K J's Educational Institute

K J College of Engineering & Management Research

(Accredited by NAAC)

Sr. No. 25 & 27, Kondhwa - Saswad Road, Near Bopdev Ghat, Pune 411048.

Department of Computer Engineering

Savitribai Phule Pune University **Second Year of Computer Engineering (2019 Course)** **410255: Laboratory Practice V**		
Teaching Scheme **Practical:** 02 Hours/Week	**Credit:**01	**Examination Scheme and Marks** **Term work:** 50 Marks **Practical:** 50 Marks
Subject Incharge	**Dr. Nikita Kulkarni/Prof. S.L.Yedge**	
Academic Year	2022-2023	
Semester	**VIII**	

List of Practicals

410251 : Deep Learning

Group 1

1) **Linear regression by using Deep Neural network:** Implement Boston housing price prediction problem by Linear regression using Deep Neural network. Use Boston House price prediction dataset.
2) **Classification using Deep neural network:** Binary classification using Deep Neural Networks Example: Classify movie reviews into positive" reviews and "negative" reviews, just based on the text content of the reviews. Use IMDB dataset
3) **Convolutional neural network (CNN):** Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories.

Group 2

4) **Mini Project:** Human Face Recognition

Group 1

EXPERIMENT 1

AIM: Linear regression by using Deep Neural network

Course Objective:
To illustrate the concepts of DNN.

Problem Statement:
Implement Boston housing price prediction problem by Linear regression using Deep Neural network. Use Boston House price prediction dataset.

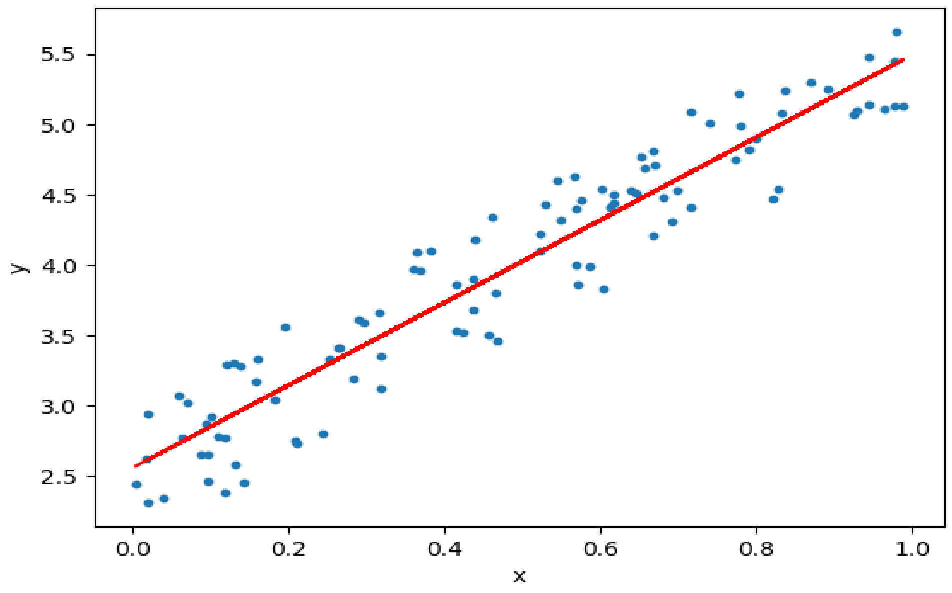
Course Outcomes:
Apply the technique of Deep Neural network for implementing Linear regression.

Software-Hardware requirements:
Python, Ubuntu Operating System 14.0 or above, 2 CPU CoreProcessor, 2GB RAM.

THEORY:

Linear Regression:
Linear Regression is usually the first machine learning algorithm that every data scientist comes across. It is a simple model but everyone needs to master it as it lays the foundation for other machine learning algorithms.
The objective of a linear regression model is to find a relationship between one or more features(independent variables) and a continuous target variable(dependent variable). When there is only feature it is called Uni-variate Linear Regression and if there are multiple features, it is called Multiple Linear Regression.

Where can Linear Regression be used?
It is a very powerful technique and can be used to understand the factors that influence profitability. It can be used to forecast sales in the coming months by analyzing the sales data for previous months. It can also be used to gain various insights about customer behaviour. By the end of the blog we will build a model which looks like the below picture i.e, determine a line which best fits the data.



Hypothesis of Linear Regression

The linear regression model can be represented by the following equation

$$Y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots. + \theta_n x_n$$

- Y is the predicted value
- θ_0 is the bias term.
- $\theta_1,\ldots,\theta_n$ are the model parameters
- $x_1, x_2,\ldots,x_n$ are the feature values.

The above hypothesis can also be represented by

$$Y = \theta^T x$$

where

- θ is the model's parameter vector including the bias term θ_0
- x is the feature vector with $x_0 = 1$

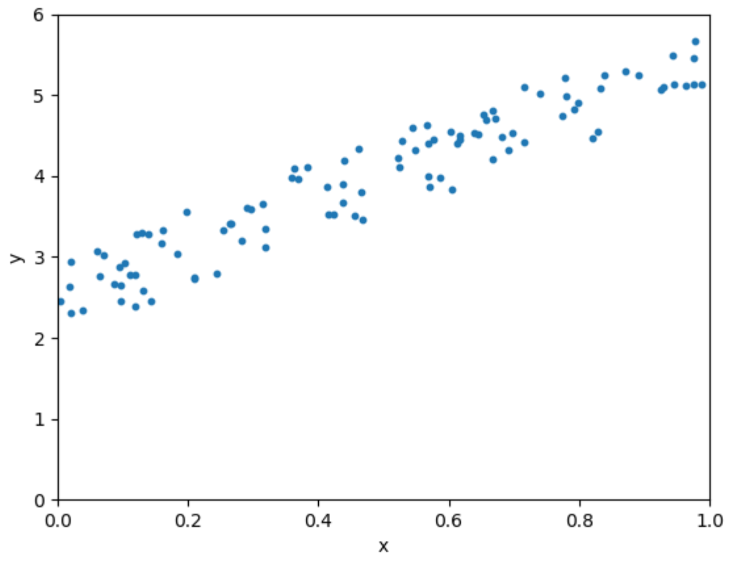
Data-set

- Let's create some random data-set to train our model.

```
#imports
            import numpy as np
            import matplotlib.pyplot as plt

            # generate random data-set
            np.random.seed(0)
            x = np.random.rand(100, 1)
            y = 2 + 3 * x + np.random.rand(100, 1)
```

```
# plot
plt.scatter(x,y,s=10)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

The plot for the data set generated using the above code is shown below:

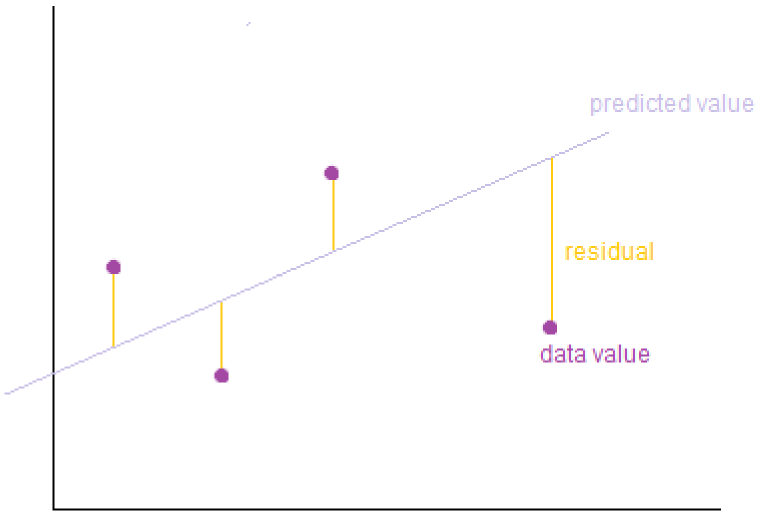


Training a Linear Regression Model

Training of the model here means to find the parameters so that the model best fits the data.

How do we determine the best fit line?

The line for which the the error between the predicted values and the observed values is minimum is called the best fit line or the regression line. These errors are also called as residuals. The residuals can be visualized by the vertical lines from the observed data value to the regression line.



To define and measure the error of our model we define the cost function as the sum of the squares of the residuals. The cost function is denoted by

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h(x^i) - y^i)^2$$

where the hypothesis function h(x) is denoted by

$$h(x) = \theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n$$

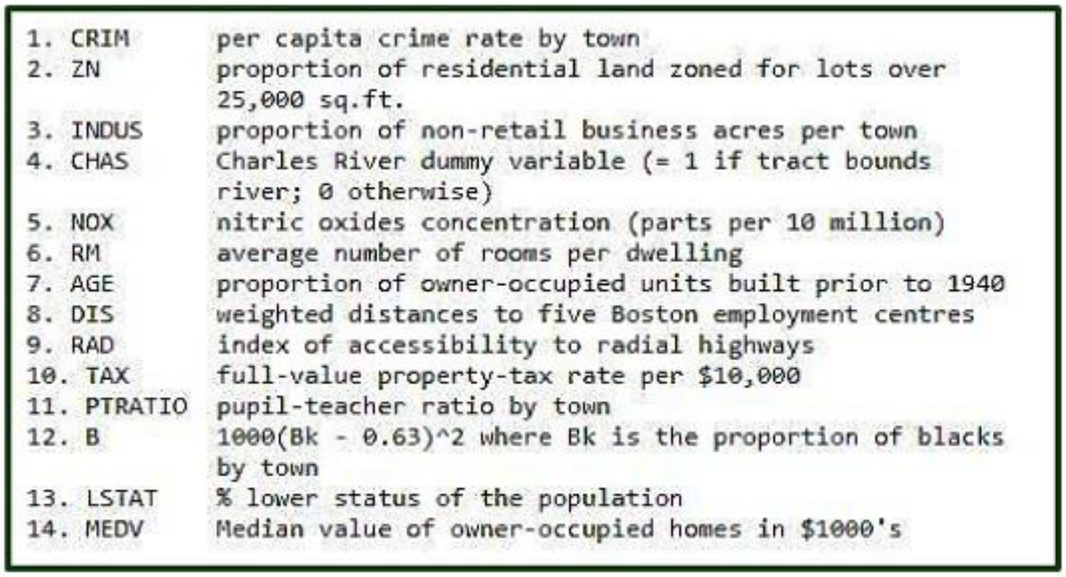
and m is the total number of training examples in our data-set.

DATASET: lib.stat.cmu.edu/datasets/boston

IMPLEMENTATION:

Boston Housing Data: This dataset was taken from the StatLib library and is maintained by Carnegie Mellon University. This dataset concerns the housing prices in the housing city of Boston. The dataset provided has 506 instances with 13 features.

The Description of the dataset is taken from the below reference as shown in the table follows:



```
1. CRIM      per capita crime rate by town
2. ZN        proportion of residential land zoned for lots over
             25,000 sq.ft.
3. INDUS     proportion of non-retail business acres per town
4. CHAS      Charles River dummy variable (= 1 if tract bounds
             river; 0 otherwise)
5. NOX       nitric oxides concentration (parts per 10 million)
6. RM        average number of rooms per dwelling
7. AGE       proportion of owner-occupied units built prior to 1940
8. DIS       weighted distances to five Boston employment centres
9. RAD       index of accessibility to radial highways
10. TAX      full-value property-tax rate per $10,000
11. PTRATIO  pupil-teacher ratio by town
12. B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks
             by town
13. LSTAT    % lower status of the population
14. MEDV     Median value of owner-occupied homes in $1000's
```

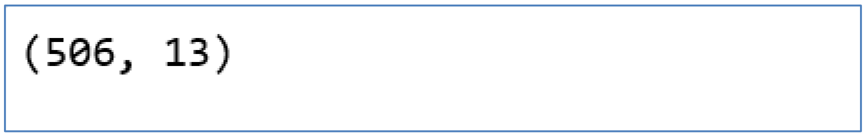
Make the Linear Regression Model, predicting housing prices by Inputting Libraries and datasets.

Step 1: import the required libraries.

```
# Importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Importing Data
from sklearn.datasets import load_boston
boston = load_boston()
```

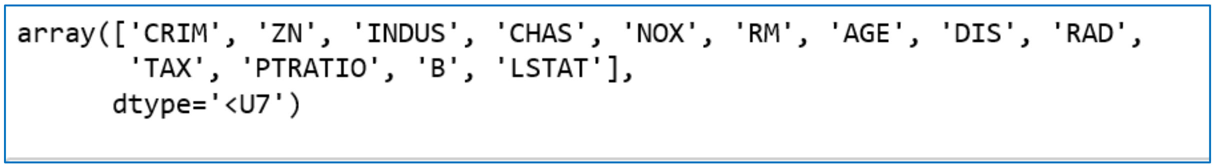
Step 2: The shape of input Boston data and getting feature_names.

```
boston.data.shape
```



```
(506, 13)
```

```
boston.feature_names
```



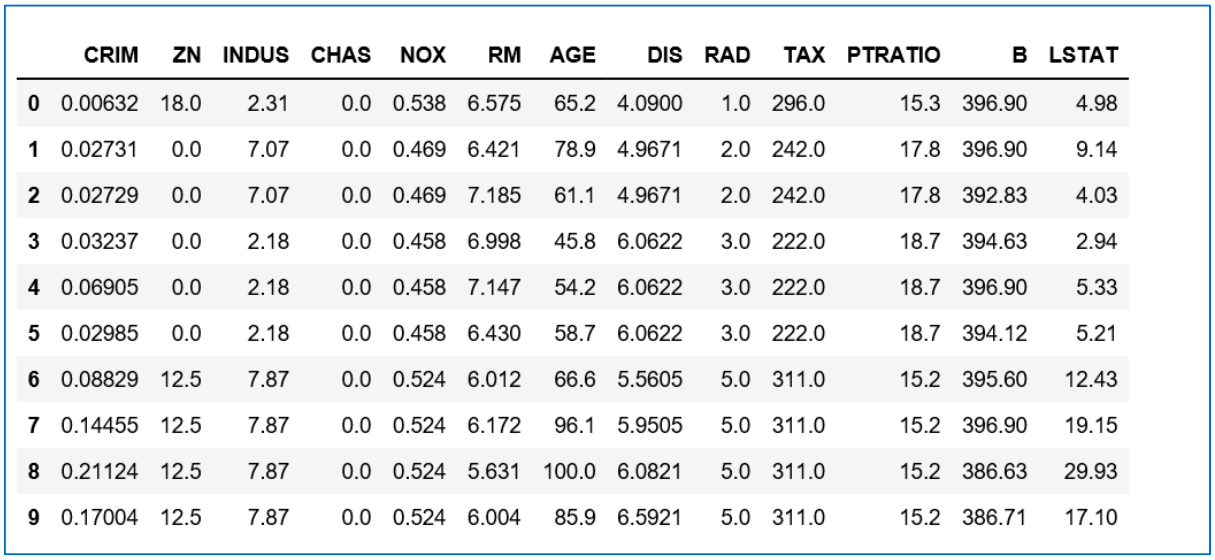
```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'],
      dtype='<U7')
```

Step 3: Converting data from nd-array to data frame and adding feature names to the data

```
data = pd.DataFrame(boston.data)
```

data.columns = boston.feature_names

data.head(10)

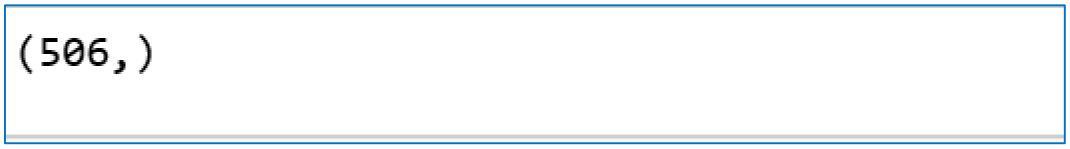


	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	5.21
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.60	12.43
7	0.14455	12.5	7.87	0.0	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396.90	19.15
8	0.21124	12.5	7.87	0.0	0.524	5.631	100.0	6.0821	5.0	311.0	15.2	386.63	29.93
9	0.17004	12.5	7.87	0.0	0.524	6.004	85.9	6.5921	5.0	311.0	15.2	386.71	17.10

Step 4: Adding the 'Price' column to the dataset

# Adding 'Price' (target) column to the data

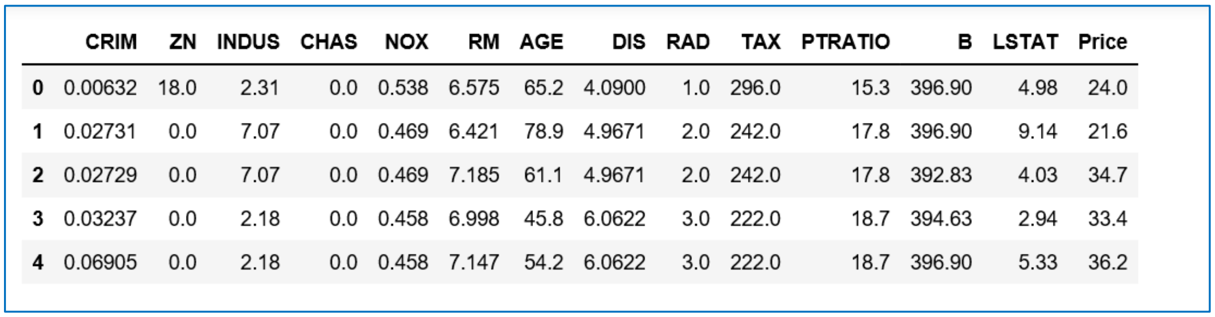
boston.target.shape



```
(506,)
```

data['Price'] = boston.target

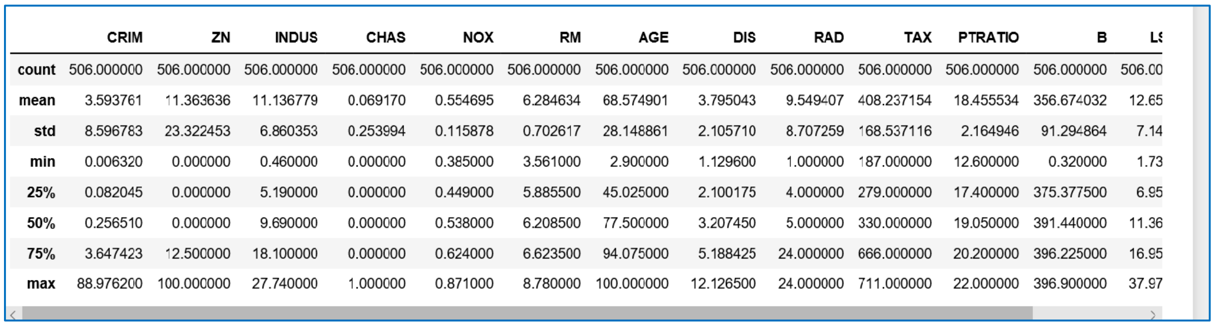
data.head()



	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Price
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

Step 5: Description of Boston dataset

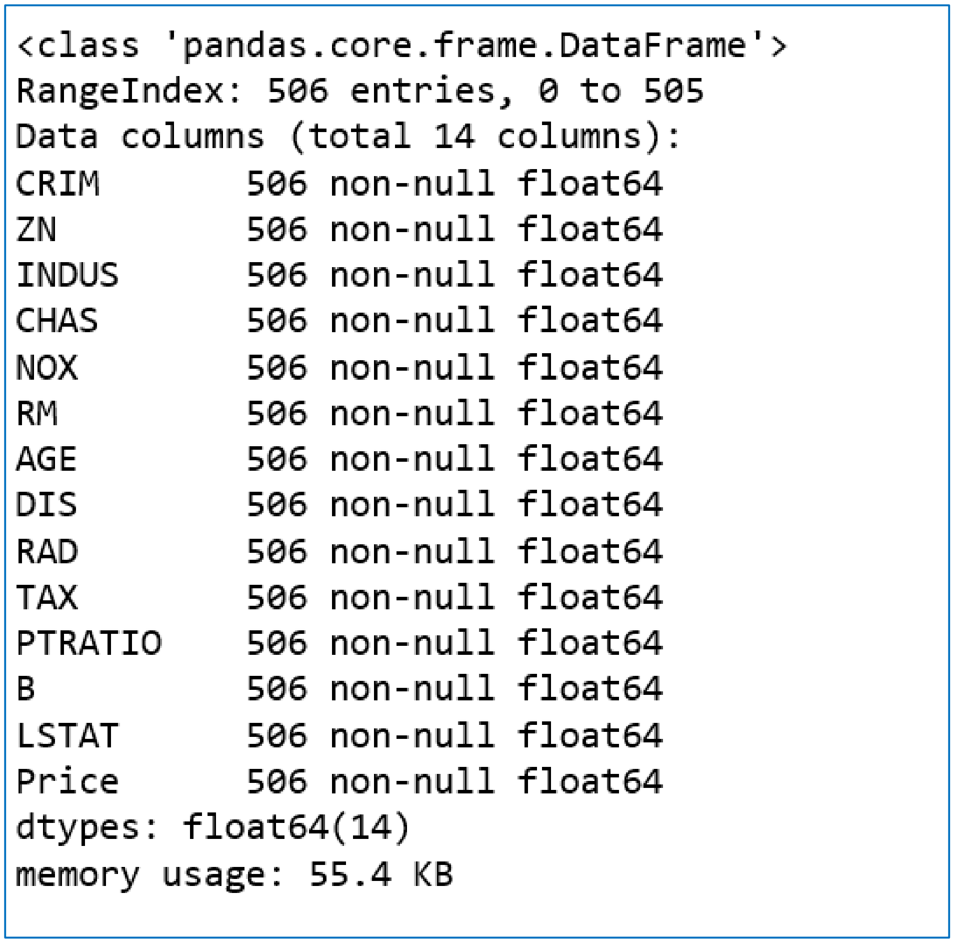
data.describe()



	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.00
mean	3.593761	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.65
std	8.596783	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.14
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000	1.73
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.95
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.36
75%	3.647423	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.95
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37.97

Step 6: Info of Boston Dataset

data.info()

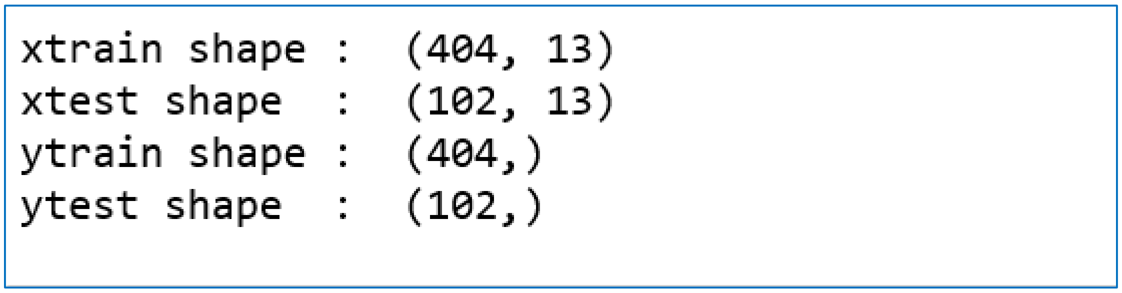


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
CRIM       506 non-null float64
ZN         506 non-null float64
INDUS      506 non-null float64
CHAS       506 non-null float64
NOX        506 non-null float64
RM         506 non-null float64
AGE        506 non-null float64
DIS        506 non-null float64
RAD        506 non-null float64
TAX        506 non-null float64
PTRATIO    506 non-null float64
B          506 non-null float64
LSTAT      506 non-null float64
Price      506 non-null float64
dtypes: float64(14)
memory usage: 55.4 KB
```

Step 7: Getting input and output data and further splitting data to training and testing dataset.

```
# Input Data
x = boston.data

# Output Data
y = boston.target
```

```
# splitting data to training and testing dataset.

#from sklearn.cross_validation import train_test_split
#the submodule cross_validation is renamed and deprecated to model_selection
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size =0.2, random_state = 0)

print("xtrain shape : ", xtrain.shape)
print("xtest shape : ", xtest.shape)
print("ytrain shape : ", ytrain.shape)
print("ytest shape : ", ytest.shape)
```



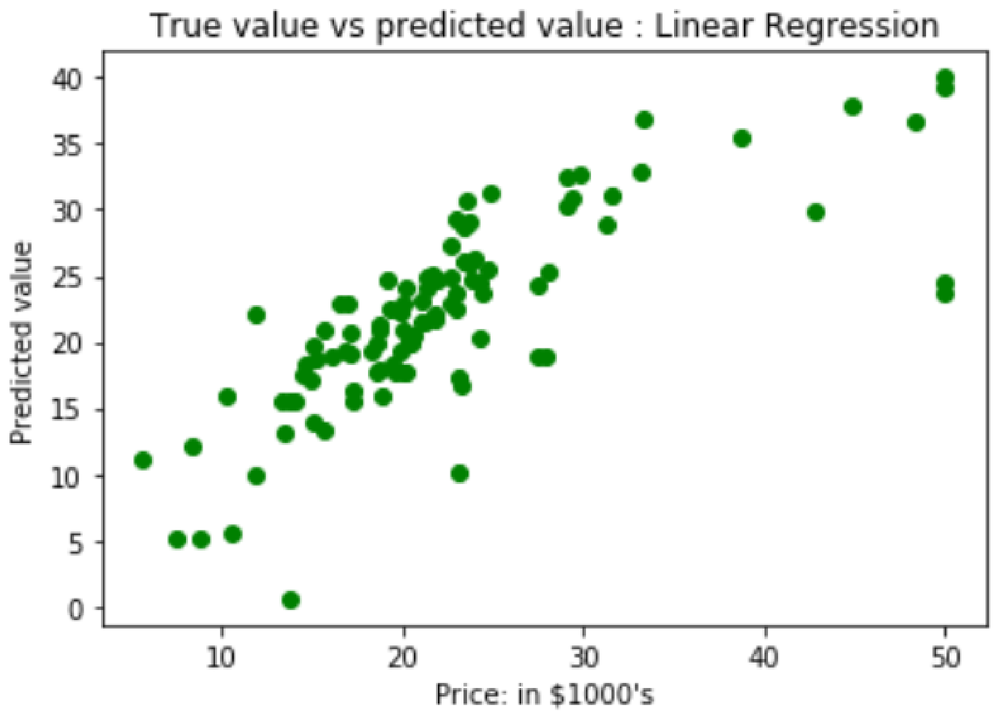
```
xtrain shape :  (404, 13)
xtest shape  :  (102, 13)
ytrain shape :  (404,)
ytest shape  :  (102,)
```

Step 8: Applying Linear Regression Model to the dataset and predicting the prices.

```
# Fitting Multi Linear regression model to training model
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(xtrain, ytrain)

# predicting the test set results
y_pred = regressor.predict(xtest)
```

Step 9: Plotting Scatter graph to show the prediction results – ‘y_true’ value vs ‘y_pred’ value.

```
# Plotting Scatter graph to show the prediction
# results - 'ytrue' value vs 'y_pred' value
plt.scatter(ytest, y_pred, c = 'green')
plt.xlabel("Price: in $1000's")
plt.ylabel("Predicted value")
plt.title("True value vs predicted value : Linear Regression")
plt.show()
```



Step 10: Results of Linear Regression i.e. Mean Squared Error and Mean Absolute Error.

```
from sklearn.metrics import mean_squared_error, mean_absolute_error
mse = mean_squared_error(ytest, y_pred)
mae = mean_absolute_error(ytest,y_pred)
print("Mean Square Error : ", mse)
print("Mean Absolute Error : ", mae)
```

```
Mean Square Error :  33.448979997676496
Mean Absolute Error :  3.8429092204444966
```

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left[h_\theta\left(x^i\right) - y_i \right]^2$$

As per the result, our model is only 66.55% accurate. So, the prepared model is not very good for predicting housing prices. One can improve the prediction results using many other possible machine learning algorithms and techniques.

CONCLUSION:

Thus, we have learnt about the concepts of Linear Regression.

EXPERIMENT 2

AIM: Classification using Deep neural network

Objective:
To classify movie reviews as positive or negative

Problem Statement:
Binary classification using Deep Neural Networks Example: Classify movie reviews into positive" reviews and "negative" reviews, just based on the text content of the reviews. Use IMDB dataset

Course Outcomes:
Apply the technique of Deep Neural network for implementing binary classification.

Software-Hardware requirements:
Python, Ubuntu Operating System 14.0 or above, 2 CPU CoreProcessor, 2GB RAM.

THEORY:

Binary Classification

Binary Classification refers to classifying samples in one of two categories. In this example, we will design a neural network to perform two-class classification, or binary classification, of reviews, from the IMDB movie reviews dataset, to determine whether the reviews are positive or negative. We will use the Python library, Keras.

The IMDB Dataset

The IMDB dataset is a set of 50,000 highly polarized reviews from the Internet Movie Database. They are split into 25000 reviews each for training and testing. Each set contains an equal number (50%) of positive and negative reviews. The IMDB dataset comes packaged with Keras. It consists of reviews and their corresponding labels (0 for negative and 1 for positive review). The reviews are a sequence of words. They come preprocessed as a sequence of integers, where each integer stands for a specific word in the dictionary. The IMDB dataset can be loaded directly from Keras and will usually download about 80 MB on your machine.

Step 1: Loading the Data

Let's load the prepackaged data from Keras. We will only include 10,000 of the most frequently occurring words.

Step 2: Preparing the Data

We cannot feed a list of integers into our deep neural network. We will need to convert them into tensors.

To prepare our data, we will One-hot Encode our lists and turn them into vectors of 0's and 1's. This would blow up all of our sequences into 10,000-dimensional vectors containing 1 at all indices corresponding to integers present in that sequence. This vector will have element 0 at all index, which is not present in the integer sequence.

Simply put, the 10,000-dimensional vector corresponding to each review will have

- Every index corresponding to a word
- Every index with value 1, is a word that is present in the review and is denoted by its integer counterpart.
- Every index containing 0 is a word not present in the review.

We will vectorize our data manually for maximum clarity. This will result in a tensor of shape (25000, 10000).

Step 3: Building the Neural Network

Our input data is vectors that need to be mapped to scaler labels (0s and 1s). This is one of the easiest setups, and a simple stack of fully-connected, Dense layers with relu activation perform quite well.

Step 4: Hidden layers

Hidden layers, simply put, are layers of mathematical functions each designed to produce an output specific to an intended result.

Hidden layers allow for the function of a neural network to be broken down into specific transformations of the data. Each hidden layer function is specialized to produce a defined output.For example, a hidden layer functions that are used to identify human eyes and ears may be used in conjunction by subsequent layers to identify faces in images. While the functions to identify eyes alone are not enough to independently recognize objects, they can function jointly within a neural network.

In this network, we will leverage hidden layers. We will define our layers as such.

Dense(16, activation='relu')

The argument being passed to each Dense layer, (16) is the number of hidden units of a layer.

The output from a Dense layer with relu activation is generated after a chain of tensor operations. This chain of operations is implemented as

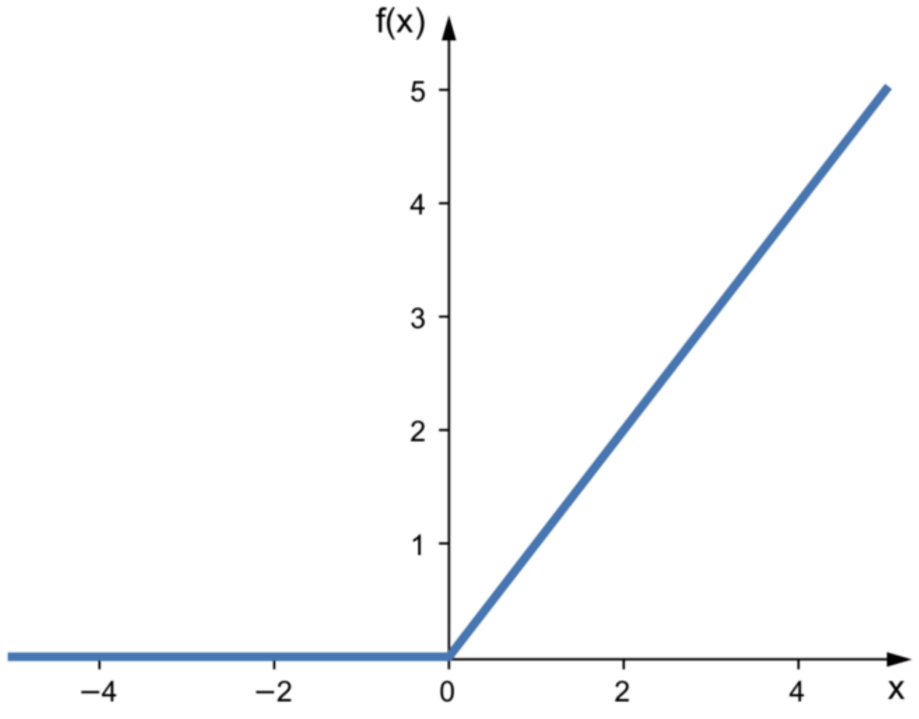
output = relu(dot(W, input) + b)

Where, W is the Weight matrix and b is the bias (tensor).

Having 16 hidden units means that the matrix W will be of the shape (input_Dimension, 16). In this case, where the dimension of the input vector is 10,000, the shape of the Weight matrix will be (10000, 16). If you were to represent this network as a graph, you would see 16 neurons in this hidden layer.

To put it in layman’s terms, there will be 16 balls in this layer.

Each of these balls or hidden units is a dimension in the representation space of the layer. Representation space is the set of all viable representations for the data. Every hidden layer composed of its hidden units aims to learn one specific transformation of the data or one feature/pattern from the data.



ReLU activation function

Model Architecture

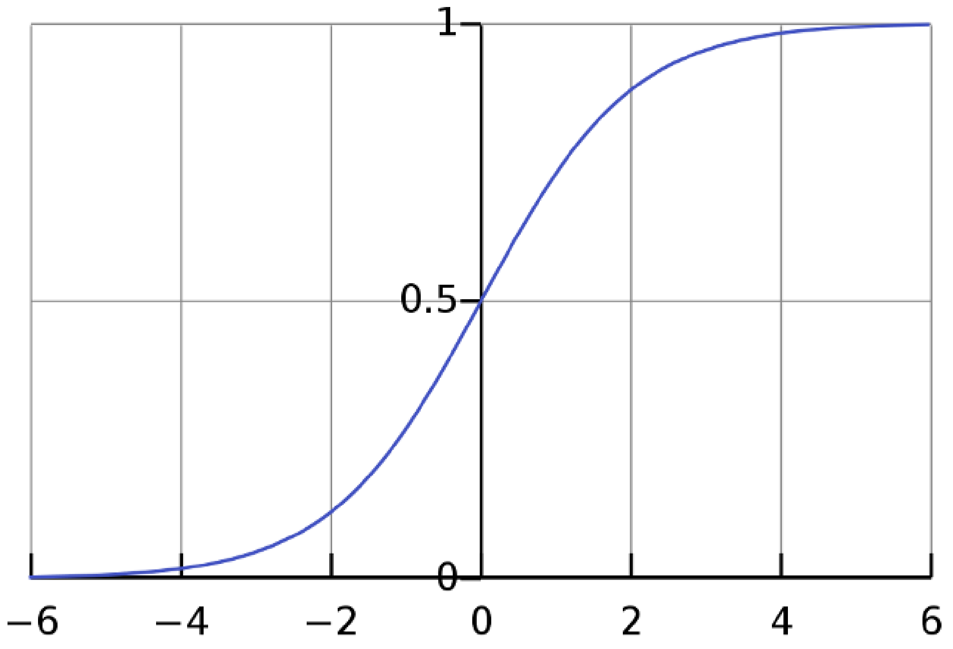
For our model, we will use

Two intermediate layers with 16 hidden units each

Third layer that will output the scalar sentiment prediction

Intermediate layers will use the relu activation function. relu or Rectified linear unit function will zero out the negative values.

Sigmoid activation for the final layer or output layer. A sigmoid function “squashes” arbitrary values into the [0,1] range.



The Sigmoid Activation Function

There are formal principles that guide our approach in selecting the architectural attributes of a model.

Compiling the model

In this step, we will choose an optimizer, a loss function, and metrics to observe. We will go forward with

- binary_crossentropy loss function, commonly used for Binary Classification
- rmsprop optimizer and
- accuracy as a measure of performance

We can pass our choices for optimizer, loss function and metrics as strings to the compile function because rmsprop, binary_crossentropy and accuracy come packaged with Keras.

```
model.complie(optimizer='rmsprop',
        loss = 'binary_crossentropy',
        metrics = ['accuracy'])
```

One could use a customized loss function or optimizer by passing a custom class instance as an argument to the loss, optimizer or mertics fields.

In this example, we will implement our default choices, but we will do so by passing class instances. This is precisely how we would do it if we had customized parameters.

Setting up Validation

We will set aside a part of our training data for validation of the accuracy of the model as it trains. A validation set enables us to monitor the progress of our model on previously unseen data as it goes through epochs during training.

Validation steps help us fine-tune the training parameters of the model.fit function to avoid Overfitting and underfitting of data.

Training our model

Initially, we will train our models for 20 epochs in mini-batches of 512 samples. We will also pass our validation set to the fit method.

Calling the fit method returns a History object. This object contains a member history which stores all data about the training process, including the values of observable or monitored quantities as the epochs proceed. We will save this object to determine the fine-tuning better to apply to the training step.

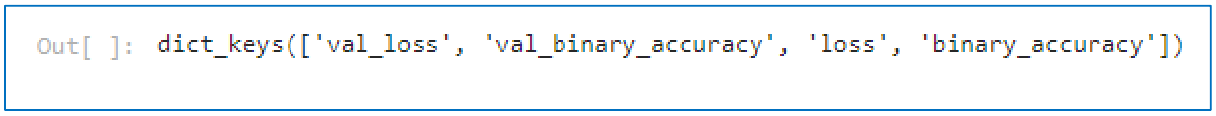
At the end of the training, we have attained a training accuracy of 99.85% and validation accuracy of 86.57%

Now that we have trained our network, we will observe its performance metrics stored in the History object.

Calling the fit method returns a History object. This object has an attribute history which is a dictionary containing four entries: one per monitored metric.

history_dict = history.history

history_dict.keys()



```
Out[ ]: dict_keys(['val_loss', 'val_binary_accuracy', 'loss', 'binary_accuracy'])
```

history_dict contains values of

- Training loss
- Training Accuracy
- Validation Loss
- Validation Accuracy
- at the end of each epoch.

Let’s use Matplotlib to plot Training and validation losses and Training and Validation Accuracy side by side.

We observe that minimum validation loss and maximum validation Accuracy is achieved at around 3–5 epochs. After that, we observe two trends:

- increase in validation loss and a decrease in training loss
- decrease in validation accuracy and an increase in training accuracy

This implies that the model is getting better at classifying the sentiment of the training data, but making consistently worse predictions when it encounters new, previously unseen data. This is the hallmark of Overfitting. After the 5th epoch, the model begins to fit too closely to the training data.

To address Overfitting, we will reduce the number of epochs to somewhere between 3 and 5. These results may vary depending on your machine and due to the very nature of the random assignment of weights that may vary from model to model.

In our case, we will stop training after 3 epochs.

Retraining our Neural Network

We retrain our neural network based on our findings from studying the history of loss and accuracy variation. This time we run it for 3 epochs so as to avoid Overfitting on training data.

In the end, we achieve a training accuracy of 99% and a validation accuracy of 86%. This is pretty good, considering we are using a very naive approach. A higher degree of accuracy can be achieved by using a better training algorithm.

Evaluating the model performance

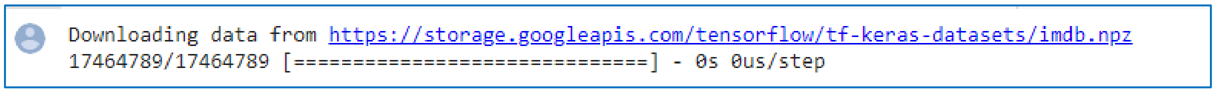
We will use our trained model to make predictions for the test data. The output is an array of floating integers that denote the probability of a review being positive. As you can see, in some cases, the network is absolutely sure the review is positive.

DATASET: https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews?resource=download

IMPLEMENTATION:

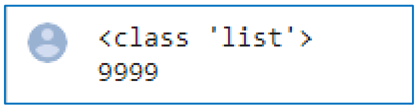
Step 1: Loading the Data

```
from keras.datasets import imdb

# Load the data, keeping only 10,000 of the most frequently occuring 
words
(train_data, train_labels), (test_data, test_labels) = imdb.load_data
(num_words = 10000)
```



```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 0s 0us/step
```

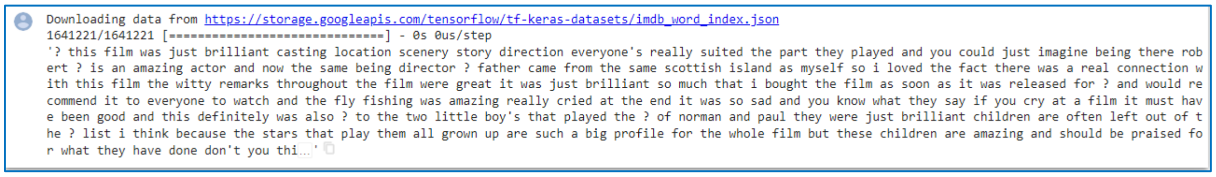
```
# Since we restricted ourselves to the top 10000 frequent words, no w
ord index should exceed 10000
# we'll verify this below

# Here is a list of maximum indexes in every review --
- we search the maximum index in this list of max indexes
print(type([max(sequence) for sequence in train_data]))

# Find the maximum of all max indexes
max([max(sequence) for sequence in train_data])
```



```
<class 'list'>
9999
```

For kicks, let’s decode the first review.

```
# Let's quickly decode a review

# step 1: load the dictionary mappings from word to integer index
word_index = imdb.get_word_index()

# step 2: reverse word index to map integer indexes to their respecti
ve words
reverse_word_index = dict([(value, key) for (key, value) in word_inde
x.items()])

# Step 3: decode the review, mapping integer indices to words
#
# indices are off by 3 because 0, 1, and 2 are reserverd indices for
"padding", "Start of sequence" and "unknown"
decoded_review = ' '.join([reverse_word_index.get(i-
3, '?') for i in train_data[0]])

decoded_review
```

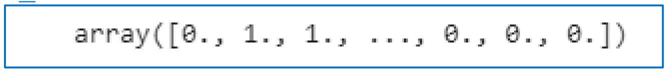


```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1641221/1641221 [==============================] - 0s 0us/step
'? this film was just brilliant casting location scenery story direction everyone's really suited the part they played and you could just imagine being there rob
ert ? is an amazing actor and now the same being director ? father came from the same scottish island as myself so i loved the fact there was a real connection w
ith this film the witty remarks throughout the film were great it was just brilliant so much that i bought the film as soon as it was released for ? and would re
commend it to everyone to watch and the fly fishing was amazing really cried at the end it was so sad and you know what they say if you cry at a film it must hav
e been good and this definitely was also ? to the two little boy's that played the ? of norman and paul they were just brilliant children are often left out of t
he ? list i think because the stars that play them all grown up are such a big profile for the whole film but these children are amazing and should be praised fo
r what they have done don't you thi…'
```

Step 2: Preparing the Data
Vectorize input data

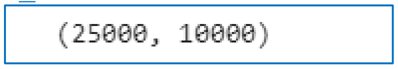
```
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))    # Creates an a
ll zero matrix of shape (len(sequences),10K)
    for i,sequence in enumerate(sequences):
        results[i,sequence] = 1                        # Sets specifi
c indices of results[i] to 1s
    return results

# Vectorize training Data
X_train = vectorize_sequences(train_data)

# Vectorize testing Data
X_test = vectorize_sequences(test_data)
```

```
X_train[0]
```



```
array([0., 1., 1., ..., 0., 0., 0.])
```

```
X_train.shape
```



```
(25000, 10000)
```

Vectorize labels

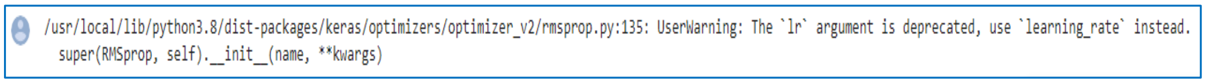
```
y_train = np.asarray(train_labels).astype('float32')
y_test  = np.asarray(test_labels).astype('float32')
```

Step 3: Model definition

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Step 4: Compiling the model

```
from keras import optimizers
from keras import losses
from keras import metrics

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss = losses.binary_crossentropy,
              metrics = [metrics.binary_accuracy])
```



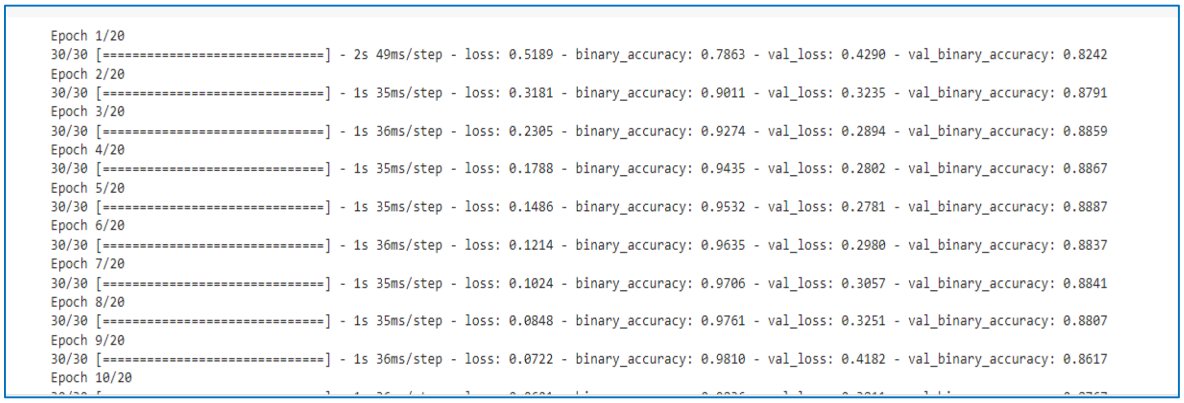
```
/usr/local/lib/python3.8/dist-packages/keras/optimizers/optimizer_v2/rmsprop.py:135: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(RMSprop, self).__init__(name, **kwargs)
```

Step 5: Setting up Validation

```
# Input for Validation
X_val = X_train[:10000]
partial_X_train = X_train[10000:]

# Labels for validation
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

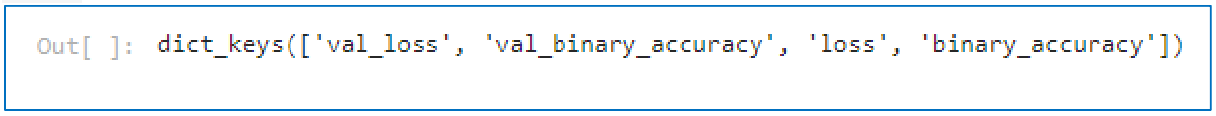
Step 6: Training our model

```
history = model.fit(partial_X_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(X_val, y_val))
```



```
Epoch 1/20
30/30 [==============================] - 2s 49ms/step - loss: 0.5189 - binary_accuracy: 0.7863 - val_loss: 0.4290 - val_binary_accuracy: 0.8242
Epoch 2/20
30/30 [==============================] - 1s 35ms/step - loss: 0.3181 - binary_accuracy: 0.9011 - val_loss: 0.3235 - val_binary_accuracy: 0.8791
Epoch 3/20
30/30 [==============================] - 1s 36ms/step - loss: 0.2305 - binary_accuracy: 0.9274 - val_loss: 0.2894 - val_binary_accuracy: 0.8859
Epoch 4/20
30/30 [==============================] - 1s 35ms/step - loss: 0.1788 - binary_accuracy: 0.9435 - val_loss: 0.2802 - val_binary_accuracy: 0.8867
Epoch 5/20
30/30 [==============================] - 1s 35ms/step - loss: 0.1486 - binary_accuracy: 0.9532 - val_loss: 0.2781 - val_binary_accuracy: 0.8887
Epoch 6/20
30/30 [==============================] - 1s 36ms/step - loss: 0.1214 - binary_accuracy: 0.9635 - val_loss: 0.2980 - val_binary_accuracy: 0.8837
Epoch 7/20
30/30 [==============================] - 1s 35ms/step - loss: 0.1024 - binary_accuracy: 0.9706 - val_loss: 0.3057 - val_binary_accuracy: 0.8841
Epoch 8/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0848 - binary_accuracy: 0.9761 - val_loss: 0.3251 - val_binary_accuracy: 0.8807
Epoch 9/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0722 - binary_accuracy: 0.9810 - val_loss: 0.4182 - val_binary_accuracy: 0.8617
Epoch 10/20
```

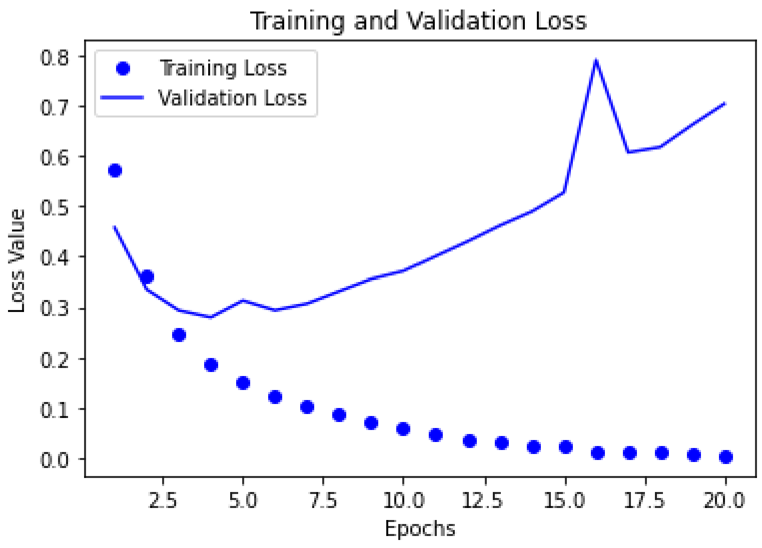
```
history_dict = history.history
history_dict.keys()
```



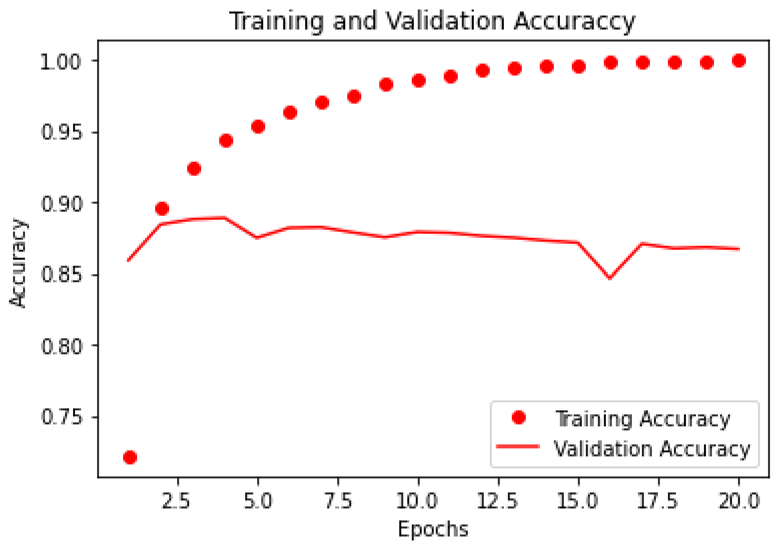
```
Out[ ]: dict_keys(['val_loss', 'val_binary_accuracy', 'loss', 'binary_accuracy'])
```

Step 7: Matplotlib to plot Training and validation losses and Training and Validation Accuracy side by side

```
import matplotlib.pyplot as plt
%matplotlib inline

# Plotting losses
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

epochs = range(1, len(loss_values) + 1)

plt.plot(epochs, loss_values, 'bo', label="Training Loss")
plt.plot(epochs, val_loss_values, 'b', label="Validation Loss")

plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss Value')
plt.legend()

plt.show()
```

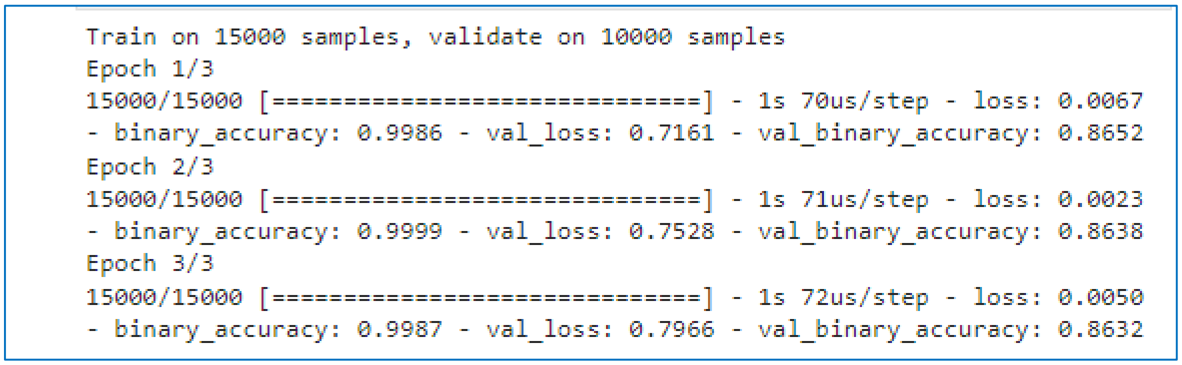


```
# Training and Validation Accuracy

acc_values = history_dict['binary_accuracy']
val_acc_values = history_dict['val_binary_accuracy']

epochs = range(1, len(loss_values) + 1)

plt.plot(epochs, acc_values, 'ro', label="Training Accuracy")
plt.plot(epochs, val_acc_values, 'r', label="Validation Accuracy")

plt.title('Training and Validation Accuraccy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



Step 8: Retraining our model

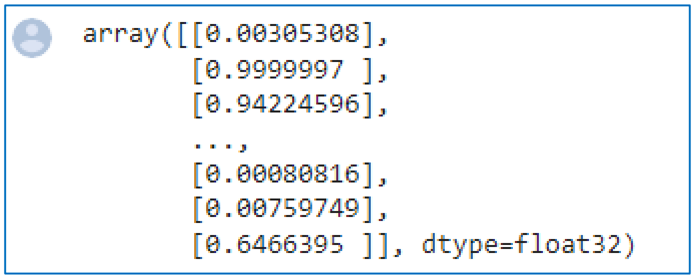
```
model.fit(partial_X_train,
                         partial_y_train,
                         epochs=3,
                         batch_size=512,
                         validation_data=(X_val, y_val))
```



```
Train on 15000 samples, validate on 10000 samples
Epoch 1/3
15000/15000 [==============================] - 1s 70us/step - loss: 0.0067
- binary_accuracy: 0.9986 - val_loss: 0.7161 - val_binary_accuracy: 0.8652
Epoch 2/3
15000/15000 [==============================] - 1s 71us/step - loss: 0.0023
- binary_accuracy: 0.9999 - val_loss: 0.7528 - val_binary_accuracy: 0.8638
Epoch 3/3
15000/15000 [==============================] - 1s 72us/step - loss: 0.0050
- binary_accuracy: 0.9987 - val_loss: 0.7966 - val_binary_accuracy: 0.8632
```

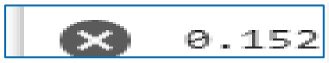
Step 9: Model Evaluation

```
# Making Predictions for testing data
np.set_printoptions(suppress=True)
result = model.predict(X_test)


result
```



```
array([[0.00305308],
       [0.9999997 ],
       [0.94224596],
       ...,
       [0.00080816],
       [0.00759749],
       [0.6466395 ]], dtype=float32)
```

```
y_pred = np.zeros(len(result))
for i, score in enumerate(result):
     y_pred[i] = 1 if score > 0.5 else 0

from sklearn.metrics import mean_absolute_error
                     mae = mean_absolute_error(y_pred, y_test)

# Error
mae
```



```
0.152
```

CONCLUSION:

Thus, we have successfully classified reviews on IMDB.

EXPERIMENT 3

AIM: Convolutional neural network (CNN)

Course Objective:
To classify fashion clothing into categories using CNN model.

Problem Statement:
Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories.

Course Outcomes:
Apply the technique of Convolution (CNN) for implementing Deep Learning models.

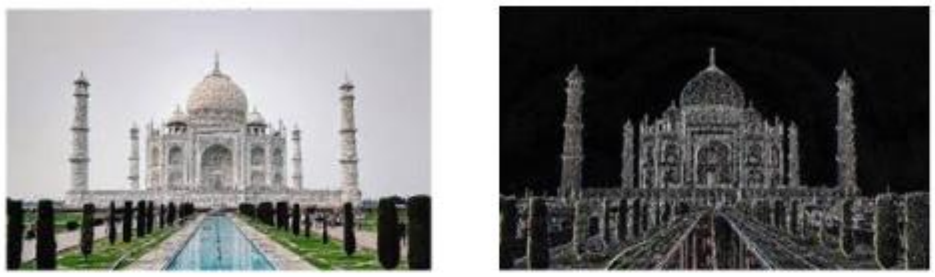
Software-Hardware requirements:
Python, Ubuntu Operating System 14.0 or above, 2 CPU CoreProcessor, 2GB RAM.

THEORY:

Convolutional neural network (CNN)

CNN is basically a model known to be Convolutional Neural Network and in recent times it has gained a lot of popularity because of its usefulness. CNN uses multilayer perceptrons to do computational works. CNN uses relatively little pre-processing compared to other image classification algorithms. This means the network learns through filters that in traditional algorithms were hand-engineered. So, for the image processing tasks CNNs are the best-suited option.
These CNN models are being used across different applications and domains, and they're especially prevalent in image and video processing projects.
A Convolutional Neural Network is a powerful neural network that uses filters to extract features from images. It also does so in such a way that position information of pixels is retained.
A convolution is a mathematical operation applied on a matrix. This matrix is usually the image represented in the form of pixels/numbers. The convolution operation extracts the features from the image.

The building blocks of CNNs are filters a.k.a. kernels. Kernels are used to extract the relevant features from the input using the convolution operation.
Let's try to grasp the importance of filters using images as input data. Convolving an image with filters results in a feature map:



Output of Convolution

They can identify faces, individuals, street signs, platypuses, and many other aspects of visual data. CNNs overlap with text analysis via optical character recognition, but they are also useful when

analyzing words as discrete textual units. They’re also good at analyzing sound.
CNN captures the spatial features from an image. Spatial features refer to the arrangement of pixels and the relationship between them in an image. They help us in identifying the object accurately, the location of an object, as well as its relation with other objects in an image
In the above image, we can easily identify that its a human’s face by looking at specific features like eyes, nose, mouth and so on. We can also see how these specific features are arranged in an image. That’s exactly what CNNs are capable of capturing.

CNN also follows the concept of parameter sharing. A single filter is applied across different parts of an input to produce a feature map:
The number of filters (kernel) you will use on the input will result in same amount of feature maps. Think of filter like a membrane that allows only the desired qualities of the input to pass through it.

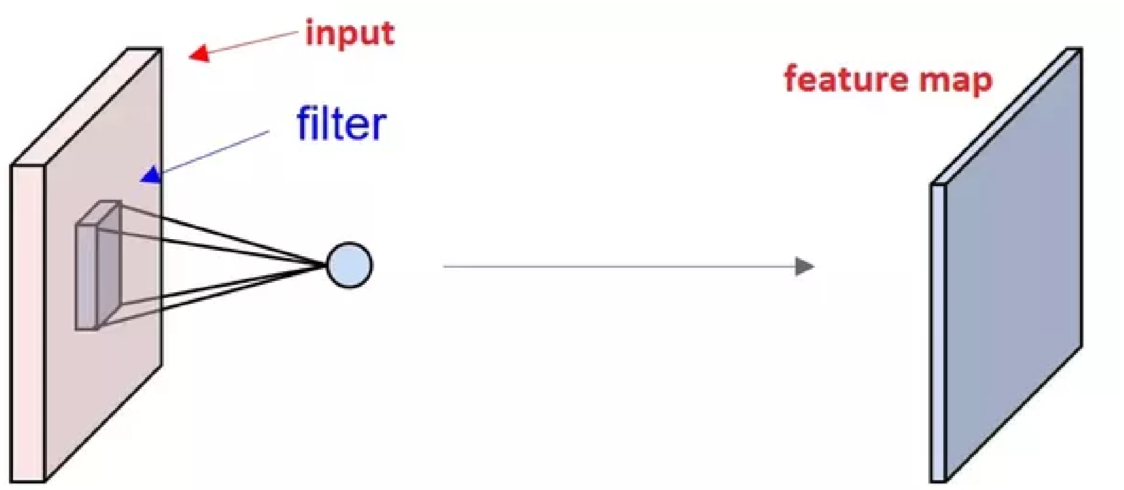
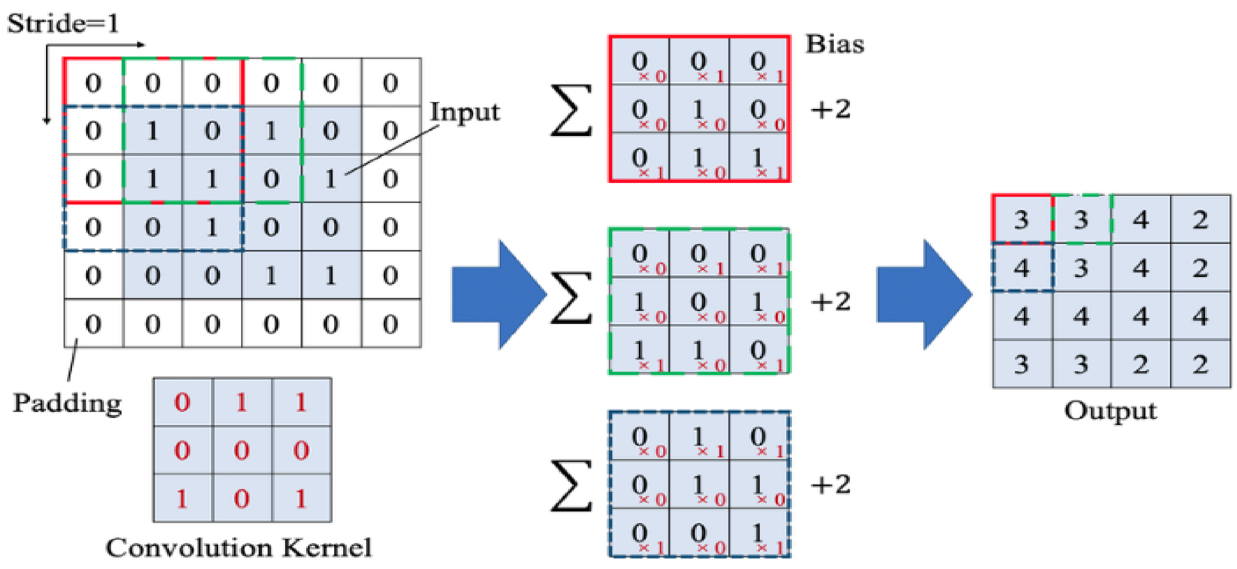
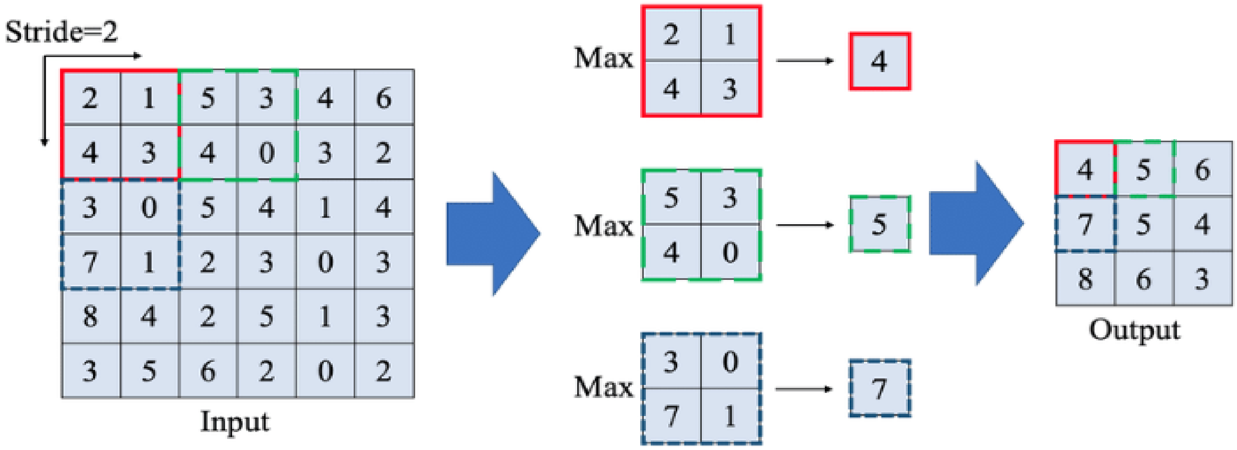


Figure: CNN

Example of Convolution Layer & Maxpooling Layer

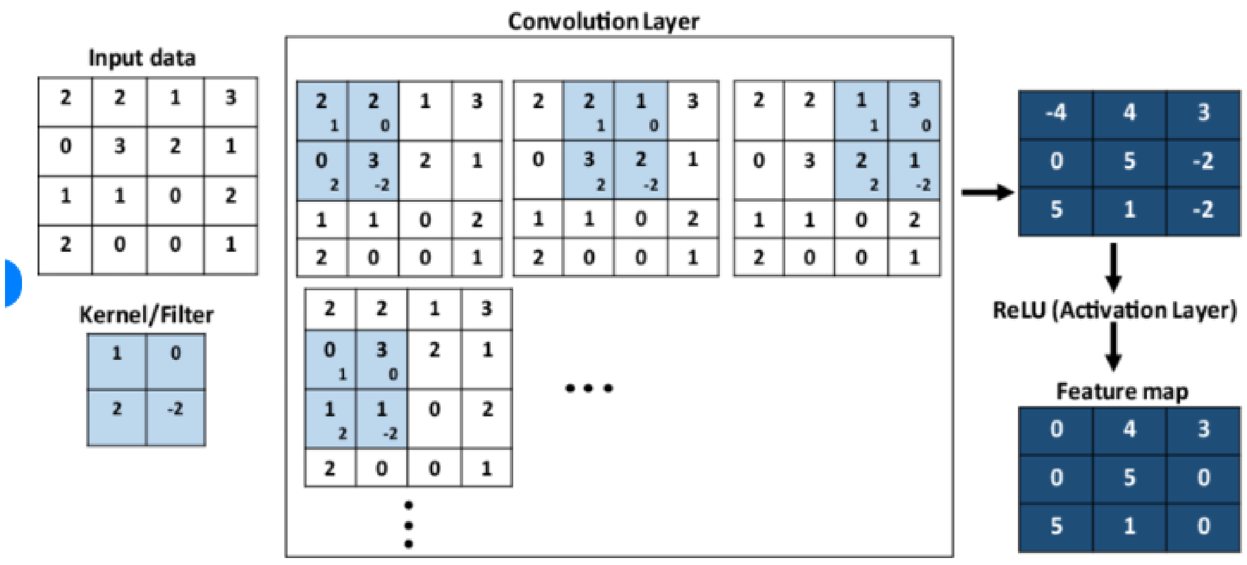


Example of Convolutional layer.



Example of MaxPooling layer.

Producing a feature map by applying convolution and the activation function ReLU to the Input Data



Producing a feature map by applying convolution and the activation function ReLU to the input data

MNIST dataset:
mnist dataset is a dataset of handwritten images as shown below in the image.

We can get 99.06% accuracy by using CNN(Convolutional Neural Network) with a functional model. The reason for using a functional model is to maintain easiness while connecting the layers.

DATASET: https://www.kaggle.com/datasets/oddrationale/mnist-in-csv

IMPLEMENTATION:

Step 1: Firstly, include all necessary libraries

```
import numpy as np
import keras
from keras.datasets import mnist
from keras.models import Model
from keras.layers import Dense, Input
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten
from keras import backend as k
```

Step 2: Create the train data and test data

Test data: Used for testing the model that how our model has been trained.
Train data: Used to train our model.

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

While proceeding further, img_rows and img_cols are used as the image dimensions. In mnist dataset, it is 28 and 28. We also need to check the data format i.e. ‘channels_first’ or ‘channels_last’. In CNN, we can normalize data before hands such that large terms of the calculations can be reduced to smaller terms. Like, we can normalize the x_train and x_test data by dividing it by 255.
Checking data-format:

```
img_rows, img_cols=28, 28

if k.image_data_format() == 'channels_first':
  x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
  x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
  inpx = (1, img_rows, img_cols)

else:
  x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
  x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
  inpx = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

Step 3: Description of the output classes:

Since the output of the model can comprise any of the digits between 0 to 9. so, we need 10 classes in output. To make output for 10 classes, use keras.utils.to_categorical function, which will provide the 10 columns. Out of these 10 columns, only one value will be one and the rest 9 will be zero and this one value of the output will denote the class of the digit.

```
y_train = keras.utils.to_categorical(y_train)
```

```
y_test = keras.utils.to_categorical(y_test)
```

- Now, the dataset is ready so let’s move towards the CNN model :

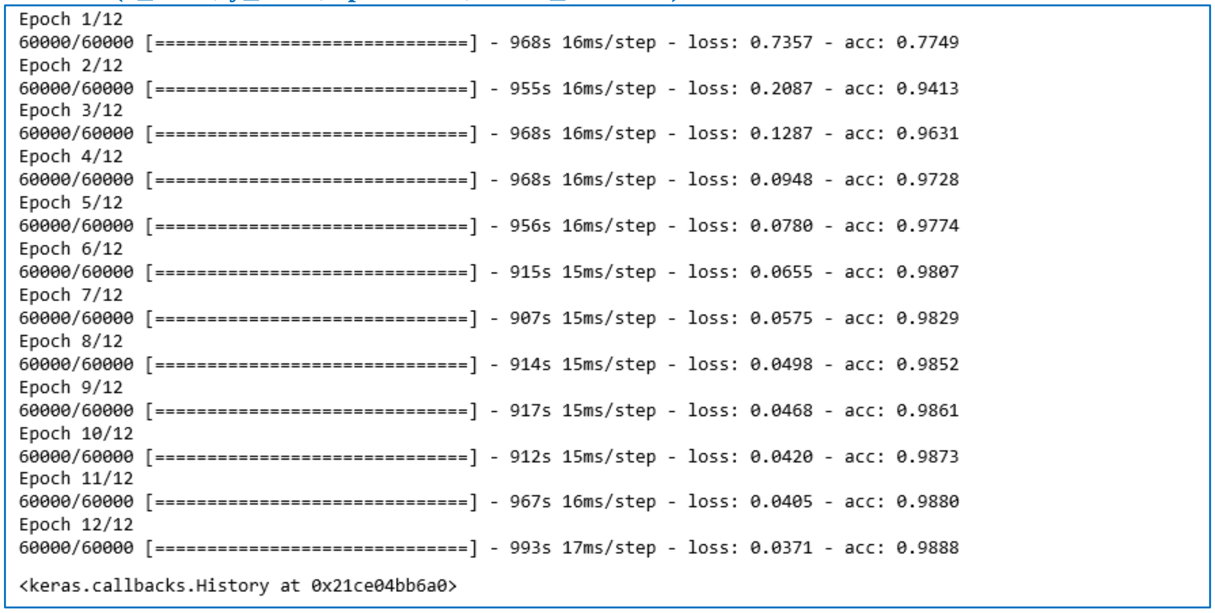
```
inpx = Input(shape=inpx)
layer1 = Conv2D(32, kernel_size=(3, 3), activation='relu')(inpx)
layer2 = Conv2D(64, (3, 3), activation='relu')(layer1)
layer3 = MaxPooling2D(pool_size=(3, 3))(layer2)
layer4 = Dropout(0.5)(layer3)
layer5 = Flatten()(layer4)
layer6 = Dense(250, activation='sigmoid')(layer5)
layer7 = Dense(10, activation='softmax')(layer6)
```

- **Explanation of the working of each layer in the CNN model:**

layer1 is the Conv2d layer which convolves the image using 32 filters each of size (3*3).
layer2 is again a Conv2D layer which is also used to convolve the image and is using 64 filters each of size (3*3).
layer3 is the MaxPooling2D layer which picks the max value out of a matrix of size (3*3).
layer4 is showing Dropout at a rate of 0.5.
layer5 is flattening the output obtained from layer4 and this flattens output is passed to layer6.
layer6 is a hidden layer of a neural network containing 250 neurons.
layer7 is the output layer having 10 neurons for 10 classes of output that is using the softmax function.

- **Calling compile and fit function:**

```
model = Model([inpx], layer7)
model.compile(optimizer=keras.optimizers.Adadelta(),
                          loss=keras.losses.categorical_crossentropy,
                          metrics=['accuracy'])

model.fit(x_train, y_train, epochs=12, batch_size=500)
```



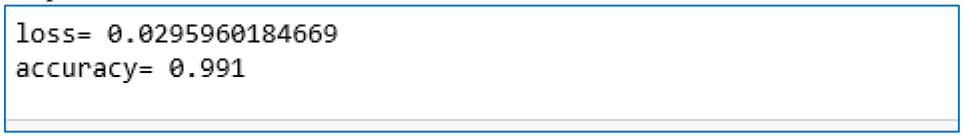
```
Epoch 1/12
60000/60000 [==============================] - 968s 16ms/step - loss: 0.7357 - acc: 0.7749
Epoch 2/12
60000/60000 [==============================] - 955s 16ms/step - loss: 0.2087 - acc: 0.9413
Epoch 3/12
60000/60000 [==============================] - 968s 16ms/step - loss: 0.1287 - acc: 0.9631
Epoch 4/12
60000/60000 [==============================] - 968s 16ms/step - loss: 0.0948 - acc: 0.9728
Epoch 5/12
60000/60000 [==============================] - 956s 16ms/step - loss: 0.0780 - acc: 0.9774
Epoch 6/12
60000/60000 [==============================] - 915s 15ms/step - loss: 0.0655 - acc: 0.9807
Epoch 7/12
60000/60000 [==============================] - 907s 15ms/step - loss: 0.0575 - acc: 0.9829
Epoch 8/12
60000/60000 [==============================] - 914s 15ms/step - loss: 0.0498 - acc: 0.9852
Epoch 9/12
60000/60000 [==============================] - 917s 15ms/step - loss: 0.0468 - acc: 0.9861
Epoch 10/12
60000/60000 [==============================] - 912s 15ms/step - loss: 0.0420 - acc: 0.9873
Epoch 11/12
60000/60000 [==============================] - 967s 16ms/step - loss: 0.0405 - acc: 0.9880
Epoch 12/12
60000/60000 [==============================] - 993s 17ms/step - loss: 0.0371 - acc: 0.9888

<keras.callbacks.History at 0x21ce04bb6a0>
```

Firstly, we made an object of the model as shown in the above-given lines, where [inpx] is the input in the model and layer7 is the output of the model. We compiled the model using the required optimizer, loss function and printed the accuracy and at the last model.fit was called along with parameters like x_train(means image vectors), y_train(means the label), number of epochs, and the batch size. Using fit function x_train, y_train dataset is fed to model in particular batch size.

Step 4: Evaluate function:
model.evaluate provides the score for the test data i.e. provided the test data to the model. Now, the model will predict the class of the data, and the predicted class will be matched with the y_test label to give us the accuracy.

```
score = model.evaluate(x_test, y_test, verbose=0)
print('loss=', score[0])
print('accuracy=', score[1])
```

Output:



```
loss= 0.0295960184669
accuracy= 0.991
```

CONCLUSION:

Thus, we have learnt how to classify fashion clothing into categories using CNN.

Group 2

MINI PROJECT

AIM: Mini Project: Human Face Recognition

Course Objective:
To illustrate the concepts of Artificial Intelligence/Machine Learning(AI/ML).

Problem Statement:
Face Detection and Recognition is one of the areas of computer vision where the research actively happens. The applications of Face Recognition include Face Unlock, Security and Defense, etc.

Course Outcomes:
Identify and apply the suitable algorithms to solve AI/ML problems.

Software-Hardware requirements:
Python, Ubuntu Operating System 14.0 or above, 2 CPU CoreProcessor, 2GB RAM.

THEORY:
Face Detection and Recognition is one of the areas of computer vision where the research actively happens. The applications of Face Recognition include Face Unlock, Security and Defense, etc. Doctors and healthcare officials use face recognition to access the medical records and history of patients and better diagnose diseases.
In this python project, we are going to build a machine learning model that recognizes the persons from an image. We use the face_recognition API and OpenCV in our project.
Face recognition is the process of identifying or verifying a person’s face from photos and video frames.
Face detection is defined as the process of locating and extracting faces (location and size) in an image for use by a face detection algorithm.
Face recognition method is used to locate features in the image that are uniquely specified. The facial picture has already been removed, cropped, scaled, and converted to grayscale in most cases.
Face recognition involves 3 steps: face detection, feature extraction, face recognition.
OpenCV is an open-source library written in C++. It contains the implementation of various algorithms and deep neural networks used for computer vision tasks.

Tools and Libraries
- Python – 3.x
- cv2 – 4.5.2
- numpy – 1.20.3
- face_recognition – 1.3.0

To install the above packages, use the following command.
```
pip install numpy opencv-python
```
To install the face_recognition, install the dlib package first.
```
pip install dlib
```
Now, install face_recognition module using the below command
```
pip install face_recognition
```

DATASET:
We can do this face recognition project using our own dataset. For this project, let's take the cast of the popular American web series "Friends" as the dataset. The dataset is included with face recognition project code

IMPLEMENTATION:

1. Prepare the dataset
Create 2 directories, train and test. Pick an image for each of the cast from the internet and download it onto our "train" directory. Make sure that the images you've selected show the features of the face well enough for the classifier.
For testing the model, let's take a picture containing all of the cast and place it onto our "test" directory.
For your comfort, we have added training and testing data with the project code.

2. Train the model
First import the necessary modules.
```
import face_recognition as fr
import cv2
import numpy as np
import os
```
The face_recognition library contains the implementation of the various utilities that help in the process of face recognition.
Now, create 2 lists that store the names of the images (persons) and their respective face encodings.
```
path = "./train/"
known_names = []
known_name_encodings = []
images = os.listdir(path)
```
Face encoding is a vector of values representing the important measurements between distinguishing features of a face like the distance between the eyes, the width of the forehead, etc.
We loop through each of the images in our train directory, extract the name of the person in the image, calculate its face encoding vector and store the information in the respective lists.
```
for _ in images:
image = fr.load_image_file(path + _)
image_path = path + _
encoding = fr.face_encodings(image)[0]
known_name_encodings.append(encoding)
known_names.append(os.path.splitext(os.path.basename(image_path))[0].capitalize())
```

3. Test the model on the test dataset
As mentioned above, our test dataset only contains 1 image with all of the persons in it.
Read the test image using the cv2 imread() method.
```
test_image = "./test/test.jpg"
image = cv2.imread(test_image)
```
The face_recognition library provides a useful method called face_locations() which locates the coordinates (left, bottom, right, top) of every face detected in the image. Using those location values we can easily find the face encodings.
```
face_locations = fr.face_locations(image)
face_encodings = fr.face_encodings(image, face_locations)
```

We loop through each of the face locations and its encoding found in the image. Then we compare this encoding with the encodings of the faces from the “train” dataset.
Then calculate the facial distance meaning that we calculate the similarity between the encoding of the test image and that of the train images. Now, we pick the minimum valued distance from it indicating that this face of the test image is one of the persons from the training dataset.
Now, draw a rectangle with the face location coordinates using the methods from the cv2 module.

```
for (top, right, bottom, left), face_encoding in zip(face_locations, face_encodings):
matches = fr.compare_faces(known_name_encodings, face_encoding)
name = ""
face_distances = fr.face_distance(known_name_encodings, face_encoding)
best_match = np.argmin(face_distances)
if matches[best_match]:
name = known_names[best_match]
cv2.rectangle(image, (left, top), (right, bottom), (0, 0, 255), 2)
cv2.rectangle(image, (left, bottom - 15), (right, bottom), (0, 0, 255), cv2.FILLED)
font = cv2.FONT_HERSHEY_DUPLEX
cv2.putText(image, name, (left + 6, bottom - 6), font, 1.0, (255, 255, 255), 1)
```

Display the image using the imshow() method of the cv2 module.

```
cv2.imshow("Result", image)
```

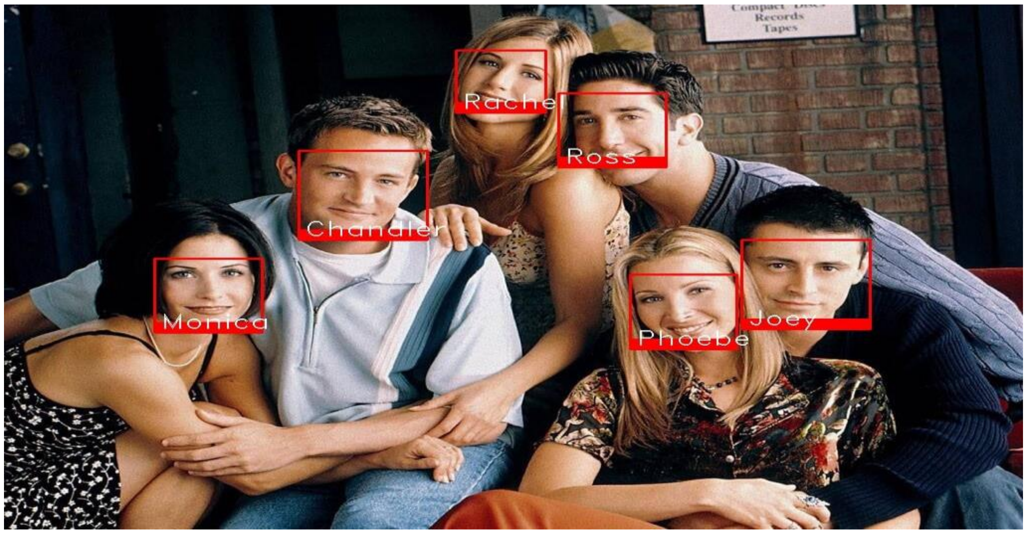
Save the image to our current working directory using the imwrite() method.

```
cv2.imwrite("./output.jpg", image)
```

Release the resources that weren’t deallocated(if any).

```
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



Python Face Recognition Output

CONCLUSION:

In this machine learning project, we developed a face recognition model in python and opencv using our own custom dataset..