FACULTY OF ENGINEERING AND TECHNOLOGY

BACHELOR OF TECHNOLOGY

MACHINE LEARNING
LAB MANUAL
(203105403)

6th SEMESTER

COMPUTER SCIENCE ENGINEERING & TECHNOLOGY

Faculty of Engineering & Technology
Subject Name: Machine Learning
Subject Code : 203105403
B Tech :CSE Year3,Semster 6

CERTIFICATE

This is to certify that Mr. **T POORNACHANDRA** with enrolment no.**(210304124393)** has successfully completed laboratory experiments in the **MACHINE LEARNING(203105403)**from the department of **CSE-AI** during the academic year **2023-24.**

Date of Submission:..... Staff In charge:.....

Head Of Department:.....

Faculty of Engineering & Technology
Subject Name: Machine Learning
Subject Code : 203105403
B Tech :CSE Year3,Semster 6

TABLE OF CONTENT

Sr. No	Experiment Title	Page No		Date of Start	Date of Completion	Sign	Marks (out of 10)
		To	From				
1	Use the Naive Bayes Classifier to differentiate between spam and non-spam(ham) messages.						
2	Apply SVM to classify images of different fashion products						
3	Build a linear regression model to predict house prices based on various features like size, location, year built.						
4	Given the actual & predicted outputs for a classification problem, implement the performance metrics in python from scratch: 1) Accuracy 2)Precision 3) F-1 Score 4) ROC						
5	Use logistic regression to identify fraudulent credit card transactions						
6	Apply XGBoost and compare the performance for the loan default estimation						
7	Predict the onset of diabetes based on diagnostic measures using Logistic Regression						
8	Implement back propagation algorithm from scratch in Python.						
9	Develop an artificial neural network to identity handwritten digits.						
10	Write a convolutional neural network to distinguish between cats and dogs images						

EXPERIMENT NO :- 1

AIM :- Use the Naive Bayes Classifier to differentiate between spam and non- spam (ham) messages.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
df=pd.read_csv('spam.csv',encoding='latin')
df.head()
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

```
df.v1.unique()
```

```
array(['ham', 'spam'], dtype=object)
```

```
df['spam']=df['v1'].apply(lambda x: 1 if x=='spam' else 0)
df.head(5)
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4	spam
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN	0
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN	0
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN	1
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN	0
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN	0

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(df.v2,df.spam,test_size=
0.2,random_state=42)

len(x train)
```

4457

```
len(x_test)
```

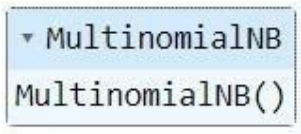
1115

```
from sklearn.feature_extraction.text import CountVectorizer
v=CountVectorizer()
cv_messages = v.fit_transform(x_train.values)
cv_messages.toarray()[0:5]
```

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

```
from sklearn.naive_bayes import MultinomialNB
model=MultinomialNB()
```

```
model.fit(cv_messages,y_train)
```




```
email = [
          'Upto 30% discount on parking, exclusive offer just for yoy.
Dont miss thi reward!',
          'Ok lar...joking wif u oni...'
]
email_count= v.transform(email)
model.predict(email count)
```

array([1, 0])

```
x_test_count=v.transform(x_test)
model.score(x_test_count,y_test)
```

0.9838565022421525

```
from sklearn.pipeline import Pipeline
clf = Pipeline([
      ('vectorizer', CountVectorizer()),
      ('nb', MultinomialNB())
]
)
clf.fit(x_train,y_train)
```

```
email = [
        'Upto 30% discount on parking, exclusive offer just for yoy.
Dont miss thi reward!',
         'Ok lar...joking wif u oni...'
]
clf.predict(email)
```

array([1, 0])

```
clf.score(x_test,y_test)
```

0.9838565022421525

```
import joblib
joblib.dump(clf,'spam_model.pkl')
```

['spam_model.pkl']

```
21# model is completed
```

EXPERIMENT NO :- 2

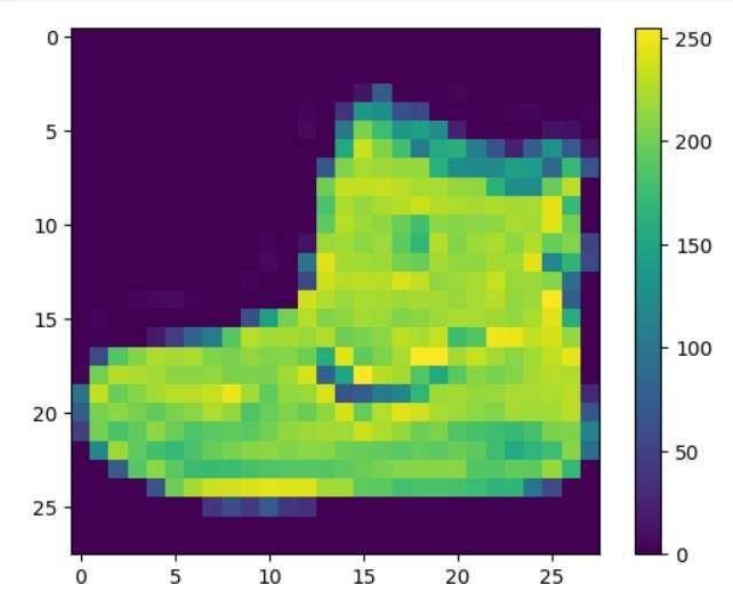
AIM :- Apply SVM to classify images of different fashion products.

```
# importing the necessary libraries
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```

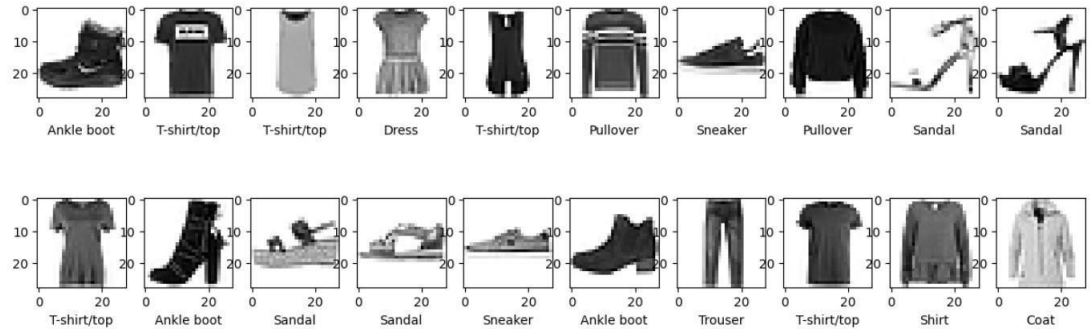
```
# storing the dataset path
clothing_fashion_mnist = tf.keras.datasets.fashion_mnist

# loading the dataset from tensorflow
(x_train, y_train),(x_test, y_test) =
clothing_fashion_mnist.load_data()

# displaying the shapes of training and testing dataset
print('Shape of training cloth images: ',x_train.shape)

print('Shape of training label: ',y_train.shape)

print('Shape of test cloth images: ',x_test.shape)

print('Shape of test labels: ',y_test.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
Shape of training cloth images:  (60000, 28, 28)
Shape of training label:  (60000,)
Shape of test cloth images:  (10000, 28, 28)
Shape of test labels:  (10000,)
```

```
# storing the class names as it is
# not provided in the dataset
label_class_names = ['T-shirt/top', 'Trouser',
           'Pullover', 'Dress', 'Coat',
           'Sandal', 'Shirt', 'Sneaker',
           'Bag', 'Ankle boot']

# display the first images
plt.imshow(x_train[0])
plt.colorbar() # to display the colourbar
plt.show()
```




```
plt.figure(figsize=(15, 5)) # figure size i = 0
while i < 20: plt.subplot(2, 10, i+1)

  # showing each image with colourmap as binary plt.imshow(x_train[i],
  cmap=plt.cm.binary)

  # giving class labels
  plt.xlabel(label_class_names[y_train[i]]) i = i+1

plt.show() # plotting the final output f
```



EXPERIMENT NO :- 3

AIM :- Build a linear regression model to predict house prices based on various features like size, location, year built.

```
import pandas as pd
```

```
data = pd.read_csv("/content/house_rent.csv")
```

```
data
```

	city	type	bhk	rent
0	0	0	1	20000
1	0	0	2	30000
2	0	1	2	40000
3	1	0	3	50000
4	1	0	4	60000
5	1	1	4	70000
6	0	1	5	80000
7	1	1	5	90000

```
data = pd.get_dummies(data, columns=['city', 'type'], drop_first=True)
```

```
x= data.drop('rent', axis=1)

y=data['rent']
```

```
x
```

	bhk	city_1	type_1
0	1	0	0
1	2	0	0
2	2	0	1
3	3	1	0
4	4	1	0
5	4	1	1
6	5	0	1
7	5	1	1

```
Y
```

```
0    20000
1    30000
2    40000
3    50000
4    60000
5    70000
6    80000
7    90000
Name: rent, dtype: int64
```

```
from sklearn.linear_model import LinearRegression
```

```
model =LinearRegression()
```

```
model.fit(x,y)
```

```
LinearRegression
LinearRegression()
```

```
city =int(input("enter the city 0 or 1:"))
type_ =int(input("enter the type 0 or 1:"))
bhk =int(input("enter the bhk:"))
```

```
enter the city 0 or 1:0
enter the type 0 or 1:1
enter the bhk:2
```

```
user_input = pd.DataFrame({
    'city_1': [0],
    'type_1': [0],
    'bhk': [bhk]
})
```

```
user_input = user_input[x.columns]
```

```
predicted_rent = model.predict(user_input)[0]
print(f'Predicted Rent: {predicted_rent}')
```

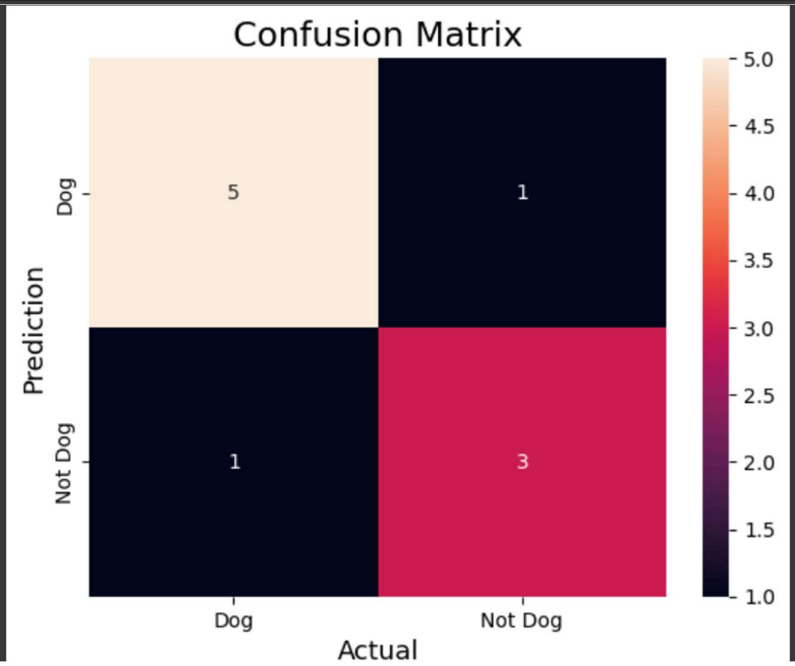
```
Predicted Rent: 30769.23076923078
```

EXPRIMENT 04

AIM: Given the actual and predict outputs for a classification problem,implement the performance metrics in python from scratch:1)Accuracy 2)precision 3) F-1 score 4) ROC

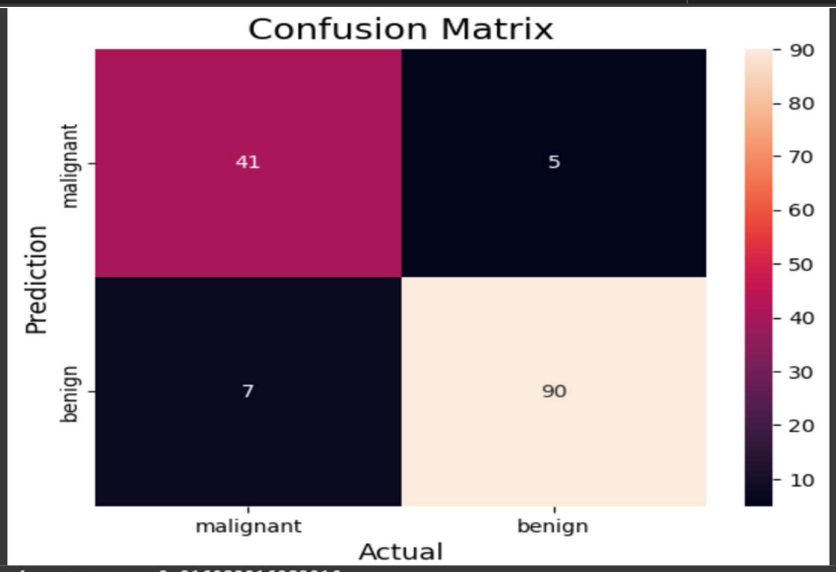
```
#Import the necessary libraries
import numpy as np
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

#Create the NumPy array for actual and predicted labels.
actual    = np.array(
  ['Dog','Dog','Dog','Not Dog','Dog','Not Dog','Dog','Dog','Not Dog','Not Dog'])
predicted = np.array(
  ['Dog','Not Dog','Dog','Not Dog','Dog','Dog','Dog','Dog','Not Dog','Not Dog'])

#compute the confusion matrix.
cm = confusion_matrix(actual,predicted)
```

```
[ ]
#Plot the confusion matrix.

sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['Dog','Not Dog'],
            yticklabels=['Dog','Not Dog'])
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()
```



```python
#Import the necessary libraries
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Load the breast cancer dataset
X, y= load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.25)

# Train the model
tree = DecisionTreeClassifier(random_state=23)
tree.fit(X_train, y_train)

# prediction
y_pred = tree.predict(X_test)

# compute the confusion matrix
cm = confusion_matrix(y_test,y_pred)
```

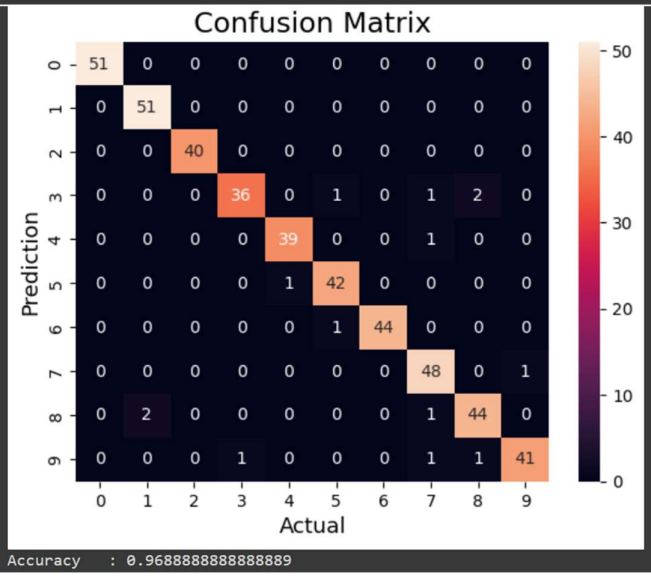
```python
#Plot the confusion matrix.
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['malignant', 'benign'],
            yticklabels=['malignant', 'benign'])
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()


# Finding precision and recall
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy   :", accuracy)
precision = precision_score(y_test, y_pred)
print("Precision :", precision)
recall = recall_score(y_test, y_pred)
print("Recall    :", recall)
F1_score = f1_score(y_test, y_pred)
print("F1-score  :", F1_score)
```




```
Accuracy   : 0.916083916083916
Precision : 0.9473684210526315
Recall     : 0.9278350515463918
F1-score   : 0.9374999999999999
```

```python
#Import the necessary libraries
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Load the breast cancer dataset
X, y= load_digits(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.25)

# Train the model
clf = RandomForestClassifier(random_state=23)
clf.fit(X_train, y_train)

# prediction
y_pred = clf.predict(X_test)

# compute the confusion matrix
cm = confusion_matrix(y_test,y_pred)
```

```python
#Plot the confusion matrix.
sns.heatmap(cm,
            annot=True,
            fmt='g')
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()


# Finding precision and recall
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy   :", accuracy)
```




```
Accuracy   : 0.9688888888888889
```

EXPERIMENT 05

AIM: Use the logistic regression to identify fraudulent credit card transactions

```
[ ]  #importation of libraries
     import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     %matplotlib inline
     import seaborn as sns
```

```
[ ]  df = pd.read_csv('card_transdata.csv')
```

```
[ ]  df.head()
```

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat_retailer	used_chip	used_pin_number	online_order	fraud
0	57.877857	0.311140	1.945940	1.0	1.0	0.0	0.0	0.0
1	10.829943	0.175592	1.294219	1.0	0.0	0.0	0.0	0.0
2	5.091079	0.805153	0.427715	1.0	0.0	0.0	1.0	0.0
3	2.247564	5.600044	0.362663	1.0	1.0	0.0	1.0	0.0
4	44.190936	0.566486	2.222767	1.0	1.0	0.0	1.0	0.0

```
[ ]  df.info()

     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 1000000 entries, 0 to 999999
     Data columns (total 8 columns):
      #   Column                          Non-Null Count    Dtype
     ---  ------                          --------------    -----
      0   distance_from_home              1000000 non-null  float64
      1   distance_from_last_transaction  1000000 non-null  float64
      2   ratio_to_median_purchase_price  1000000 non-null  float64
      3   repeat_retailer                 1000000 non-null  float64
      4   used_chip                       1000000 non-null  float64
      5   used_pin_number                 1000000 non-null  float64
      6   online_order                    1000000 non-null  float64
      7   fraud                           1000000 non-null  float64
     dtypes: float64(8)
     memory usage: 61.0 MB
```

```
     df[['fraud']].value_counts()

     fraud
     0.0      912597
     1.0       87403
     dtype: int64
```

```
[ ]  df.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
distance_from_home	1000000.0	26.628792	65.390784	0.004874	3.878008	9.967760	25.743985	10632.723672
distance_from_last_transaction	1000000.0	5.036519	25.843093	0.000118	0.296671	0.998650	3.355748	11851.104565
ratio_to_median_purchase_price	1000000.0	1.824182	2.799589	0.004399	0.475673	0.997717	2.096370	267.802942
repeat_retailer	1000000.0	0.881536	0.323157	0.000000	1.000000	1.000000	1.000000	1.000000
used_chip	1000000.0	0.350399	0.477095	0.000000	0.000000	0.000000	1.000000	1.000000
used_pin_number	1000000.0	0.100608	0.300809	0.000000	0.000000	0.000000	0.000000	1.000000
online_order	1000000.0	0.650552	0.476796	0.000000	0.000000	1.000000	1.000000	1.000000
fraud	1000000.0	0.087403	0.282425	0.000000	0.000000	0.000000	0.000000	1.000000

```
df_fraud = df.loc[df['fraud']==1]
df_fraud
```

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat_retailer	used_chip	used_pin_number	online_order
13	2.131956	56.372401	6.358667	1.0	0.0	0.0	1.0
24	3.803057	67.241081	1.872950	1.0	0.0	0.0	1.0
29	15.694986	175.989182	0.855623	1.0	0.0	0.0	1.0
35	26.711462	1.552008	4.603601	1.0	1.0	0.0	1.0
36	10.664474	1.565769	4.886521	1.0	0.0	0.0	1.0
...	...	...	...	...	...	...	...
999908	45.296658	0.882736	8.856861	1.0	0.0	0.0	1.0
999916	167.139756	0.282253	0.308468	1.0	0.0	0.0	1.0
999919	124.640118	0.004416	0.434885	1.0	0.0	0.0	1.0
999939	51.412900	3.429330	29.914254	1.0	0.0	0.0	1.0
999949	15.724799	1.875906	11.009366	1.0	1.0	0.0	1.0

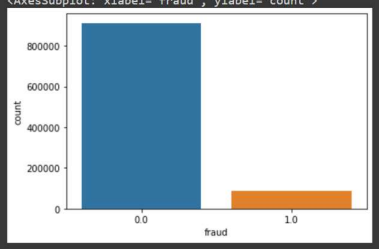
87403 rows × 8 columns

```
[ ] #a quick look at mean value of the features for fraudulent and non-fraudulent transactions
    df.groupby('fraud').mean()
```

fraud	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat_retailer	used_chip	used_pin_number	online_order
0.0	22.832976	4.301391	1.423642	0.881672	0.359402	0.109944	0.622225
1.0	66.261876	12.712185	6.006323	0.880119	0.256399	0.003123	0.946318

```
#visualizing the count plot and the unbalanced nature of the dataset
sns.countplot(x = 'fraud', data = df)
```

```
<AxesSubplot: xlabel='fraud', ylabel='count'>
```

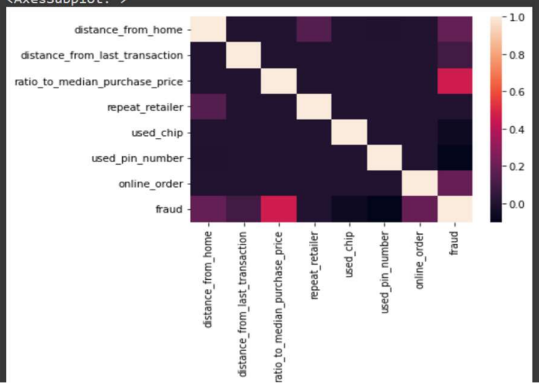



```
[ ] #correlation of each of the variables with fraud
    df.corr()['fraud'].sort_values()
```

```
used_pin_number                -0.100293
used_chip                      -0.060975
repeat_retailer                -0.001357
distance_from_last_transaction  0.091917
distance_from_home              0.187571
online_order                    0.191973
ratio_to_median_purchase_price  0.462305
fraud                           1.000000
Name: fraud, dtype: float64
```

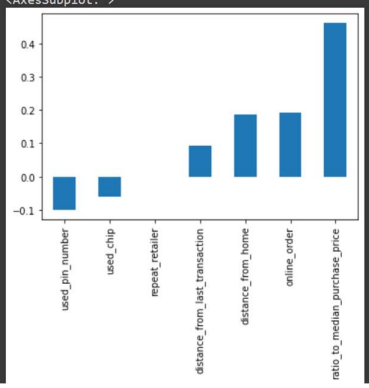
```
sns.heatmap(df.corr())
```

```
<AxesSubplot: >
```



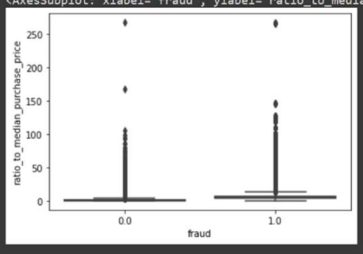

```
df.corr()['fraud'][:-1].sort_values().plot(kind='bar')
```

```
<AxesSubplot: >
```



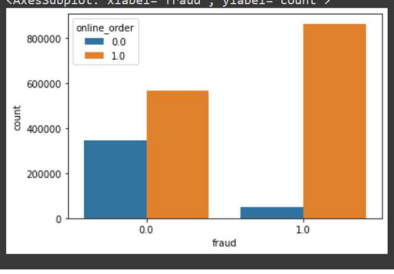

```
#ration to median purchase price higher for fraudulent transactions
sns.boxplot(x='fraud',y='ratio_to_median_purchase_price',data=df_over)
```

```
<AxesSubplot: xlabel='fraud', ylabel='ratio_to_median_purchase_price'>
```



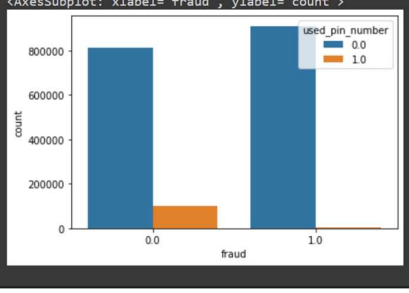

```
[ ] #online orders are more likely to be fraudulent
    sns.countplot(x='fraud',hue='online_order',data=df_over)
```

```
<AxesSubplot: xlabel='fraud', ylabel='count'>
```



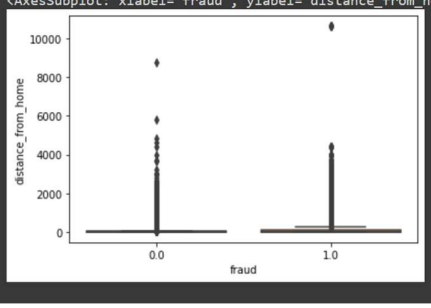
```
[ ] #transactions are less likely to be fraudulent when pin numbers are used
    sns.countplot(x='fraud',hue='used_pin_number',data=df_over)
```

```
<AxesSubplot: xlabel='fraud', ylabel='count'>
```




```
[ ] sns.boxplot(x='fraud',y='distance_from_home',data=df_over)
```

```
<AxesSubplot: xlabel='fraud', ylabel='distance_from_home'>
```




```
[ ] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X_over, y_over, test_size=0.3, random_state = 43)
```

```
[ ] #as part of preprocessing, I shall transform the independent variable to make it appropriate to be passed into logistic regression model
    from sklearn.preprocessing import StandardScaler
    columns = X_over.columns
    scaler = StandardScaler()
    X_train_transformed = scaler.fit_transform(X_train)
    X_train_transformed = pd.DataFrame(X_train_transformed, columns = columns)
    X_train_transformed.head()
```

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat_retailer	used_chip	used_pin_number	online_order
0	-0.268167	-0.181586	2.053853	0.368009	1.499015	-0.244671	0.525001
1	-0.346399	-0.227679	-0.134354	0.368009	-0.667105	4.087127	0.525001
2	-0.377079	-0.201560	-0.198613	0.368009	-0.667105	-0.244671	0.525001
3	-0.299126	-0.180417	-0.502093	0.368009	-0.667105	-0.244671	0.525001
4	-0.416190	-0.093649	-0.609292	-2.717328	-0.667105	-0.244671	0.525001

```
[ ] from sklearn.linear_model import LogisticRegression

[ ] logmodel = LogisticRegression(max_iter = 200)

[ ] logmodel.fit(X_train_transformed,y_train)

    LogisticRegression(max_iter=200)

[ ] X_test_transformed = scaler.transform(X_test)
    X_test_transformed = pd.DataFrame(X_test_transformed, columns = columns)
    X_test_transformed.head()
```

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat_retailer	used_chip	used_pin_number	online_order
0	0.731859	-0.098570	-0.629725	0.368009	-0.667105	-0.244671	0.525001
1	-0.358836	0.079219	-0.499421	0.368009	-0.667105	-0.244671	0.525001
2	-0.413249	-0.041578	-0.475285	-2.717328	1.499015	-0.244671	0.525001
3	-0.385277	-0.126289	1.313159	0.368009	1.499015	-0.244671	0.525001
4	0.600462	-0.216363	0.006923	0.368009	-0.667105	-0.244671	0.525001

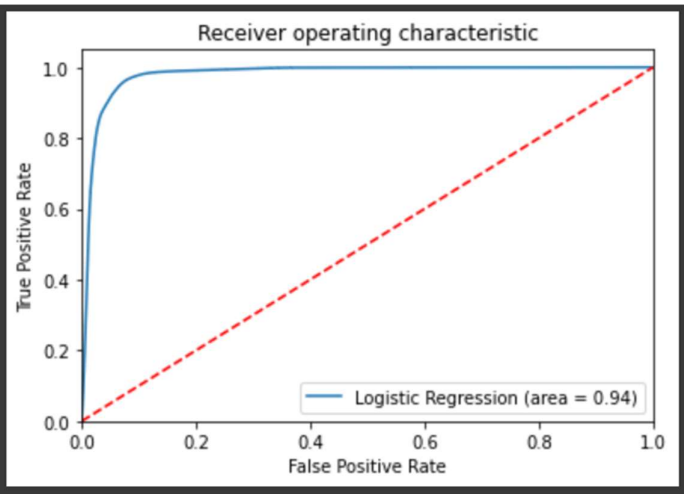
```
[ ] predictions = logmodel.predict(X_test_transformed)

[ ] from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

[ ] print(classification_report(y_test, predictions))
```

```
                  precision    recall  f1-score   support

             0.0       0.95      0.93      0.94    273968
             1.0       0.93      0.95      0.94    273591

        accuracy                           0.94    547559
       macro avg       0.94      0.94      0.94    547559
    weighted avg       0.94      0.94      0.94    547559
```

```
[ ] confusion_matrix(y_test, predictions)

    array([[255487,  18481],
           [ 13649, 259942]], dtype=int64)
```

```
[ ] accuracy_score(y_test, predictions)

    0.9413213918500107
```

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test, logmodel.predict(X_test_transformed))
fpr, tpr, thresholds = roc_curve(y_test, logmodel.predict_proba(X_test_transformed)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```




```
[ ] #predict a fraudulent transaction
    df_1fraud=df[df['fraud']==1].head(1)
    df_1fraud=df_1fraud.drop('fraud',axis=1)
    df_1fraud
```

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat_retailer	used_chip	used_pin_number	online_order
13	2.131956	56.372401	6.358667	1.0	0.0	0.0	1.0

```
[ ] df_1fraud = scaler.transform(df_1fraud)

[ ] df_1fraud

    array([[-0.40851561,  1.29597531,  0.55667687,  0.36800856, -0.6671046 ,
            -0.24467067,  0.52500085]])

[ ] df_1fraud = pd.DataFrame(df_1fraud, columns = columns)

[ ] #correctly predicted as fraudulent
    logmodel.predict(df_1fraud)

    array([1.])
```

```
import joblib
#save model
joblib.dump(logmodel, 'agada_credit_card_fraud_prediction.sav')
#save the scaler for transforming new records
joblib.dump(scaler,'agada_credit_card_fraud_prediction_scaler.pkl')
```

```
['agada_credit_card_fraud_prediction_scaler.pkl']
```

```
model = joblib.load('agada_credit_card_fraud_prediction.sav')
scaler =joblib.load('agada_credit_card_fraud_prediction_scaler.pkl')
```

```
def return_prediction(model,scaler,sample_json):

    ft_a = sample_json['distance_from_home']
    ft_b = sample_json['distance_from_last_transaction']
    ft_c = sample_json['ratio_to_median_purchase_price']
    ft_d = sample_json['repeat_retailer']
    ft_e = sample_json['used_chip']
    ft_f = sample_json['used_pin_number']
    ft_g = sample_json['online_order']

    columns = ['distance_from_home',
               'distance_from_last_transaction',
               'ratio_to_median_purchase_price',
               'repeat_retailer',
               'used_chip',
               'used_pin_number',
               'online_order']
```

```
        transaction = [[ft_a,ft_b,ft_c,ft_d,ft_e,ft_f,ft_g]]

        transaction = pd.DataFrame(transaction, columns = columns)

        transaction = scaler.transform(transaction)

        transaction = pd.DataFrame(transaction, columns = columns)

        classes = np.array(['not fraudulent', 'fraudulent'])

        class_ind = model.predict(transaction)

        class_ind = class_ind[0]

        return classes[int(class_ind)]
```

```
[ ] # this is a fraudulent transaction
    df_fraud2 = df[df['fraud']==1].tail(1)
    df_fraud2
```

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat_retailer	used_chip	used_pin_number	online_order
999949	15.724799	1.875906	11.009366	1.0	1.0	0.0	1.0

```
[ ] transaction_example = {'distance_from_home':15.724799,
                           'distance_from_last_transaction':1.875906,
                           'ratio_to_median_purchase_price':11.009366,
                           'repeat_retailer':1.0,
                           'used_chip':1.0,
                           'used_pin_number':0.0,
                           'online_order':1.0
                          }
```

```
# the transaction is correctly predicted as fraudulent
return_prediction(model,scaler,transaction_example)
```

```
'fraudulent'
```

```
[ ] #this transaction is not fraudulent
    df_not_fraud = df.head(1)
    df_not_fraud
```

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat_retailer	used_chip	used_pin_number	online_order	fraud
0	57.877857	0.31114	1.94594	1.0	1.0	0.0	0.0	0.0

```
[ ] #predict a non fraudulent transaction
    df.head(1)
```

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat_retailer	used_chip	used_pin_number	online_order	fraud
0	57.877857	0.31114	1.94594	1.0	1.0	0.0	0.0	0.0

```
[ ] transaction_example2 = {'distance_from_home':57.877857,
                            'distance_from_last_transaction':0.31114,
                            'ratio_to_median_purchase_price':1.94594,
                            'repeat_retailer':1.0,
                            'used_chip':1.0,
                            'used_pin_number':0.0,
                            'online_order':0.0
                           }
```

```
[ ] #correct prediction
    return_prediction(model,scaler,transaction_example2)
```

```
'not fraudulent'
```

EXPERIMENT NO :- 6

AIM :- Apply XGBoost and compare the performance for the loan default estimation.

Importing Required Libraries:-

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns
%matplotlib inline
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
import sklearn.metrics as metrics
import os
```

Reading the CSV File:-

```
data = pd.read_csv('loan-prediction-based-on-customer-behavior/Training Data.csv')
```

```
data.head() #displaying head of the .csv file
```


ENROLLMENT:210304124393

```
data.head() #displaying head of the .csv file
```

Python

	Id	Income	Age	Experience	Married/Single	House_Ownership	Car_Ownership	Profession	CITY	STATE	CURRENT_JOB_YRS	CURRENT_HOUSE_YRS	Risk_Flag
0	1	1303834	23	3	single	rented	no	Mechanical_engineer	Rewa	Madhya_Pradesh	3	13	0
1	2	7574516	40	10	single	rented	no	Software_Developer	Parbhani	Maharashtra	9	13	0
2	3	3991815	66	4	married	rented	no	Technical_writer	Alappuzha	Kerala	4	10	0
3	4	6256451	41	2	single	rented	yes	Software_Developer	Bhubaneswar	Odisha	2	12	1
4	5	5768871	47	11	single	rented	no	Civil_servant	Tiruchirappalli[10]	Tamil_Nadu	3	14	1

Dropping the Unecessary Columns:-

```
data = data.drop("Id", axis = 1)
data = data.drop("CITY", axis=1)
```

```
data.head() #displaying head of the .csv file
```

	Income	Age	Experience	Married/Single	House_Ownership	Car_Ownership	Profession	STATE	CURRENT_JOB_YRS	CURRENT_HOUSE_YRS	Risk_Flag
0	1303834	23	3	single	rented	no	Mechanical_engineer	Madhya_Pradesh	3	13	0
1	7574516	40	10	single	rented	no	Software_Developer	Maharashtra	9	13	0
2	3991815	66	4	married	rented	no	Technical_writer	Kerala	4	10	0
3	6256451	41	2	single	rented	yes	Software_Developer	Odisha	2	12	1
4	5768871	47	11	single	rented	no	Civil_servant	Tamil_Nadu	3	14	1

Checking for NULL Values:-

```
data.isnull().sum()
```

```
Income              0
Age                 0
Experience          0
Married/Single      0
House_Ownership     0
Car_Ownership       0
Profession          0
STATE               0
CURRENT_JOB_YRS     0
CURRENT_HOUSE_YRS   0
Risk_Flag           0
dtype: int64
```

Describing the Data:-

```
data.describe()
```

	Income	Age	Experience	CURRENT_JOB_YRS	CURRENT_HOUSE_YRS	Risk_Flag
count	2.520000e+05	252000.000000	252000.000000	252000.000000	252000.000000	252000.000000
mean	4.997117e+06	49.954071	10.084437	6.333877	11.997794	0.123000
std	2.878311e+06	17.063855	6.002590	3.647053	1.399037	0.328438
min	1.031000e+04	21.000000	0.000000	0.000000	10.000000	0.000000
25%	2.503015e+06	35.000000	5.000000	3.000000	11.000000	0.000000
50%	5.000694e+06	50.000000	10.000000	6.000000	12.000000	0.000000
75%	7.477502e+06	65.000000	15.000000	9.000000	13.000000	0.000000
max	9.999938e+06	79.000000	20.000000	14.000000	14.000000	1.000000

Encoding the Data:-

```
en = LabelEncoder()
catCols = ['Married/Single','House_Ownership','Car_Ownership','Profession','STATE']
for cols in catCols:
    data[cols] = en.fit_transform(data[cols])
```

```
data.head() #displaying head of the .csv file
```

	Income	Age	Experience	Married/Single	House_Ownership	Car_Ownership	Profession	STATE	CURRENT_JOB_YRS	CURRENT_HOUSE_YRS	Risk_Flag
0	1303834	23	3	1	2	0	33	13	3	13	0
1	7574516	40	10	1	2	0	43	14	9	13	0
2	3991815	66	4	0	2	0	47	12	4	10	0
3	6256451	41	2	1	2	1	43	17	2	12	1
4	5768871	47	11	1	2	0	11	22	3	14	1

Splitting Dataset into Train and Test:-

```
y = data["Risk_Flag"]
X = data.drop("Risk_Flag",axis = 1)
```

```
Y=pd.DataFrame(y)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size=0.8, test_size=0.2)
```

```
from xgboost import XGBClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score, accuracy_score, roc_auc_score
# Define the model


    # defining the model
clf = XGBClassifier(learning_rate=0.1,
                    n_estimators=1000,
                    use_label_encoder=False,
                    random_state=42)

clf.fit(X_train, y_train, eval_metric='logloss')
predictions = clf.predict(X_test)
print("accuracy_score: " + str(accuracy_score(y_test, predictions)))
```

Creating XGBOOST Model:-

```
accuracy_score: 0.8971825396825397
```

Calculating the ROC Score of Model:-

```
    print("ROC AUC score: " + str(roc_auc_score(y_test, predictions)))
```

```
ROC AUC score: 0.6837972465278378
```

EXPERIMENT-7

AIM: Predict the onset of diabetes based on diagonostic measures using Logistic Regression

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import pickle
```

```
df=pd.read_csv('diabetes.csv')
```

```
X=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)
```

```
classifier=RandomForestClassifier()
classifier.fit(X_train,y_train)
```

```
RandomForestClassifier()
```

```
y_pred=classifier.predict(X_test)
```

```
from sklearn.metrics import accuracy_score
score=accuracy_score(y_test,y_pred)
```

```
score
```

```
0.7272727272727273
```

```
pickle_out = open("classifier.pkl","wb")
pickle.dump(classifier, pickle_out)
pickle_out.close()
```

```
classifier.predict([[2,3,4,1]])
```

```
array([0], dtype=int64)
```

Diabetes Prediction

Enter the Pregnancies value: 4,00

Enter the Glucose value: 110,00

Enter the Blood Pressure value: 92,00

Enter the Skin Thickness value: 0,00

Enter the Insulin value: 0,00

Enter the BMI value: 37,60

Enter the Diabetes Pedigree Function value: 0,19

Enter the Age value: 30,00

Diabetes Prediction Test

The patient does not have diabetes

Experiment-8

AIM :- Implement back propogation algorithm from scratch in python.

Imports and Data Downloads

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

data = pd.read_csv('/kaggle/input/mnist-digit-recognizer/train.csv')
```

```
data.head(10)
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
6	7	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
7	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
8	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
9	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0

10 rows × 785 columns

```
data.shape
```

```
(42000, 785)
```


ENROLLMENT:210304124393

Split the Data into Train and Test Sets:-

```
data = np.array(data)
m, n = data.shape
np.random.shuffle(data) # shuffle before splitting into test and training sets

data_test = data[0:2000].T
Y_test = data_test[0]
X_test = data_test[1:n]
X_test = X_test / 255.

data_train = data[2000:m].T
Y_train = data_train[0]
X_train = data_train[1:n]
X_train = X_train / 255.
_,m_train = X_train.shape
```

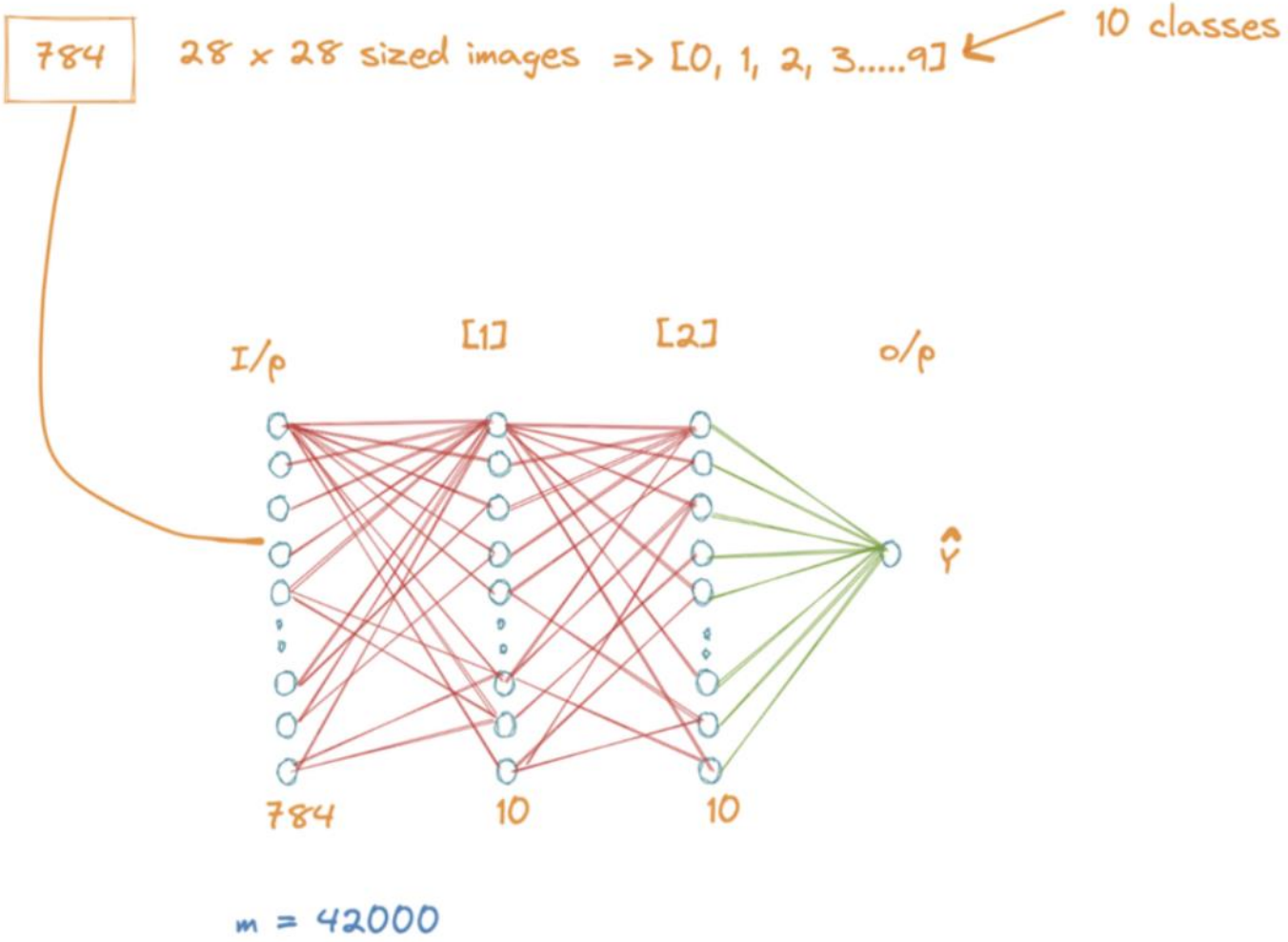
```
X_train.shape
```

```
(784, 40000)
```

```
Y_train
```

```
array([4, 2, 0, ..., 5, 1, 9])
```

Neural Net Design:-



$m = 42000$

size of input layer = 784×42000
size of first hidden layer [1] = 10×42000
size of second hidden layer [2] = 10×42000

Forward Propogation:-

$$A^{[0]} = X$$ Input layer, X is the image matrix

$$Z^{[1]} = W^{[1]}A^{[0]} + b^{[1]}$$ W = Weights, b = biases

$$A^{[1]} = g(Z^{[1]}) = ReLu(Z^{[1]})$$ Activated output

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = g(Z^{[2]}) = Softmax(Z^{[2]})$$ Softmax activated output

Backward Propogation:-

$$dZ^{[2]} = 2(A^{[2]} - Y)$$ Derivative of loss wrt o/p at layer 2

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} . A^{[1]T}$$ Derivative of loss wrt weights of layer 2

$$db^{[2]} = \frac{1}{m} \sum dZ^{[2]}$$

$$dZ^{[1]} = dW^{[2][T]} . dZ^{[2]} * g'(Z^{[1]})$$ Undo the activation function of ReLU

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} . X^{T}$$

$$db^{[1]} = \frac{1}{m} \sum dZ^{[1]}$$

ENROLLMENT:210304124393

Updating the Weights:-

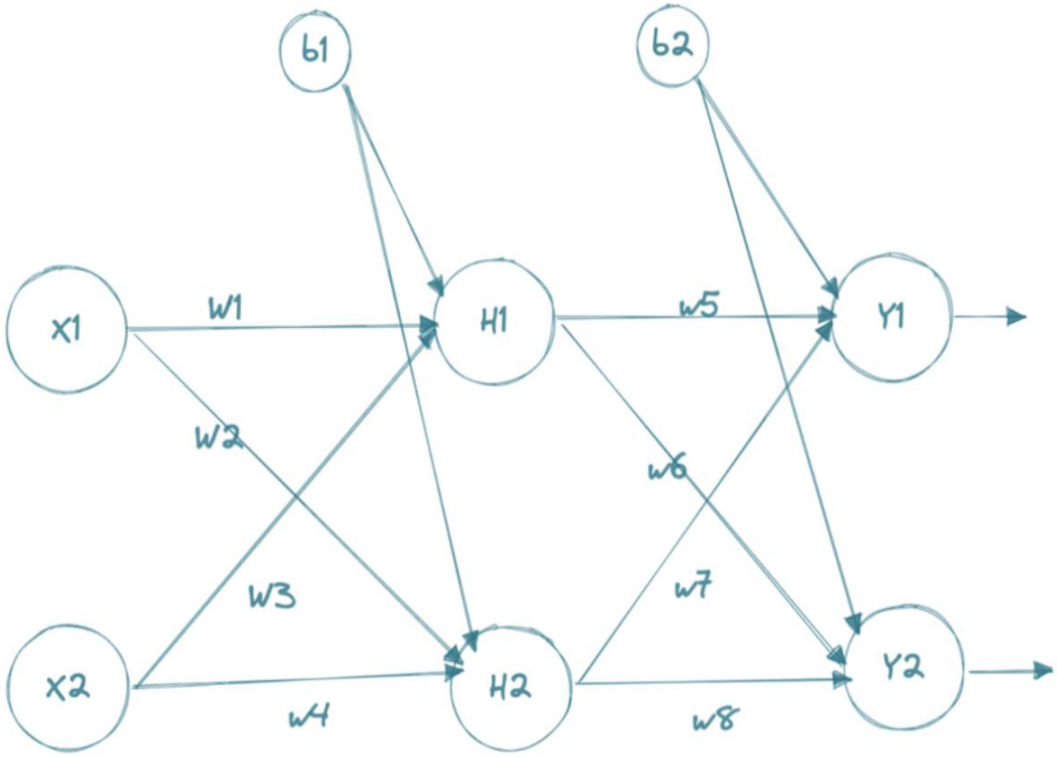
$$W^{[1]} = W^{[1]} - \alpha * dW^{[1]}$$

$$b^{[1]} = b^{[1]} - \alpha * db^{[1]}$$

$$W^{[2]} = W^{[2]} - \alpha * dW^{[2]}$$

$$b^{[2]} = b^{[2]} - \alpha * db^{[2]}$$

The Calculas involves the Chain rule:-



o/p at H1 = W1 X1 + W3 X2 + b1

$$w*_5 = w_5 - \alpha * \frac{\partial Total_{Error}}{\partial w_5}$$

$$Total\ Error = \frac{1}{2}(T_1 - Output_{y_1})^2 + \frac{1}{2}(T_2 - Output_{y_2})^2$$

$$\frac{\partial Total_{Error}}{\partial w_5} = \frac{\partial Total_{Error}}{\partial out_{y_1}} * \frac{\partial out_{y_1}}{\partial y_1} * \frac{\partial y_1}{\partial w_5}$$

Code:-

```
def init_params():
    W1 = np.random.rand(10, 784) - 0.5
    b1 = np.random.rand(10, 1) - 0.5
    W2 = np.random.rand(10, 10) - 0.5
    b2 = np.random.rand(10, 1) - 0.5
    return W1, b1, W2, b2

def ReLU(Z):
    return np.maximum(Z, 0)

def softmax(Z):
    A = np.exp(Z) / sum(np.exp(Z))
    return A

def forward_prop(W1, b1, W2, b2, X):
    Z1 = W1.dot(X) + b1
    A1 = ReLU(Z1)
    Z2 = W2.dot(A1) + b2
    A2 = softmax(Z2)
    return Z1, A1, Z2, A2

def ReLU_deriv(Z):
    return Z > 0

def one_hot(Y):
    one_hot_Y = np.zeros((Y.size, Y.max() + 1))
    one_hot_Y[np.arange(Y.size), Y] = 1
    one_hot_Y = one_hot_Y.T
    return one_hot_Y
```

```python
def backward_prop(Z1, A1, Z2, A2, W1, W2, X, Y):
    one_hot_Y = one_hot(Y)
    dZ2 = 2*(A2 - one_hot_Y)
    dW2 = 1 / m * dZ2.dot(A1.T)
    db2 = 1 / m * np.sum(dZ2)
    dZ1 = W2.T.dot(dZ2) * ReLU_deriv(Z1)
    dW1 = 1 / m * dZ1.dot(X.T)
    db1 = 1 / m * np.sum(dZ1)
    return dW1, db1, dW2, db2

def update_params(W1, b1, W2, b2, dW1, db1, dW2, db2, alpha):
   # print(W1.shape)
   # print(dW1.shape)
    W1 = W1 - alpha * dW1
    b1 = b1 - alpha * db1
    W2 = W2 - alpha * dW2
    b2 = b2 - alpha * db2
    return W1, b1, W2, b2
```

```python
def get_predictions(A2):
    return np.argmax(A2, 0)

def get_accuracy(predictions, Y):
    print(predictions, Y)
    return np.sum(predictions == Y) / Y.size

def gradient_descent(X, Y, alpha, iterations):
    W1, b1, W2, b2 = init_params()
    for i in range(iterations):
        Z1, A1, Z2, A2 = forward_prop(W1, b1, W2, b2, X)
        dW1, db1, dW2, db2 = backward_prop(Z1, A1, Z2, A2, W1, W2, X, Y)
        #print(W1.shape)
        #print(dW1.shape)
        W1, b1, W2, b2 = update_params(W1, b1, W2, b2, dW1, db1, dW2, db2, alpha)
        if i % 10 == 0:
            print("Iteration: ", i)
            predictions = get_predictions(A2)
            print(get_accuracy(predictions, Y))
    return W1, b1, W2, b2
```


ENROLLMENT:210304124393

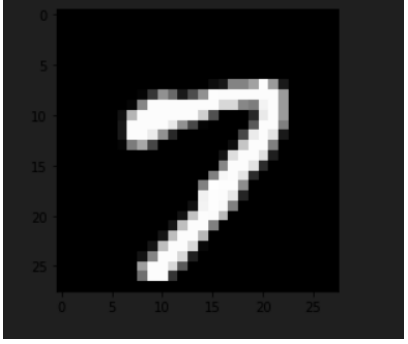
```
W1, b1, W2, b2 = gradient_descent(X_train, Y_train, 0.10, 500)
```

```
Iteration:  0
[0 0 7 ... 7 8 0] [4 2 0 ... 5 1 9]
0.077025
Iteration:  10
[0 0 7 ... 8 8 2] [4 2 0 ... 5 1 9]
0.234275
Iteration:  20
[6 2 6 ... 2 1 6] [4 2 0 ... 5 1 9]
0.347
Iteration:  30
[2 2 6 ... 2 1 6] [4 2 0 ... 5 1 9]
0.404225
Iteration:  40
[4 2 6 ... 2 1 6] [4 2 0 ... 5 1 9]
0.456375
Iteration:  50
[4 2 6 ... 2 1 6] [4 2 0 ... 5 1 9]
0.5175
Iteration:  60
[4 2 6 ... 2 1 6] [4 2 0 ... 5 1 9]
0.5732
Iteration:  70
[4 2 6 ... 2 1 6] [4 2 0 ... 5 1 9]
0.61445
Iteration:  80
...
0.877925
Iteration:  490
[4 2 0 ... 0 1 9] [4 2 0 ... 5 1 9]
0.878675
```

```
def make_predictions(X, W1, b1, W2, b2):
    _, _, _, A2 = forward_prop(W1, b1, W2, b2, X)
    predictions = get_predictions(A2)
    return predictions

def test_prediction(index, W1, b1, W2, b2):
    current_image = X_train[:, index, None]
    prediction = make_predictions(X_train[:, index, None], W1, b1, W2, b2)
    label = Y_train[index]
    print("Prediction: ", prediction)
    print("Label: ", label)

    current_image = current_image.reshape((28, 28)) * 255
    plt.gray()
    plt.imshow(current_image, interpolation='nearest')
    plt.show()
```

```
test_prediction(0, W1, b1, W2, b2)
test_prediction(1, W1, b1, W2, b2)
test_prediction(2, W1, b1, W2, b2)
test_prediction(100, W1, b1, W2, b2)
test_prediction(200, W1, b1, W2, b2)
```

```
Prediction:  [7]
Label:  7
```



```
test_predictions = make_predictions(X_test, W1, b1, W2, b2)
get_accuracy(test_predictions, Y_test)
```

```
[1 9 9 ... 6 3 1] [1 4 9 ... 6 3 1]

0.882
```

Experiment-9

Aim:- Develop an artificial neural network to identity handwritten digits.

```
import tensorflow as tf
```

```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import cv2
```

```
(X_train, y_train) , (X_test, y_test) = keras.datasets.mnist.load_data()
```

```
len(X_train)
```

```
60000
```

```
len(X_test)
```

```
10000
```

```
X_train[0].shape
```

```
(28, 28)
```

```
X_train.shape
```

```
(60000, 28, 28)
```

```
x_train[0]
```

```
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,
```

```
# selecting the first sample
x = X_train[1]
print("The dimension of x is 2D matrix as ", x.shape)
# Resizing the image
x = cv2.resize(x, (18,18))
```

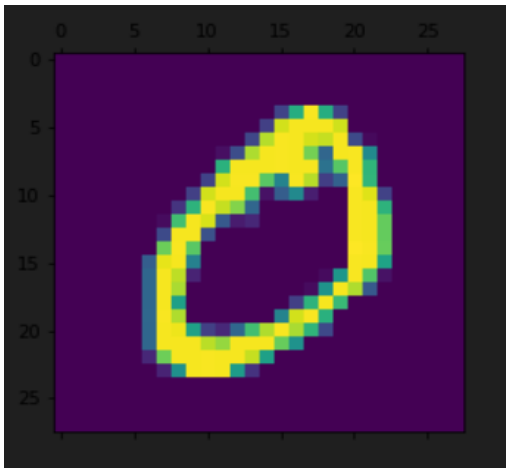
```
The dimension of x is 2D matrix as  (28, 28)
```

```
print("The range of pixel varies between 0 to 255")
print("The pixel having black is more close to 0 and pixel which is white is more close to 255")
print(x)
```

```
The range of pixel varies between 0 to 255
The pixel having black is more close to 0 and pixel which is white is more close to 255
[[  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   6  55  84  22   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  96 244 250 228   0   0   0   0   0]
 [  0   0   0   0   0   0   0   9 108 243 252 196 247 110   0   0   0   0]
 [  0   0   0   0   0   0   2 181 252 247 251 189 178 210   0   0   0   0]
 [  0   0   0   0   0   2 112 247 220  84 159  69  30 234  29   0   0   0]
 [  0   0   0   0   1  68 223 201 103   0   0   0   0 252 160   0   0   0]
 [  0   0   0   0  21 232 166  17   7   0   0   0   0 252 184   0   0   0]
 [  0   0   0   0 116 248  65   0   0   0   0   0   0 253 172   0   0   0]
 [  0   0   0   0 167 223  15   0   0   0   0   2 107 225  33   0   0   0]
 [  0   0   0   0 168 182   0   0   0   0  16 111 219  90   0   0   0   0]
 [  0   0   0   0 169 208   0   0   0  30 207 217  18   0   0   0   0   0]
 [  0   0   0   0 169 248 162 130 202 234 184  62   2   0   0   0   0   0]
 [  0   0   0   0 108 245 253 251 229  99   0   0   0   0   0   0   0   0]
 [  0   0   0   0   5  52  98  91  26   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]]
```

```
plt.matshow(X_train[1])
```

```
<matplotlib.image.AxesImage at 0x7f87e641cf50>
```




```
y_train[1]
```

```
0
```

```
X_train
```

```
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],
        ...,
```


ENROLLMENT:210304124393

```
X_test
```

```
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]],

       ...,
...
```

```
X_train = X_train / 255
X_test = X_test / 255
```

```
X_train[0]
```

```
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.        ,
...
```

```
X_train_flattened = X_train.reshape(len(X_train), 28*28)
X_test_flattened = X_test.reshape(len(X_test), 28*28)
```

```
X_train_flattened.shape
```

```
(60000, 784)
```

```
X_train_flattened[0]
```

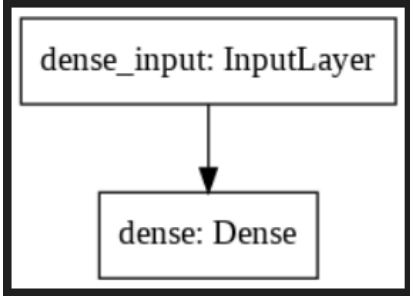
```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 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.        ,
...
```

```
model = keras.Sequential([
    keras.layers.Dense(10, input_shape=(784,), activation='sigmoid')
])

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

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
dense (Dense)               (None, 10)                7850
=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
_________________________________________________________________
```

```
import pydot
keras.utils.plot_model(model)
```




```
model.fit(X_train_flattened, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.4685 - accuracy: 0.8769
Epoch 2/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.3033 - accuracy: 0.9151
Epoch 3/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.2830 - accuracy: 0.9204
Epoch 4/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.2729 - accuracy: 0.9242
Epoch 5/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.2664 - accuracy: 0.9263

<tensorflow.python.keras.callbacks.History at 0x7f87e3361790>
```

```
model.evaluate(X_test_flattened, y_test)
```

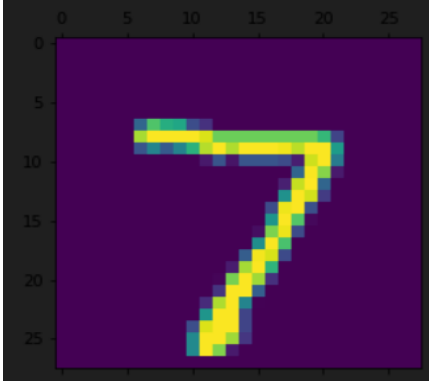
```
313/313 [==============================] - 0s 1ms/step - loss: 0.2696 - accuracy: 0.9248

[0.26961490511894226, 0.9247999787330627]
```

```
y_predicted = model.predict(X_test_flattened)
y_predicted[0]
```

```
array([2.6710808e-02, 2.0082869e-07, 4.0452570e-02, 9.5325053e-01,
       1.7535686e-03, 9.9020720e-02, 1.4245377e-06, 9.9976116e-01,
       8.6834908e-02, 5.9089357e-01], dtype=float32)
```

```
plt.matshow(X_test[0])
```

```
<matplotlib.image.AxesImage at 0x7f87e64062d0>
```



```
np.argmax(y_predicted[0])
```

```
7
```

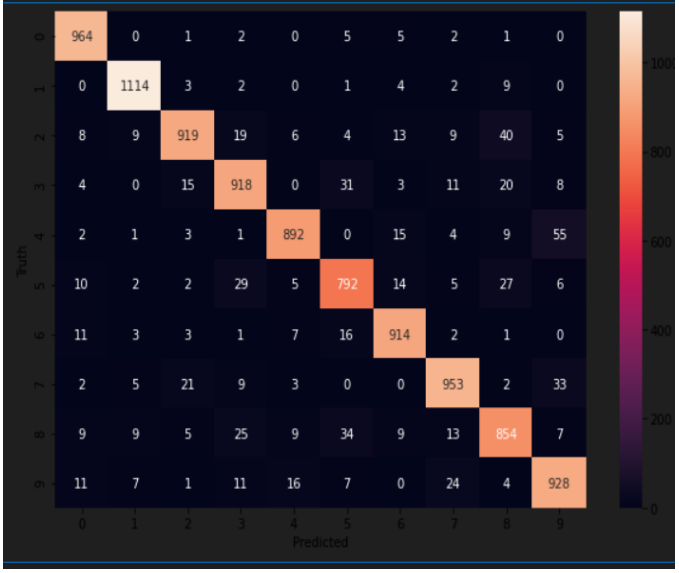
```
y_predicted_labels = [np.argmax(i) for i in y_predicted]
```

```
y_predicted_labels[:5]
```

```
[7, 2, 1, 0, 4]
```

```
import seaborn as sn
plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
Text(69.0, 0.5, 'Truth')
```



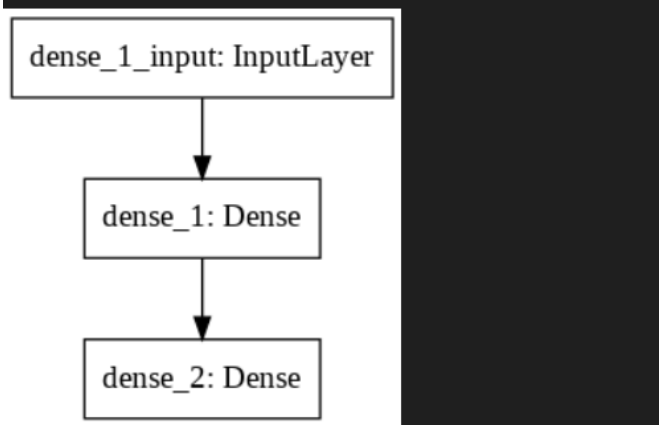
ENROLLMENT:210304124393

```
model = keras.Sequential([
    keras.layers.Dense(100, input_shape=(784,), activation='relu'),
    keras.layers.Dense(10, activation='sigmoid')
])

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

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 100)               78500
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010
=================================================================
Total params: 79,510
Trainable params: 79,510
Non-trainable params: 0
_________________________________________________________________
```

```
import pydot
keras.utils.plot_model(model)
```

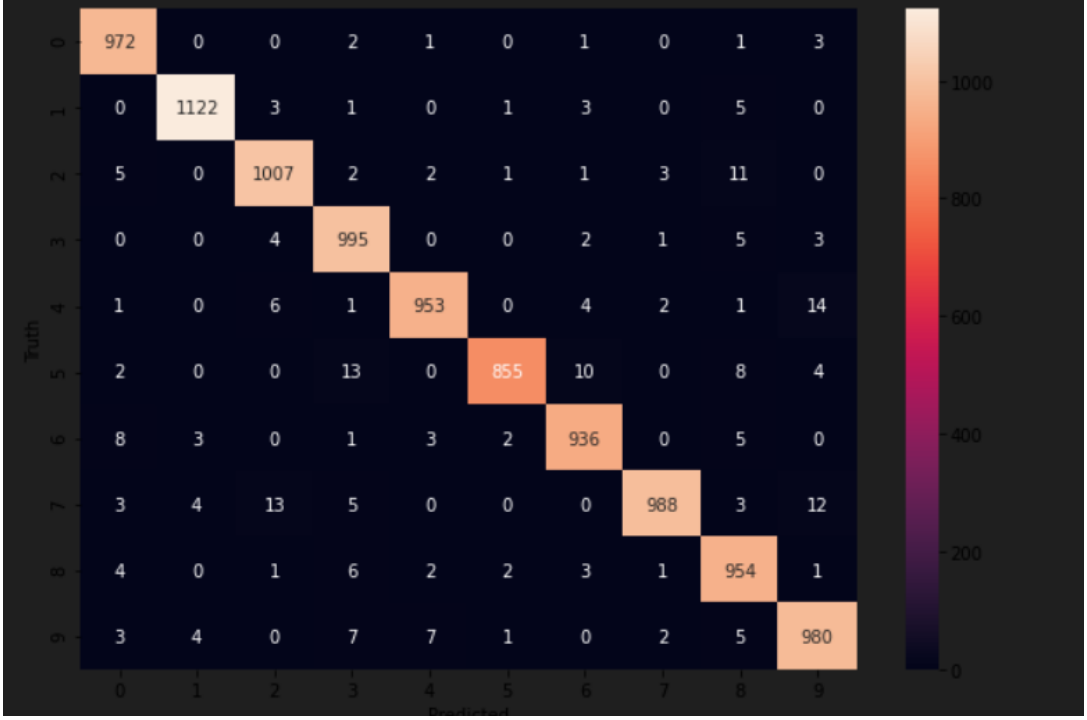



```
model.fit(X_train_flattened, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2763 - accuracy: 0.9221
Epoch 2/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1264 - accuracy: 0.9633
Epoch 3/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0863 - accuracy: 0.9741
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0676 - accuracy: 0.9791
Epoch 5/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0529 - accuracy: 0.9837

<tensorflow.python.keras.callbacks.History at 0x7f87d1691b10>
```

```
y_predicted = model.predict(X_test_flattened)
y_predicted_labels = [np.argmax(i) for i in y_predicted]
cm = tf.math.confusion_matrix(labels=y_test,predictions=y_predicted_labels)

plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
Text(69.0, 0.5, 'Truth')
```




```
model.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 1s 1ms/step - loss: 0.0854 - accuracy: 0.9762

[0.08542901277542114, 0.9761999845504761]
```

EXPERIMENT-10

AIM: Write a Convolutional neural network to distinguish between cat's and dog's images

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.callbacks import TensorBoard
```

```
from warnings import filterwarnings
filterwarnings('ignore')
```

```
classifier = Sequential()
classifier.add(Conv2D(32,(3,3),input_shape=(64,64,3),activation = 'relu'))
classifier.add(MaxPooling2D(pool_size=(2,2),strides=2)) #if stride not given it equal to pool filter size
classifier.add(Conv2D(32,(3,3),activation = 'relu'))
classifier.add(MaxPooling2D(pool_size=(2,2),strides=2))
classifier.add(Flatten())
classifier.add(Dense(units=128,activation='relu'))
classifier.add(Dense(units=1,activation='sigmoid'))
adam = tensorflow.keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
classifier.compile(optimizer=adam,loss='binary_crossentropy',metrics=['accuracy'])
#tensorboard = TensorBoard(log_dir="logs/{}".format(time()))
```

Data Augmentation:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.1,
                                   zoom_range=0.1,
                                   horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)

#Training Set
train_set = train_datagen.flow_from_directory('train',
                                              target_size=(64,64),
                                              batch_size=32,
                                              class_mode='binary')
#Validation Set
test_set = test_datagen.flow_from_directory('test',
                                            target_size=(64,64),
                                            batch_size = 32,
                                            class_mode='binary',
                                            shuffle=False)
#Test Set /no output available
test_set1 = test_datagen.flow_from_directory('test1',
                                             target_size=(64,64),
                                             batch_size=32,
                                             shuffle=False)
```

```
Found 19998 images belonging to 2 classes.
Found 5000 images belonging to 2 classes.
Found 12500 images belonging to 1 classes.
```

```
%%capture
classifier.fit_generator(train_set,
                         steps_per_epoch=800,
                         epochs = 200,
                         validation_data = test_set,
                         validation_steps = 20,
                         #callbacks=[tensorboard]
                         );

#Some Helpful Instructions:

#finetune you network parameter in last by using low learning rate like 0.00001
#classifier.save('resources/dogcat_model_bak.h5')
#from tensorflow.keras.models import load_model
#model = load_model('partial_trained1')
#100 iteration with learning rate 0.001 and after that 0.0001
```

```
from tensorflow.keras.models import load_model
classifier = load_model('resources/dogcat_model_bak.h5')
```

Prediction of Single image:

```
#Prediction of image
%matplotlib inline
import tensorflow
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt
import numpy as np
img1 = image.load_img('test/Cat/10.jpg', target_size=(64, 64))
img = image.img_to_array(img1)
img = img/255
# create a batch of size 1 [N,H,W,C]
img = np.expand_dims(img, axis=0)
prediction = classifier.predict(img, batch_size=None,steps=1) #gives all class prob.
if(prediction[:,:]>0.5):
    value ='Dog :%1.2f'%(prediction[0,0])
    plt.text(20, 62,value,color='red',fontsize=18,bbox=dict(facecolor='white',alpha=0.8))
else:
    value ='Cat :%1.2f'%(1.0-prediction[0,0])
    plt.text(20, 62,value,color='red',fontsize=18,bbox=dict(facecolor='white',alpha=0.8))

plt.imshow(img1)
plt.show()
```

ENROLLMENT:210304124393

```
import pandas as pd
test_set.reset
ytesthat = classifier.predict_generator(test_set)
df = pd.DataFrame({
    'filename':test_set.filenames,
    'predict':ytesthat[:,0],
    'y':test_set.classes
})
```

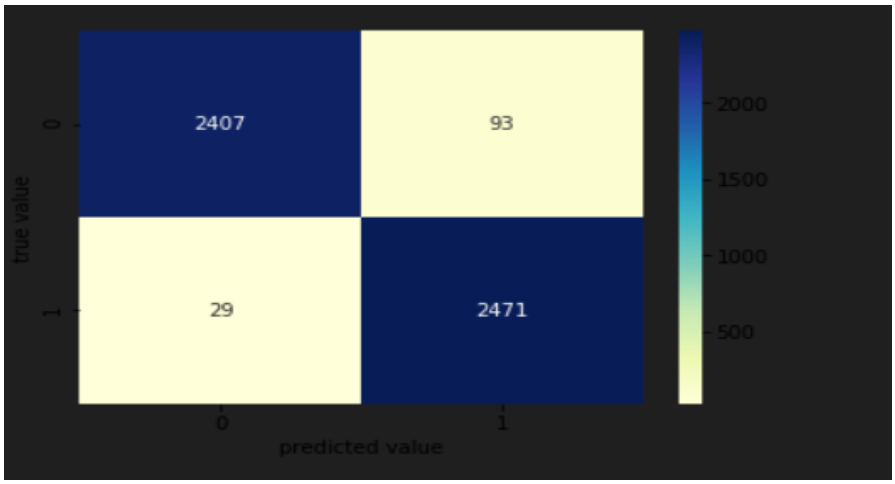
```
pd.set_option('display.float_format', lambda x: '%.5f' % x)
df['y_pred'] = df['predict']>0.5
df.y_pred = df.y_pred.astype(int)
df.head(10)
```

	filename	predict	y	y_pred
0	Cat/0.jpg	0.00000	0	0
1	Cat/1.jpg	0.00000	0	0
2	Cat/10.jpg	0.00000	0	0
3	Cat/100.jpg	0.99970	0	1
4	Cat/10001.jpg	0.00000	0	0
5	Cat/10009.jpg	0.02340	0	0
6	Cat/1001.jpg	0.00001	0	0
7	Cat/10012.jpg	0.00000	0	0
8	Cat/10017.jpg	0.00000	0	0
9	Cat/10018.jpg	0.00000	0	0

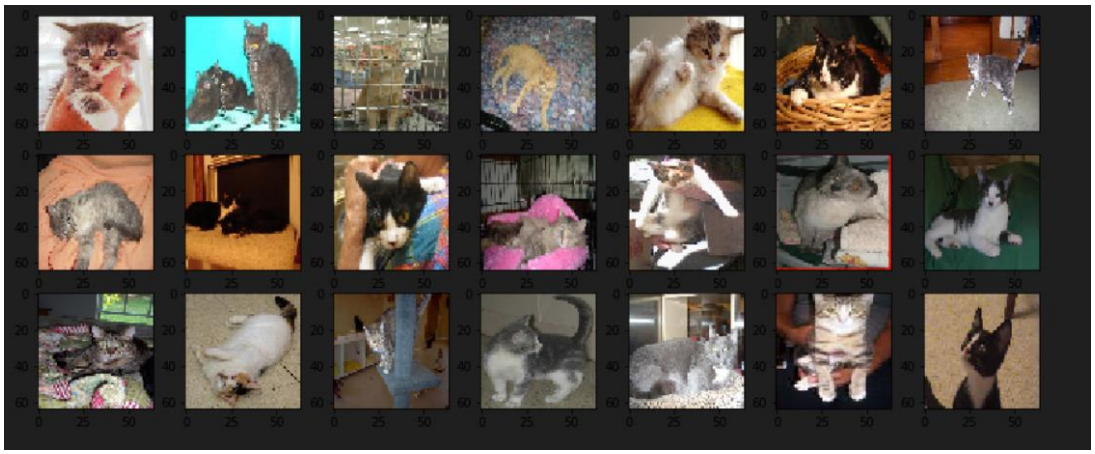
```
misclassified = df[df['y']!=df['y_pred']]
print('Total misclassified image from 5000 Validation images : %d'%misclassified['y'].count())
```

```
Total misclassified image from 5000 Validation images : 122
```

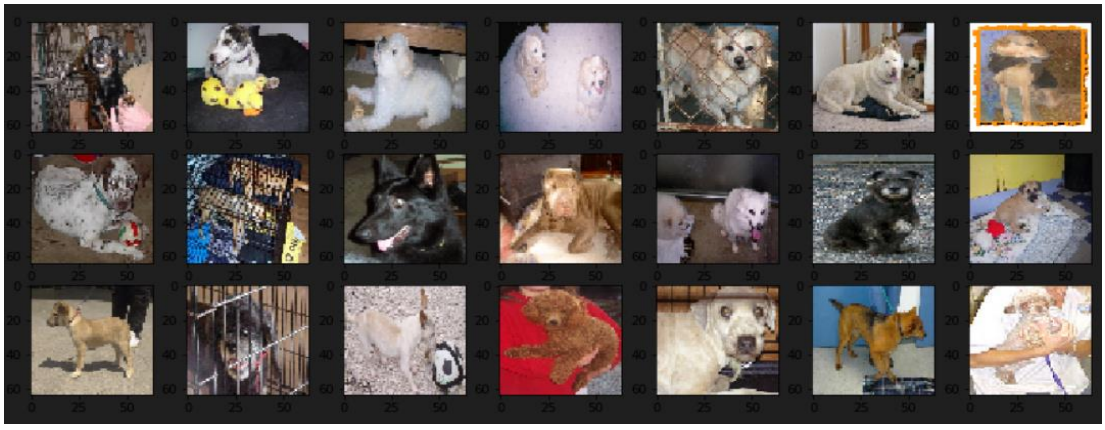
```
#Prediction of test set
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

conf_matrix = confusion_matrix(df.y,df.y_pred)
sns.heatmap(conf_matrix,cmap="YlGnBu",annot=True,fmt='g');
plt.xlabel('predicted value')
plt.ylabel('true value');
```



```
#Some of Cat image misclassified as Dog.
import matplotlib.image as mpimg

CatasDog = df['filename'][(df.y==0)&(df.y_pred==1)]
fig=plt.figure(figsize=(15, 6))
columns = 7
rows = 3
for i in range(columns*rows):
    #img = mpimg.imread()
    img = image.load_img('test/'+CatasDog.iloc[i], target_size=(64, 64))
    fig.add_subplot(rows, columns, i+1)
    plt.imshow(img)

plt.show()
```



```
#Some of Dog image misclassified as Cat.
import matplotlib.image as mpimg

DogasCat = df['filename'][(df.y==1)&(df.y_pred==0)]
fig=plt.figure(figsize=(15, 6))
columns = 7
rows = 3
for i in range(columns*rows):
    #img = mpimg.imread()
    img = image.load_img('test/'+DogasCat.iloc[i], target_size=(64, 64))
    fig.add_subplot(rows, columns, i+1)
    plt.imshow(img)
plt.show()
```



ENROLLMENT:210304124393

```
classifier.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 62, 62, 32)        896
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 31, 31, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 29, 29, 32)        9248
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               802944
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 129
=================================================================
Total params: 813,217
Trainable params: 813,217
Non-trainable params: 0
_________________________________________________________________
```

Visualization of Layers Output:

```
#Input Image for Layer visualization
img1 = image.load_img('test/Cat/14.jpg')
plt.imshow(img1);
#preprocess image
img1 = image.load_img('test/Cat/14.jpg', target_size=(64, 64))
img = image.img_to_array(img1)
img = img/255
img = np.expand_dims(img, axis=0)
```


```
model_layers = [ layer.name for layer in classifier.layers]
print('layer name : ',model_layers)
```

```
layer name :  ['conv2d_6', 'max_pooling2d_6', 'conv2d_7', 'max_pooling2d_7', 'flatten_3', 'dense_6', 'dense_7']
```

```
from tensorflow.keras.models import Model
conv2d_6_output = Model(inputs=classifier.input, outputs=classifier.get_layer('conv2d_6').output)
conv2d_7_output = Model(inputs=classifier.input,outputs=classifier.get_layer('conv2d_7').output)
```

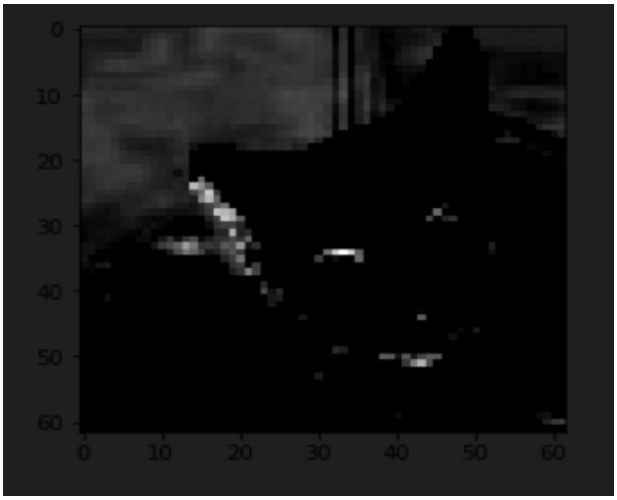
```
    conv2d_6_features = conv2d_6_output.predict(img)
    conv2d_7_features = conv2d_7_output.predict(img)
    print('First conv layer feature output shape : ',conv2d_6_features.shape)
    print('First conv layer feature output shape : ',conv2d_7_features.shape)
```

```
First conv layer feature output shape :  (1, 62, 62, 32)
First conv layer feature output shape :  (1, 29, 29, 32)
```

Single Convolution Filter Output:

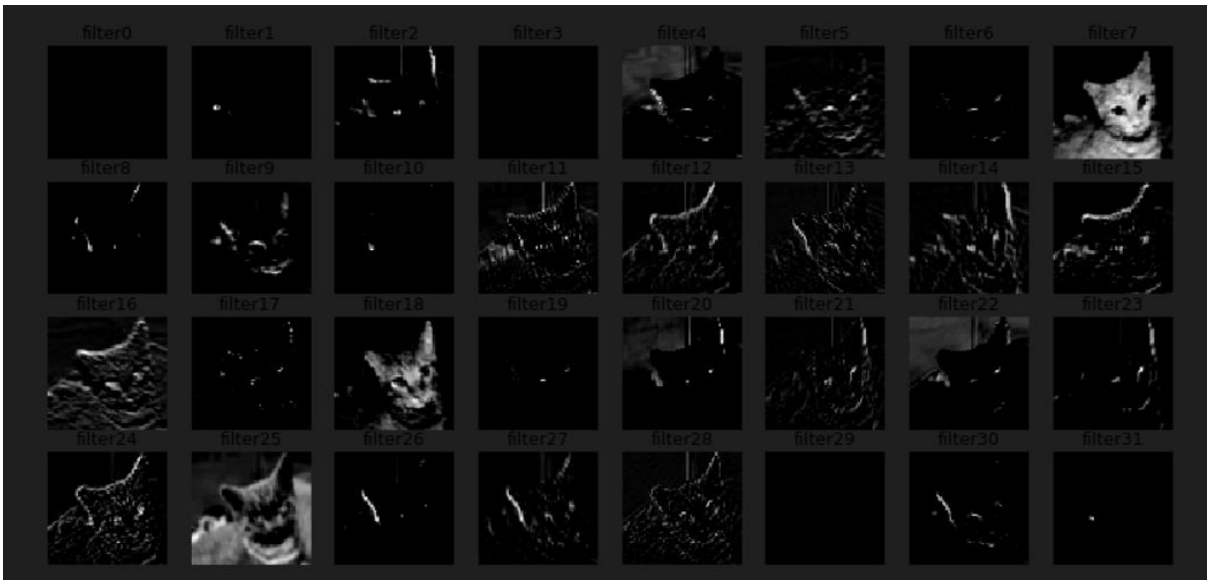
```
    plt.imshow(conv2d_6_features[0, :, :, 4], cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f3b1c90f978>
```



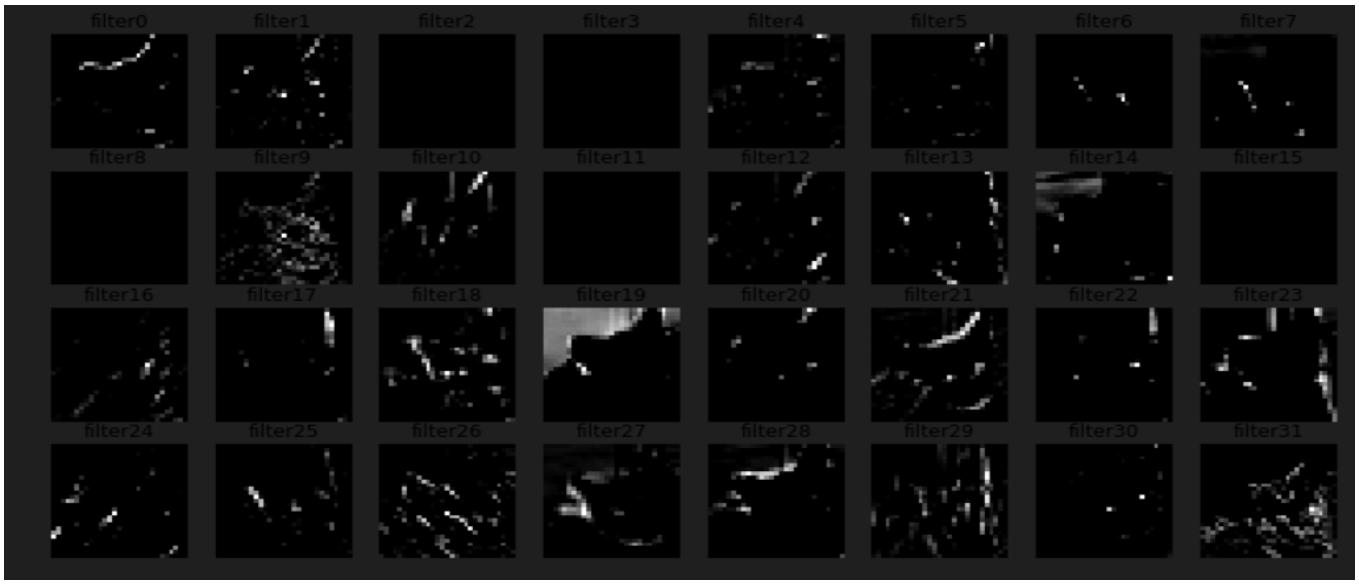
First Convolution Layer Output:

```
import matplotlib.image as mpimg

fig=plt.figure(figsize=(14,7))
columns = 8
rows = 4
for i in range(columns*rows):
    #img = mpimg.imread()
    fig.add_subplot(rows, columns, i+1)
    plt.axis('off')
    plt.title('filter'+str(i))
    plt.imshow(conv2d_6_features[0, :, :, i], cmap='gray')
plt.show()
```



Second Convolution Layer Output:

```
fig=plt.figure(figsize=(14,7))
columns = 8
rows = 4
for i in range(columns*rows):
    #img = mpimg.imread()
    fig.add_subplot(rows, columns, i+1)
    plt.axis('off')
    plt.title('filter'+str(i))
    plt.imshow(conv2d_7_features[0, :, :, i], cmap='gray')
plt.show()
```

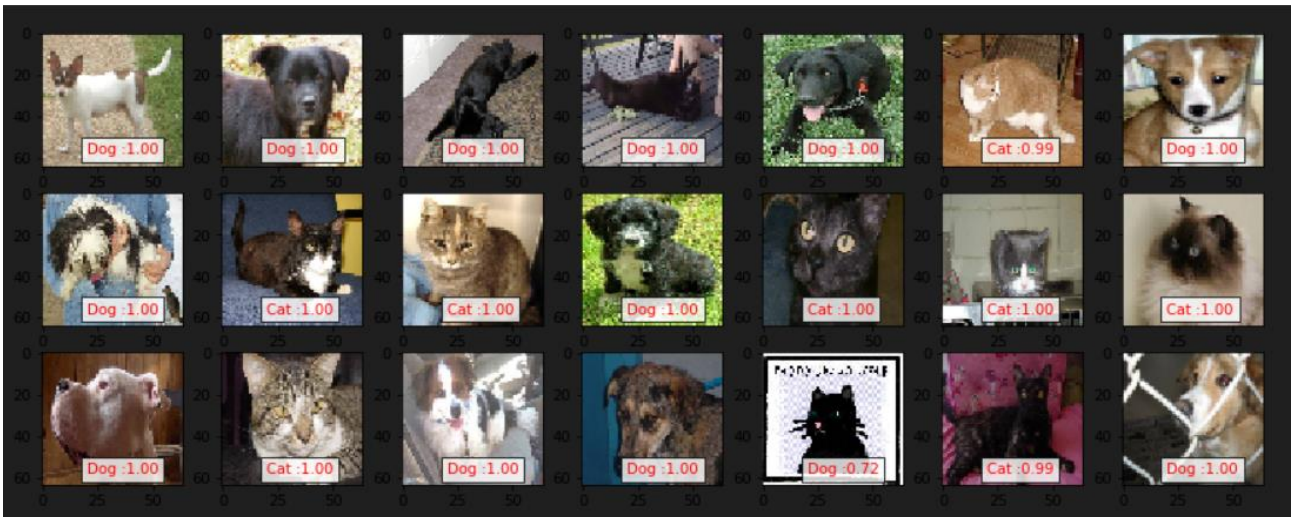


ENROLLMENT:210304124393

Model Performance on Unseen Data:

```
# for generator image set u can use
# ypred = classifier.predict_generator(test_set)

fig=plt.figure(figsize=(15, 6))
columns = 7
rows = 3
for i in range(columns*rows):
    fig.add_subplot(rows, columns, i+1)
    img1 = image.load_img('test1/'+test_set1.filenames[np.random.choice(range(12500))], target_size=(64, 64))
    img = image.img_to_array(img1)
    img = img/255
    img = np.expand_dims(img, axis=0)
    prediction = classifier.predict(img, batch_size=None,steps=1) #gives all class prob.
    if(prediction[:,:]>0.5):
        value ='Dog :%1.2f'%(prediction[0,0])
        plt.text(20, 58,value,color='red',fontsize=10,bbox=dict(facecolor='white',alpha=0.8))
    else:
        value ='Cat :%1.2f'%(1.0-prediction[0,0])
        plt.text(20, 58,value,color='red',fontsize=10,bbox=dict(facecolor='white',alpha=
    plt.imshow(img1)
```

(function) target_size: Any




```
%%capture
# Model Accuracy
x1 = classifier.evaluate_generator(train_set)
x2 = classifier.evaluate_generator(test_set)
```

```
print('Training Accuracy  : %1.2f%%     Training loss  : %1.6f'%(x1[1]*100,x1[0]))
print('Validation Accuracy: %1.2f%%     Validation loss: %1.6f'%(x2[1]*100,x2[0]))
```

```
Training Accuracy  : 99.96%     Training loss  : 0.002454
Validation Accuracy: 97.56%     Validation loss: 0.102678
```