Setting up the Spyder IDE environment and executing ANN program

Aim:

To write a program to implement classification and regression by Setting up the Spyder IDE environment and executing ANN program.

Program:

Classification:

```
#Importing the necessary libraries
import pandas as pd
import numpy as np
from sklearn.datasets import make_classification, make_regression
import tensorflow as tf
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
import sklearn
dir(sklearn.datasets)

#importing the dataset
data = make_classification(200,4,random_state=1)
data

x = data[0]
y = data[1]

#Model Creation
model = Sequential()
model.add(Dense(1135, activation='tanh', input_dim=4))
model.add(Dense(624, activation='relu'))
model.add(Dense(114, activation='relu'))
model.add(Dense(1,activation='sigmoid'))
adam = Adam(0.001)

#model compilation
model.compile(optimizer=adam, loss='binary_crossentropy', metrics=['accuracy'])
print(model.summary())

history = model.fit(x, y, epochs=150, batch_size=5, validation_split=0.2)
pd.DataFrame(model.history.history).reset_index().plot('index', kind='line')

data = pd.DataFrame(history.history)
data.loc[data['accuracy'].idxmax()]
```

Regression:

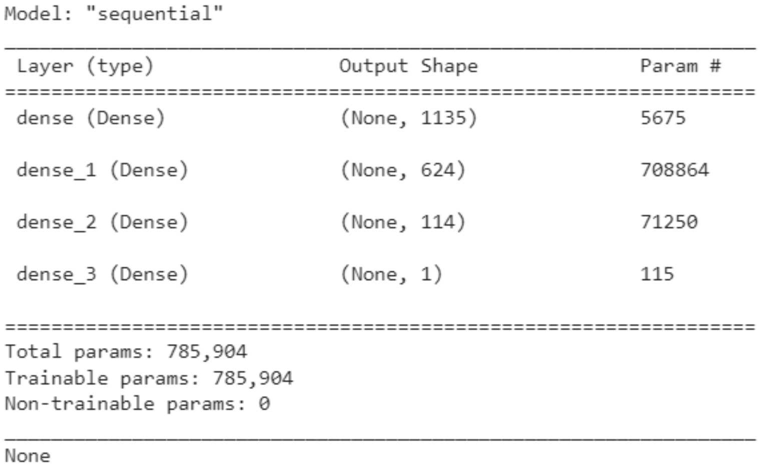
```
#importing the dataset
data = make_regression(200,4,random_state=1)
x=data[0]
y=data[1]

#r2
from keras import backend as k
def r2(y_true,y_pred):
  ss_res = k.sum(k.square(y_true-y_pred))
  ss_tot = k.sum(k.square(y_true-k.mean(y_true)))
  return (1-ss_res/(ss_tot))

#Model Creation
model = Sequential()
```

```
model.add(Dense(1135,activation= 'tanh', input_dim = 4))
model.add(Dense(624, activation='relu'))
model.add(Dense(114))
model.add(Dense(1))
adam = Adam(0.0001)

#Model compilation
model.compile(optimizer=adam, loss='mean_squared_error', metrics=[r2])
print(model.summary())

history = model.fit(x,y,epochs=150, batch_size=5, validation_split=0.2)

pd.DataFrame(model.history.history)[['r2','val_r2']].reset_index().plot('index', kind='line')

data = pd.DataFrame(history.history)
data.loc[data['r2'].idxmax()]
```

OUTPUT:

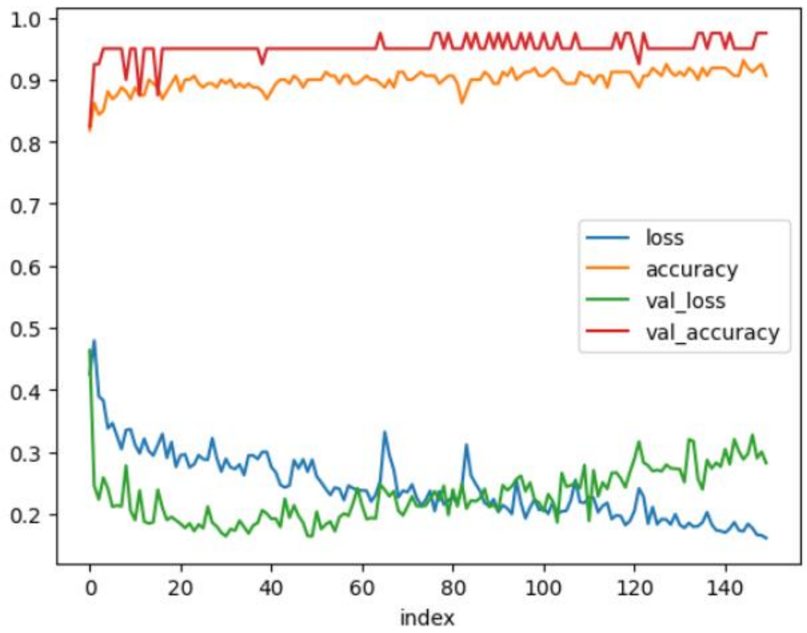
Classification:

#Model Summary

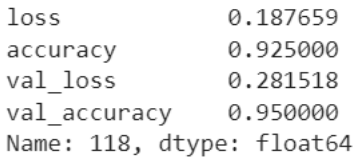


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 1135)              5675

 dense_1 (Dense)             (None, 624)               708864

 dense_2 (Dense)             (None, 114)               71250

 dense_3 (Dense)             (None, 1)                 115

=================================================================
Total params: 785,904
Trainable params: 785,904
Non-trainable params: 0
_________________________________________________________________
None
```

#Model history



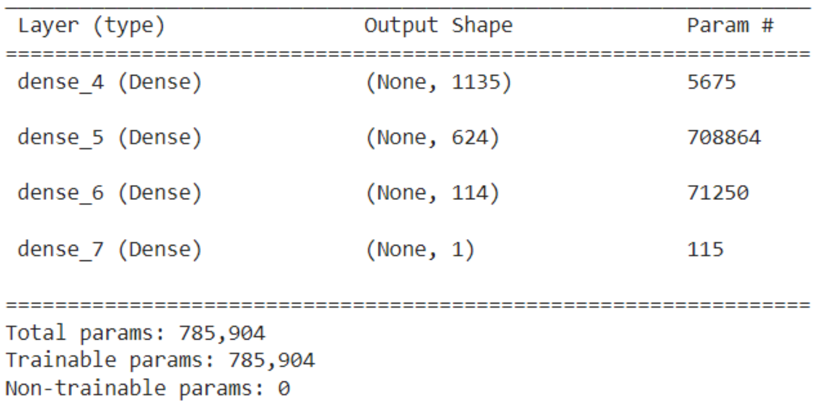
#Max accuracy details



```
loss            0.187659
accuracy        0.925000
val_loss        0.281518
val_accuracy    0.950000
Name: 118, dtype: float64
```

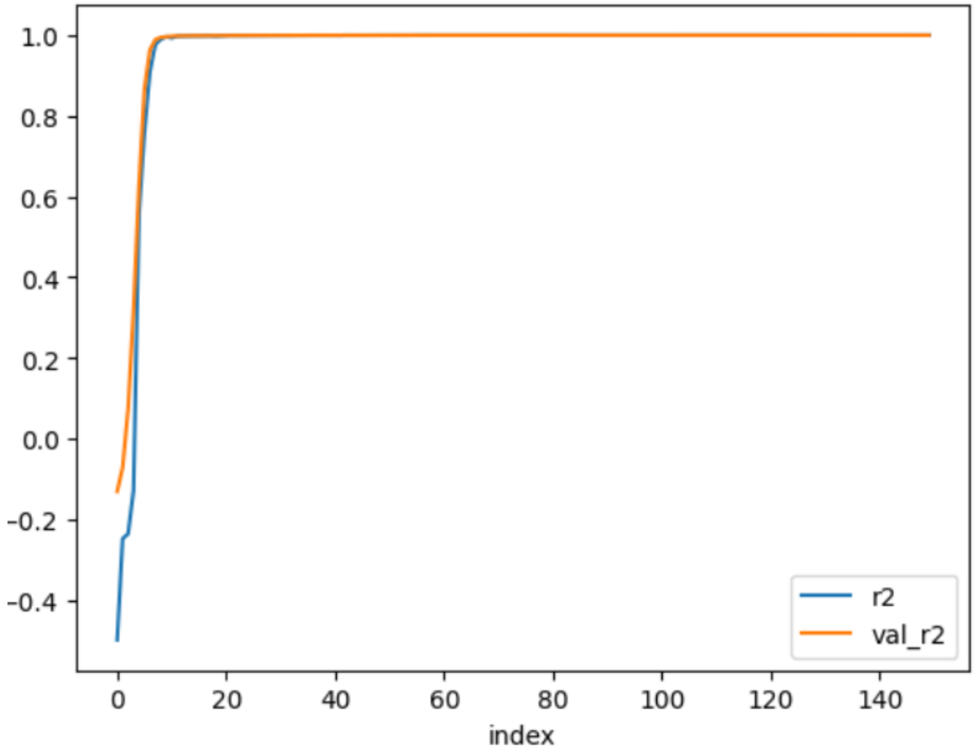
Regression:

#Model Summary

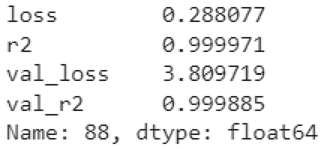


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 1135)              5675

 dense_5 (Dense)             (None, 624)               708864

 dense_6 (Dense)             (None, 114)               71250

 dense_7 (Dense)             (None, 1)                 115

=================================================================
Total params: 785,904
Trainable params: 785,904
Non-trainable params: 0
```

#Model history



#Max r2 details



```
loss        0.288077
r2          0.999971
val_loss    3.809719
val_r2      0.999885
Name: 88, dtype: float64
```

Result:

Thus, classification and regression was implemented successfully by using ANN in the Spyder IDE environment.

Artificial Neural Network

Aim:

To write a program to implement classification and regression using Artificial Neural Networks in python.

Program:

Classification:

```
#Importing the libraries
import numpy as np
import pandas as pd

#Importing the data
data = pd.read_csv('/content/drive/MyDrive/Dataset/healthcare-dataset-stroke-data.csv')
data.head()

data=data.dropna()
data.columns

df=data[['age','hypertension','heart_disease','avg_glucose_level','bmi','stroke']]
df.head()

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['gender'] = le.fit_transform(data['gender'])
df['ever_married'] = le.fit_transform(data['ever_married'])
df['work_type'] = le.fit_transform(data['work_type'])
df['Residence_type'] = le.fit_transform(data['Residence_type'])
df['smoking_status'] = le.fit_transform(data['smoking_status'])
```

```
df.head()

data.shape

df.shape

x = df.drop('stroke',axis=1)
y = df['stroke']

x.shape
y.shape

import seaborn as sns
import matplotlib.pyplot as plt
sns.countplot(df,x='stroke')
plt.xticks([0,1],['NO','YES'])
plt.title('COUNT PLOT')

from sklearn.model_selection import train_test_split as tts
x_train, x_test, y_train, y_test= tts(x,y,test_size=0.2)

#Model Creation and compilation
import tensorflow as tf
from tensorflow.keras.layers import Dense
ann = tf.keras.Sequential()
ann.add(Dense(units=25, activation='relu'))
ann.add(Dense(units=25,activation='relu'))
ann.add(Dense(units=1,activation='sigmoid'))
ann.compile('adam','binary_crossentropy',metrics=['accuracy'])

result = ann.fit(x_train,y_train,epochs=10)
```

```
y_pred=[]
for i in ann.predict(x_test):
  if i>0.5:
    y_pred.append(1)
  if i<0.5:
    y_pred.append(0)

from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,y_pred)

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

Regression:

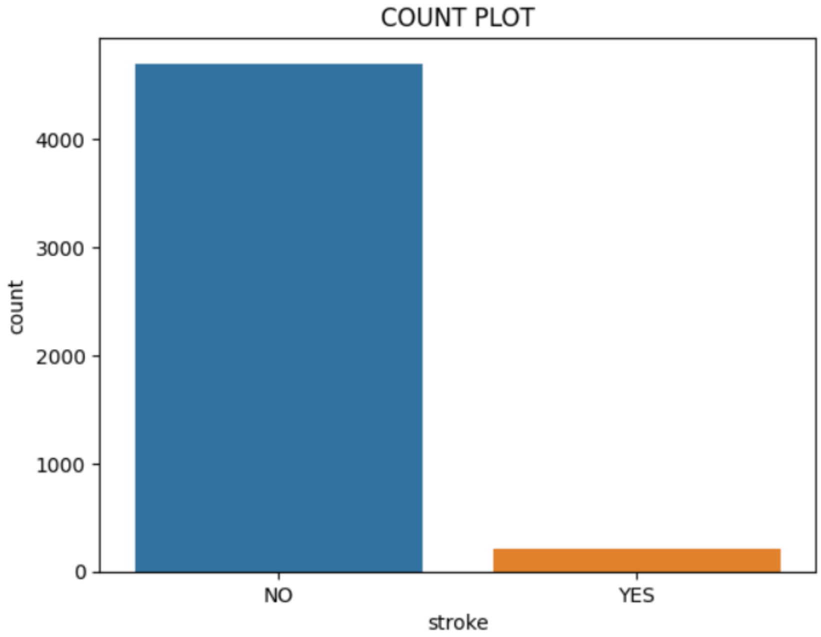
```
#Importing the necessary libraries
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Dense
from tensorflow.keras import Sequential

#Importing the data
df = pd.read_csv('/content/drive/MyDrive/Dataset/CarPrice_Assignment (1).csv')
df

df.columns
```

```
df = pd.get_dummies(df)
df.head()

x = df.drop(['price'],axis=1)

y = df.price

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)

x_train.shape

#r2
from keras import backend as k
def r2(y_true,y_pred):
  ss_res = k.sum(k.square(y_true-y_pred))
  ss_tot = k.sum(k.square(y_true-k.mean(y_true)))
  return (1-ss_res/(ss_tot))

#Model creation and compilation
model = Sequential()
model.add(Dense(400,activation='relu',input_dim=198))
model.add(Dense(240,activation='relu'))
model.add(Dense(1))

model.compile(optimizer='Adam',loss='mean_absolute_error',metrics=[r2])

hist = model.fit(x_train,y_train,epochs=150, batch_size=5, validation_split=0.3)

pd.DataFrame(model.history.history).reset_index().plot('index',kind='line')
pd.DataFrame(model.history.history)[['r2','val_r2']].reset_index().plot('index', kind='line')

data = pd.DataFrame(hist.history)
data.loc[data['r2'].idxmax()]
```

Output:

Classification:

#Count plot of y



#Confusion matrix

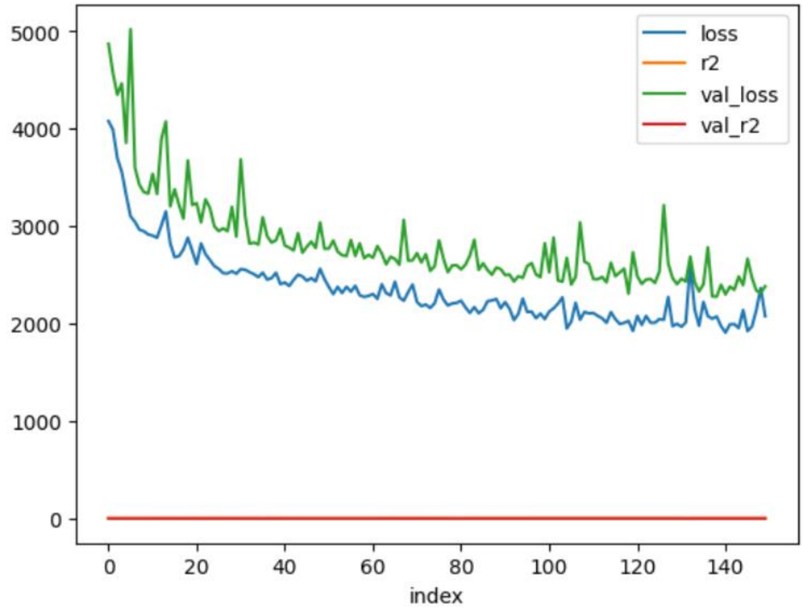
```
array([[935,   4],
       [ 43,   0]])
```

#Accuracy:

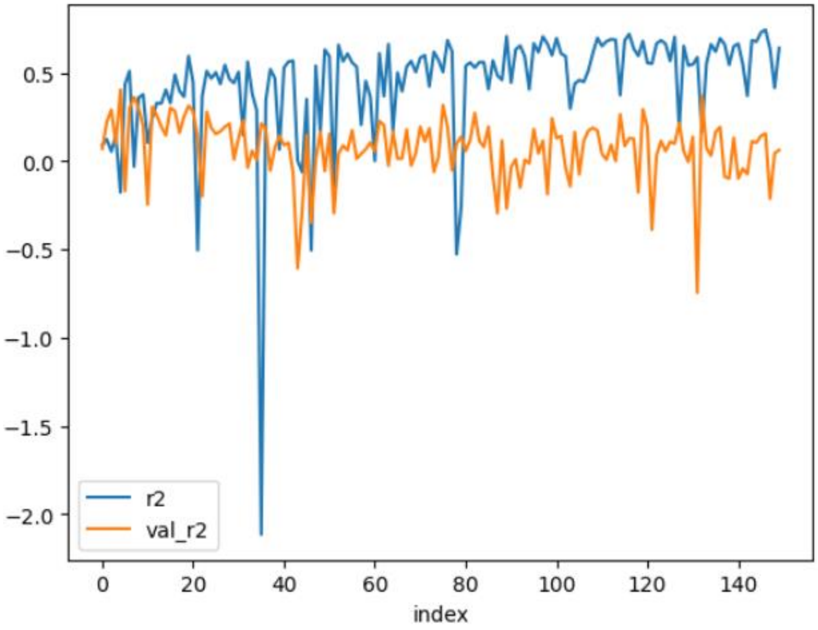
```
0.9521384928716904
```

Regression:

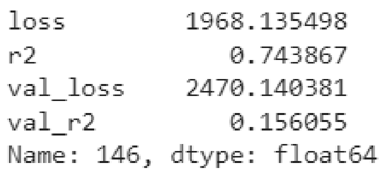
#Model history



#r2 vs val_r2



#Max r2 details



```
loss        1968.135498
r2             0.743867
val_loss    2470.140381
val_r2         0.156055
Name: 146, dtype: float64
```

Result:

Thus, classification and regression was implemented using Artificial Neural Networks in python and executed successfully.

Convolutional Neural Network with Text data

Aim:

To write a program to implement text data images classification using convolution neural network.

Program:

Classification:

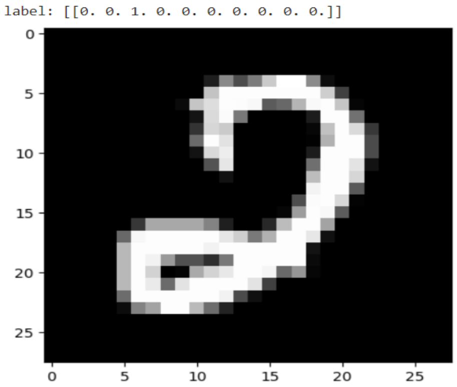
```
#Importing the libraries
import numpy as np
import tensorflow as tf
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
import warnings
warnings.filterwarnings('ignore')

#initializing the values
batch_size = 128
num_classes = 10
epochs = 10

#splitting the dataset into train and test data
(x_train,y_train),(x_test,y_test) = mnist.load_data()

x_train = x_train.reshape(60000,28,28,1)
x_test = x_test.reshape(10000,28,28,1)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
```

```
#normalize values 0 to 1 range
x_train /= 255
x_test /= 255
print(x_train.shape[0],'train samples')
print(x_test.shape[0],'test samples')

#Convert class vectors to binary class matrices
from tensorflow.keras.utils import to_categorical
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)

#viewing an image
import pylab as plt
print('label:',y_test[400:401])
plt.imshow(x_test[400:401].reshape(28,28),cmap='gray')
plt.show()

#model creation
model = Sequential()
model.add(Conv2D(8,kernel_size=(3,3),activation='relu',input_shape=(28,28,1)))
model.add(Conv2D(filters=16, kernel_size=(3,3), activation = 'relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(32,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.summary()
```

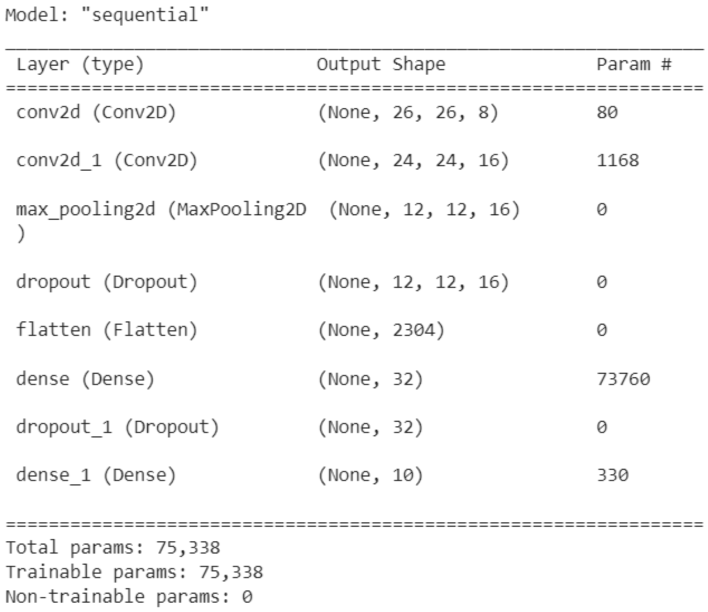
```
#model compilation
model.compile(loss=keras.losses.categorical_crossentropy,optimizer=tf.keras.optimizers.Adam(),metrics=['accuracy'])

model.fit(x_train,y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(x_test,y_test))

#model evaluation
score = model.evaluate(x_test,y_test,verbose=0)
print('Test Loss: ',score[0])
print('Test Accuracy: ', score[1])

#comparing the original image with predicted value
import pylab as plt
plt.imshow(x_test[706:707].reshape(28,28),cmap='gray')

prediction = model.predict(x_test[706:707])
print('Prediction score: \n', prediction[0])
thresholded = (prediction>0.5)*1
print('\n Thresholded score: \n',thresholded[0])
print('Predicted digit: \n',np.where(thresholded==1)[1][0])
```

Output:

#Visualizing a data

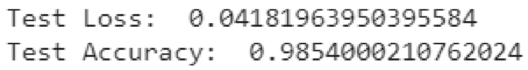


#Model Summary



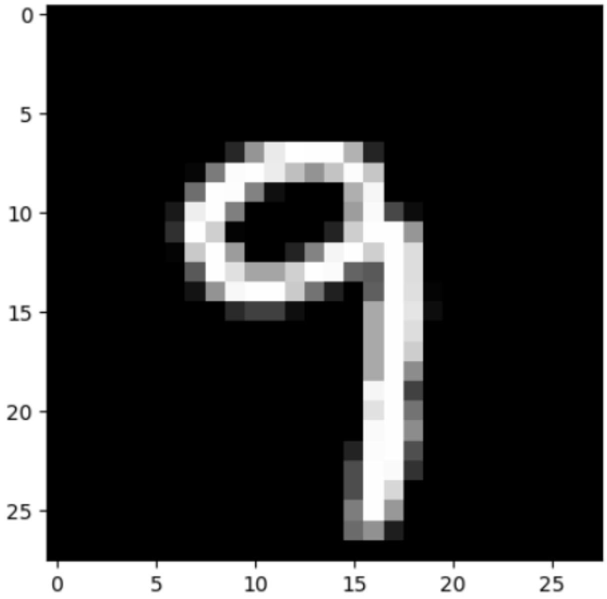
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 8)         80

 conv2d_1 (Conv2D)           (None, 24, 24, 16)        1168

 max_pooling2d (MaxPooling2D  (None, 12, 12, 16)       0
 )

 dropout (Dropout)           (None, 12, 12, 16)        0

 flatten (Flatten)           (None, 2304)              0

 dense (Dense)               (None, 32)                73760

 dropout_1 (Dropout)         (None, 32)                0

 dense_1 (Dense)             (None, 10)                330

=================================================================
Total params: 75,338
Trainable params: 75,338
Non-trainable params: 0
```

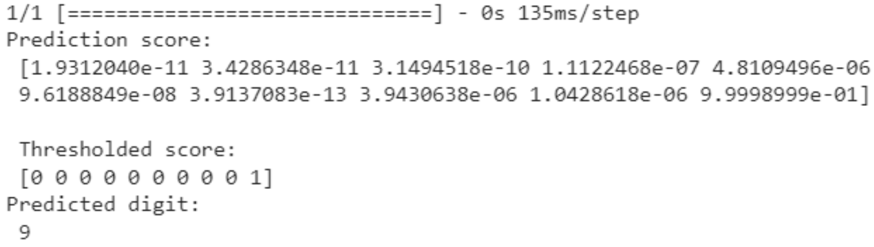
#Model Evaluation



```
Test Loss:  0.04181963950395584
Test Accuracy:  0.9854000210762024
```

#comparing the original image with predicted value





```
1/1 [==============================] - 0s 135ms/step
Prediction score:
 [1.9312040e-11 3.4286348e-11 3.1494518e-10 1.1122468e-07 4.8109496e-06
 9.6188849e-08 3.9137083e-13 3.9430638e-06 1.0428618e-06 9.9998999e-01]

 Thresholded score:
 [0 0 0 0 0 0 0 0 0 1]
Predicted digit:
 9
```

Result:

Thus, text data image classification was implemented using convolution neural network and executed successfully.

Convolutional Neural Network with Ciar-10 data

Aim:

To write a program to implement cifar-10 images classification using convolution neural network in python.

Program:

Classification:

```
#Importing the libraries
import numpy as np
import tensorflow as tf
import keras
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
import matplotlib.pyplot as plt
from tensorflow.keras.utils import to_categorical
import warnings
warnings.filterwarnings('ignore')

#Importing data
(x_train,y_train),(x_test,y_test) = cifar10.load_data()
print(f"x_train shape: {x_train.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"x_test shape: {x_test.shape}")
print(f"y_test shape: {y_test.shape}")

# Define the labels of the dataset
labels = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

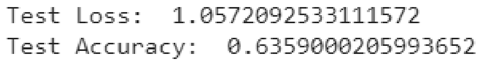
```
#visulaization
fig, axes = plt.subplots(10,10, figsize = (17,17))
axes = axes.ravel()
n_train = len(x_train)
for i in np.arange(0, 100):
  index = np.random.randint(0,n_train)
  axes[i].imshow(x_train[index,1:])
  label_index = int(y_train[index])
  axes[i].set_title(labels[label_index], fontsize = 8)
  axes[i].axis('off')
plt.subplots_adjust(hspace=0.4)

print('label:',labels[int(y_train[666])])
plt.figure(figsize=(.8,1))
plt.imshow(x_train[666,1:])
plt.show()

classes, counts = np.unique(y_train, return_counts=True)
plt.barh(labels, counts)
plt.title('Class distribution in training set')

classes, counts = np.unique(y_test, return_counts=True)
plt.barh(labels, counts)
plt.title('Class distribution in testing set')

#normalize values 0 to 1 range
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

```
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

num_classes = 10

#model creation
model = Sequential()
model.add(Conv2D(8,kernel_size=(3,3),activation='relu',input_shape=(32,32,3)))
model.add(Conv2D(filters=16, kernel_size=(3,3), activation = 'relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(32,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.summary()

#model compilation
model.compile(loss=keras.losses.categorical_crossentropy,optimizer=tf.keras.optimizers.Adam()
,metrics=['accuracy'])

model.fit(x_train,y_train,batch_size=100,epochs=50,verbose=1,validation_data=(x_test,y_test))

#model evaluation
score = model.evaluate(x_test,y_test,verbose=0)
print('Test Loss: ',score[0])
print('Test Accuracy: ', score[1])
```

```
#Visualize and evaluate
plt.imshow(x_test[856])

threshold=[]
for i in prediction[0]:
  if i==max(prediction[0]):
    threshold.append(1)
  else:
    threshold.append(0)
thresholded=threshold
thresholded=np.array(thresholded)
print('\n Thresholded score: \n',thresholded)
im_index = np.where(thresholded==1)
print('Predicted image: \n', labels[im_index[0][0]])
```

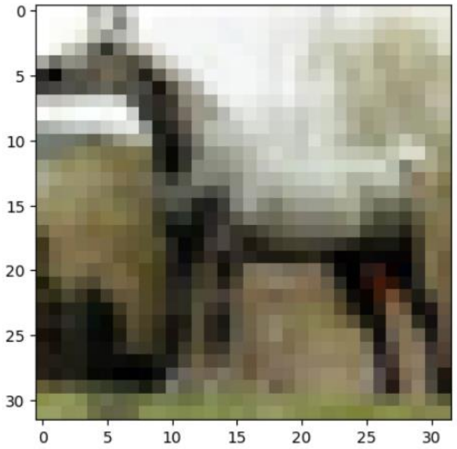
Output:

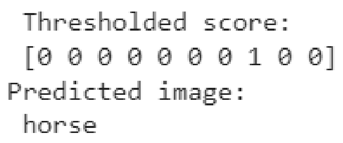
#Model Evaluation



```
Test Loss:  1.0572092533111572
Test Accuracy:  0.6359000205993652
```

#Visualization and evaluation





```
 Thresholded score:
 [0 0 0 0 0 0 0 1 0 0]
Predicted image:
 horse
```

Result:

Thus, cifar-10 images classification was implemented using convolution neural network in python and executed successfully.

Recurrent Neural Network

Aim:

To write a program to implement sequence classification using recurrent neural network in python.

Program:

```
# Importing the libraries
import pandas as pd
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from keras.datasets import reuters
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from keras.layers import Dense, SimpleRNN, Activation, LSTM, GRU
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import accuracy_score

#Initializing the values
num_words = None
maxlen = 50
test_split = 0.3

#Splitting the dataset into train and test sets
(x_train,y_train),(x_test,y_test) = reuters.load_data(num_words = num_words, maxlen = maxlen, test_split = test_split)
```

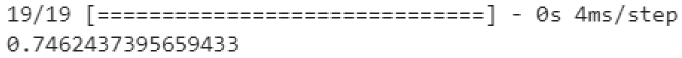
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)

from numpy.core.fromnumeric import shape
x_train = pad_sequences(x_train, padding = 'post')
x_test = pad_sequences(x_test, padding = 'post')
x_train = np.array(x_train).reshape((x_train.shape[0],x_train.shape[1],1))
x_test = np.array(x_test).reshape((x_test.shape[0],x_test.shape[1],1))

x_train.shape, x_test.shape

y_train.shape, y_test.shape

y_data = np.concatenate((y_train,y_test))
y_data = to_categorical(y_data)

y_train = y_data[:1395]
y_test = y_data[1395:]

#Simple RNN
model = Sequential()
model.add(SimpleRNN(50, input_shape=(49,1)))
model.add(Dense(46))
model.add(Activation('softmax'))
adam = Adam(learning_rate = 0.001)

#Model compilation
model.compile(loss = 'categorical_crossentropy', optimizer = adam, metrics = ['accuracy'])
model.fit(x_train,y_train, epochs = 100, validation_split=0.3)
```

```
#Model Evaluation
y_pred = np.argmax(model.predict(x_test),axis = 1)
y_test = np.argmax(y_test, axis = 1)
print(accuracy_score(y_pred,y_test))

#LSTM
#Model Creation
ls = Sequential()
ls.add(LSTM(50, input_shape=(49,1)))
ls.add(Dense(46))
ls.add(Activation('softmax'))

from keras import metrics
adam = Adam(learning_rate = 0.001)

#Model compilation
ls.compile(loss = 'categorical_crossentropy', optimizer = adam, metrics = ['accuracy'])
ls.fit(x_train,y_train, epochs = 100, validation_split=0.3)

#Model Evaluation
y_pred = np.argmax(ls.predict(x_test), axis = 1)
#y_test = np.argmax(y_test,axis = 1)
print(accuracy_score(y_pred, y_test))

#GRU
#Model Creation
gr = Sequential()
gr.add(GRU(50, input_shape=(49,1)))
gr.add(Dense(46))
gr.add(Activation('sigmoid'))
```

```
#Model compilation
gr.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
gr.fit(x_train,y_train, epochs = 10, validation_split = 0.3)

#Model Evaluation
y_pred = np.argmax(gr.predict(x_test), axis = 1)
#y_test = np.argmax(y_test,axis = 1)
print(accuracy_score(y_pred, y_test))
```

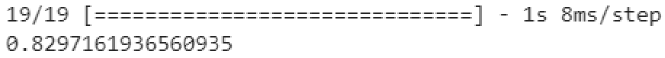
Output:

#Simple RNN accuracy



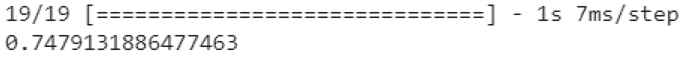
```
19/19 [==============================] - 0s 4ms/step
0.7462437395659433
```

#LSTM accuracy



```
19/19 [==============================] - 1s 8ms/step
0.8297161936560935
```

#GRU accuracy



```
19/19 [==============================] - 1s 7ms/step
0.7479131886477463
```

Result:

Thus, sequence classification program was implemented using recurrent neural network and executed successfully.

Image Segmentation

Aim:

To demonstrate and compare different thresholding techniques in image segmentation and highlight their effects on image.

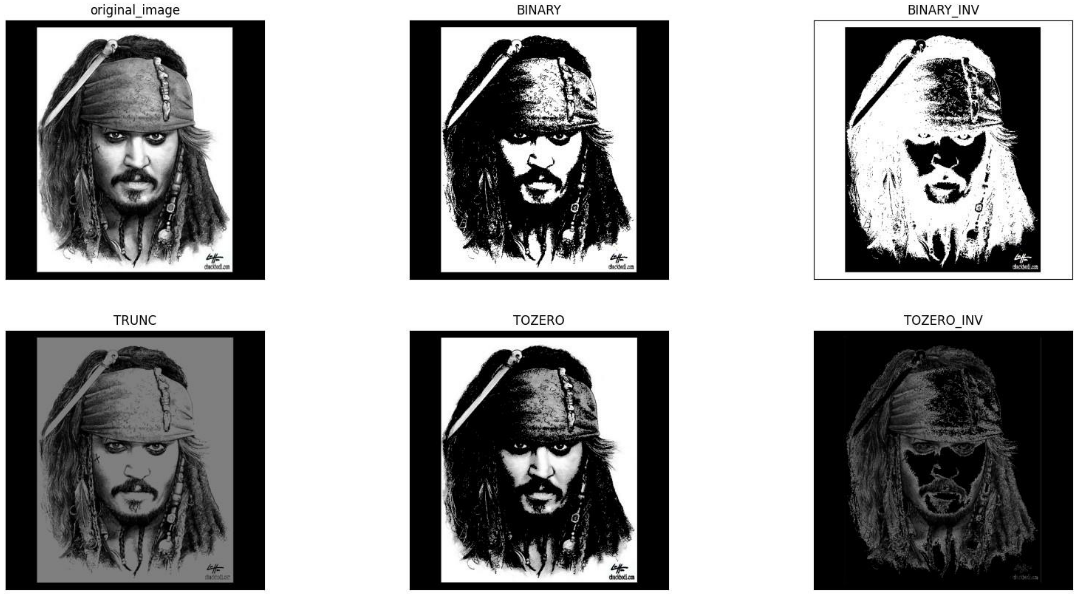
Program:

```
# Importing the libraries
import cv2
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

#Loading the image
image = cv2.imread('/content/jack_sparrow.jpg',flags =-1) #cv2.IMREAD_GRAYSCALE
image

#Simple Thresholding
ret, thresh1 = cv2.threshold(image,127, 255, cv2.THRESH_BINARY)
ret, thresh2 = cv2.threshold(image,127, 255, cv2.THRESH_BINARY_INV)
ret, thresh3 = cv2.threshold(image,127, 255, cv2.THRESH_TRUNC)
ret, thresh4 = cv2.threshold(image,127, 255, cv2.THRESH_TOZERO)
ret, thresh5 = cv2.threshold(image,127, 255, cv2.THRESH_TOZERO_INV)

titles = ['original_image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
images = [image,thresh1,thresh2,thresh3,thresh4,thresh5]
```

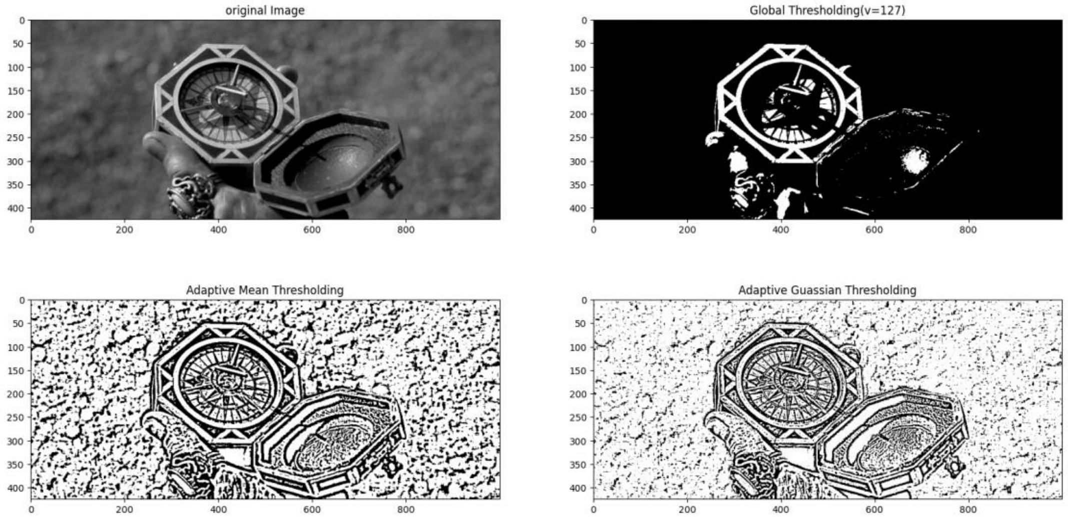
```
plt.figure(figsize=(20,10))
for i in range(6):
  plt.subplot(2,3,i+1)
  plt.imshow(images[i]),'gray')
  plt.xticks([])
  plt.yticks([])
  plt.title(titles[i])
plt.show()

# Adaptive Thresholding
img = cv2.imread('/content/campass.jpg')
img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
thresh2 = cv2.adaptiveThreshold(img,225,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,11,2)
thresh3 = cv2.adaptiveThreshold(img,225,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,11,2)

titles = ['original Image','Global Thresholding(v=127)','Adaptive Mean Thresholding','Adaptive Guassian Thresholding']
images = [img,thresh1,thresh2,thresh3]
plt.figure(figsize=(20,10))
for i in range(4):
  plt.subplot(2,2,i+1)
  plt.imshow(images[i],'gray')
  plt.title(titles[i])
  plt.xticks()
  plt.yticks()
plt.show()
```

Output:

#Simple thresholding



#Adaptive thresholding


original Image
Global Thresholding(v=127)
Adaptive Mean Thresholding
Adaptive Guassian Thresholding
0
50
100
150
200
250
300
350
400
0
200
400
600
800

Result:

Thus, the thresholding techniques in image segmentation was executed successfully and highlighted their effects on image.

Image Transformations

Aim:

To demonstrate various image transformations, such as Morphological and Geometric operations using Computer Vision.

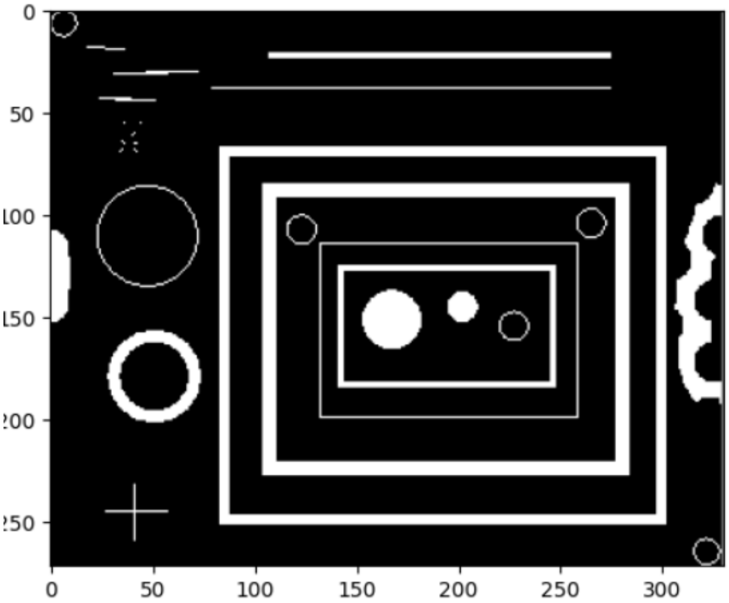
Program:

```
#Importing the necessary libraries
import cv2
import cv2
import matplotlib.pyplot as plt
import numpy as np

# Geometric Transformations of Images
# Scaling
img = cv2.imread('/content/morph_dilate.png')
height, width = img.shape[:2]
res = cv2.resize(img,(2*width, 5*height), interpolation = cv2.INTER_NEAREST)

plt.imshow(img)

print(res.shape)
plt.imshow(res)

# Translation
img = cv2.imread('/content/morph_dilate.png',0)
rows,cols = img.shape
M = np.float32([[1,0,500],[0,1,400]])
dst = cv2.warpAffine(img,M,(cols,rows))
plt.imshow(dst)
```

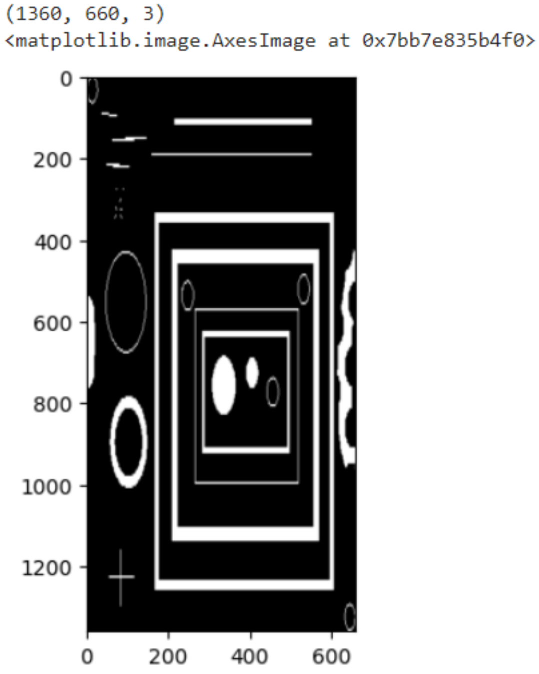
```
# Rotation
M = cv2.getRotationMatrix2D((cols/2,rows/2),50,1)
dst = cv2.warpAffine(img,M,(cols,rows))
plt.imshow(dst,'gray')

# Morphological Transformations of Images
# Erosion
img = cv2.imread('/content/morph_dilate.png',0)
kernel = np.ones((3,3),np.uint8)
plt.imshow(img,'gray')

erosion = cv2.erode(img,kernel,iterations = 1)
plt.imshow(erosion,'gray')

# Dilation
dilation = cv2.dilate(img,kernel,iterations = 1)
plt.imshow(dilation,'gray')

# Opening
img=cv2.imread('/content/morph_dilate.png')[0:600,0:1000]
plt.imshow(img)

opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
plt.imshow(opening)

# Closing
closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
plt.imshow(closing)
```

Output:

#Original image

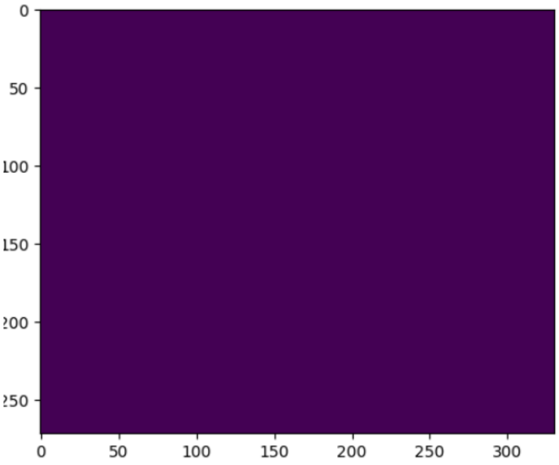


#scaled image

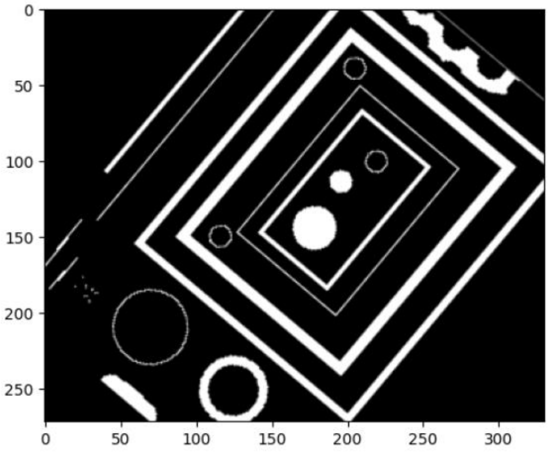


```
(1360, 660, 3)
<matplotlib.image.AxesImage at 0x7bb7e835b4f0>
```

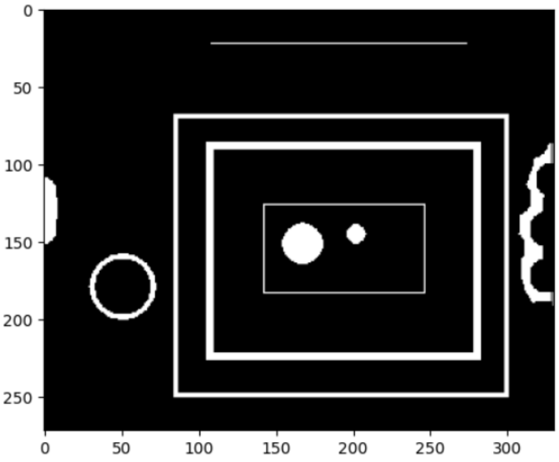
#Translation transformation



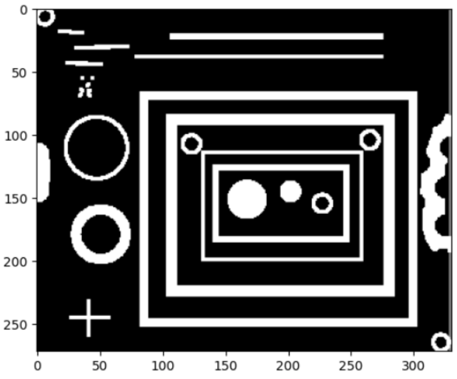
#Rotational transformation



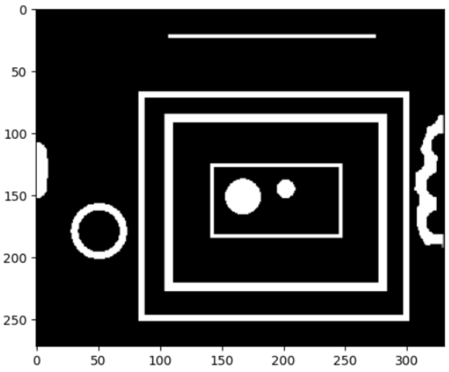
#Erosion operation



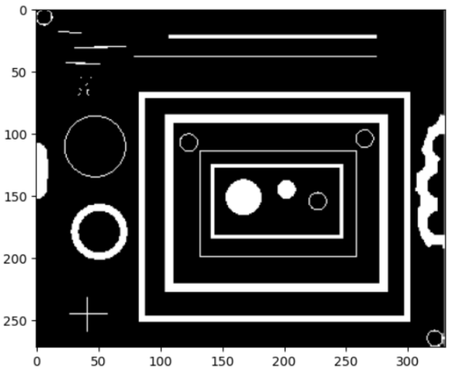
#Dilation operation



#Opening operation



#Closing operation



Result:

Thus, the image transformations such as Morphological and Geometric transformations was executed successfully using Computer Vision.

Image Gradients and Edge Detection

Aim:

To write a program explore image gradients and edge detection techniques using Computer Vision.

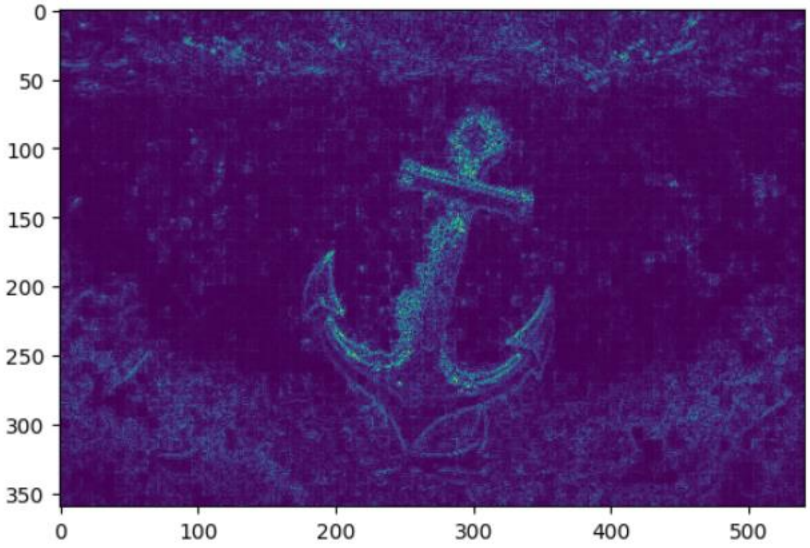
Program:

```
# Importing the libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt

#Loading the Image
image1 = cv2.imread('/content/Anchor.jpg')
image = cv2.cvtColor(image1,cv2.COLOR_BGR2GRAY)

# Importing the libraries
lap = cv2.Laplacian(image,cv2.CV_64F)
lap = np.uint8(np.absolute(lap))
plt.imshow(lap)

lm = cv2.cvtColor(image1,cv2.COLOR_BGRA2GRAY)
plt.imshow(lm)

sobelx = cv2.Sobel(image1,cv2.CV_64F,1,0)
sobely = cv2.Sobel(image1,cv2.CV_64F,0,1)
sobelx = np.uint8(np.absolute(sobelx))
sobely = np.uint8(np.absolute(sobely))
Sobelcombinbed = cv2.bitwise_or(sobelx,sobely)
```

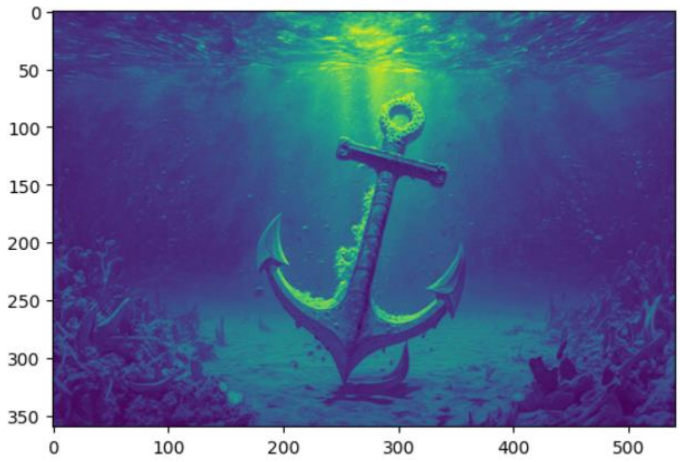
```
plt.imshow(sobelx)

plt.imshow(sobely)

img=cv2.imread('/content/Anchor.jpg')
edges=cv2.Canny(img,100,200)

plt.imshow(img)
plt.title('original')
plt.show()

plt.imshow(edges)
plt.title('Canny Edge Detection')
plt.show()
```

Output:

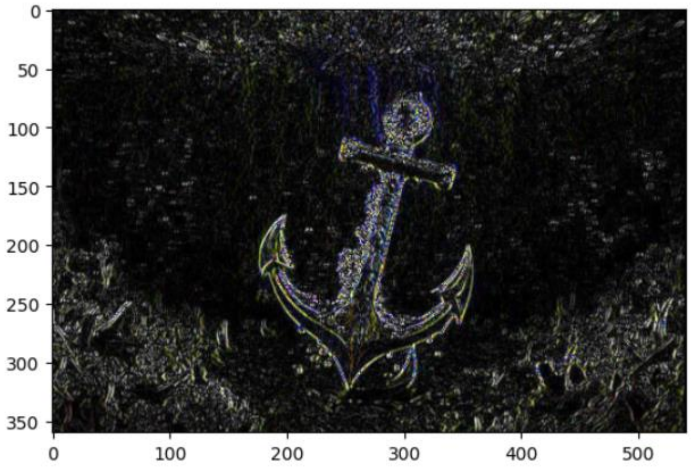
#Laplacian gradient



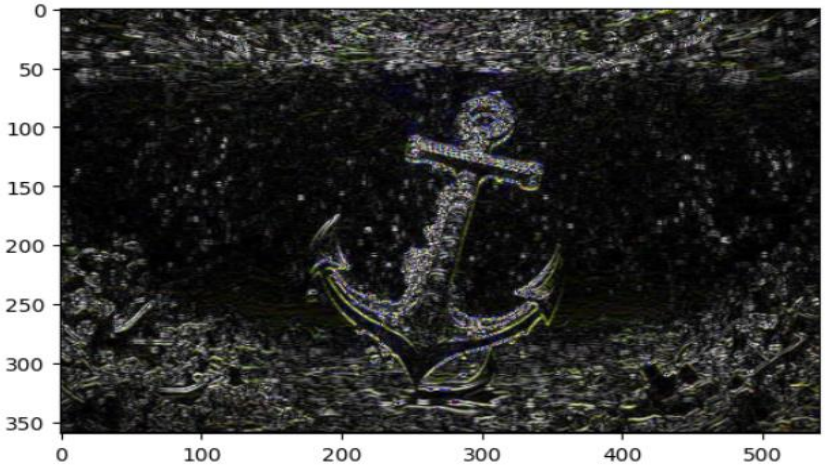
#Grayscale conversion



#Sobel Gradient – X

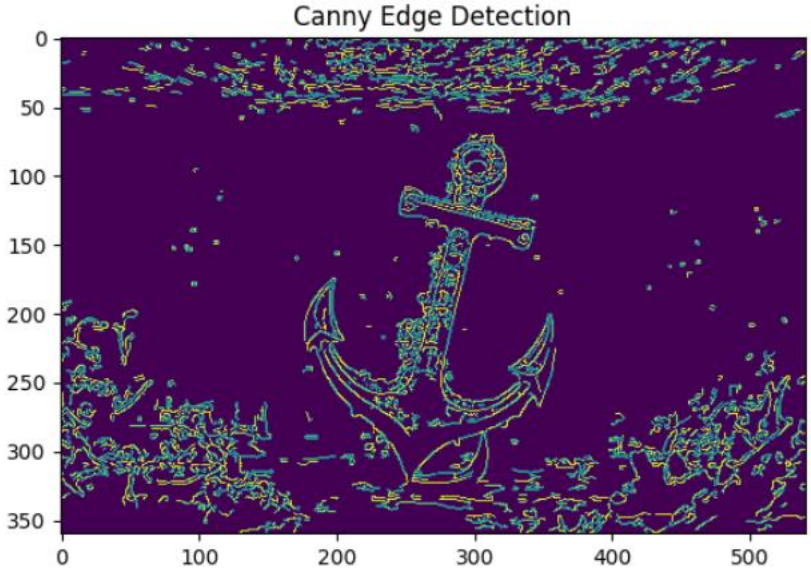


#Sobel Gradient – Y



#Original

#Canny Edge Detection



Result:

Thus, image gradients such as Sobel-X, Sobel-Y and SobelXY and Canny edge detection technique were implemented successfully.

Harris Corner Detection

Aim:

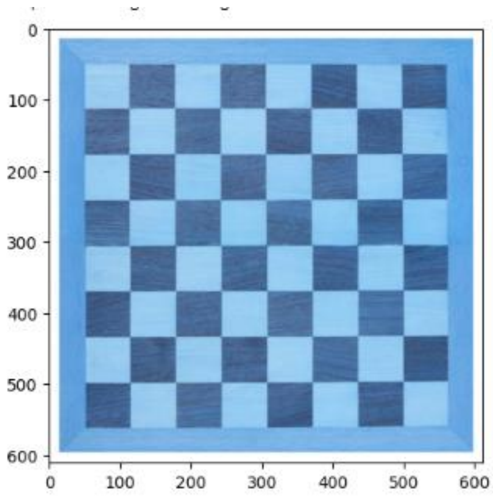
To write a program to implement Harris corner detection using computer vision.

Program:

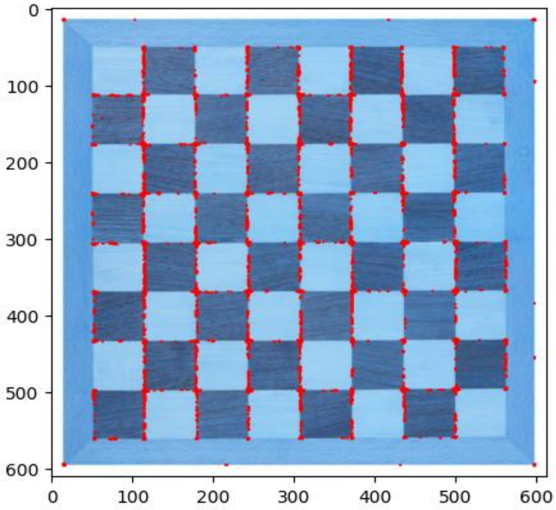
```
# Importing the libraries
import cv2
import matplotlib.pyplot as plt

#loading the image
img = cv2.imread('/content/drive/MyDrive/Deep learning/Images/chess.jpg')
plt.imshow(img)

#converting the image color
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
plt.imshow(gray)

#Harris corner detection
dst = cv2.cornerHarris(gray,2,3,0,.4)
dst = cv2.dilate(dst,None)
plt.imshow(dst)

img[dst>0.01*dst.max()] = [255,0,0]
plt.figure(figsize=(10,10))
plt.imshow(img)
```

Output:

#Original image



#Corner detected image



Result:

Thus, the program to perform the Harris corner detection has been completed successfully.

Image Contours

Aim:

To demonstrate how to find and visualize contours in an image using Python and OpenCV.

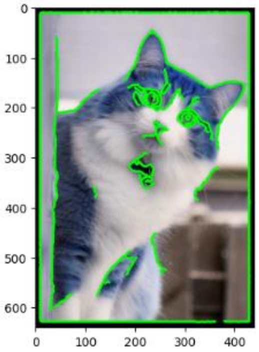
Program:

```
# Importing the libraries
import cv2
import matplotlib.pyplot as plt

#Image Contours
image = cv2.imread('/content/cat.jpg')
gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(gray,(11,11),0)
edged = cv2.Canny(blurred,30,150)
cnts,_ =
cv2.findContours(edged.copy(),cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
coins = image.copy()
cv2.drawContours(coins,cnts,-1,(0,255,0),3)

plt.imshow(coins)

plt.imshow(cv2.cvtColor(image,cv2.COLOR_BGR2RGB))

plt.imshow(edged)
```

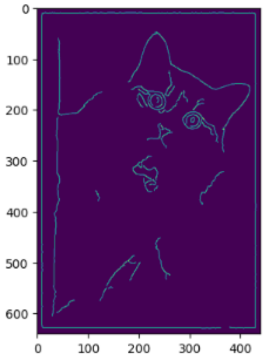
Output:

#Original image

#Image with contours



#Canny (edged)



Result:

Thus, the contours were visualized successfully in an image using Python and OpenCV.

Face Detection using Haar Cascade

Aim:

To import image and detect the face using HAAR cascade.

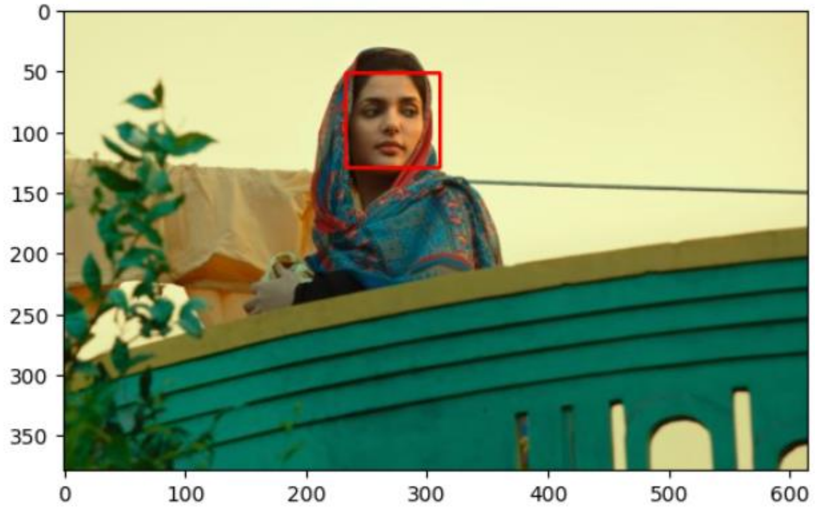
Program:

```
# Importing the libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Importing the xml file
facecascade = cv2.CascadeClassifier('/content/drive/MyDrive/Dataset/xml file/face.xml')

# Importing the image
img = cv2.imread("/content/Gypsy.jpg")
img2 = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.imshow(img2)

#Face Detection using detectMultiScale
face_img=img2.copy()
face_rect=facecascade.detectMultiScale(face_img,scaleFactor=1.2,minNeighbors=5)
for(x,y,w,h)in face_rect:
  cv2.rectangle(face_img,(x,y),(x+w,y+h),(245,5,0),2)
plt.imshow(face_img)
```

Output:

#Original image

#detection of face



Result:

Thus, the program to perform the face detection using Haar Cascade has been completed successfully.

Chatbot Creation

Aim:

To build a simple chatbot using natural language processing and deep learning techniques.

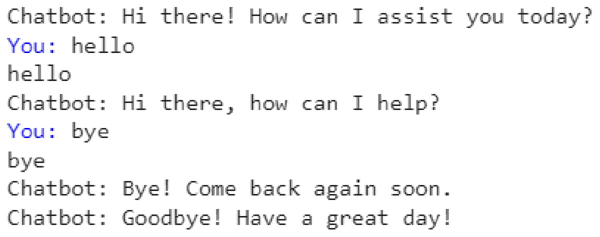
Program:

```
 %%writefile content.json
 {"intents": [
        {"tag": "greeting",
         "patterns": ["Hi there", "How are you", "Is anyone there?","Hey","Hola", "Hello", "Good day"],
         "responses": ["Hello", "Good to see you again", "Hi there, how can I help?"],
         "context": [""]
        },
        {"tag": "goodbye",
         "patterns": ["Bye", "See you later", "Goodbye", "Nice chatting to you, bye", "Till next time"],
         "responses": ["See you!", "Have a nice day", "Bye! Come back again soon."],
         "context": [""]
        },
        {"tag": "thanks",
         "patterns": ["Thanks", "Thank you", "That's helpful", "Awesome, thanks", "Thanks for helping me"],
         "responses": ["My pleasure", "You're Welcome"],
         "context": [""]
        },
        {"tag": "query",
         "patterns": ["What is Simplilearn?"],
         "responses": ["Simplilearn is the popular online Bootcamp & online courses learning platform "],
         "context": [""]
        }
     ]}
```

```
#importing necessary libraries
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, GlobalAveragePooling1D
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence  import pad_sequences
from sklearn.preprocessing import LabelEncoder
import json

#Loading the intents.json file
with open('/content/content.json') as file:
  data = json.load(file)

training_sentences = []
training_labels = []
labels = []
responses = []
for intent in data['intents']:
  for pattern in intent['patterns']:
    training_sentences.append(pattern)
    training_labels.append(intent['tag'])
  responses.append(intent['responses'])
  if intent['tag'] not in labels:
    labels.append(intent['tag'])
num_classes = len(labels)

lbl_encoder = LabelEncoder()
lbl_encoder.fit(training_labels)
training_labels = lbl_encoder.transform(training_labels)

vocab_size = 1000
embedding_data = 16
max_len = 20
oov_token = "<oov>"
tokenizer = Tokenizer(num_words = vocab_size, oov_token = oov_token)
tokenizer.fit_on_texts(training_sentences)
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(training_sentences)
padded_sequences = pad_sequences(sequences,truncating = 'post',maxlen = max_len)

model = Sequential()
model.add(Embedding(vocab_size, embedding_data, input_length=max_len))
model.add(GlobalAveragePooling1D())
model.add(Dense(16, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
epochs = 500
history = model.fit(padded_sequences, np.array(training_labels), epochs=epochs)
```

```
# to save the trained model
model.save('chat_model')
import pickle

# to save the fitted tokenizer
with open('tokenizer.pickle','wb') as handle:
 pickle.dump(tokenizer,handle,protocol = pickle.HIGHEST_PROTOCOL)

# to save the fitted label encoder
with open('label_encoder.pickle','wb') as ecn_file:
 pickle.dump(lbl_encoder,ecn_file,protocol = pickle.HIGHEST_PROTOCOL)

# Import necessary libraries
pip install colorama
import json
import colorama
colorama.init()
from colorama import Fore, Style, Back
import random
import pickle

# Load intents from 'intents.json'
with open('content.json') as file:
    data = json.load(file)

# Define a function to handle user input and generate responses
def chat():
    print(Fore.YELLOW + "Chatbot: Hi there! How can I assist you today?" +
Style.RESET_ALL)
    while True:
        user_input = input(Fore.BLUE + "You: " + Style.RESET_ALL)
        response = generate_response(user_input)
        print(Fore.YELLOW + "Chatbot: " + response + Style.RESET_ALL)
        if user_input.lower() == "bye":
            print(Fore.YELLOW + "Chatbot: Goodbye! Have a great day!" + Style.RESET_ALL)
            break

# Define the generate_response function here
def generate_response(user_input):
    user_input = user_input.lower()
    response = "I'm sorry, I don't understand. Please try asking a different question."

    for intent in data['intents']:
        for pattern in intent['patterns']:
            if user_input in pattern.lower():
                response = random.choice(intent['responses'])
                break
    return response

# Start the chat when the script is run
if __name__ == "__main__":
 chat()
```

Output:



```
Chatbot: Hi there! How can I assist you today?
You: hello
hello
Chatbot: Hi there, how can I help?
You: bye
bye
Chatbot: Bye! Come back again soon.
Chatbot: Goodbye! Have a great day!
```

Result:

Thus, a simple chatbot was built using deep learning techniques and executed successfully.