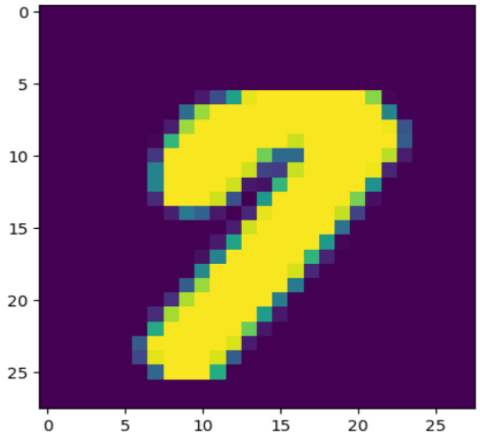
```
Epoch 1/4
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1576 - accuracy: 0.9557 - val_loss: 0.1582 - val_accuracy:
0.9555
Epoch 2/4
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1495 - accuracy: 0.9582 - val_loss: 0.1505 - val_accuracy:
0.9571
Epoch 3/4
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1421 - accuracy: 0.9603 - val_loss: 0.1442 - val_accuracy:
0.9588
Epoch 4/4
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1356 - accuracy: 0.9622 - val_loss: 0.1402 - val_accuracy:
0.9607
```

In [15]:
```
test_loss,test_accuracy= model.evaluate(x_test,y_test)
print('loss=%.3f' %test_loss)
print('Accuracy=%.3f' %test_accuracy)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.1402 - accuracy: 0.9607
loss=0.140
Accuracy=0.961
```

making prediction on new data

In [78]:
```
n = random.randint(0,9999)
plt.imshow(x_test[n])
plt.show()
```

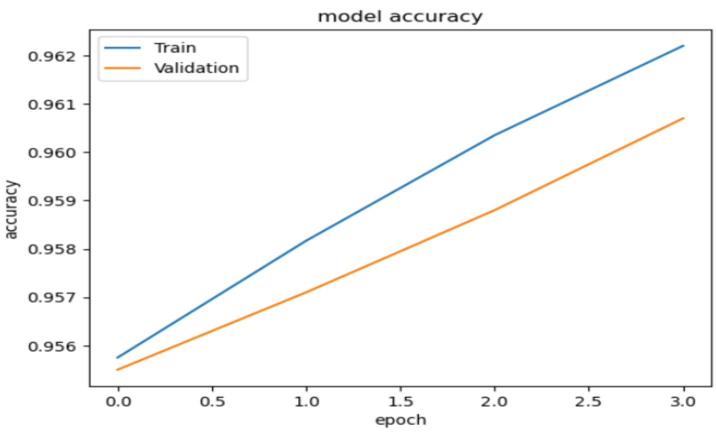


predicted

In [79]:
```
predicted_value = model.predict(x_test)
print('hand written number in the image is = %d' %np.argmax(predicted_value[n]))
```

```
313/313 [==============================] - 1s 2ms/step
hand written number in the image is = 7
```

In [21]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Train','Validation'],loc='upper left')
plt.show()
```



RESULT :

Finally using the feed-forward neural network we work with mnist sample dataset successfully completed.

EX.NO : 03	MULTI LAYER PERCEPTRON (MLP) ON REAL-TIME DATASET
DATE : 24/07/2023	

AIM :

To create Multi Layer Perceptron neural network on real time datasets.

DESCRIBE :

The perceptron is very useful for classifying data sets that are linearly separable. They encounter serious limitations with data sets that do not conform to this pattern as discovered with the XOR problem. The XOR problem shows that for any classification of four points that there exists a set that are not linearly separable.

The Multi Layer Perceptron (MLPs) breaks this restriction and classifies datasets which are not linearly separable. They do this by using a more robust and complex architecture to learn regression and classification models for difficult datasets.

ABOUT DATASET :

Here we using mnist inbuild dataset import from the keras deep learning framework. This dataset contain numeric values images from zero to nine around 60000 samples.

PROCEDURE :

- Import the needed libraries.
- Load the inbuild dataset from tensorflow keras.
- Split the dataset to tarin and test the model.
- Preprocess the dataset and build the model.
- Compile all the layer using compile function.
- Finally predict the model using the new data.

CODES :

importing modules

In [1]:
```
import tensorflow as tf
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.neural_network import MLPRegressor
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Activation
import matplotlib.pyplot as plt
```

Data split

In [2]:
```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

Cast the records into float values

In [3]:
```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
```

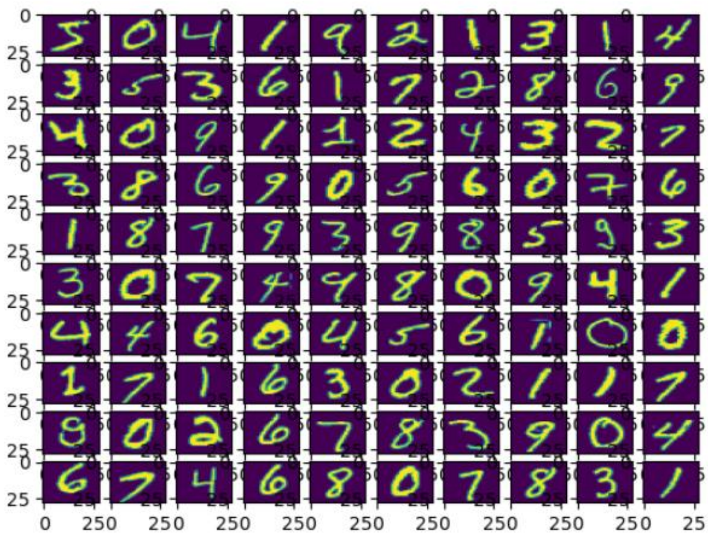
normalize image pixel values by dividing

In [4]:
```
gray_scale = 255
x_train /= gray_scale
x_test /= gray_scale
```

In [5]:
```
print("Feature matrix:", x_train.shape)
print("Target matrix:", x_test.shape)
print("Feature matrix:", y_train.shape)
print("Target matrix:", y_test.shape)
```

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

In [6]:
```
fig, ax = plt.subplots(10, 10)
k = 0
for i in range(10):
    for j in range(10):
        ax[i][j].imshow(x_train[k].reshape(28, 28),
                        aspect='auto')
        k += 1
plt.show()
```



Model building

In [7]:
```
model = Sequential([Flatten(input_shape=(28, 28)),Dense(256, activation='sigmoid'),
                    Dense(128, activation='sigmoid'),Dense(10, activation='sigmoid')])
```

Model Compile

In [8]:
```
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

In [9]:
```
model.fit(x_train, y_train, epochs=4,batch_size=2000,validation_split=0.2)
```

```
Epoch 1/4
24/24 [==============================] - 3s 102ms/step - loss: 2.1472 - accuracy: 0.3800 - val_loss: 1.8295 - val_accuracy: 0.6
954
Epoch 2/4
24/24 [==============================] - 2s 77ms/step - loss: 1.5190 - accuracy: 0.7017 - val_loss: 1.1697 - val_accuracy: 0.79
33
Epoch 3/4
24/24 [==============================] - 2s 83ms/step - loss: 0.9777 - accuracy: 0.7980 - val_loss: 0.7647 - val_accuracy: 0.84
83
Epoch 4/4
24/24 [==============================] - 2s 89ms/step - loss: 0.6822 - accuracy: 0.8509 - val_loss: 0.5577 - val_accuracy: 0.88
38
```

Out[9]:
```
<keras.src.callbacks.History at 0x191af5be700>
```

In [10]:
```
results = model.evaluate(x_test, y_test, verbose = 0)
print('test loss, test acc:', results)
```

```
test loss, test acc: [0.5654992461204529, 0.8794999718666077]
```

RESULT :

Thus using Multi Layer perceptron network we successfully completed with 0.8794 accuracy.

EX.NO : 04	CONVOLUTION NEURAL NETWORK ON BINARY CLASSIFICATION TASK: CAT AND DOG DATASET
DATE : 07/08/2023	

AIM :

To build convolution neural network on binary classification using cat and dog data set.

DESCRIBE :

A convolutional neural network (CNN or ConvNet) is a network architecture for deep learning that learns directly from data. CNNs are particularly useful for finding patterns in images to recognize objects, classes, and categories. They can also be quite effective for classifying audio, time-series, and signal data.

ABOUT DATASET :

The Dogs vs. Cats dataset is a standard computer vision dataset that involves classifying photos as either containing a dog or cat. This dataset is provided as a subset of photos from a much larger dataset of 10 thousand manually annotated photos.

PROCEDURE :

- Import the needed libraries.
- Import the dogs and cats image dataset to classifying.
- Split the dataset to tarin and test the model.
- Preprocess the dataset and build the model.
- Compile all the layer using compile function.
- Finally predict the model using the new data.

CODES :

```
In [1]: !pip install keras

Requirement already satisfied: keras in c:\users\abinashkumar\anaconda3\lib\site-packages (2.13.1)
```

IMPORT THE LIBRARIES

```
In [2]: import tensorflow as tf
        from tensorflow import keras
        import os
        import matplotlib.pyplot as plt
```

```
In [3]: import cv2
        import imghdr
```

```
In [4]: image_exts = ['jpeg','jpg','bmp','png']
```

LOAD THE DATSEAT

```
In [5]: data_dir = r"C:\Users\Abinashkumar\Downloads\cats and dogs\training_set\training_set"
```

```
In [6]: for image_class in os.listdir(data_dir):
            for image in os.listdir(os.path.join(data_dir,image_class)):
                print(image)
```

```
cat.103.jpg
cat.1030.jpg
cat.1031.jpg
cat.1032.jpg
cat.1033.jpg
cat.1034.jpg
cat.1035.jpg
cat.1036.jpg
cat.1037.jpg
cat.1038.jpg
cat.1039.jpg
cat.104.jpg
cat.1040.jpg
cat.1041.jpg
cat.1042.jpg
cat.1043.jpg
cat.1044.jpg
```

```
In [7]: for image_class in os.listdir(data_dir):
            for image in os.listdir(os.path.join(data_dir,image_class)):
                image_path = os.path.join(data_dir,image_class,image)
                try:
                    img = cv2.imread(image_path)
                    tip = imghdr.what(image_path)
                    if tip not in image_exts:
                        print('Image not in ext list {}'.format(image_path))
                        os.remove(image_path)
                except Exception as e:
                    print('Issue with image {}'.format(image_path))
```

```
In [8]: data = tf.keras.utils.image_dataset_from_directory(r"C:\Users\Abinashkumar\Downloads\cats and dogs\training_set\training_set")
```

```
Found 8005 files belonging to 2 classes.
```

```
In [9]: data_iterator = data.as_numpy_iterator()
        data_iterator
```

```
Out[9]: <tensorflow.python.data.ops.dataset_ops._NumpyIterator at 0x1ee937c8c00>
```

```
In [10]: batch = data_iterator.next()
         batch
```

```
            [[254.4052   , 255.       , 255.       ],
             [253.7859   , 254.75977  , 254.49023  ],
             [252.58772  , 252.6678   , 252.82796  ],
             ...,
             [255.       , 255.       , 255.       ],
             [255.       , 255.       , 255.       ],
             [255.       , 255.       , 255.       ]],

            [[255.       , 255.       , 255.       ],
             [254.14146  , 254.14146  , 254.14146  ],
             [253.4641   , 253.4641   , 253.4641   ],
             ...,
             [255.       , 255.       , 255.       ],
             [255.       , 255.       , 255.       ],
             [255.       , 255.       , 255.       ]],

            [[253.41602  , 253.41602  , 253.41602  ],
             [254.       , 254.       , 254.       ],
             [254.       , 254.       , 254.       ],
```

```
In [11]: len(batch)
```

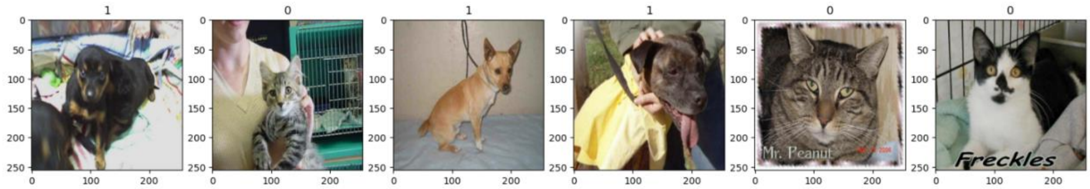
```
Out[11]: 2
```

```
In [12]: batch[0].shape
```

```
Out[12]: (32, 256, 256, 3)
```

```
In [13]: tf.keras.utils.image_dataset_from_directory(r"C:\Users\Abinashkumar\Downloads\cats and dogs\training_set\training_set",batch
```

In [15]:
```
fig, ax = plt.subplots(ncols=6, figsize = (20,20))
for idx, img in enumerate(batch[0][:6]):
    ax[idx].imshow(img.astype(int))
    ax[idx].title.set_text(batch[1][idx])
```



In [16]:
```
scaled = batch[0]/255
```

In [17]:
```
scaled.max()
```

Out[17]: 1.0

PREPROCESSING THE DATA

In [18]:
```
data = data.map(lambda x,y: (x/255,y))
```

In [19]:
```
scaled_iteration = data.as_numpy_iterator()
```

In [20]:
```
scaled_iteration.next()[0].min()
```

Out[20]: 0.0

In [21]:
```
batch = scaled_iteration.next()
```

SPLIT THE DATA

In [23]:
```
len(data)
```

Out[23]: 251

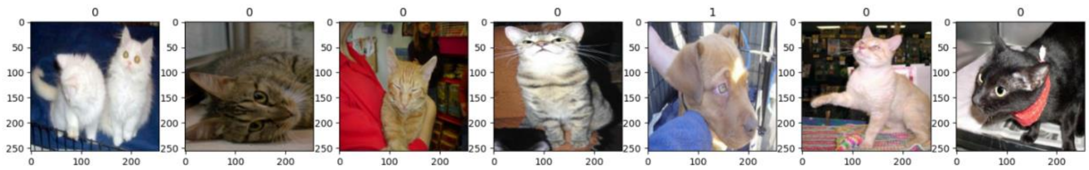
In [24]:
```
251*.251
```

Out[24]: 63.001

In [25]:
```
train_size = int(len(data)*.7)
val_size = int(len(data)*.2)+1
test_size = int(len(data)*.1)+1
```

In [21]:
```
batch = scaled_iteration.next()
```

In [22]:
```
fig, ax = plt.subplots(ncols=7, figsize = (20,20))
for idx, img in enumerate(batch[0][:7]):
    ax[idx].imshow(img)
    ax[idx].title.set_text(batch[1][idx])
```



```
In [27]: train = data.take(train_size)
         val = data.skip(train_size).take(val_size)
         test = data.skip(train_size+val_size).take(test_size)
```

```
In [28]: len(test)
Out[28]: 25
```

Model building

```
In [29]: from tensorflow.keras.models import Sequential
         from tensorflow.keras.layers import Conv2D,MaxPooling2D,Dense,Flatten,Dropout
```

```
In [30]: model = Sequential()
```

```
In [31]: model.add(Conv2D(16, (3, 3),1, input_shape=(256,256,3), activation='relu'))
         model.add(MaxPooling2D())
```

```
In [32]: model.add(Conv2D(32, (3, 3),1, activation='relu'))
         model.add(MaxPooling2D())


         model.add(Conv2D(16, (3, 3),1, activation='relu'))
         model.add(MaxPooling2D())


         model.add(Flatten())



         model.add(Dense(256, activation='relu'))
         model.add(Dense(1, activation='sigmoid'))
```

```
In [33]: model.compile('adam', loss=tf.losses.BinaryCrossentropy(),metrics=['accuracy'])
```

```
In [34]: model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 16)      448

 max_pooling2d (MaxPooling2  (None, 127, 127, 16)      0
 D)

 conv2d_1 (Conv2D)           (None, 125, 125, 32)      4640

 max_pooling2d_1 (MaxPoolin  (None, 62, 62, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 60, 60, 16)        4624
```

```
======================================================================
Total params: 3696625 (14.10 MB)
Trainable params: 3696625 (14.10 MB)
Non-trainable params: 0 (0.00 Byte)
______________________________________________________________________
```

```
In [38]: # Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Data preprocessing
train_datagen = ImageDataGenerator(
    rescale=1.0/255,  # Rescale pixel values to the range [0, 1]
    shear_range=0.2,  # Shear transformations
    zoom_range=0.2,   # Random zoom
    horizontal_flip=True  # Randomly flip images horizontally
)

train_generator = train_datagen.flow_from_directory(
    r"C:\Users\Abinashkumar\my diractory\deep learning\training_sets\training_set",
    target_size=(64, 64),  # Resize input images to 64x64 pixels
    batch_size=32,
    class_mode='binary'  # Binary classification (cat vs. dog)
)

# Train the model
model.fit(train_generator, epochs=10, steps_per_epoch=len(train_generator))
```


```
# Train the model
model.fit(train_generator, epochs=10, steps_per_epoch=len(train_generator))
```


```
Found 8005 images belonging to 2 classes.
Epoch 1/10
251/251 [==============================] - 212s 838ms/step - loss: 0.6833 - accuracy: 0.5628
Epoch 2/10
251/251 [==============================] - 113s 451ms/step - loss: 0.6233 - accuracy: 0.6533
Epoch 3/10
251/251 [==============================] - 116s 464ms/step - loss: 0.5698 - accuracy: 0.7001
Epoch 4/10
251/251 [==============================] - 112s 447ms/step - loss: 0.5366 - accuracy: 0.7312
Epoch 5/10
251/251 [==============================] - 114s 455ms/step - loss: 0.5046 - accuracy: 0.7565
Epoch 6/10
251/251 [==============================] - 111s 443ms/step - loss: 0.4972 - accuracy: 0.7599
Epoch 7/10
251/251 [==============================] - 113s 450ms/step - loss: 0.4687 - accuracy: 0.7814
Epoch 8/10
251/251 [==============================] - 114s 453ms/step - loss: 0.4512 - accuracy: 0.7868
Epoch 9/10
251/251 [==============================] - 113s 451ms/step - loss: 0.4411 - accuracy: 0.7915
Epoch 10/10
251/251 [==============================] - 115s 457ms/step - loss: 0.4195 - accuracy: 0.8062
```

RESULT :

Thus using the cats and dogs data set we build convolution neural network model with accuracy 80.62 %.

EX.NO : 05	CONVOLUTION NEURAL NETWORK ON MULTI-CLASSIFICATION TASK: DOG BREED CLASSIFICATIONS
DATE : 14/08/2023	

AIM :

To build convolution neural network on multi classification using the dogs breed dataset.

DESCRIBE :

Multiclass Image Classification is one of the very primary yet powerful computer vision tasks that can be performed using CNN networks. In this method, we have more than two classes of images that are labeled according to their categories. (eg. CIFAR, Fashion MNIST).

ABOUT DATASET :

1. **Breed Names:** A list of various dog breeds, ranging from popular ones like Labrador Retriever, German Shepherd, and Poodle to more obscure breeds.
2. **Classification Labels:** Each breed would be associated with a classification label, which might denote the breed's group (e.g., Sporting, Herding, Terrier) based on recognized kennel club standards.
3. **Physical Attributes:** Information about the breed's physical characteristics such as size, coat type (long, short, curly), coat color, and general appearance.

PROCEDURE :

- Import the dogs breed dataset
- Read the each image from the dataset
- Build the convolution neural network model
- Compile the model
- Making prediction

CODES :

```
In [1]: import tensorflow as tf
        from tensorflow import keras
        import os
        import matplotlib.pyplot as plt
```

```
In [2]: import cv2
        import imghdr
```

```
In [3]: image_exts = ['jpeg','jpg','bmp','png']
```

```
In [4]: data_dir = r"C:\Users\Abinashkumar\my diractory\deep learning\big dog breed classification\dog_v1"
```

In [5]:
```
for image_class in os.listdir(data_dir):
    for image in os.listdir(os.path.join(data_dir,image_class)):
        print(image)
```

```
0200259a-2722-4576-86fb-6ead7393d8a0.jpg
0c28471d0832854c0206dc3d4a563e93.jpg
100e702566c23d7f711b7d69f415c735.jpg
14747-2116.jpg
14959-1992.jpg
15072-4692.jpg
15491-2365.jpg
21767-8375.jpg
21768-1518.jpg
22310-5589.jpg
```

In [6]:
```
for image_class in os.listdir(data_dir):
    for image in os.listdir(os.path.join(data_dir,image_class)):
        image_path = os.path.join(data_dir,image_class,image)
        try:
            img = cv2.imread(image_path)
            tip = imghdr.what(image_path)
            if tip not in image_exts:
                print('Image not in ext list {}'.format(image_path))
                os.remove(image_path)
        except Exception as e:
            print('Issue with image {}'.format(image_path))
```

In [7]:
```
data = tf.keras.utils.image_dataset_from_directory(r"C:\Users\Abinashkumar\my diractory\deep learning\big dog breed classificatic
```

```
Found 1030 files belonging to 5 classes.
```

In [8]:
```
data_iterator = data.as_numpy_iterator()
data_iterator
```

Out[8]:
```
<tensorflow.python.data.ops.dataset_ops._NumpyIterator at 0x260a7ba1f00>
```

In [9]:
```
batch = data_iterator.next()
batch
```

Out[9]:
```
(array([[[[163.19336  , 161.19336  , 174.19336  ],
          [163.6869   , 161.6869   , 174.6869   ],
          [161.17188  , 159.17188  , 172.17188  ],
          ...,
          [172.80664  , 165.80664  , 173.80664  ],
          [170.80664  , 165.80664  , 172.80664  ],
          [169.11934  , 164.11934  , 171.11934  ]],

         [[163.41992  , 161.41992  , 174.41992  ],
          [164.41992  , 162.41992  , 175.41992  ],
          [164.79204  , 162.79204  , 175.79204  ],
```

In [10]:
```
len(batch)
```

Out[10]:
```
2
```

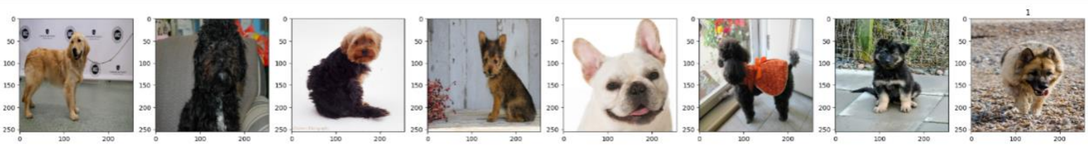
In [11]:
```
batch[1].shape
```

Out[11]:
```
(32,)
```

In [12]:
```
tf.keras.utils.image_dataset_from_directory(r"C:\Users\Abinashkumar\my diractory\deep learning\big dog breed classification\dog_v
```

```
Found 1030 files belonging to 5 classes.
```

In [13]:
```
fig, ax = plt.subplots(ncols=8, figsize = (30,30))
for idx, img in enumerate(batch[0][:8]):
    ax[idx].imshow(img.astype(int))
ax[idx].title.set_text(batch[1][idx])
```



In [14]:
```
scaled = batch[0]/255
```

In [15]:
```
scaled.max()
```

Out[15]: 1.0

preprocessing

In [16]:
```
data = data.map(lambda x,y: (x/255,y))
```

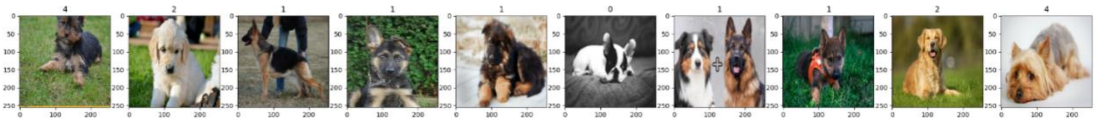
In [17]:
```
scaled_iteration = data.as_numpy_iterator()
```

In [18]:
```
scaled_iteration.next()[0].min()
```

Out[18]: 0.0

In [19]:
```
batch = scaled_iteration.next()
```

In [20]:
```
fig, ax = plt.subplots(ncols=10, figsize = (30,30))
for idx, img in enumerate(batch[0][:10]):
    ax[idx].imshow(img)
    ax[idx].title.set_text(batch[1][idx])
```



Split data

In [21]:
```
len(data)
```

Out[21]: 33

In [22]:
```
251*.251
```

Out[22]: 63.001

In [23]:
```
train_size = int(len(data)*.7)
val_size = int(len(data)*.2)+1
test_size = int(len(data)*.1)+1
```

In [24]:
```
train_size+val_size+test_size
```

Out[24]: 34

In [25]:
```
train = data.take(train_size)
val = data.skip(train_size).take(val_size)
test = data.skip(train_size+val_size).take(test_size)
```

In [26]:
```
len(test)
```

Out[26]: 3

Model building

In [27]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,MaxPooling2D,Dense,Flatten,Dropout
```

In [28]:
```
model = Sequential()
```

In [29]:
```
model.add(Conv2D(16, (3, 3),1, input_shape=(256,256,3), activation='relu'))
model.add(MaxPooling2D())
```

```
In [35]: model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 16)      448

 max_pooling2d (MaxPooling2  (None, 127, 127, 16)      0
 D)

 conv2d_1 (Conv2D)           (None, 125, 125, 32)      4640

 max_pooling2d_1 (MaxPoolin  (None, 62, 62, 32)        0
 g2D)
```

```
In [36]: import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define the CNN model
model = Sequential()

# Convolutional Layers
model.add(Conv2D(32, (3, 3), input_shape=(64, 64, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

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

# Flatten the output to feed into a fully connected layer
model.add(Flatten())

# Fully connected layers
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))  # Dropout to reduce overfitting
model.add(Dense(1, activation='sigmoid'))  # Output Laye
```

```
In [8]: # 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))

31
Epoch 95/100
2/2 [==============================] - 22s 12s/step - loss: 2.7648 - accuracy: 0.8359 - val_loss: 3.0329 - val_accuracy: 0.73
85
Epoch 96/100
2/2 [==============================] - 21s 11s/step - loss: 2.7360 - accuracy: 0.8385 - val_loss: 3.0229 - val_accuracy: 0.72
31
Epoch 97/100
2/2 [==============================] - 21s 11s/step - loss: 2.7461 - accuracy: 0.8333 - val_loss: 3.0317 - val_accuracy: 0.72
31
Epoch 98/100
2/2 [==============================] - 21s 11s/step - loss: 2.7155 - accuracy: 0.8438 - val_loss: 3.0290 - val_accuracy: 0.72
31
Epoch 99/100
2/2 [==============================] - 21s 11s/step - loss: 2.7359 - accuracy: 0.8125 - val_loss: 3.0073 - val_accuracy: 0.75
38
Epoch 100/100
2/2 [==============================] - 21s 11s/step - loss: 2.6964 - accuracy: 0.8411 - val_loss: 2.9923 - val_accuracy: 0.70
77
```

RESULT :

Thus using the dogs breed dataset we successfully build the convolution neural network with an accuracy 70.77 %.

EX.NO : 06	TRANSFER LEARNING USING PRE TRAINED ARCHITECTURES
DATE : 28/08/2023	

AIM :

To apply the transfer learning using pre trained model architecture and predict the image.

DESCRIBE :

Transfer learning speeds up the training process. Pre-trained CNNs have already learned general features, so fine-tuning the model on a specific task requires less time compared to training from scratch. It also reduces the computational resources needed for training.

ABOUT DATASET :

The Dogs vs. Cats dataset it is a already pre-trained model using the convolution neural network. Here we can applying the transfer learning to predict the image of cats and dogs.

PROCEDURE :

- Import the already pre-trained model dataset.
- Applying the transfer learning to the pre-trained dataset.
- Build the transfer learning model.
- Compile the model using model Function.
- Predict the image using the transfer learning model.

CODES :

IMPORT THE LIBRARIES

In [1]:
```
import os
from tensorflow.keras import layers
from tensorflow.keras import Model
```

In [2]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```


IMPORT THE LIBRARIES ¶

In [1]:
```
import os
from tensorflow.keras import layers
from tensorflow.keras import Model
```

In [2]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

LOAD THE PRE-TRAINED DATASET

In [3]:
```
train_datagen = ImageDataGenerator(rescale=1.0/255,rotation_range=20,width_shift_range=0.2,height_shift_range=0.2,shear_range=0.2
    zoom_range=0.2,horizontal_flip=True,fill_mode='nearest')

train_generator = train_datagen.flow_from_directory(
r"C:\Users\Abinashkumar\my diractory\deep learning\training_sets\training_set",target_size=(224, 224),batch_size=32,class_mode='b

val_datagen = ImageDataGenerator(rescale=1.0/255)

val_generator = val_datagen.flow_from_directory(r"C:\Users\Abinashkumar\my diractory\deep learning\test_set\test_set",target_size
    batch_size=32,class_mode='binary')
```

```
Found 8005 images belonging to 2 classes.
Found 2023 images belonging to 2 classes.
```

In [4]:
```
# Example: Load a pre-trained model from TensorFlow Hub
base_model = keras.applications.MobileNetV2(input_shape=(224, 224, 3),include_top=False,weights='imagenet')

# Freeze the base model's layers
base_model.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ord
ering_tf_kernels_1.0_224_no_top.h5
9406464/9406464 [==============================] - 30s 3us/step
```

MODEL CREATION

In [5]:
```
# Add your custom classification head
model = keras.Sequential([base_model,layers.GlobalAveragePooling2D(),layers.Dense(1, activation='sigmoid')])
```

COMPILE THE MODEL

In [6]:
```
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

In [7]:
```
history = model.fit(train_generator,steps_per_epoch=len(train_generator),epochs=10,validation_data=val_generator,
                    validation_steps=len(val_generator))
```

```
Epoch 1/10
251/251 [==============================] - 809s 3s/step - loss: 0.1459 - accuracy: 0.9449 - val_loss: 0.0633 - val_accuracy: 0.
9778
Epoch 2/10
251/251 [==============================] - 708s 3s/step - loss: 0.0839 - accuracy: 0.9658 - val_loss: 0.0485 - val_accuracy: 0.
9812
Epoch 3/10
251/251 [==============================] - 675s 3s/step - loss: 0.0748 - accuracy: 0.9699 - val_loss: 0.0470 - val_accuracy: 0.
9827
Epoch 4/10
251/251 [==============================] - 686s 3s/step - loss: 0.0672 - accuracy: 0.9721 - val_loss: 0.0421 - val_accuracy: 0.
9837
Epoch 5/10
251/251 [==============================] - 676s 3s/step - loss: 0.0650 - accuracy: 0.9739 - val_loss: 0.0392 - val_accuracy: 0.
9842
Epoch 6/10
251/251 [==============================] - 488s 2s/step - loss: 0.0698 - accuracy: 0.9711 - val_loss: 0.0423 - val_accuracy: 0.
9847
Epoch 7/10
251/251 [==============================] - 293s 1s/step - loss: 0.0612 - accuracy: 0.9778 - val_loss: 0.0421 - val_accuracy: 0.
9852
Epoch 8/10
251/251 [==============================] - 281s 1s/step - loss: 0.0639 - accuracy: 0.9748 - val_loss: 0.0497 - val_accuracy: 0.
9807
Epoch 9/10
251/251 [==============================] - 291s 1s/step - loss: 0.0618 - accuracy: 0.9749 - val_loss: 0.0484 - val_accuracy: 0.
9822
Epoch 10/10
251/251 [==============================] - 272s 1s/step - loss: 0.0627 - accuracy: 0.9765 - val_loss: 0.0376 - val_accuracy: 0.
9881
```

EVALUATE THE MODEL

In [10]:
```
# Evaluate the model on a test dataset
test_loss, test_accuracy = model.evaluate(val_generator)
print(f'Test Accuracy: {test_accuracy * 100:.2f}%')
```

```
64/64 [==============================] - 41s 638ms/step - loss: 0.0376 - accuracy: 0.9881
Test Accuracy: 98.81%
```

RESULT :

Thus, we successfully created the model with an accuracy of 98.81%.

EX.NO : 07	HYPER PARAMETER OPTIMIZATION ON CNN MODELS
DATE : 04/09/2023	

AIM :

To create a hyper parameter optimization on convolution neural network model.

DESCRIBE :

Performance of a multi-layer neural network always depends on hyper-parameters such as learning rate, mini batch size, dropout rate, starting learning rate, and learning rate etc. Optimizing hyper-parameters of a multi-layer neural network is always a challenging task. This repository implements two ways of optimizing hyper-parameters of a convolutional neural network and compares their performances: 1) Grid Search and 2) Bayesian Optimization. This repository optimizes three particular hyper-parameters: learning rate, dropout for first fully connected layer and dropout for second fully connected layer.

ABOUT DATASET :

Here we using inbuild dataset it's there in the tensorflow frame work itself. It is fashion mnist dataset its contains fashion dress images, using this dataset we create the hyper parameter optimization Algorithm with convolution neural network.

PROCEDURE :

- Import the needed libraries from the tensorflow frame work.
- Import the inbuild the dataset.
- Split the dataset.
- Tune the model using hyper parameter algorithm.
- Compile the model.
- Validate the tuned model.

CODES :

IMPORT THE LIBRARIES

In [1]:
```
import tensorflow as tf
from tensorflow import keras
import numpy as np
```

IMPORT THE INBUILD DATASET

In [3]:
```
fashion_mnist=keras.datasets.fashion_mnist
```

SPLIT THE DATASET

```
In [4]: (train_images,train_labels),(test_images,test_labels)=fashion_mnist.load_data()
```

RESHAPE THE DATASET

```
In [5]: train_images=train_images/255.0
        test_images=test_images/255.0
```

```
In [6]: train_images[0].shape

Out[6]: (28, 28)
```

```
In [7]: train_images=train_images.reshape(len(train_images),28,28,1)
        test_images=test_images.reshape(len(test_images),28,28,1)
```

BUILD AND COMPILE THE MODEL

```
In [8]: def build_model(hp):
            model = keras.Sequential([keras.layers.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)),
            keras.layers.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'),keras.layers.Flatten(),keras.layers.Dense(
                units=hp.Int('dense_1_units', min_value=32, max_value=128, step=16),activation='relu'),
            keras.layers.Dense(10, activation='softmax')])

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

        return model
```

IMPORT THE ALGORITHM

```
In [11]: from kerastuner import RandomSearch
         from kerastuner.engine.hyperparameters import HyperParameters
```

```
In [12]: tuner_search=RandomSearch(build_model,
                                   objective='val_accuracy',
                                   max_trials=3,directory='output',project_name="Mnist Fashion")

         Reloading Tuner from output\Mnist Fashion\tuner0.json
```

TUNE THE MODEL

```
In [13]: tuner_search.search(train_images,train_labels,epochs=3,validation_split=0.1)

         Trial 5 Complete [00h 08m 21s]
         val_accuracy: 0.9049999713897705

         Best val_accuracy So Far: 0.9164999723434448
         Total elapsed time: 00h 08m 21s
```

```
In [14]: model=tuner_search.get_best_models(num_models=1)[0]
```

FIT THE MODEL

```
In [16]: model.fit(train_images, train_labels, epochs=4, validation_split=0.1, initial_epoch=3)

         Epoch 4/4
         1688/1688 [==============================] - 166s 98ms/step - loss: 0.1259 - accuracy: 0.9521 - val_loss: 0.2921 - val_accurac
         y: 0.9122

Out[16]: <keras.src.callbacks.History at 0x209964a49d0>
```

RESULT :

Thus, successfully we apply hyper parameter optimization techniques with convolution neural network on inbuild fashion mnist dataset.

EX.NO : 08	RECURRENT NEURAL NETWORK ON STOCK PRICE PREDICTION
DATE : 11/09/2023	

AIM :

To using recurrent neural network to analysis and predict the stock price using stock dataset.

DESCRIBE :

Recurrent neural networks (RNNs) are a class of neural network that are helpful in modelling sequence data. Derived from feedforward networks, RNNs exhibit similar behaviour to how human brains function.

ABOUT DATASET :

Here we using the stock dataset in specifically use nifty50 dataset, it contains some features about top fifty companies using this data we use Recurrent neural network to predict the next 10 days price of the companies.

PROCEDURE :

- Import the needed libraries from the TensorFlow frame work.
- Import and read the dataset using the panda's library.
- Split the dataset as train data and test data.
- Add the LSTM and Dense layers.
- Transform the data and calculate the mean squared error.
- Create the model and compile the model.
- Finally calculate or predict the model.

CODES :

In [1]:
```
import pandas as pd
```

In [2]:
```
df=pd.read_csv(r"C:\Users\Abinashkumar\my diractory\deep learning\stock market dataset for rnn 8\NIFTY50_all.csv")
```

In [4]:
```
df.head()
```

Out[4]:

	Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close	VWAP	Volume	Turnover	Trades	Deliverable Volume	%Deliverble
0	2007-11-27	MUNDRAPORT	EQ	440.00	770.00	1050.00	770.0	959.0	962.90	984.72	27294366	2.687719e+15	NaN	9859619.0	0.3612
1	2007-11-28	MUNDRAPORT	EQ	962.90	984.00	990.00	874.0	885.0	893.90	941.38	4581338	4.312765e+14	NaN	1453278.0	0.3172
2	2007-11-29	MUNDRAPORT	EQ	893.90	909.00	914.75	841.0	887.0	884.20	888.09	5124121	4.550658e+14	NaN	1069678.0	0.2088

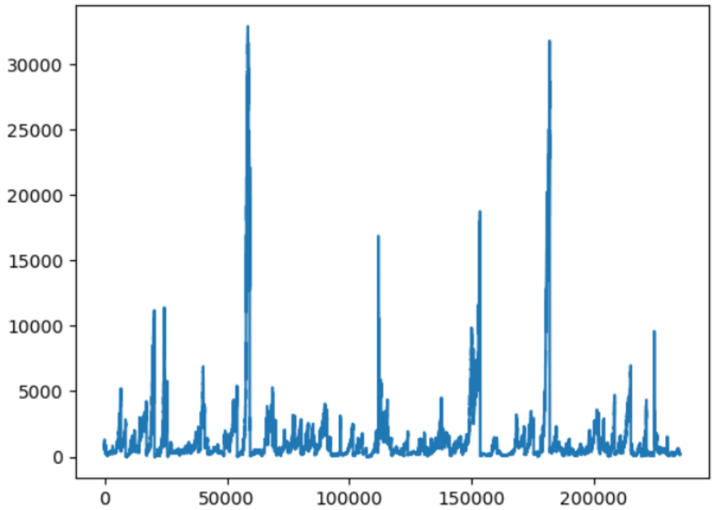
In [6]:
```
df1=df.reset_index()['Close']
```

In [7]:
```
df1
```

Out[7]:
```
0        962.90
1        893.90
2        884.20
3        921.55
4        969.30
```

```
In [8]: import matplotlib.pyplot as plt
        plt.plot(df1)
```

```
Out[8]: [<matplotlib.lines.Line2D at 0x18906ad4130>]
```



PREPROCESSING THE DATA

```
In [12]: from sklearn.preprocessing import MinMaxScaler
         scaler=MinMaxScaler(feature_range=(0,1))
         df1=scaler.fit_transform(np.array(df1).reshape(-1,1))
```

```
In [13]: print(df1)

[[0.02903101]
 [0.02693073]
 [0.02663548]
 ...
 [0.00547746]
 [0.00539984]
 [0.00537093]]
```

SPLIT THE DATASET

```
In [14]: ##splitting dataset into train and test split
         training_size=int(len(df1)*0.65)
         test_size=len(df1)-training_size
         train_data,test_data=df1[0:training_size,:],df1[training_size:len(df1),:1]
```

```
In [15]: training_size,test_size

Out[15]: (152874, 82318)
```

ADD THE LAYERS

```
In [23]: model=Sequential()
         model.add(LSTM(50,return_sequences=True,input_shape=(100,1)))
         model.add(LSTM(50,return_sequences=True))
         model.add(LSTM(50))
         model.add(Dense(1))
         model.compile(loss='mean_squared_error',optimizer='adam')
```

```
In [25]: model.fit(X_train,y_train,validation_data=(X_test,ytest),epochs=1,batch_size=64,verbose=1)

2388/2388 [==============================] - 1042s 433ms/step - loss: 7.4870e-05 - val_loss: 3.7269e-05

Out[25]: <keras.src.callbacks.History at 0x2baf0a029d0>
```

TRANSFORM THE DATA IN ORIGINAL FORMAT

```
In [28]: ##Transformback to original form
         train_predict=scaler.inverse_transform(train_predict)
         test_predict=scaler.inverse_transform(test_predict)
```

CALCULATE MEAN SQUARED ERROR

In [30]:
```
### Test Data RMSE
math.sqrt(mean_squared_error(ytest,test_predict))
```

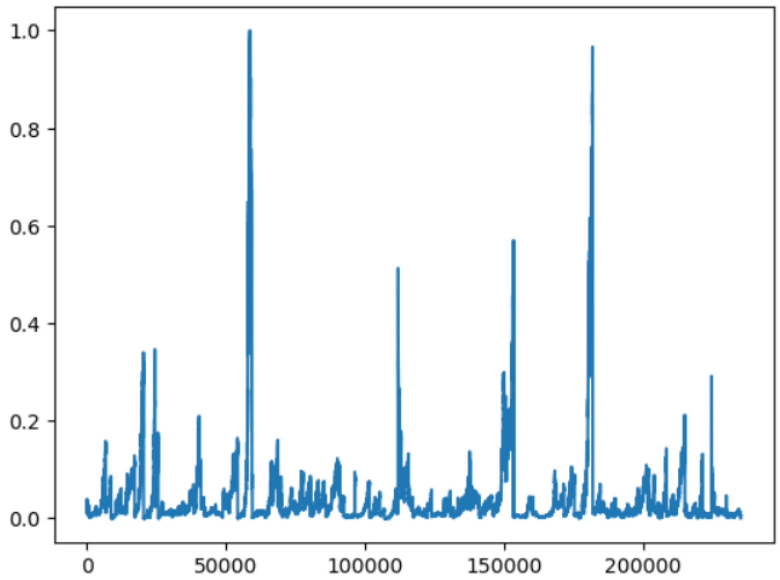
Out[30]:
```
293.07652201244144
```

In [37]:
```
# demonstrate prediction for next 10 days
from numpy import array

lst_output=[]
n_steps=70801
i=0
while(i<10):

    if(len(temp_input)>1516):
        #print(temp_input)
        x_input=np.array(temp_input[1:])
        print("{} day input {}".format(i,x_input))
        x_input=x_input.reshape(1,-1)
        x_input = x_input.reshape((1, n_steps, 1))
        #print(x_input)
        yhat = model.predict(x_input, verbose=0)
        print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        #print(temp_input)
        lst_output.extend(yhat.tolist())
        i=i+1
    else:
        x_input = x_input.reshape((1, n_steps,1))
        yhat = model.predict(x_input, verbose=0)
        print(yhat[0])
        temp_input.extend(yhat[0].tolist())
        print(len(temp_input))
        lst_output.extend(yhat.tolist())
        i=i+1


print(lst_output)
```

```
0 day input [0.00269231 0.00275015 0.00275015 ... 0.00547746 0.00539984 0.00537093]
0 day output [[0.00054192]]
1 day input [0.00275015 0.00275015 0.00282472 ... 0.00539984 0.00537093 0.00054192]
1 day output [[0.00054477]]
2 day input [0.00275015 0.00282472 0.00278363 ... 0.00537093 0.00054192 0.00054477]
2 day output [[0.0005521]]
3 day input [0.00282472 0.00278363 0.00273493 ... 0.00054192 0.00054477 0.0005521 ]
3 day output [[0.00056054]]
4 day input [0.00278363 0.00273493 0.00277145 ... 0.00054477 0.0005521  0.00056054]
4 day output [[0.00056676]]
5 day input [0.00273493 0.00277145 0.00272732 ... 0.0005521  0.00056054 0.00056676]
5 day output [[0.00056831]]
6 day input [0.00277145 0.00272732 0.00277754 ... 0.00056054 0.00056676 0.00056831]
6 day output [[0.00056389]]
7 day input [0.00272732 0.00277754 0.00276993 ... 0.00056676 0.00056831 0.00056389]
7 day output [[0.0005532]]
8 day input [0.00277754 0.00276993 0.00280494 ... 0.00056831 0.00056389 0.0005532 ]
8 day output [[0.00053668]]
9 day input [0.00276993 0.00280494 0.00270144 ... 0.00056389 0.0005532  0.00053668]
9 day output [[0.00051523]]
[[0.0005419168155640364], [0.0005447704461403191], [0.0005520955892279744], [0.0005605424521490932], [0.0005667571676895022],
[0.0005683057243004441], [0.0005638866568915546], [0.0005531990900635719], [0.0005366802215576172], [0.0005152273806743324]]
```

```
In [62]: df3=df1.tolist()
         df3.extend(lst_output)
         plt.plot(df3[10:])
```

```
Out[62]: [<matplotlib.lines.Line2D at 0x189153052e0>]
```



RESULT :

Thus, we use the stock market nifty50 dataset build the recurrent neural network model and predicted.

EX.NO : 09	GATED RECURRENT NEURAL NETWORK ON IMAGE SEGMENTATION TASK
DATE : 25/10/2023	

AIM :

To build a gated recurrent neural network on image segmentation task.

DESCRIBE :

The Gated Recurrent Unit (GRU) is a type of Recurrent Neural Network (RNN) that, in certain cases, has advantages over long short term memory (LSTM). GRU uses less memory and is faster than LSTM, however, LSTM is more accurate when using datasets with longer sequences.

ABOUT DATASETS :

Here we use the inbuild fashion mnist dataset from the TensorFlow frame work it's a kind of clothing images. To use this image dataset we create a gated recurrent neural network model and classifies the clothing images based on their similarity. Finally we calculate the accuracy of the model.

PROCEDURE :

- Import the needed libraries from the TensorFlow frame work.
- Import the inbuild dataset from the Keras and TensorFlow framework.
- Split the dataset as train data and test data.
- Use the GAN Algorithm and add the needed neural network layers.
- Define the loss and optimizer.
- Finally calculate or predict the model.

CODES :

Generative Adversarial Network for an FASHION MNIST Clothing From Scratch in Keras

In [2]:
```
# loading the mnist dataset
from tensorflow.keras.datasets.fashion_mnist import load_data

# load the images into memory
(trainX, trainy), (testX, testy) = load_data()
# summarize the shape of the dataset
print('Train', trainX.shape, trainy.shape)
print('Test', testX.shape, testy.shape)
```

```
Train (60000, 28, 28) (60000,)
Test (10000, 28, 28) (10000,)
```

```
In [4]: from tensorflow.keras.datasets.fashion_mnist import load_data
        from matplotlib import pyplot

        (trainX, trainy), (testX, testy) = load_data()
        for i in range(25):
            pyplot.subplot(5, 5, 1 + i)
            pyplot.axis('off')
            pyplot.imshow(trainX[i], cmap='gray_r')
        pyplot.show()
```

FASHION MNIST dataset to train the generator and the discriminator.

```
In [5]: import glob
        import imageio
        import matplotlib.pyplot as plt
        import numpy as np
        import os
        import PIL
        from tensorflow.keras import layers
        import time
        import tensorflow as tf

        from IPython import display
```

```
In [6]: (train_images, train_labels), (_, _) = tf.keras.datasets.fashion_mnist.load_data()
```

```
In [7]: train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype('float32')
        train_images = (train_images - 127.5) / 127.5  # Normalize the images to [-1, 1]
```

```
In [8]: BUFFER_SIZE = 60000
        BATCH_SIZE = 256
```

```
In [9]: # Batch and shuffle the data
        train_dataset = tf.data.Dataset.from_tensor_slices(train_images).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
```

The Generator

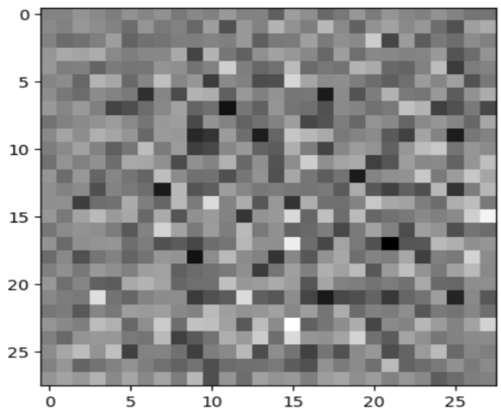
The generator uses **tf.keras.layers.Conv2DTranspose (upsampling) layers** to produce an image from a random noise.

In [10]:
```
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((7, 7, 256)))
    assert model.output_shape == (None, 7, 7, 256)  # Note: None is the batch size
    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False))
    assert model.output_shape == (None, 7, 7, 128)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

   # upsample to 14x14
    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False))
    assert model.output_shape == (None, 14, 14, 64)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    # upsample to 28x28
    model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same', use_bias=False, activation='tanh'))
    assert model.output_shape == (None, 28, 28, 1)

    return model
```

Use the (as yet untrained) generator to create an image

In [11]:
```
# sample image generated by the the generator
generator = make_generator_model()

noise = tf.random.normal([1, 100]) #latent space
generated_image = generator(noise, training=False)

plt.imshow(generated_image[0, :, :, 0], cmap='gray')
```

Out[11]: <matplotlib.image.AxesImage at 0x275d005c910>



Define the loss and optimizers

In [14]:
```
# This method returns a helper function to compute cross entropy loss
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
```

Define the training loop

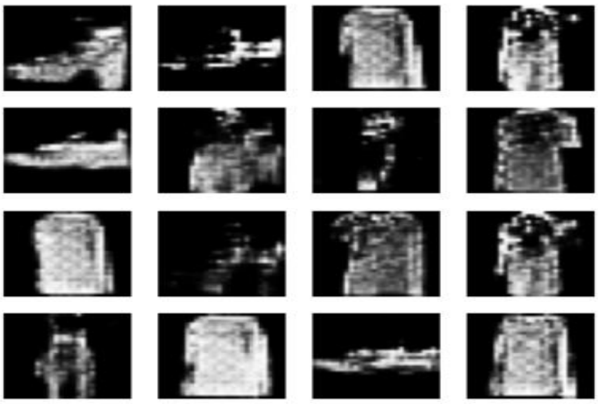
In [19]:
```

EPOCHS = 50
noise_dim = 100
num_examples_to_generate = 16

# You will reuse this seed overtime (so it's easier)
# to visualize progress in the animated GIF)
seed = tf.random.normal([num_examples_to_generate, noise_dim])
```

Train the model

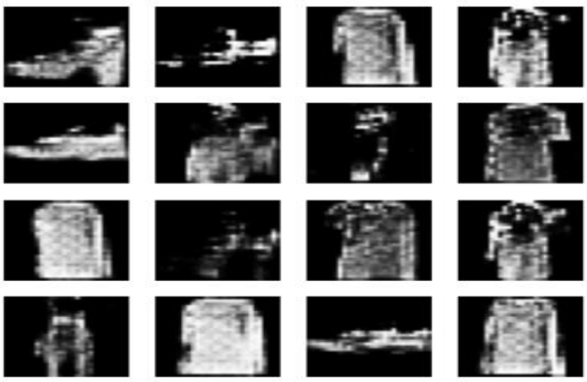
In [21]:
```
def train(dataset, epochs):
  for epoch in range(epochs):
    start = time.time()

    for image_batch in dataset:
      train_step(image_batch)

    # Produce images for the GIF as you go
    display.clear_output(wait=True)
    generate_and_save_images(generator,
                             epoch + 1,
                             seed)

    # Save the model every 15 epochs
    if (epoch + 1) % 15 == 0:
      checkpoint.save(file_prefix = checkpoint_prefix)

    print ('Time for epoch {} is {} sec'.format(epoch + 1, time.time()-start))

  # Generate after the final epoch
  display.clear_output(wait=True)
  generate_and_save_images(generator,
                           epochs,
                           seed)
```

In [26]:
```
display_image(EPOCHS)
```

Out[26]:



In [23]:
```
train(train_dataset, EPOCHS)
```



RESULT :

Thus, finally we successfully working with Gated recurrent neural network on image segmentation task.

EX.NO : 10	LSTM ON PRICE PREDICTION
DATE : 09/10/2023	

AIM :

To build Rnn model of Long-short-term-memory and work on stock price prediction dataset.

DESCRIBE :

LSTM stands for long short-term memory networks, used in the field of Deep Learning. It is a variety of recurrent neural networks (RNNs) that are capable of learning long-term dependencies, especially in sequence prediction problems.

ABOUT DATASET :

Here we use the AMZN stock dataset, it took from the Kaggle platform. AMZN have some features in the dataset (i.e Date, Open, Close, High, Low, Volume, etc). These dataset having 1256 row and 7 columns.

PROCEDURE :

- Import the library from the TensorFlow.
- Preprocessing the dataset and split the dataset.
- Transform the data into array format.
- Add the Dence layer and LSTM layer from Keras TensorFlow.
- Finally calculate the mean squared error to the corresponding dataset.

CODES :

Stock Market Prediction And Forecasting Using Stacked LSTM ¶

READ THE DATASET

In [19]:
```
import pandas as pd
```

In [20]:
```
df=pd.read_csv(r"D:\m.sc Data science\data set\stock market datasets\AMZN.csv")
```

In [21]:
```
df.head()
```

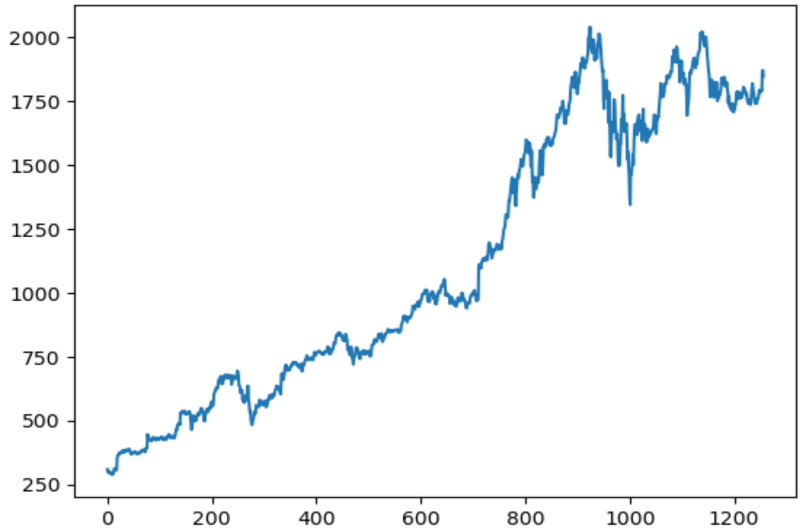
Out[21]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2015-01-02	312.579987	314.750000	306.959991	308.519989	308.519989	2783200
1	2015-01-05	307.010010	308.380005	300.850006	302.190002	302.190002	2774200
2	2015-01-06	302.239990	303.000000	292.380005	295.290009	295.290009	3519000
3	2015-01-07	297.500000	301.279999	295.329987	298.420013	298.420013	2640300
4	2015-01-08	300.320007	303.140015	296.109985	300.459991	300.459991	3088400

PLOT THE CLOSE INDEX

In [25]:
```
import matplotlib.pyplot as plt
plt.plot(df1)
```

Out[25]: [<matplotlib.lines.Line2D at 0x1e804ea5a60>]



PREPROCESSING THE DATSET

In [30]:
```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(-1,1))
df1=scaler.fit_transform(np.array(df1).reshape(-1,1))
```

In [31]:
```
print(df1)
```

```
[[-0.97538461]
 [-0.98260831]
 [-0.9904825 ]
 ...
 [ 0.80515362]
 [ 0.80632907]
 [ 0.78018442]]
```

SPLIT THE DATASET

In [32]:
```
##splitting dataset into train and test split
training_size=int(len(df1)*0.65)
test_size=len(df1)-training_size
train_data,test_data=df1[0:training_size,:],df1[training_size:len(df1),:1]
```

CREATE THE MODEL USING LAYERS

In [41]:
```
model=Sequential()
model.add(LSTM(50,return_sequences=True,input_shape=(100,1)))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
```

```
In [42]: model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100, 50)           10400

 lstm_1 (LSTM)               (None, 100, 50)           20200

 lstm_2 (LSTM)               (None, 50)                20200

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 50851 (198.64 KB)
Trainable params: 50851 (198.64 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

FIT THE MODEL

```
In [44]: model.fit(X_train,y_train,validation_data=(X_test,ytest),epochs=100,batch_size=64,verbose=1)
```

```
Epoch 92/100
12/12 [==============================] - 4s 361ms/step - loss: 8.2946e-04 - val_loss: 0.0195
Epoch 93/100
12/12 [==============================] - 4s 359ms/step - loss: 8.4587e-04 - val_loss: 0.0190
Epoch 94/100
12/12 [==============================] - 4s 366ms/step - loss: 8.3406e-04 - val_loss: 0.0248
Epoch 95/100
12/12 [==============================] - 4s 348ms/step - loss: 8.2570e-04 - val_loss: 0.0236
Epoch 96/100
12/12 [==============================] - 4s 376ms/step - loss: 9.0487e-04 - val_loss: 0.0212
Epoch 97/100
12/12 [==============================] - 4s 362ms/step - loss: 7.8214e-04 - val_loss: 0.0258
Epoch 98/100
12/12 [==============================] - 4s 369ms/step - loss: 7.9016e-04 - val_loss: 0.0295
Epoch 99/100
12/12 [==============================] - 4s 355ms/step - loss: 7.6546e-04 - val_loss: 0.0351
Epoch 100/100
12/12 [==============================] - 4s 349ms/step - loss: 7.9519e-04 - val_loss: 0.0273
```

```
Out[44]: <keras.src.callbacks.History at 0x1e80fafd040>
```

CALCULATE THE MEAN SQUARED ERROR

```
In [47]: import math
         from sklearn.metrics import mean_squared_error
         math.sqrt(mean_squared_error(y_train,train_predict))
```

```
Out[47]: 866.1477214035598
```

CALCULATE THE ROOTMEAN SQUARED ERROR

```
In [48]: ### Test Data RMSE
         math.sqrt(mean_squared_error(ytest,test_predict))
```

```
Out[48]: 1662.0877882466116
```

RESULT :

Thus, finally we completed the LSTM on the stock market dataset and we predict the stock price successfully.

EX.NO : 11	LSTM ON IMAGE SEGMENTATION
DATE : 16/10/2023	

AIM :

To build Long-short-term-memory model and work on image segmentation dataset using LSTM.

DESCRIBE :

Image segmentation using Long Short-Term Memory (LSTM) networks is a technique that involves using recurrent neural networks (RNNs), specifically LSTM units, to perform pixel-wise classification of images. Image segmentation aims to partition an image into multiple segments or regions, where each segment corresponds to a particular object or region of interest. LSTM networks are primarily designed for sequential data, such as text and time series, but they can also be adapted for image segmentation tasks.

ABOUT DATASET :

Here we use the mnist dataset it already there in the keras TensorFlow library itself. It's an hand written numeric number (from zero to nine) images dataset.

PROCEDURE :

- Import the needed libraries from the TensorFlow keras framework.
- Import the mnist dataset from TensorFlow framework.
- Processing and normalizing the dataset.
- Labeling image from the dataset.
- Build and compile the LSTM model.
- Feed the model and visualizing the model.
- Finally predict the LSTM model.

CODES :

IMPORT THE LIBRARIES

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import tensorflow as tf
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import LSTM,Dense
        from tensorflow.keras import datasets
```

LOAD THE INBUILD DATSET

```
In [2]: (train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.mnist.load_data()
```

PREPROCESSING THE DATA - NORMALIZATION

```
In [3]: train_images, test_images = train_images / 255.0, test_images/255.0
```

```
In [4]: class_names = ['Zero','one', 'two','three','four','five','six','seven','eight','nine']
```

SHAPE OF TRAINING AND TEST IMAGES

In [6]:
```
print('shape of training images : ',train_images.shape)
print('shape of training images : ',train_labels.shape)
```

```
shape of training images :  (60000, 28, 28)
shape of training images :  (60000,)
```

LABEL VALUES

In [8]:
```
print('label values :',np.min(train_images), "to" ,np.max(train_images))
```

```
label values : 0.0 to 1.0
```

BUILD THE MODEL

In [10]:
```
model = Sequential()
model.add(LSTM(100,input_shape=(28,28)))
model.add(Dense(10, activation = 'softmax'))
```

In [11]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100)               51600

 dense (Dense)               (None, 10)                1010

=================================================================
Total params: 52610 (205.51 KB)
Trainable params: 52610 (205.51 KB)
Non-trainable params: 0 (0.00 Byte)
```

COMPILE THE MODEL

In [12]:
```
model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=['accuracy'])
```

FEED THE MODEL

In [14]:
```
history = model.fit(train_images,train_labels, epochs=20,validation_data= (test_images,test_labels))
```

```
Epoch 1/20
1875/1875 [==============================] - 61s 31ms/step - loss: 0.3455 - accuracy: 0.8913 - val_loss: 0.1264 - val_accuracy:
0.9613
Epoch 2/20
1875/1875 [==============================] - 60s 32ms/step - loss: 0.1093 - accuracy: 0.9679 - val_loss: 0.1037 - val_accuracy:
0.9694
Epoch 3/20
1875/1875 [==============================] - 54s 29ms/step - loss: 0.0741 - accuracy: 0.9779 - val_loss: 0.0775 - val_accuracy:
0.9771
Epoch 4/20
1875/1875 [==============================] - 47s 25ms/step - loss: 0.0606 - accuracy: 0.9819 - val_loss: 0.0627 - val_accuracy:
0.9815
Epoch 5/20
1875/1875 [==============================] - 44s 23ms/step - loss: 0.0501 - accuracy: 0.9850 - val_loss: 0.0550 - val_accuracy:
```

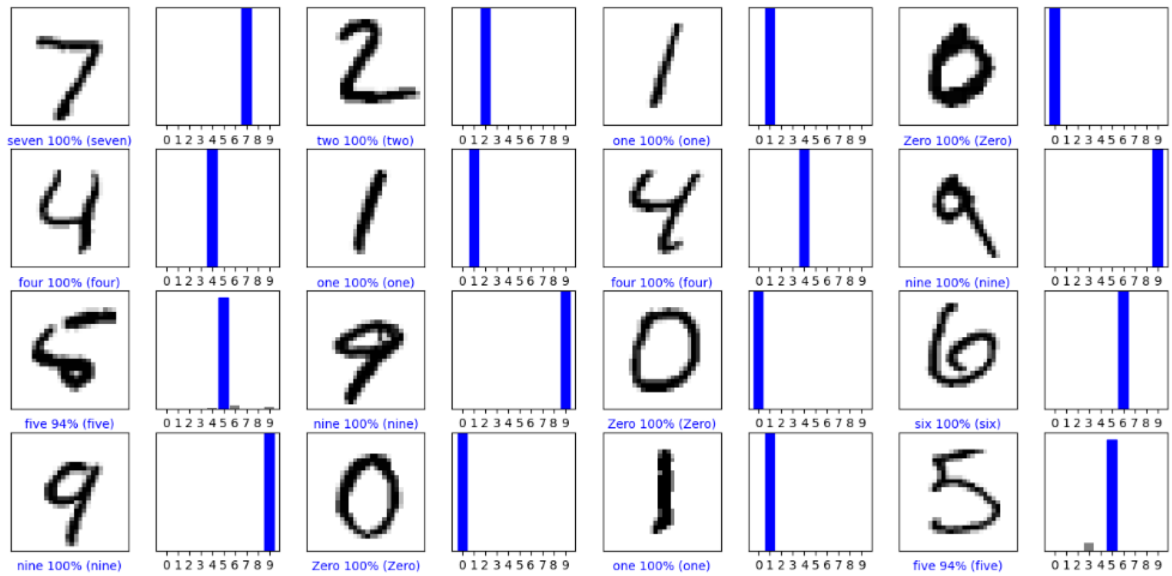
In [20]:
```
test_loss,test_acc = model.evaluate(test_images, test_labels, verbose = 2)
print('Test accuracy : ', round(test_acc*100), '%')
```

```
313/313 - 3s - loss: 0.0498 - accuracy: 0.9871 - 3s/epoch - 11ms/step
Test accuracy :  99 %
```

VERIFY AND VISULIZE PREDICTION

In [41]:
```
predictions = model.predict(test_images)

plt.figure(figsize = (17,8))

for i in range(16):
    plt.subplot(4,8,2*i+1)
    plt.xticks([])
    plt.yticks([])
    if np.argmax(prediction[i]) == test_labels[i]:
        color = 'blue'
    else:
        color = 'red'
    plt.imshow(test_images[i], cmap=plt.cm.binary)
    plt.xlabel('{} {:2.0f}% ({})'.format(class_names[np.argmax(predictions[i])],
                                         100*np.max(predictions[i]),class_names[test_labels[i]]),
                                         color=color)
    plt.subplot(4,8,2*i+2)
    plt.xticks(range(10))
    plt.yticks([])
    thisplot = plt.bar(range(10), prediction[i], color ='gray')
    plt.ylim([0,1])

    thisplot[np.argmax(predictions[i])].set_color('red')
    thisplot[test_labels[i]].set_color('blue')
plt.show()
```

```
313/313 [==============================] - 4s 11ms/step
```



USE THE TRAIN MODEL

In [42]:
```
img = test_images[50]
print(img.shape)
```

```
(28, 28)
```

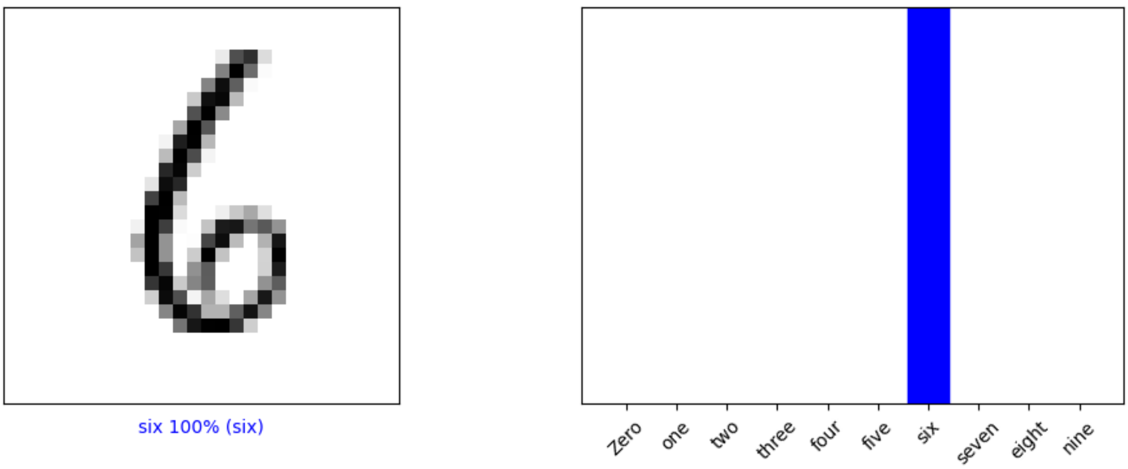
THEN, IMAGES CAN BE FED INTO THE MODEL FOR PREDICTION

In [47]:
```
predictions_img = model.predict(img)

max_confidence = np.argmax(predictions_img[0])

print('the highest confidence is : ', max_confidence)
print('the predicted class is :', class_names[max_confidence])
```

```
1/1 [==============================] - 0s 54ms/step
the highest confidence is :  6
the predicted class is : six
```

LET'S VISUVALIZE THE RESULTS.

In [57]:
```
plt.figure(figsize = (12,4))

i=140
plt.subplot(1,2,1)
plt.xticks([])
plt.yticks([])
if np.argmax(prediction[i]) == test_labels[i]:
    color = 'blue'
else:
    color = 'red'
plt.imshow(test_images[i], cmap=plt.cm.binary)
plt.xlabel('{} {:2.0f}% ({})'.format(class_names[np.argmax(predictions[i])],
                    100*np.max(predictions[i]),class_names[test_labels[i]]),color=color)
plt.subplot(1,2,2)
plt.xticks(range(10),class_names,rotation = 45)
plt.yticks([])
thisplot = plt.bar(range(10), prediction[i], color ='gray')
plt.ylim([0,1])

thisplot[np.argmax(predictions[i])].set_color('red')
thisplot[test_labels[i]].set_color('blue')
plt.show()
```



RESULT :

Thus, successfully we work with Long-Short-Term-Memory algorithm on image segmentation using inbuild dataset.

EX.NO : 12	ANOMALY DETECTION USING AUTOENCODER
DATE : 30/10/2023	

AIM :

To work with anomaly detection using auto encoder algorithm.

DESCRIBE :

Anomaly detection using autoencoders is a machine learning technique for identifying rare and abnormal data points within a dataset. Autoencoders are neural networks designed for data compression and reconstruction. In the context of anomaly detection, the autoencoder learns to reconstruct normal data patterns during training, and when presented with unseen or anomalous data, it struggles to reconstruct them accurately, resulting in a higher reconstruction error.

ABOUT DATASET :

Here we using the anomaly dataset it took from the Kaggle platform. This dataset having only two columns there are ‘Normal data’, ‘Anomaly data’ and 317 columns are there in the dataset. It’s an anomaly sample dataset.

PROCEDURE :

- Import the needed libraries.
- Import the dataset and read the dataset using pandas.
- Define the model and encode the data.
- Decode the data and Compile the model
- Train the auto encoder data.
- Finally calculate the reconstruction of the data.

CODES :

IMPORT THE LIBRARIES

```
In [1]: import numpy as np
        import tensorflow as tf
        from tensorflow import keras
        import pandas as pd
```

IMPORT THE DATASET

```
In [2]: data = pd.read_csv(r"D:\Abinashkumar\m.sc Data science\data set\anomely detection dataset\cv_server_data.csv")
```

DEFINE THE SHAPE OF THE DATASET

```
In [3]: # Define the autoencoder architecture
        input_dim = data.shape[1]
        encoding_dim = 10
        input_dim

Out[3]: 2
```

USE THE MODEL AND ENCODE THE DATASET

In [4]:
```
# Encoder
input_data = keras.layers.Input(shape=(input_dim,))
encoded = keras.layers.Dense(encoding_dim, activation='relu')(input_data)
```

DECODE THE DATASET

In [5]:
```
# Decoder
decoded = keras.layers.Dense(input_dim, activation='sigmoid')(encoded)

# Create the autoencoder model
autoencoder = keras.models.Model(input_data, decoded)
```

COMPILE THE MODEL

In [6]:
```
autoencoder.compile(optimizer='adam', loss='mean_squared_error')
```

TRAIN THE AUTO ENCODER DATA

In [7]:
```
# Train the autoencoder on the data
autoencoder.fit(data, data, epochs=50, batch_size=32)
```

```
Epoch 1/50
10/10 [==============================] - 0s 3ms/step - loss: 192.7829
Epoch 2/50
10/10 [==============================] - 0s 2ms/step - loss: 190.2946
Epoch 3/50
10/10 [==============================] - 0s 4ms/step - loss: 188.3486
Epoch 4/50
10/10 [==============================] - 0s 3ms/step - loss: 187.1501
Epoch 5/50
10/10 [==============================] - 0s 5ms/step - loss: 186.5014
Epoch 6/50
10/10 [==============================] - 0s 1ms/step - loss: 186.1439
Epoch 7/50
10/10 [==============================] - 0s 3ms/step - loss: 185.9383
Epoch 8/50
10/10 [==============================] - 0s 2ms/step - loss: 185.8093
Epoch 9/50
10/10 [==============================] - 0s 2ms/step - loss: 185.7206
Epoch 10/50
10/10 [==============================] - 0s 2ms/step - loss: 185.6545
Epoch 11/50
10/10 [==============================] - 0s 2ms/step - loss: 185.6067
Epoch 12/50
10/10 [==============================] - 0s 2ms/step - loss: 185.5672
Epoch 13/50
10/10 [==============================] - 0s 2ms/step - loss: 185.5367
Epoch 14/50
10/10 [==============================] - 0s 2ms/step - loss: 185.5107
```

ENCODE AND DECODE THE DATA

In [8]:
```
# Encode and decode data
encoded_data = autoencoder.predict(data)
```

```
10/10 [==============================] - 0s 1ms/step
```

CALCULATE THE RECONSTRUCTION ERROR (MSE)

In [9]:
```
reconstruction_error = np.mean(np.square(data - encoded_data), axis=1)
reconstruction_error
```

In [10]:
```
threshold = 0.1
```

IDENTIFY TH EANOMALIES

In [11]:
```
# Identify anomalies
anomalies = data[reconstruction_error > threshold]
```

In [12]:
```
# Print the anomalies
print("Anomalies:")
print(anomalies)
```

```
Anomalies:
     Normal_data  anomalous_data
0       13.64000         15.3300
1       14.86600         16.4740
2       13.58500         13.9890
```

RESULT :

Thus, we successfully work with anomaly dataset using auto encoder algorithm.