AD18611 COMPUTER VISION AND NATURAL LANGUAGE PROCESSING LAB

EX NO: 1a **2D TO 3D TRANSFORMATION IN POINTS**

DATE:

AIM:

To implement 2D transformation in points using python.

ALGORITHM:

- Start
- Import all necessary libraries.
- Represent the point in 2D space using coordinates (x, y).
- Choose the appropriate transformation matrix based on the type of transformation required translation, rotation, scaling.
- Convert the 2D point to homogeneous coordinates by adding a third coordinate.
- Multiply the transformation matrix with the homogeneous coordinate to apply the transformation.
- Print the transformed points.
- Stop.

PROGRAM:

```
import math
point=[100,100]

#translation
translation=[0,0]
rotation_angle=0
scaling_factor=1

point[0]+= translation[0]
point[0]+= translation[0]

x= point[0]
y=point[1]

#rotation
new_x=x*math.cos(rotation_angle)-y*math.sin(rotation_angle)
new_y=x*math.sin(rotation_angle)-y*math.cos(rotation_angle)

x=new_x
y=new_y

x*=scaling_factor
y*=scaling_factor
print("transformed points",(x,y))
```

OUTPUT:

```
transformed points (100.0, -100.0)
```

RESULT:

Therefore the 2D points are transformed into 3D points successfully.

AD18611 COMPUTER VISION AND NATURAL LANGUAGE PROCESSING LAB

EX NO: 1b **2D TO 3D TRANSFORMATION IN IMAGES**

DATE:

AIM:

To implement 2D to 3D transformation in images using python.

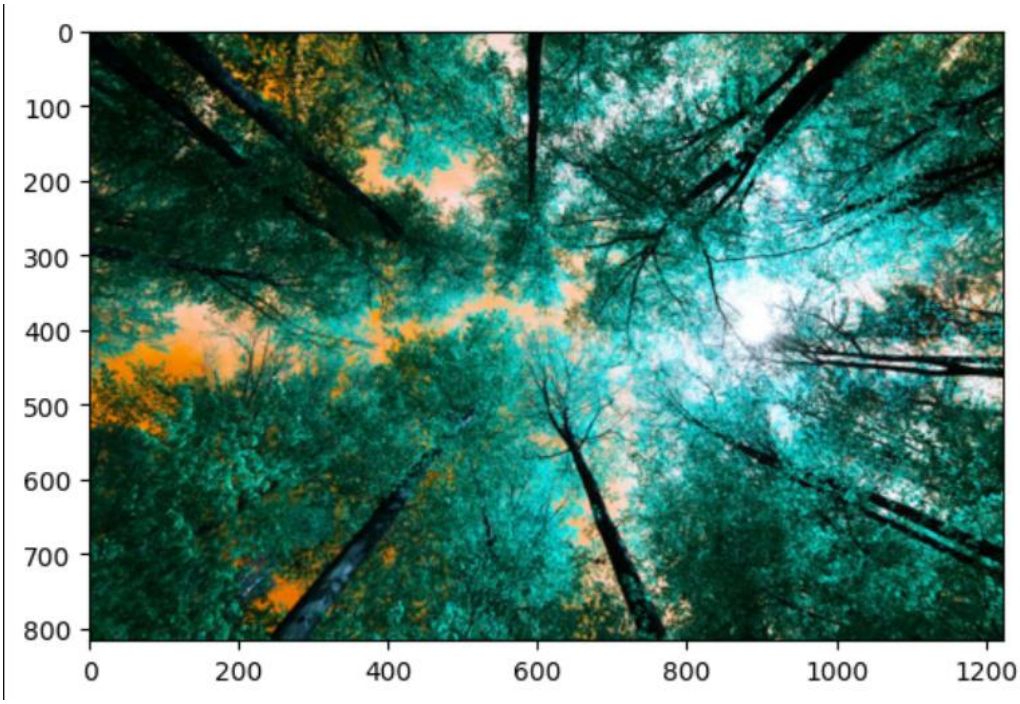
ALGORITHM:

- Start
- Import all necessary libraries
- Read the 2D image using OpenCV's cv2.imread() function.
- Convert the 2D image coordinates to homogeneous coordinates by adding an extra dimension
- Multiply each point in the image by the transformation matrix to apply the 2D to 3D transformation.
- Display or save the transformed image using OpenCV's cv2.imshow()
- Stop.

PROGRAM:

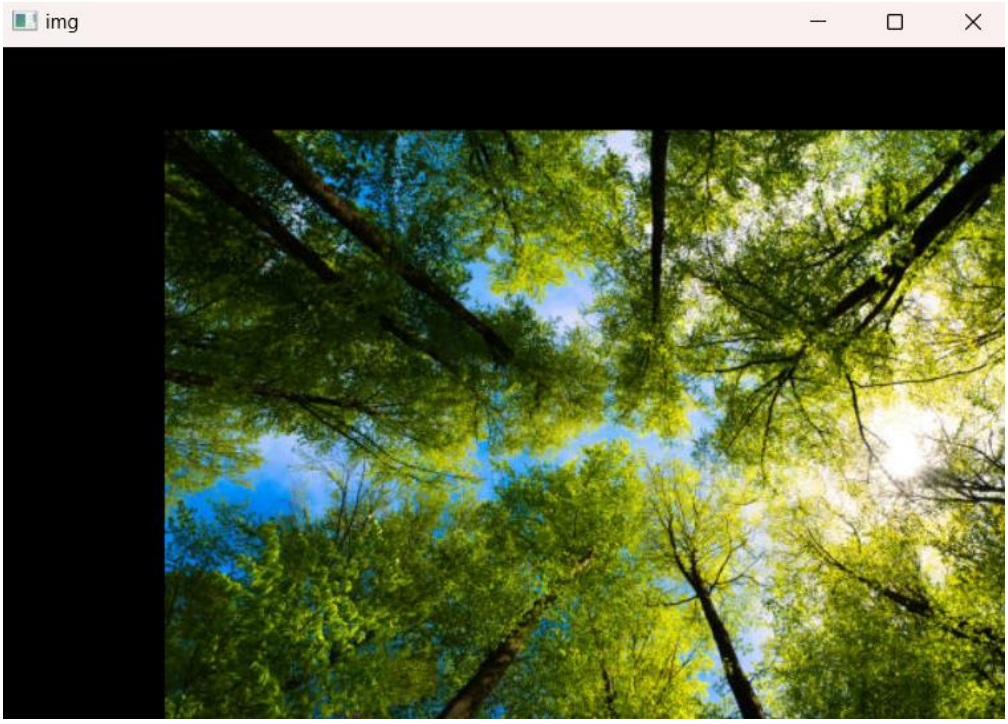
```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
img = cv.imread(r'C:\Users\sgoku\Downloads\abc.jpg')

# Assuming your image is loaded into the variable 'image'
res = cv.resize(img, None, fx=2, fy=2, interpolation=cv.INTER_AREA)
height,width = img.shape[:2]
res1 = cv.resize(img,(2*width,2*height), interpolation = cv.INTER_CUBIC)
plt.imshow(res1)
img = cv.imread(r'C:\Users\sgoku\Downloads\abc.jpg')
rows = img.shape[0]
cols = img.shape[1]
M = np.float32( [[1,0,100],[0,1,50]])
dst = cv.warpAffine(img,M,(cols,rows))
cv.imshow("img",dst)
cv.waitKey(0)
cv.destroyAllWindows()
M = cv.getRotationMatrix2D((cols/2.0, rows /2.0), 90,0.5)
dst = cv.warpAffine(img,M,(cols,rows))
cv.imshow("img",dst)
cv.waitKey(0)
cv.destroyAllWindows()
```

INPUT:

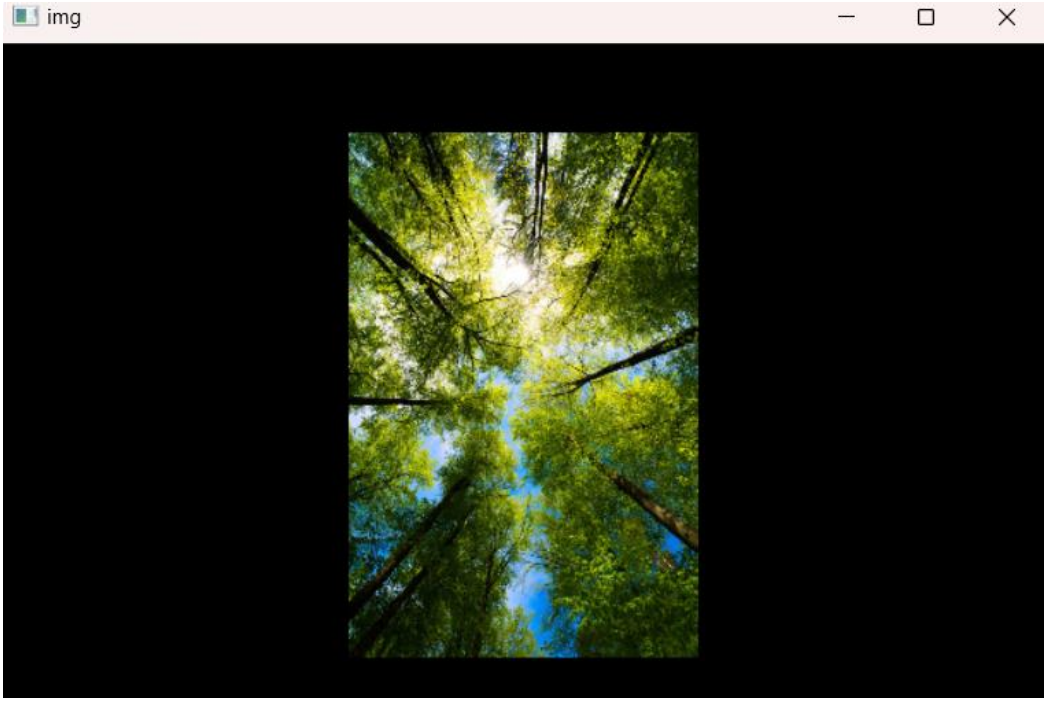


OUTPUT:

2D IMAGE:



3D IMAGE:



RESULT:

Therefore 2D to 3D transformation in images using python is implemented successfully.

AD18611 COMPUTER VISION AND NATURAL LANGUAGE PROCESSING LAB

EX NO: 2a **ACTIVE CONTOUR SEGMENTATION**

DATE:

AIM:

To implement active contour segmentation using python.

ALGORITHM:

- Start
- Import all necessary libraries
- Convert the astronaut image to grayscale and apply Gaussian noise reduction.
- Define a circular initial contour around a specified center point.
- Use the active contour algorithm to segment the preprocessed image based on the initial contour.
- Plot the original image with the initial and segmented contours.
- Define a function to convert text to uppercase and demonstrate its functionality.
- Stop.

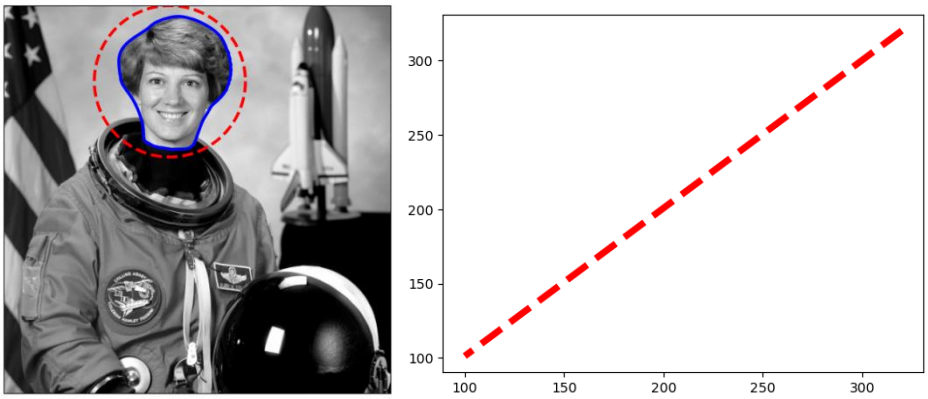
PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data
from skimage.color import rgb2gray
from skimage.filters import gaussian
from skimage.segmentation import active_contour
astronaut=data.astronaut()
grayAstronaut=rgb2gray(astronaut)
grayAstronautNoiseless=gaussian(grayAstronaut,1)

x1=220+100*np.cos(np.linspace(0,2*np.pi,500))
x2=100+100*np.sin(np.linspace(0,2*np.pi,500))

snake=np.array([x1,x2])

astronautSnake=active_contour(grayAstronautNoiseless,snake)

fig=plt.figure(figsize=(10,10))
ax=fig.add_subplot(111)
plt.imshow(grayAstronautNoiseless)
ax.plot(astronautSnake[:,0],astronautSnake[:,1],"--b",lw=5)

plt.plot(snake[:,0],snake[:,1],"--r",lw=5)

pass
import numpy as np
import matplotlib.pyplot as plt
from skimage.color import rgb2gray
from skimage import data
from skimage.filters import gaussian
from skimage.segmentation import active_contour
img = data.astronaut()
img = rgb2gray(img)
s = np.linspace(0, 2*np.pi, 400)
```

```
r = 100 + 100*np.sin(s)
c = 220 + 100*np.cos(s)
init = np.array([r, c]).T

snake = active_contour(gaussian(img, 3, preserve_range=False),
                       init, alpha=0.015, beta=10, gamma=0.001)

fig, ax = plt.subplots(figsize=(7, 7))
ax.imshow(img, cmap=plt.cm.gray)
ax.plot(init[:, 1], init[:, 0], '--r', lw=3)
ax.plot(snake[:, 1], snake[:, 0], '-b', lw=3)
ax.set_xticks([]), ax.set_yticks([])
ax.axis([0, img.shape[1], img.shape[0], 0])
plt.show()
def shout(text):
        return text.upper()
print (shout('Hello'))
yell = shout
print (yell)
```

OUTPUT:

```
[<matplotlib.lines.Line2D at 0x7f771fc51160>]
```




```
HELLO
<function shout at 0x7f771f9ff1f0>
```

RESULT:

The above transformation of image is done successfully

EX NO: 2b **GRABCUT ALGORITHM**

DATE:

AIM:

To implement Grabcut algorithm using python.

ALGORITHM:

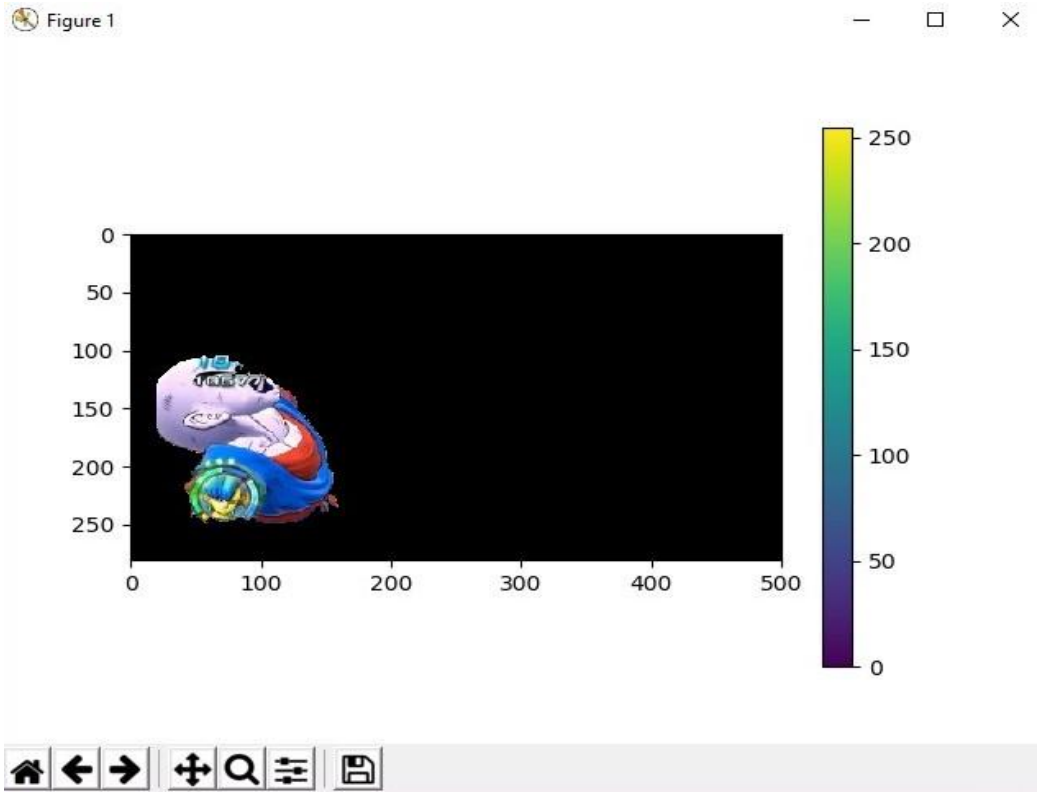
- Start
- Import all necessary libraries.
- Load the image and initialize a mask for segmentation.
- Prepare background and foreground models for GrabCut.
- Define a rectangular region of interest (ROI) around the object.
- Apply GrabCut algorithm to segment the image based on the defined ROI.
- Generate the segmented image and display it alongside the original.
- Stop

PROGRAM:

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

image = cv2.imread("/home/user/Desktop/snake.jpg")
mask = np.zeros(image.shape[:2], np.uint8)

backgroundModel = np.zeros((1, 65), np.float64)
foregroundModel = np.zeros((1, 65), np.float64)

rectangle = (500,950,3000,950)

cv2.grabCut(image, mask, rectangle, backgroundModel, foregroundModel,3, cv2.GC_INIT_WITH_RECT)
mask2 = np.where((mask == 2)|(mask == 0), 0, 1).astype('uint8')
image_segmented = image * mask2[:, :, np.newaxis]

# output segmented image with colorbar
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis('off')

# Display the segmented image
plt.subplot(1, 2, 2)
plt.title('Segmented Image')
plt.imshow(cv2.cvtColor(image_segmented, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

INPUT:

OUTPUT:



RESULT:

Therefore GrabCut algorithm is implemented successfully using python.

EX NO: 3 **PATTERN RECOGNITION**

DATE:

AIM:

To implement pattern recognition for images using python.

ALGORITHM:

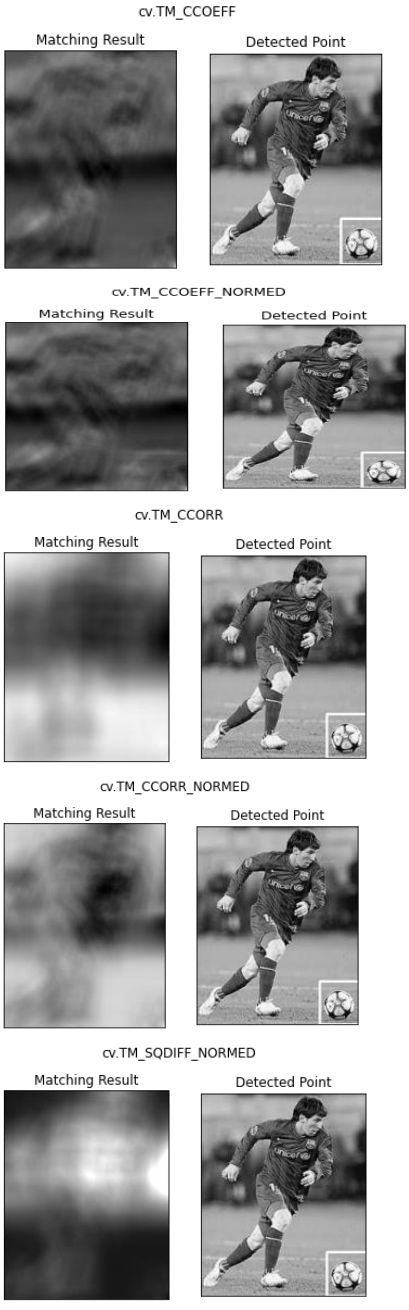
- Start
- Import all necessary libraries.
- Load the target and template images, and define template matching methods.
- Iterate through the template matching methods and apply them to find matches.
- Determine the location of the best match in the target image.
- Draw a rectangle around the detected region of interest (ROI) in the target image.
- Display the result of template matching and the original image with the detected ROI.
- Stop

PROGRAM:

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
img = cv.imread("/content/man.jpeg", cv.IMREAD_GRAYSCALE)
plt.imshow(img)

img2 = img.copy()
template = cv.imread("/content/man2.jpeg", cv.IMREAD_GRAYSCALE)
plt.imshow(template)
w, h = template.shape[::1]
methods = ["cv.TM_CCOEFF", "cv.TM_CCOEFF_NORMED", "cv.TM_CCORR", "cv.TM_CCORR_NORMED",
"cv.TM_SQDIFF", "cv.TM_SQDIFF_NORMED"]
for meth in methods:
   img = img2.copy()
   method = eval(meth)

res = cv.matchTemplate(img, template, method)
min_val, max_val, min_loc, max_loc = cv.minMaxLoc(res)

if method in [cv.TM_SQDIFF, cv.TM_SQDIFF_NORMED]:
   top_left = min_loc
else:
   top_left = max_loc
   bottom_right = (left[0] + w,left[1] + h)
   cv.rectangle(img, top_left, bottom_right, 255, 2)

plt.subplot(121), plt.imshow(res, cmap = "gray")
plt.subplot(122), plt.imshow(img, cmap = "gray")
plt.show()
```

INPUT:

OUTPUT:



RESULT:

Therefore the pattern recognition is done in image using python.

AD18611 COMPUTER VISION AND NATURAL LANGUAGE PROCESSING LAB

EX NO: 4 **MOTION DETECTION AND TRACKING**

DATE:

AIM:

To implement motion detection and tracking in videoframes using python.

ALGORITHM:

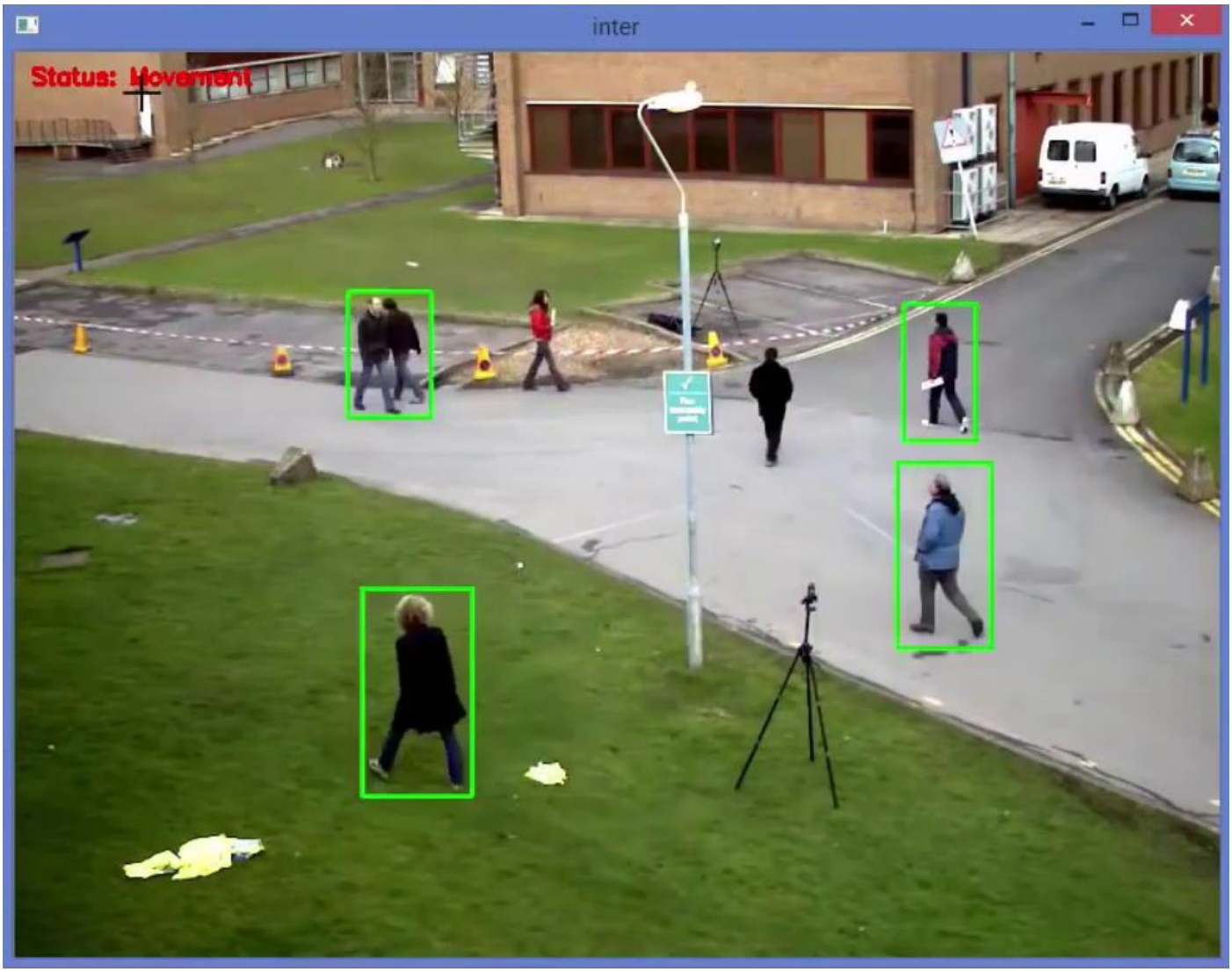
- Import all necessary libraries.
- Open the video file and define parameters like frame width, height, and codec for writing output.
- Iterate through the video frames, detecting motion by computing frame differences, thresholding, and contour detection.
- Identify significant motion by filtering contours based on area, draw rectangles around detected motion, and display "Movement" status.
- Resize annotated frames and write them to an output video. Display frames with motion detection feedback.
- Update frame variables, exit loop if the 'Esc' key is pressed, release resources, and close windows.

PROGRAM:

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow

cap = cv2.VideoCapture("/content/input(1).mp4")
fwidth = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
fheight = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fcc = cv2.VideoWriter_fourcc("X", "V", "I", "D")
out = cv2.VideoWriter("output.avi", fcc, 5.0, (1280, 720))
ret, frame1 = cap.read()
ret, frame2 = cap.read()

while cap.isOpened():
   diff = cv2.absdiff(frame1, frame2)
   gray = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)
   blur = cv2.GaussianBlur(gray, (5,5), 0)
   _, thresh = cv2.threshold(blur, 20, 255, cv2.THRESH_BINARY)
   dilated = cv2.dilate(thresh, None, iterations = 3)
   contours,_ = cv2.findContours(dilated, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

   for contour in contours:
      (x, y, w, h) = cv2.boundingRect(contour)
      if cv2.contourArea(contour) <900:
         continue
      cv2.rectangle(frame1, (x, y), (x+w, y+h), (0,255, 0), 2)
      cv2.putText(frame1, "Status:{}".format("Movement"), (10, 20), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0,
255), 3)

      image = cv2.resize(frame1, (1280, 720))
      out.write(image)
      cv2.imshow("feed", frame1)
      frame1 = frame2
```

```
        ret, frame2 = cap.read()
        if cv2.waitKey(40) == 27:
            break

cv2.destroyAllWindows()
cap.release()
out.release()
```

OUTPUT:



RESULT:

Therefore the implementation of motion detection and tracking in videoframes is done successfully.

AD18611 COMPUTER VISION AND NATURAL LANGUAGE PROCESSING LAB

EX NO: 5(a) **EDGE DETECTION - CANNY , ROBERT**

DATE:

AIM:

To perform edge detection for an image using the algorithms canny and robert.

ALGORITHM:

- Start
- Import all necessary libraries.
- Load images and convert them to grayscale for edge detection.
- Apply Canny edge detection to the grayscale image and visualize the result.
- Convert the tiger image to grayscale and apply Gaussian blur.
- Use Robert operator to detect edges in X and Y directions, and visualize the results.
- Display the edge-detected images using OpenCV's imshow function.
- Stop

PROGRAM:

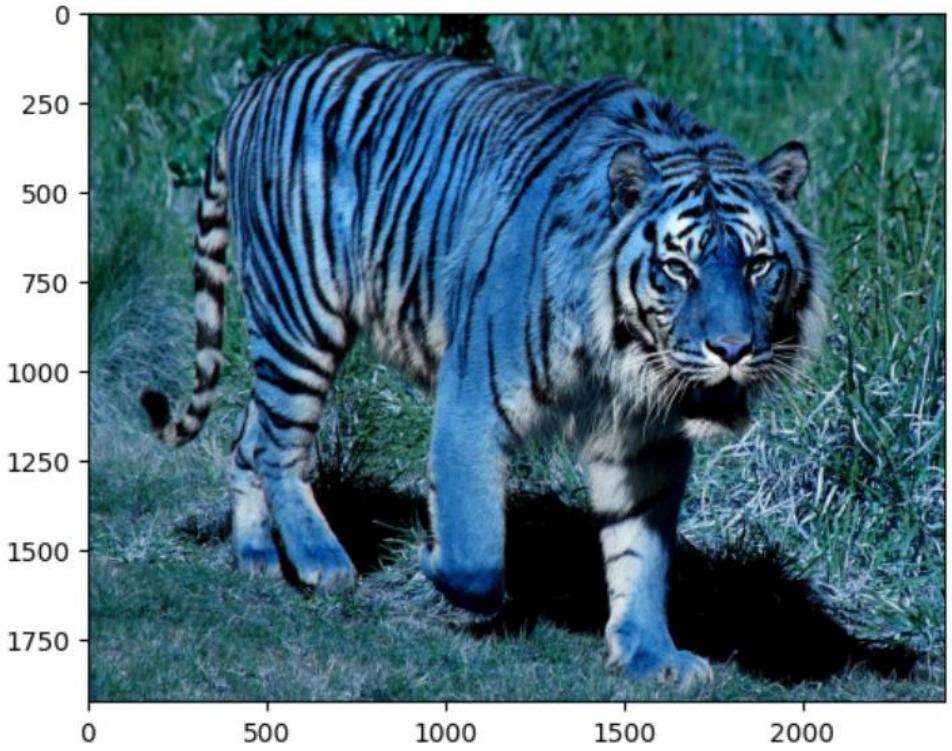
#Robert

```
import cv2
import numpy as np
from scipy import ndimage
import matplotlib.pyplot as plt

roberts_cross_v = np.array( [[1, 0 ], [0,-1 ]] )
roberts_cross_h = np.array( [[ 0, 1 ], [ -1, 0 ]] )

img = cv2.imread(r"C:\Users\sgoku\Downloads\Tiger.jpeg",0).astype('float64')
img/=255.0
vertical = ndimage.convolve( img, roberts_cross_v )
horizontal = ndimage.convolve( img, roberts_cross_h )
edged_img = np.sqrt( np.square(horizontal) + np.square(vertical))
edged_img*=255
plt.imshow(edged_img)
```

#Canny

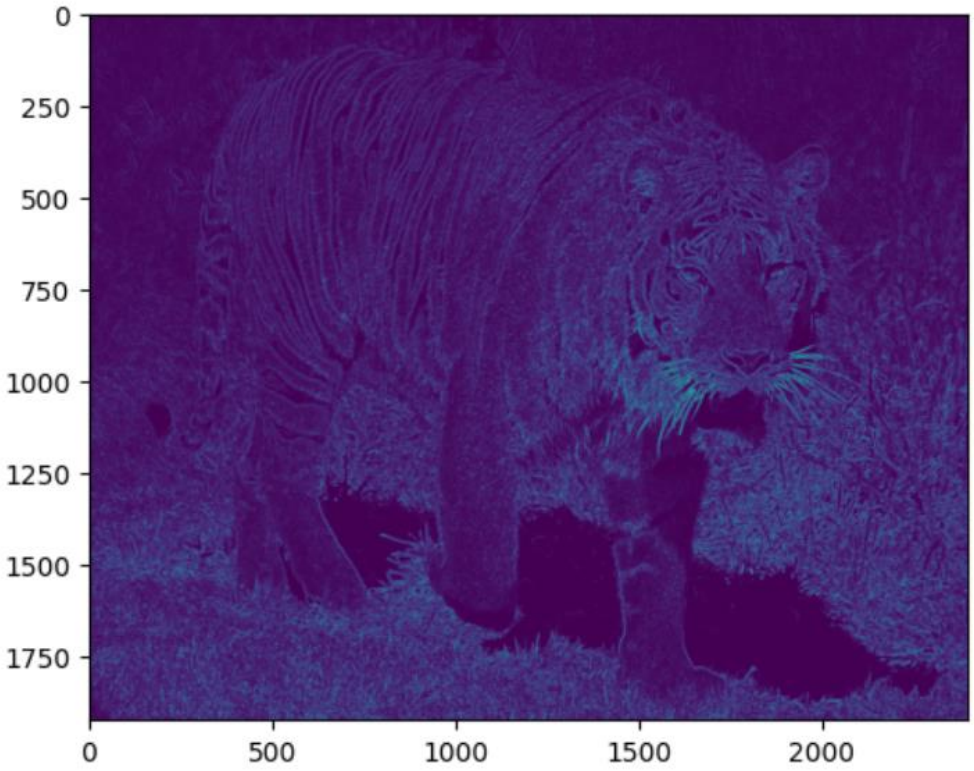
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image=cv2.imread(r"C:\Users\sgoku\Downloads\Tiger.jpeg")
plt.imshow(image)
edges=cv2.Canny(image,threshold1=30,threshold2=100)#blur or gray or image
plt.imshow(edges,cmap="gray")
plt.show()
```

Input:

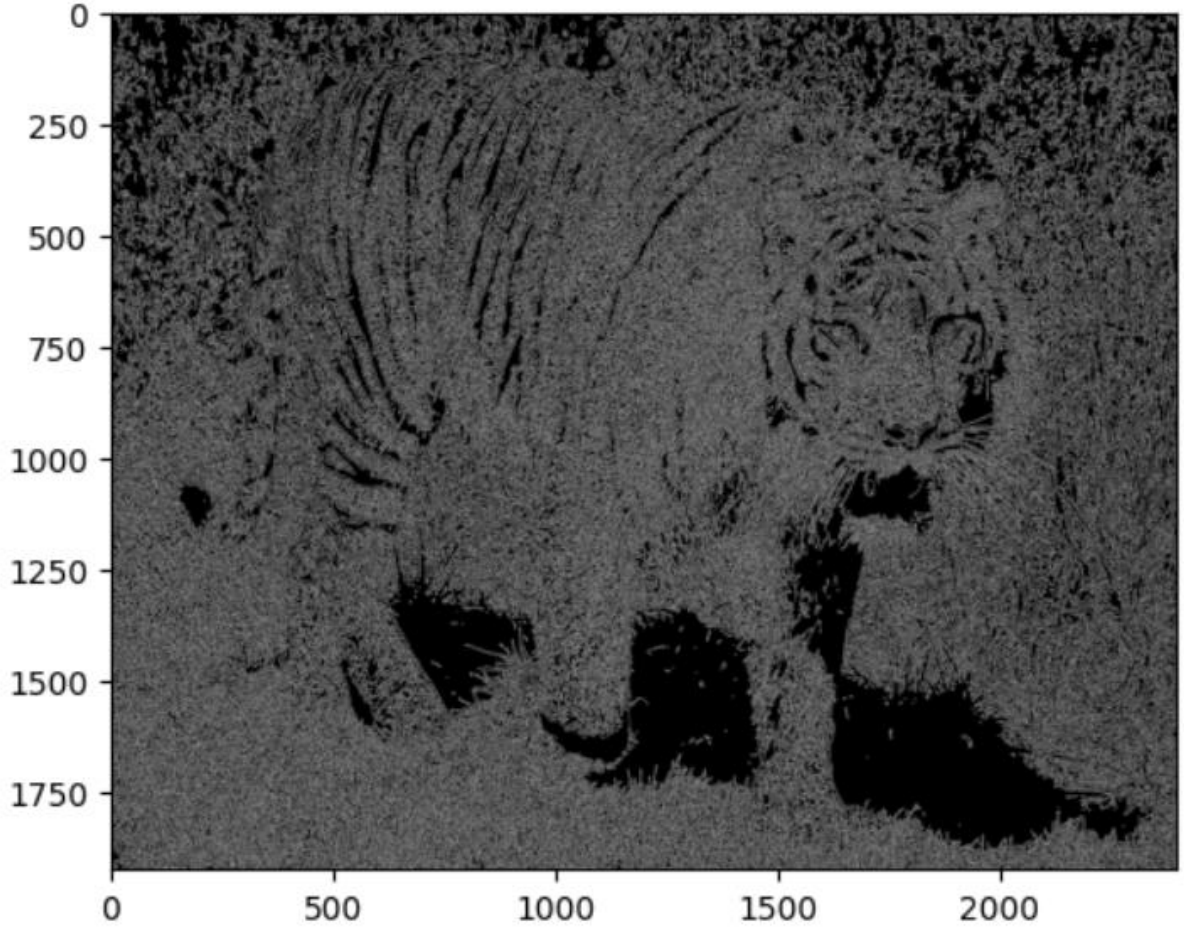


Output:

#Robert



#Canny



RESULT:

The image is loaded and edge detection is done using the algorithms successfully.

AD18611 COMPUTER VISION AND NATURAL LANGUAGE PROCESSING LAB

EX NO: 5(b) **EDGE DETECTION - ROBERT, SOBEL**

DATE:

AIM:

To perform edge detection for an image using the algorithms Robert and Sobel.

ALGORITHM:

- Start
- Import all necessary libraries.
- Load images and convert them to grayscale for edge detection.
- Apply Sobel edge detection to the grayscale image and visualize the result.
- Convert the tiger image to grayscale and apply Gaussian blur.
- Use Robert operator to detect edges in X and Y directions, and visualize the results.
- Display the edge-detected images using OpenCV's imshow function.
- Stop

PROGRAM:

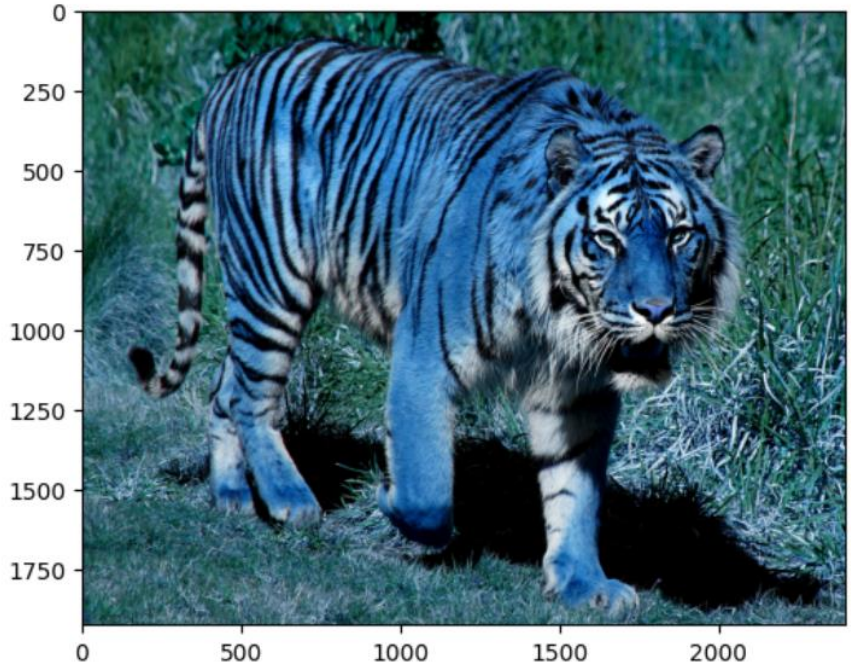
#Robert

```
import cv2
import numpy as np
from scipy import ndimage
import matplotlib.pyplot as plt

roberts_cross_v = np.array( [[1, 0 ], [0,-1 ]] )
roberts_cross_h = np.array( [[ 0, 1 ], [ -1, 0 ]] )

img = cv2.imread(r"C:\Users\sgoku\Downloads\Tiger.jpeg",0).astype('float64')
img/=255.0
vertical = ndimage.convolve( img, roberts_cross_v )
horizontal = ndimage.convolve( img, roberts_cross_h )
edged_img = np.sqrt( np.square(horizontal) + np.square(vertical))
edged_img*=255
plt.imshow(edged_img)
```

#Sobel

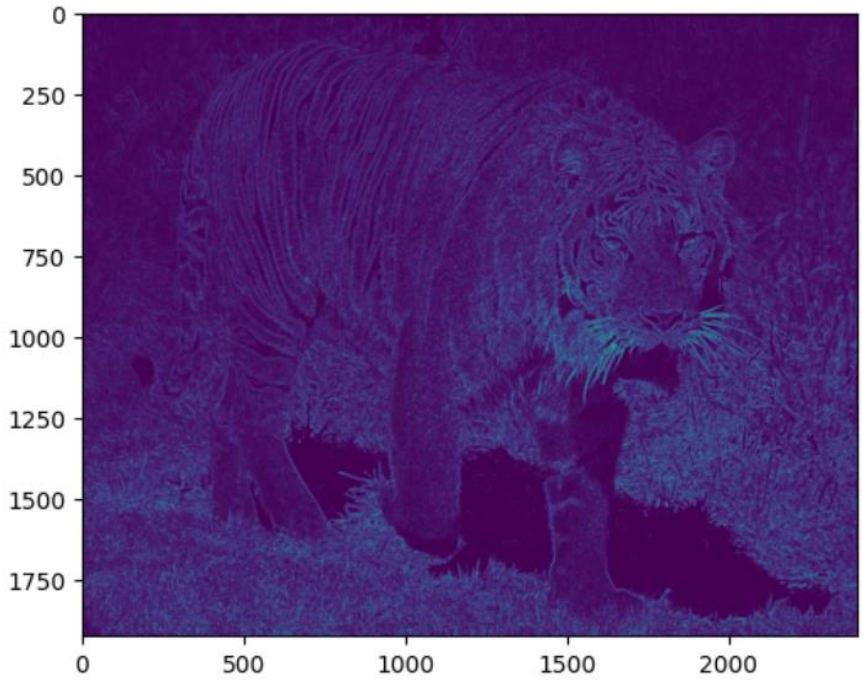
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image=cv2.imread(r"C:\Users\sgoku\Downloads\Tiger.jpeg")
plt.imshow(image)
sobelx=cv2.Sobel(src=image,ddepth=cv2.CV_64F,dx=1,dy=0,ksize=5)
sobely=cv2.Sobel(src=image,ddepth=cv2.CV_64F,dx=0,dy=1,ksize=5)
sobelxy=cv2.Sobel(src=image,ddepth=cv2.CV_64F,dx=1,dy=1,ksize=5)
plt.imshow(sobelx)
plt.imshow(sobely)
plt.imshow(sobelxy)
```

Input:

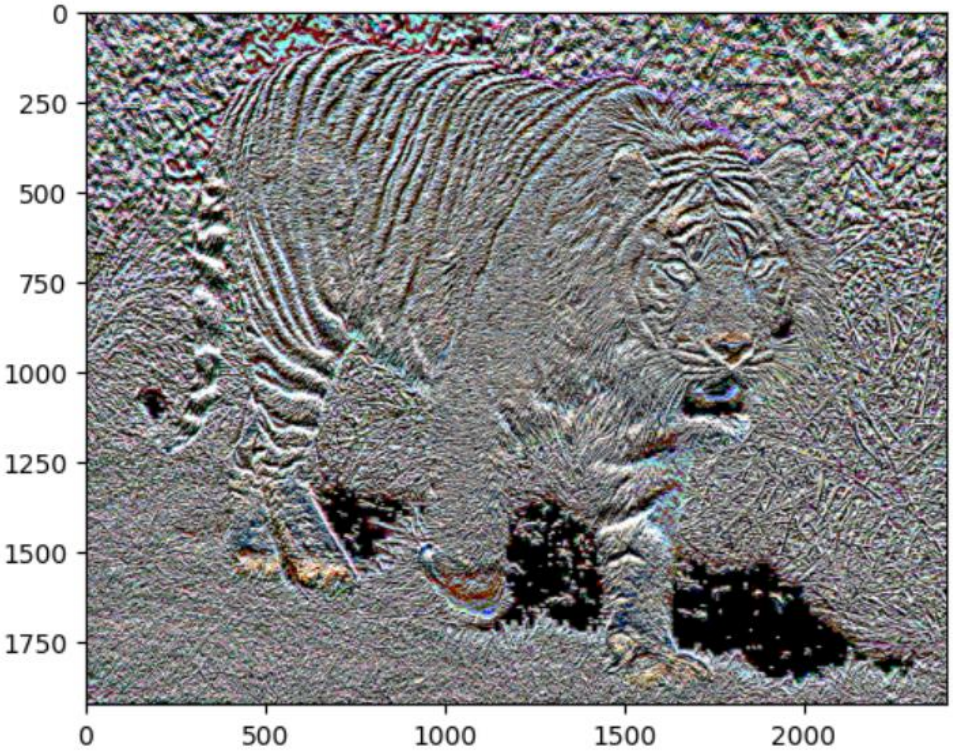


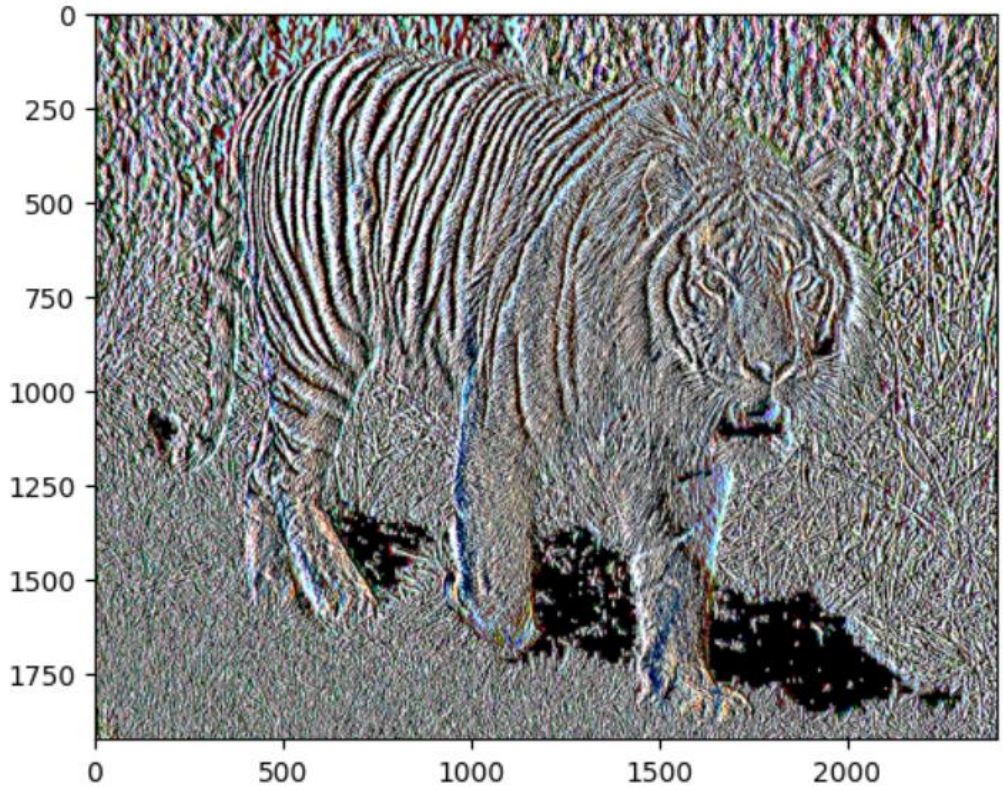
Output:

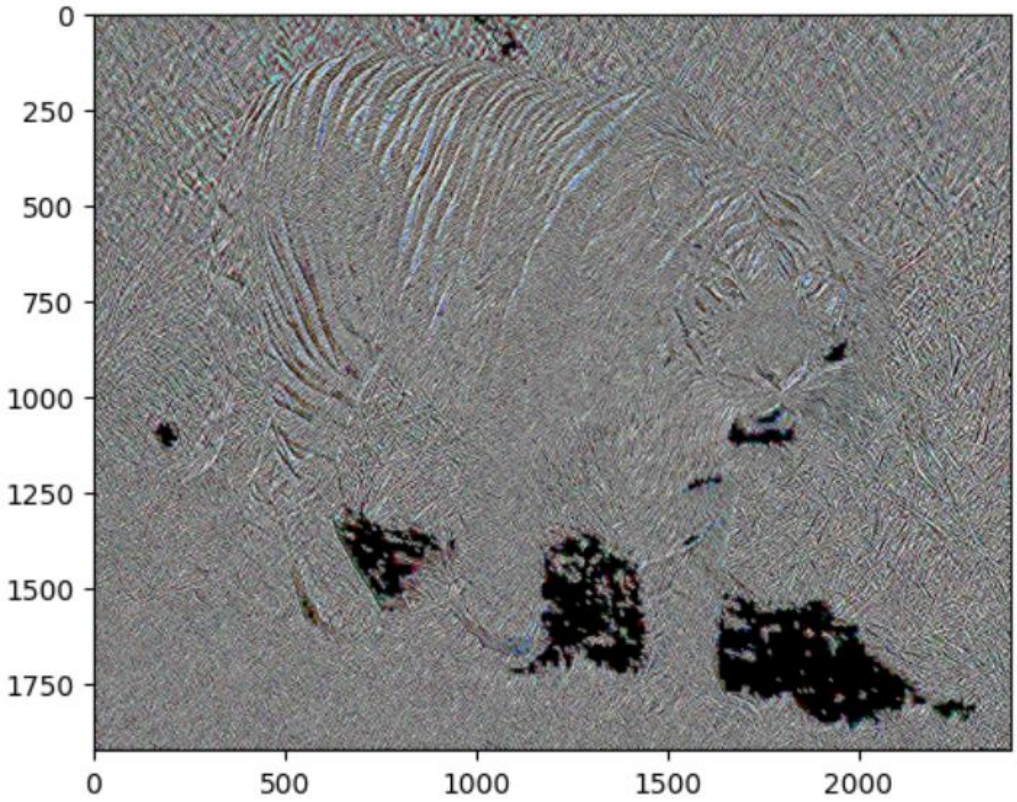
#Robert



#Sobel


0
250
500
750
1000
1250
1500
1750
0
500
1000
1500
2000


0
250
500
750
1000
1250
1500
1750
0
500
1000
1500
2000



RESULT:

The image is loaded and edge detection is done using the algorithms successfully.

AD18611 COMPUTER VISION AND NATURAL LANGUAGE PROCESSING LAB

EX NO: 5(c) **EDGE DETECTION - CANNY, SOBEL**

DATE:

AIM:

To perform edge detection for an image using the algorithms Robert and Sobel.

ALGORITHM:

- Start
- Import all necessary libraries.
- Load images and convert them to grayscale for edge detection.
- Apply Canny edge detection to the grayscale image and visualize the result.
- Convert the tiger image to grayscale and apply Gaussian blur.
- Use Sobel operator to detect edges in X and Y directions, and visualize the results.
- Display the edge-detected images using OpenCV's imshow function.
- Stop

PROGRAM:

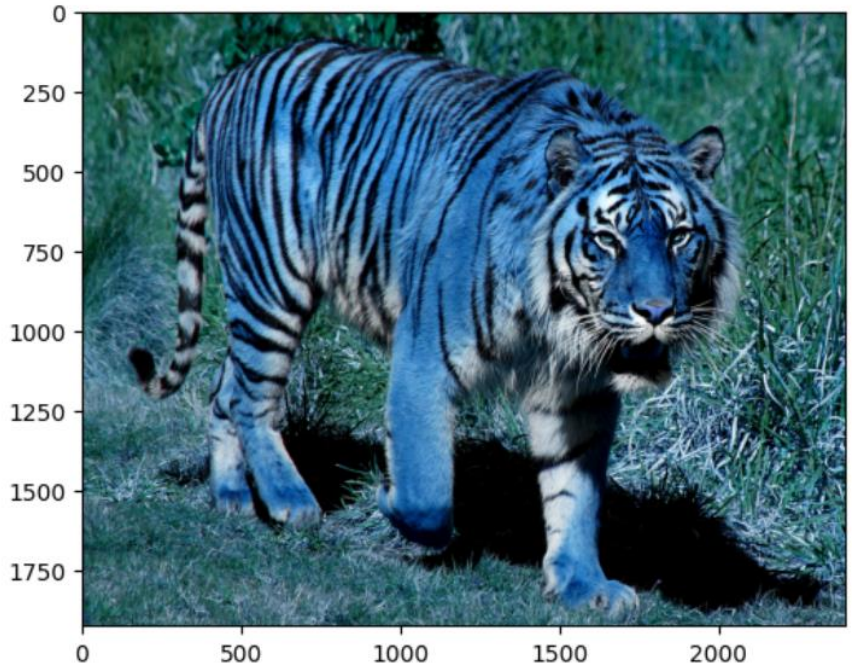
#Canny

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image=cv2.imread(r"C:\Users\sgoku\Downloads\Tiger.jpeg")
plt.imshow(image)
edges=cv2.Canny(image,threshold1=30,threshold2=100)#blur or gray or image
plt.imshow(edges,cmap="gray")
plt.show()
```

#Sobel

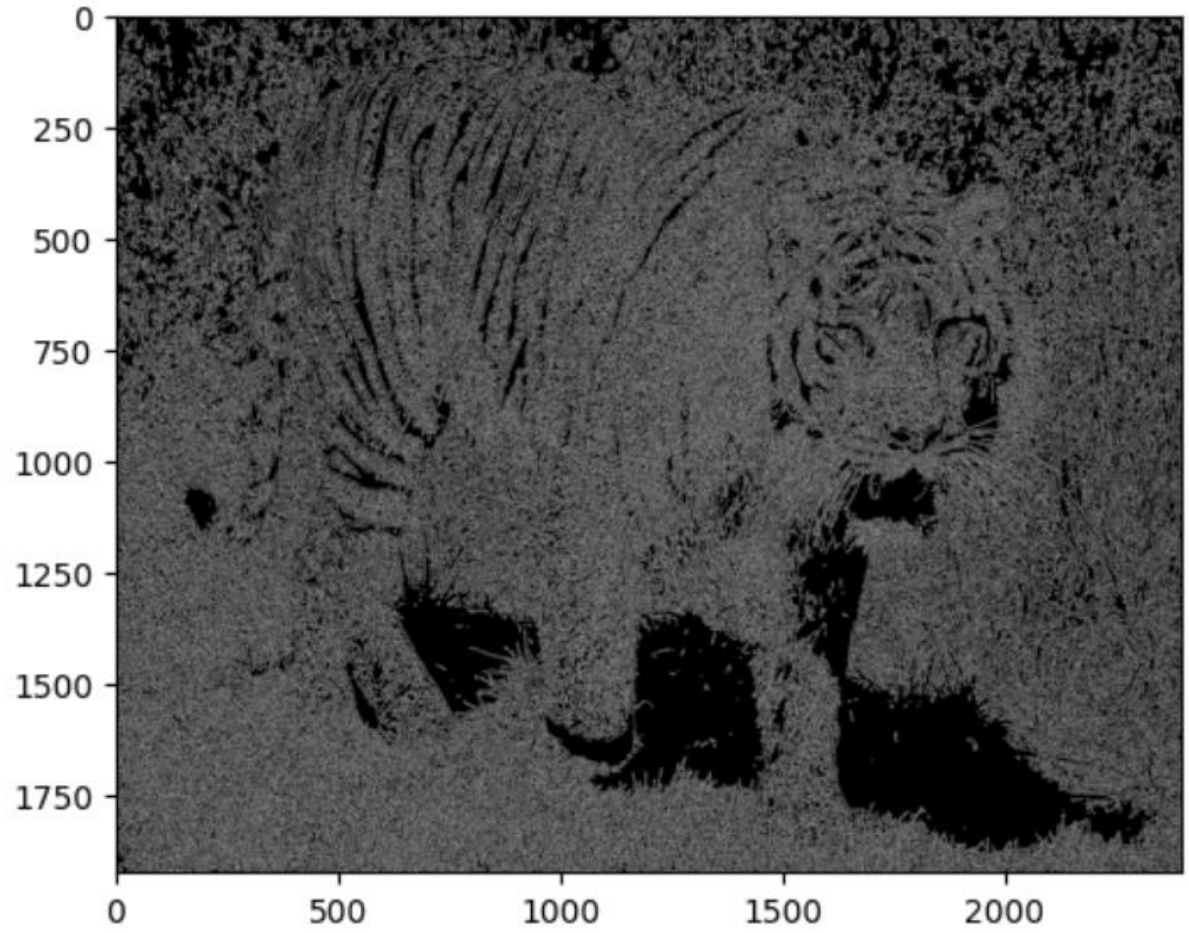
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image=cv2.imread(r"C:\Users\sgoku\Downloads\Tiger.jpeg")
plt.imshow(image)
sobelx=cv2.Sobel(src=image,ddepth=cv2.CV_64F,dx=1,dy=0,ksize=5)
sobely=cv2.Sobel(src=image,ddepth=cv2.CV_64F,dx=0,dy=1,ksize=5)
sobelxy=cv2.Sobel(src=image,ddepth=cv2.CV_64F,dx=1,dy=1,ksize=5)
plt.imshow(sobelx)
plt.imshow(sobely)
plt.imshow(sobelxy)
```

Input:

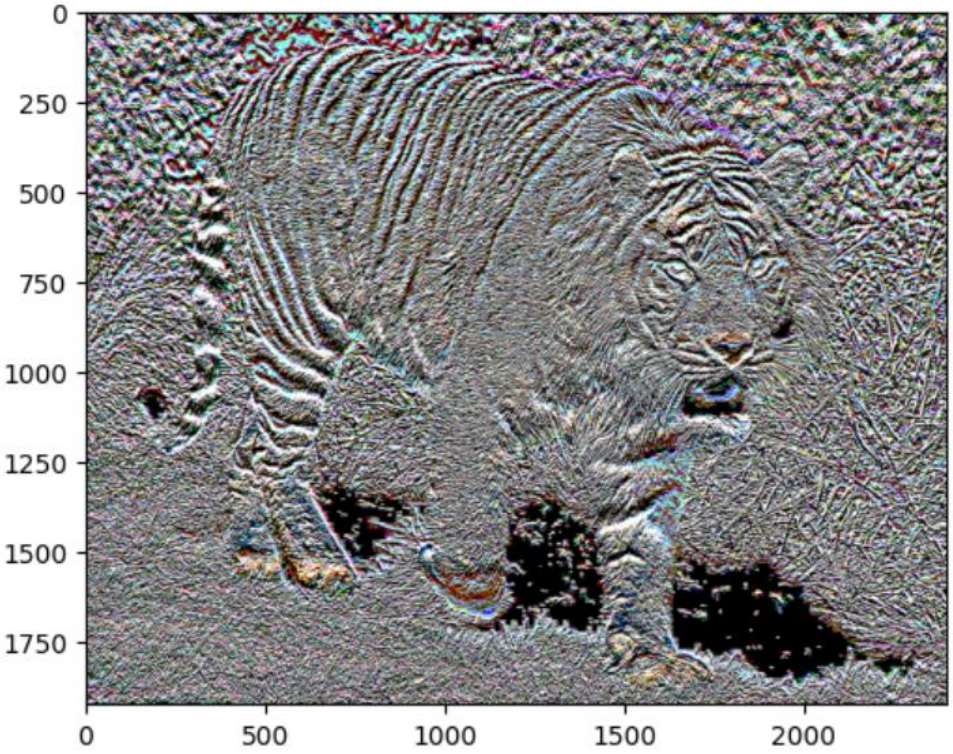


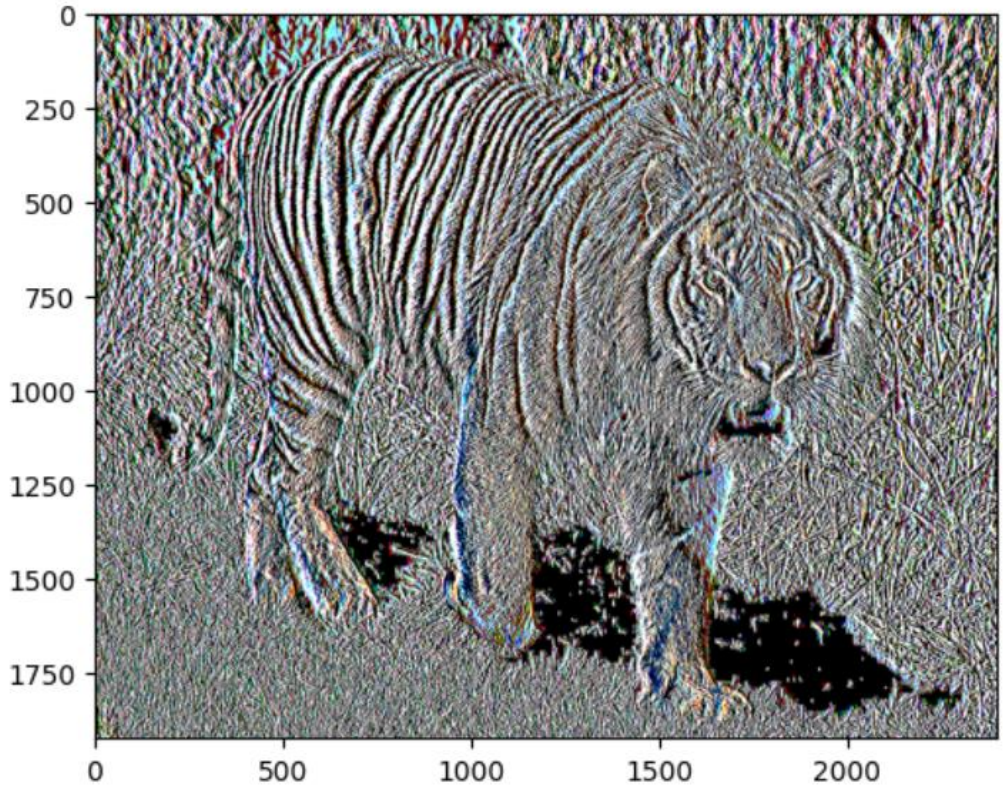
Output:

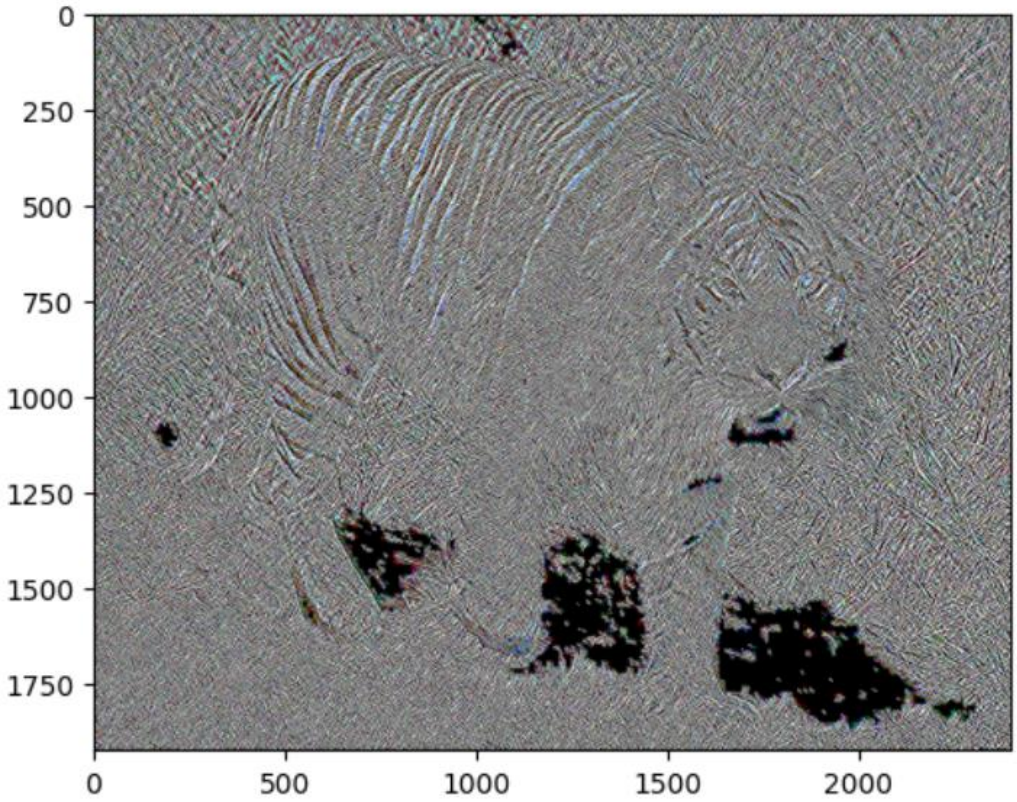
#Canny



#Sobel


0
250
500
750
1000
1250
1500
1750
0
500
1000
1500
2000


0
250
500
750
1000
1250
1500
1750
0
500
1000
1500
2000



RESULT:

The image is loaded and edge detection is done using the algorithms successfully

EX NO: 6(a) **TEXT PREPROCESSING WITH PACKAGES**

DATE:

AIM:

To implement text preprocessing techniques with packages using python.

ALGORITHM:

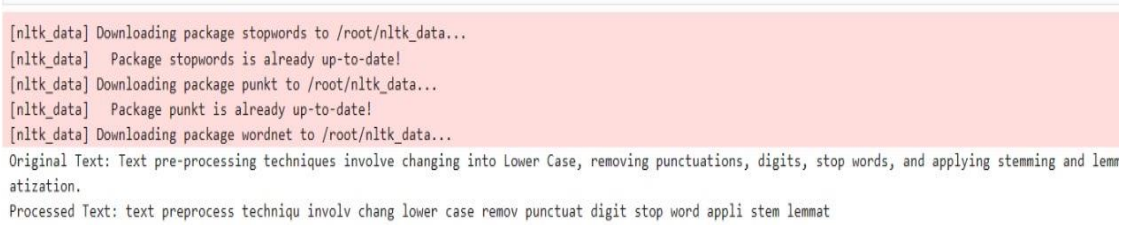
- Start
- Import required packages.
- Define a pre-processing function which obtains string as input and perform various pre-processing steps including Stemming, Lemmatization and removal of stopwords, using nltk methods, then return the resulting string.
- Display an example text and its processed text.
- Stop

PROGRAM:

```
!pip install nltk
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.tokenize import word_tokenize

# Download required resources
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

def text_preprocessing(text):
    # Change into Lower Case
    text = text.lower()

    # Remove Punctuations
    text = re.sub(r'[^\w\s]', '', text)

    # Remove words and digits containing digits
    text = re.sub(r'\w*\d\w*', '', text)

    # Remove Stop words
    stop_words = set(stopwords.words('english'))
    words = word_tokenize(text)
    words = [word for word in words if word.lower() not in stop_words]

    # Text Stemming
    stemmer = PorterStemmer()
    words = [stemmer.stem(word) for word in words]

    # Text Lemmatization
    lemmatizer = WordNetLemmatizer()
    words = [lemmatizer.lemmatize(word) for word in words]

    # Join the processed words back into a string
```

```
    processed_text = ' '.join(words)

    return processed_text

# Example usage:
input_text = "Text pre-processing techniques involve changing into Lower Case, removing punctuations, digits, stop words, and applying stemming and lemmatization."
processed_text = text_preprocessing(input_text)
print("Original Text:", input_text)
print("Processed Text:", processed_text)
```

OUTPUT:



RESULT:

Therefore text preprocessing techniques using packages was implemented successfully.

EX NO: 6(b) **TEXT PREPROCESSING WITHOUT PACKAGES**

DATE:

AIM:

To implement text preprocessing techniques without packages using python.

ALGORITHM:

- Start
- Define a pre-processing function which obtains string as input and perform various pre-processing steps including Stemming, Lemmatization and removal of stopwords, using string manipulation techniques then return the resulting string.
- Display an example text and its processed text.
- Stop

PROGRAM:

```
import string

def simple_text_preprocessing(text):
    # Convert to lowercase
    text = text.lower()

    # Remove punctuation
    text = ''.join([char for char in text if char not in string.punctuation])

    # Remove digits
    text = ''.join([char for char in text if not char.isdigit()])

    # Define a simple list of stop words
    stop_words = set([
        'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your', 'yours', 'yourself', 'yourselves',
        'he', 'him', 'his', 'himself', 'she', 'her', 'hers', 'herself', 'it', 'its', 'itself', 'they', 'them', 'their',
        'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those', 'am', 'is', 'are', 'was',
        'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and',
        'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between',
        'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off',
        'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any',
        'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so',
        'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', 'should', 'now', 'd', 'll', 'm', 'o', 're', 've', 'y',
        'ain', 'aren', 'couldn', 'didn', 'doesn', 'hadn', 'hasn', 'haven', 'isn', 'ma', 'mightn', 'mustn', 'needn', 'shan',
        'shouldn', 'wasn', 'weren', 'won', 'wouldn'
    ])

    # Remove stop words
    words = text.split()
    words = [word for word in words if word not in stop_words]

    # Perform stemming (using a simple rule-based approach)
    words = [word[:-1] if word.endswith('s') else word for word in words]
```

```
    # Perform lemmatization (using a simple rule-based approach)
    words = [word[:-2] if word.endswith('es') else word for word in words]

    # Join the processed words back into a string
    processed_text = ' '.join(words)

    return processed_text

# Example usage:
input_text = "Text pre-processing techniques involve changing into Lower Case, removing punctuations, digits, stop words, and applying stemming and lemmatization."
processed_text = simple_text_preprocessing(input_text)
print("Original Text:", input_text)
print("Processed Text:", processed_text)
```

OUTPUT:

```
Original Text: Text pre-processing techniques involve changing into Lower Case, removing punctuations, digits, stop words, and applying stemming and lemm
atization.
Processed Text: text preprocessing technique involve changing lower case removing punctuation digit stop word applying stemming lemmatization
```

RESULT:

Therefore text preprocessing techniques without using packages was implemented successfully.

EX NO: 6(c) **TEXT PREPROCESSING USING WORDCLOUD**

DATE:

AIM:

To implement text preprocessing techniques using wordCloud in python.

ALGORITHM:

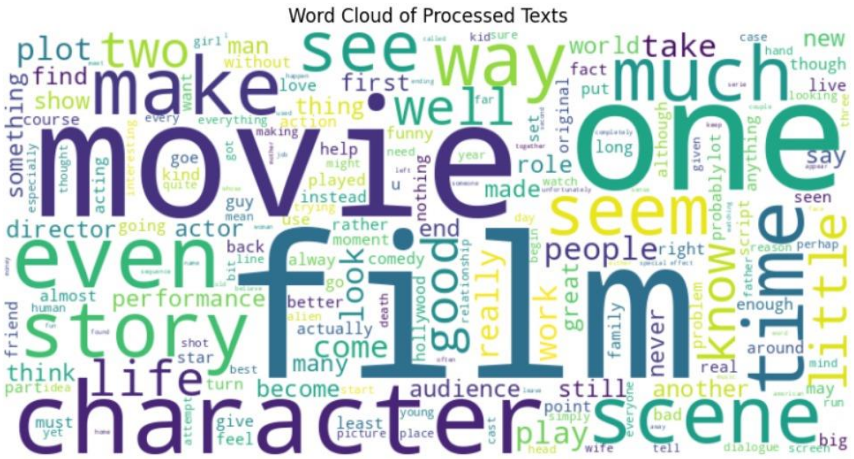
- Start
- Import required python pachages.
- Define a pre-processing function which obtains string as input and perform various pre-processing steps including Stemming, Lemmatization and removal of stopwords, using string manipulation techniques then return the resulting string.
- Load the IMDB dataset and perform pre-processing.
- Visualize the dataset using Word Cloud.
- Display sample positive and negative reviews and visualize using bar plot.
- Stop

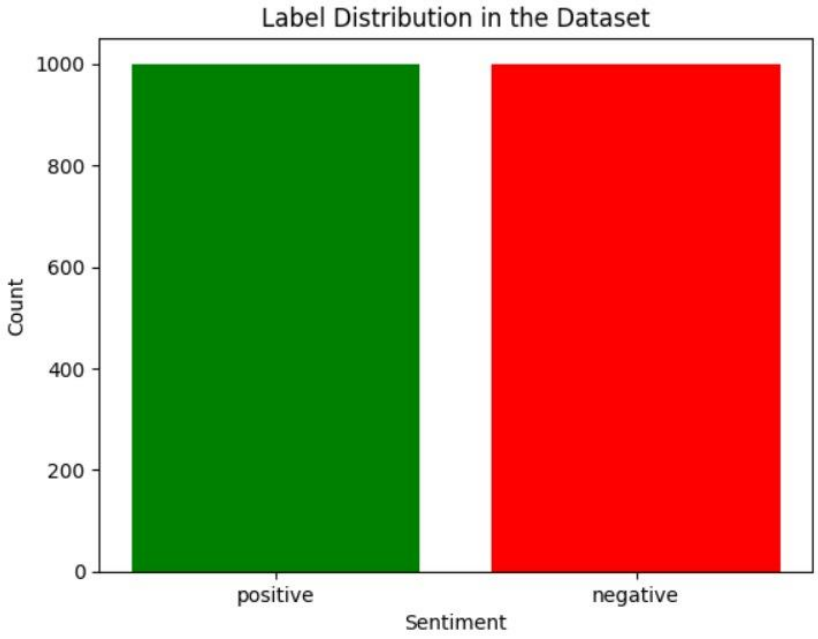
PROGRAM:

```
!pip install matplotlib wordcloud
import nltk
from nltk.corpus import movie_reviews
import random
import string
from wordcloud import WordCloud
import matplotlib.pyplot as plt

nltk.download('movie_reviews')

def simple_text_preprocessing(text):
    # Convert to lowercase
    text = text.lower()

    # Remove punctuation
    text = ''.join([char for char in text if char not in string.punctuation])

    # Remove digits
    text = ''.join([char for char in text if not char.isdigit()])

    # Define a simple list of stop words
    stop_words = set(nltk.corpus.stopwords.words('english'))

    # Remove stop words
    words = text.split()
    words = [word for word in words if word not in stop_words]

    # Perform stemming (using a simple rule-based approach)
    words = [word[:-1] if word.endswith('s') else word for word in words]

    # Perform lemmatization (using a simple rule-based approach)
    words = [word[:-2] if word.endswith('es') else word for word in words]
```

```
    # Join the processed words back into a string
    processed_text = ' '.join(words)

    return processed_text

# Load the IMDb movie reviews dataset
documents = [(list(movie_reviews.words(fileid)), category)
             for category in movie_reviews.categories()
             for fileid in movie_reviews.fileids(category)]

# Shuffle the documents
random.shuffle(documents)

# Extract the text and labels from the dataset
texts = [' '.join(words) for words, _ in documents]
labels = [category for _, category in documents]

# Apply text preprocessing to each text in the dataset
processed_texts = [simple_text_preprocessing(text) for text in texts]

# Visualize word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(' '.join(processed_texts))
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of Processed Texts')
plt.show()

# Visualize label distribution
label_counts = {'positive': labels.count('pos'), 'negative': labels.count('neg')}
plt.bar(label_counts.keys(), label_counts.values(), color=['green', 'red'])
plt.title('Label Distribution in the Dataset')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.show()

# Display original and processed texts for the first few examples
for i in range(5):
    print(f"Example {i + 1}:")
    print("Original Text:", texts[i][:100] + "...")
    print("Processed Text:", processed_texts[i][:100] + "...")
    print("Label:", labels[i])
    print("-" * 50)
```

OUTPUT:






```
Example 1:
Original Text: jean - luc picard ( patrick stewart ) and the rest of the crew of the u . s . s . enterprise are bac...
Processed Text: jean luc picard patrick stewart rest crew u enterprise back third outing ninth film based serie star...
Label: pos
--------------------------------------------------
Example 2:
Original Text: titanic is , without a doubt , the best movie i ' ve seen all year . and believe me , i ' m not an e...
Processed Text: titanic without doubt best movie seen year believe easy critic impr english patient long men black h...
Label: pos
--------------------------------------------------
Example 3:
Original Text: richard gere is not one of my favorite actors . however , i like courtroom dramas , and this film lo...
Processed Text: richard gere one favorite actor however like courtroom drama film looked good enough take chance dis...
Label: pos
--------------------------------------------------
Example 4:
Original Text: ingredients : down - on - his - luck evangelist , church synopsis : sonny dewey ( robert duvall ) is...
Processed Text: ingredient luck evangelist church synopsi sonny dewey robert duvall tirel texa pentecostal preacher ...
Label: pos
--------------------------------------------------
Example 5:
Original Text: truman ( " true - man " ) burbank is the perfect name for jim carrey ' s character in this film . pr...
Processed Text: truman true man burbank perfect name jim carrey character film president truman unassuming man becam...
Label: pos
```

RESULT:

Therefore text preprocessing techniques using WordCloud was implemented successfully.

AD18611 COMPUTER VISION AND NATURAL LANGUAGE PROCESSING LAB

EX NO: 7(a) WORD EMBEDDING TECHNIQUES USING WORDVECTOR

DATE:

AIM:

To implement word embedding techniques using wordvector in python.

ALGORITHM:

- Start
- Import required python packages.
- Obtain sample text as corpus and perform simple pre-processing steps like Tokenization.
- Train the model on the tokenized corpus.
- Display similar words and their probabilities for a sample word in the corpus.
- Stop

PROGRAM:

```
pip install gensim nltk
import nltk
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
nltk.download('punkt')

# Sample dataset
corpus = [
    "Word embeddings are a type of word representation.",
    "They capture semantic relationships between words.",
    "Word2Vec and GloVe are popular word embedding techniques.",
    "Natural Language Processing (NLP) involves the interaction between computers and humans using natural
language.",
]

# Tokenize the sentences
tokenized_corpus = [word_tokenize(sentence.lower()) for sentence in corpus]

# Train Word2Vec model
model = Word2Vec(sentences=tokenized_corpus, vector_size=100, window=5, min_count=1, workers=4)

# Save the model
model.save("word2vec_model")

# Access word vectors
word_vectors = model.wv

# Example: Getting the vector for a specific word
vector_for_word = word_vectors['word']
print("Vector for 'word':", vector_for_word)

# Example: Finding similar words
similar_words = word_vectors.most_similar('word', topn=3)
print("Words similar to 'word':", similar_words)
```

OUTPUT:

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
Vector for 'word': [-8.6214943e-03  3.6666086e-03  5.1863561e-03  5.7479362e-03
  7.4698664e-03 -6.1657466e-03  1.1110231e-03  6.0486598e-03
 -2.8446759e-03 -6.1718328e-03 -4.0855753e-04 -8.3704004e-03
 -5.5984668e-03  7.1084672e-03  3.3468201e-03  7.2257668e-03
  6.8099531e-03  7.5308117e-03 -3.7880018e-03 -5.7504204e-04
  2.3535413e-03 -4.5187781e-03  8.3976341e-03 -9.8576695e-03
  6.7629823e-03  2.9166730e-03 -4.9383035e-03  4.4003702e-03
 -1.7430126e-03  6.7122187e-03  9.9729933e-03 -4.3587615e-03
 -5.9502776e-04 -5.6976750e-03  3.8471792e-03  2.7887861e-03
  6.8890750e-03  6.1017354e-03  9.5340982e-03  9.2719821e-03
  7.9043768e-03 -6.9954009e-03 -9.1557913e-03 -3.5223819e-04
 -3.0947481e-03  7.8947041e-03  5.9328787e-03 -1.5522755e-03
  1.5146433e-03  1.7873152e-03  7.8192893e-03 -9.5101381e-03
 -2.0686894e-04  3.4677568e-03 -9.4532111e-04  8.3759837e-03
  9.0088267e-03  6.5301349e-03 -7.1530917e-04  7.7060936e-03
 -8.5331239e-03  3.1999508e-03 -4.6292576e-03 -5.0863591e-03
  3.5838669e-03  5.3740251e-03  7.7675935e-03 -5.7657524e-03
  7.4270051e-03  6.6268407e-03 -3.7140714e-03 -8.7413518e-03
  5.4362016e-03  6.5118251e-03 -7.8320602e-04 -6.7144330e-03
 -7.0811189e-03 -2.4906253e-03  5.1452550e-03 -3.6648987e-03
 -9.3742162e-03  3.8200899e-03  4.8822258e-03 -6.4248568e-03
  1.2036825e-03 -2.0810959e-03  3.0742885e-05 -9.8831896e-03
  2.6895360e-03 -4.7508357e-03  1.0896772e-03 -1.5745433e-03
  2.1983176e-03 -7.8827152e-03 -2.7143217e-03  2.6560172e-03
  5.3513078e-03 -2.3885074e-03 -9.5095290e-03  4.5117657e-03]
Words similar to 'word': [('techniques', 0.18890692293643951), ('word2vec', 0.18854855000972748), ('representation', 0.16081246733665466)]
```

RESULT:

Therefore word embedding techniques using WordVector was implemented successfully.

EX NO: 7(b) WORD EMBEDDING TECHNIQUES USING BERT MODEL

DATE:

AIM:

To implement word embedding techniques using BERT model in python.

ALGORITHM:

- Start
- Import required python packages.
- Obtain sample text as corpus and perform simple pre-processing steps like Tokenization and convert them into IDs.
- Convert the pre-processed IDs into Pytorch Tensors.
- Load the BERT model and train it on the tokenized corpus.
- Display BERT sentence embeddings.
- Stop

PROGRAM:

```
!pip install transformers
from transformers import BertTokenizer, BertModel
import torch

# Sample dataset
corpus = [
    "Word embeddings are a type of word representation.",
    "They capture semantic relationships between words.",
    "Word2Vec and GloVe are popular word embedding techniques.",
    "Natural Language Processing (NLP) involves the interaction between computers and humans using natural
language.",
]

# Tokenize the sentences
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
tokenized_corpus = [tokenizer.tokenize(sentence.lower()) for sentence in corpus]

# Convert tokens to input IDs
input_ids = [tokenizer.convert_tokens_to_ids(tokens) for tokens in tokenized_corpus]

# Pad sequences for equal length (optional)
max_len = max(len(ids) for ids in input_ids)
padded_input_ids = [ids + [0] * (max_len - len(ids)) for ids in input_ids]

# Convert to PyTorch tensor
input_tensors = torch.tensor(padded_input_ids)

# Load pre-trained BERT model
bert_model = BertModel.from_pretrained('bert-base-uncased')

# Forward pass to obtain embeddings
with torch.no_grad():
```

```
    embeddings = bert_model(input_tensors)

# Extract embeddings for the [CLS] token (representing the entire sentence)
sentence_embeddings = embeddings.last_hidden_state[:, 0, :]
# Display the sentence embeddings
print("BERT Sentence Embeddings:")
print(sentence_embeddings)
```

OUTPUT:

```
BERT Sentence Embeddings:
tensor([[-0.2000,  0.0831, -0.1475,  ..., -0.5986,  0.5957,  0.0060],
        [ 0.1985, -0.2514, -0.4085,  ..., -0.6974,  0.5334,  0.0074],
        [-0.0263,  0.0282,  0.1037,  ..., -0.4708,  0.6874,  0.1999],
        [ 0.2062,  0.2665, -0.2157,  ..., -0.3500,  0.5833,  0.8502]])
```

RESULT:

Therefore word embedding techniques using bert model was implemented successfully.

EX NO: 7(c) WORD EMBEDDING TECHNIQUES USING DOCVECTOR

DATE:

AIM:

To implement word embedding techniques using DocVector in python.

ALGORITHM:

- Start
- Import required python packages.
- Obtain sample text as corpus and perform simple pre-processing steps like Tokenization and convert them into Tagged Documents.
- Load the Doc2Vec model train it on the tokenized corpus.
- Display sample Doc2Vec Embeddings for a sentence.
- Stop

PROGRAM:

```
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from nltk.tokenize import word_tokenize
import nltk

# Sample dataset
corpus = [
    "Word embeddings are a type of word representation.",
    "They capture semantic relationships between words.",
    "Word2Vec and GloVe are popular word embedding techniques.",
    "Natural Language Processing (NLP) involves the interaction between computers and humans using natural language.",
]

# Tokenize the sentences
tokenized_corpus = [word_tokenize(sentence.lower()) for sentence in corpus]

# Create TaggedDocuments for Doc2Vec
tagged_data = [TaggedDocument(words=words, tags=[str(idx)]) for idx, words in enumerate(tokenized_corpus)]

# Train Doc2Vec model
doc2vec_model = Doc2Vec(vector_size=100, window=5, min_count=1, workers=4, epochs=20)
doc2vec_model.build_vocab(tagged_data)
doc2vec_model.train(tagged_data, total_examples=doc2vec_model.corpus_count, epochs=doc2vec_model.epochs)
```

```
# Example: Get document embedding for a specific sentence
doc_embedding = doc2vec_model.dv['0']  # '0' is the tag for the first sentence
print("Doc2Vec Embedding for the first sentence:", doc_embedding)
```

OUTPUT:

```
Doc2Vec Embedding for the first sentence: [-5.3700991e-03 -6.0185017e-03 -9.8932339e-03  8.6782621e-03
  3.5981950e-03  1.6649648e-04 -9.8907724e-03 -5.0714216e-03
 -9.7357109e-03  1.9821469e-03  2.8171067e-03  4.5662252e-03
 -4.3045571e-03 -3.1398097e-03 -3.0249134e-03 -8.8189924e-03
  2.2555238e-03  9.3067456e-03 -9.5601268e-03 -3.4719289e-03
 -3.8439352e-03  2.6559310e-03 -5.7229432e-03  2.7582194e-03
  5.7966914e-03 -8.0883270e-03 -8.4761158e-03 -9.9851377e-03
  4.9572280e-03 -9.2483144e-03  5.9034163e-03  6.8113264e-03
 -6.4920895e-03 -4.5636557e-03 -1.3571890e-03  1.6708201e-03
 -1.5194117e-03 -8.6795827e-03 -3.6605806e-03  1.6838842e-03
 -2.0035934e-03 -7.1952720e-03  4.2390870e-03 -8.6154360e-03
  2.6799776e-03 -4.6913279e-03  5.8437703e-04 -1.9856899e-03
  5.3962735e-03 -8.0493735e-03 -2.2123959e-03 -6.0368635e-05
 -6.7193829e-03 -6.6004046e-03 -2.0138042e-03  8.8751614e-03
 -1.2841403e-03  3.6772077e-03 -5.8526397e-03  8.9026419e-03
  3.0154150e-03  9.4703091e-03  4.4839485e-03 -4.2298739e-03
  2.2754201e-03 -4.3681632e-03  5.8233188e-03  1.8291845e-03
 -2.3639083e-03 -5.8411551e-03 -8.1485780e-03 -8.6645043e-04
 -9.0143066e-03 -9.2772907e-03 -7.8426199e-03  2.1756911e-03
 -6.4603779e-03 -7.9079047e-03  2.1076049e-03  2.1117711e-03
  8.3131474e-03  4.7377036e-03 -9.5840981e-03 -2.6258244e-04
  7.8148106e-03  2.8047781e-03  2.6757887e-03 -4.9354131e-03
  6.5099564e-03  1.6502647e-03 -7.7382429e-03  6.9300346e-03
 -9.9014193e-03 -8.1988089e-03 -4.8497431e-03  1.0082494e-02
  3.1258885e-03 -1.9847457e-03  9.0515176e-03  2.3515564e-03]
```

RESULT:

Therefore word embedding techniques using DocVector was implemented successfully.

EX NO: 8 **TEXT CLASSIFICATION**

DATE:

AIM:

To implement text classification using python.

ALGORITHM:

- Start.
- Import required python packages.
- Load IMDB dataset and divide the data into training and test data.
- Build a LSTM model and train it using IMDB training data.
- Evaluate the model using the test data and display metrics and Word Cloud.
- Stop.

PROGRAM:

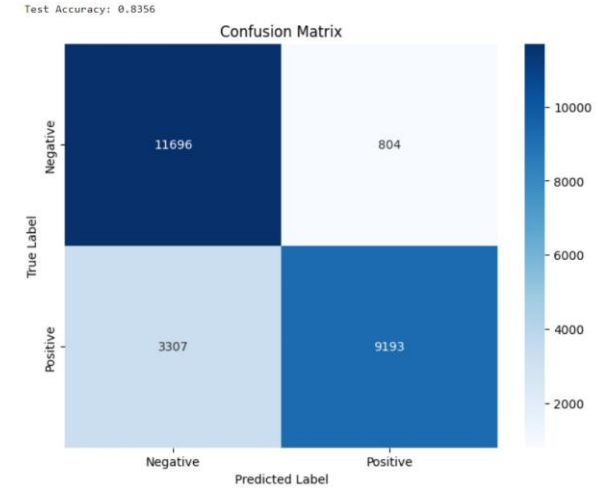
```
import os
import tarfile
import urllib.request
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, SpatialDropout1D
from sklearn.metrics import confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns

# IMDb dataset URL
url = 'https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz'
download_path = './aclImdb_v1.tar.gz'
extract_path = './aclImdb'

# Download and extract the dataset
if not os.path.exists(download_path):
    urllib.request.urlretrieve(url, download_path)

if not os.path.exists(extract_path):
    with tarfile.open(download_path, 'r:gz') as tar:
        tar.extractall(extract_path)

# Load IMDb dataset
max_words = 10000
max_len = 200
embedding_dim = 128

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_words)
```

```
# Pad sequences to a fixed length
x_train = pad_sequences(x_train, maxlen=max_len)
x_test = pad_sequences(x_test, maxlen=max_len)

# Build the LSTM model
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=max_len))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

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

# Train the model
batch_size = 64
epochs = 5
model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size, validation_data=(x_test, y_test))

# Evaluate the model
y_pred = (model.predict(x_test) > 0.5).astype('int32')  # Convert probabilities to binary predictions

# Calculate confusion matrix and accuracy
conf_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)

print(f"Test Accuracy: {accuracy:.4f}")

# Visualize confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative', 'Positive'],
yticklabels=['Negative', 'Positive'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

!pip install wordcloud

import os
import tarfile
import urllib.request
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, SpatialDropout1D
from tensorflow.keras.preprocessing.text import Tokenizer
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import seaborn as sns
import re

# IMDb dataset URL
url = 'https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz'
```

```
download_path = './aclImdb_v1.tar.gz'
extract_path = './aclImdb'

# Download and extract the dataset
if not os.path.exists(download_path):
    urllib.request.urlretrieve(url, download_path)

if not os.path.exists(extract_path):
    with tarfile.open(download_path, 'r:gz') as tar:
        tar.extractall(extract_path)

# Load IMDb dataset
max_words = 10000
max_len = 200
embedding_dim = 128

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_words)

# Convert indices back to text
word_index = imdb.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
x_train_text = [' '.join([reverse_word_index.get(i - 3, '?') for i in review]) for review in x_train]
x_test_text = [' '.join([reverse_word_index.get(i - 3, '?') for i in review]) for review in x_test]

# Text Preprocessing
def preprocess_text(text):
    # Convert to lowercase
    text = text.lower()

    # Remove HTML tags
    text = re.sub(r'<.*?>', '', text)

    # Remove special characters and punctuation
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    # Remove stopwords
    stop_words = set(['the', 'and', 'of', 'to', 'is', 'in', 'it', 'that', 'was', 'for', 'on', 'with', 'as', 'at', 'by', 'but', 'not'])
    text = ' '.join([word for word in text.split() if word not in stop_words])

    return text

x_train_text = [preprocess_text(text) for text in x_train_text]
x_test_text = [preprocess_text(text) for text in x_test_text]

# Tokenization
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(x_train_text)
x_train_seq = tokenizer.texts_to_sequences(x_train_text)
x_test_seq = tokenizer.texts_to_sequences(x_test_text)

# Pad sequences to a fixed length
x_train = pad_sequences(x_train_seq, maxlen=max_len)
x_test = pad_sequences(x_test_seq, maxlen=max_len)
```

```
# Build the LSTM model
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=max_len))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

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

# Train the model
batch_size = 64
epochs = 5
model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size, validation_data=(x_test, y_test))

# Create a Word Cloud from the training text
all_text = ' '.join(x_train_text)
wordcloud = WordCloud(width=800, height=400, random_state=21, max_font_size=110).generate(all_text)

# Plot the Word Cloud
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.show()

# Evaluate the model
y_pred = (model.predict(x_test) > 0.5).astype('int32')  # Convert probabilities to binary predictions

# Calculate confusion matrix and accuracy
from sklearn.metrics import confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
conf_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)

print(f"Test Accuracy: {accuracy:.4f}")

# Visualize confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative', 'Positive'],
yticklabels=['Negative', 'Positive'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

OUTPUT:



RESULT:

Therefore text classification was implemented successfully.

EX NO: 9(a) **TEXT MINING**

DATE:

AIM:

To implement text mining using python.

ALGORITHM:

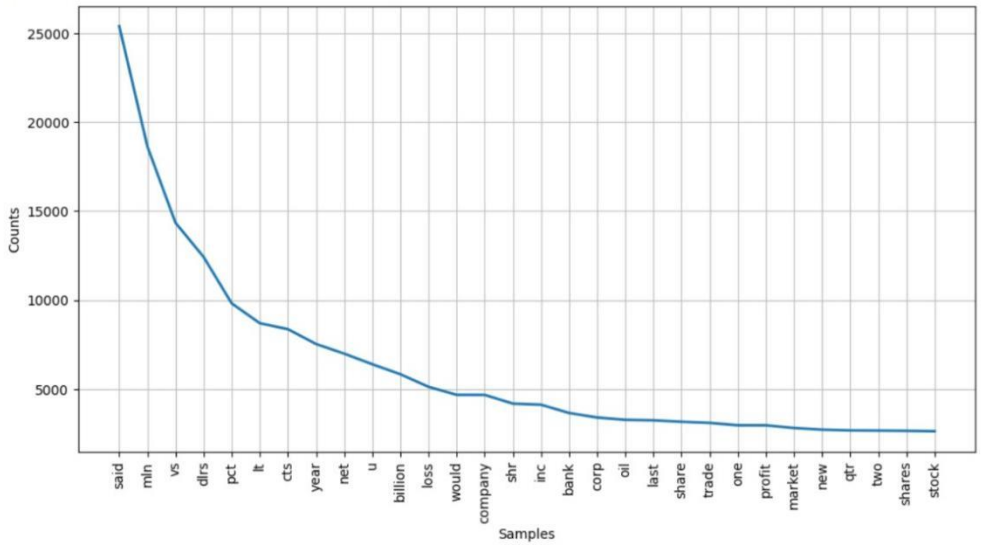
- Start.
- Import required python packages.
- Load the Reuters corpus.
- Perform basic pre-processing steps.
- Display most common words and visualize the frequency distribution plot.
- Stop.

PROGRAM:

```
pip install nltk
import nltk
from nltk.corpus import reuters, stopwords
from nltk import FreqDist
import string
import matplotlib.pyplot as plt

# Download the reuters dataset and stop words (if not already downloaded)
nltk.download('reuters')
nltk.download('stopwords')

# Load the reuters corpus
corpus = reuters.words()

# Preprocess the text: remove stop words and symbols, and lowercase
stop_words = set(stopwords.words('english'))
filtered_corpus = [word.lower() for word in corpus if word.isalpha() and word.lower() not in stop_words]

# Calculate word frequencies
word_freq = FreqDist(filtered_corpus)

# Display the most common words
print("Most common words after preprocessing:")
print(word_freq.most_common(10))

# Plot the word frequency distribution
plt.figure(figsize=(12, 6))
word_freq.plot(30, cumulative=False)
plt.show()
```

OUTPUT:

```
Most common words after preprocessing:
[('said', 25383), ('mln', 18623), ('vs', 14341), ('dlrs', 12417), ('pct', 9810), ('lt', 8696), ('cts', 8361), ('year', 7529), ('net', 6989), ('u', 6392)]
```



RESULT:

Therefore text mining was implemented successfully.

EX NO: 9(b) **TEXT MINING USING COSINE SIMILARITIES**

DATE:

AIM:

To implement text mining using cosine similarities in python.

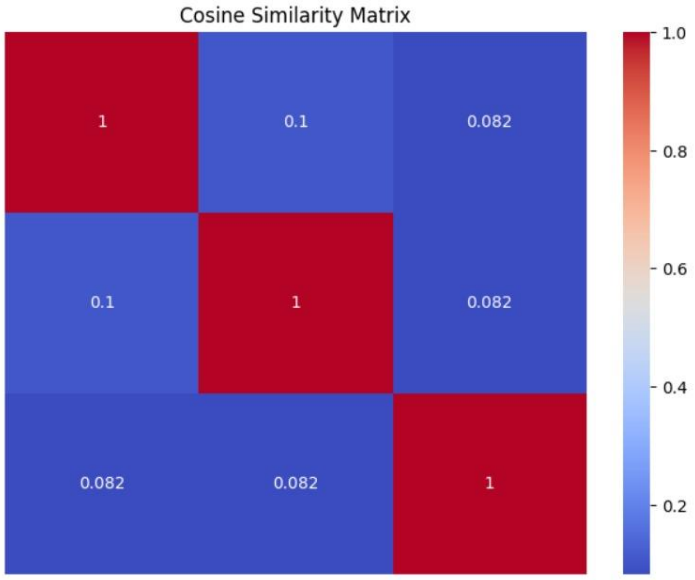
ALGORITHM:

- Start.
- Import required python packages.
- Obtain a sample corpus.
- Perform basic pre-processing steps and then vectorize it using TF-IDF.
- Display cosine similarities of the corpus and visualize the Similarity matrix using heatmap.
- Stop.

PROGRAM:

```
pip install nltk matplotlib seaborn
import nltk
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Download the NLTK punkt dataset (if not already downloaded)
nltk.download('punkt')

# Sample documents
documents = [
    "This is a sample document.",
    "Another document for demonstration.",
    "A third document to show the process.",
]

# Preprocess the documents
preprocessed_documents = [' '.join([word.lower() for word in nltk.word_tokenize(doc) if word.isalpha()]) for doc in
documents]

# Vectorize the documents using TF-IDF
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(preprocessed_documents)

# Calculate cosine similarity between documents
cosine_similarities = cosine_similarity(tfidf_matrix, tfidf_matrix)

# Plot the similarity matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cosine_similarities, annot=True, cmap='coolwarm', xticklabels=False, yticklabels=False)
plt.title('Cosine Similarity Matrix')
plt.show()
```

OUTPUT:



RESULT:

Therefore text mining using cosine similarities was implemented successfully.

EX NO: 9(c) **TEXT MINING USING CHUNKS**

DATE:

AIM:

To implement text mining using chunks in python.

ALGORITHM:

- Start.
- Import required python packages.
- Obtain a sample corpus.
- Perform basic pre-processing steps and assign PoS Tags to the entities.
- Create chunks of the corpus using the nltk.chunk method
- Display words from the corpus and their corresponding PoS Tags.
- Stop

PROGRAM:

```
import nltk
nltk.download('averaged_perceptron_tagger')

nltk.download('words')

# Download the NLTK 'punkt' and 'maxent_ne_chunker' resources (if not already downloaded)
nltk.download('punkt')
nltk.download('maxent_ne_chunker')

# Sample text
text = "Apple Inc. was founded by Steve Jobs and Steve Wozniak on April 1, 1976, in Cupertino, California."

# Tokenize the text into words
words = nltk.word_tokenize(text)

# Perform part-of-speech tagging
pos_tags = nltk.pos_tag(words)

# Perform named entity recognition
chunks = nltk.ne_chunk(pos_tags)

# Display named entities
for chunk in chunks:
    if hasattr(chunk, 'label'):
        print(f"Named Entity: {' '.join(c[0] for c in chunk)} - Type: {chunk.label()}")
```

OUTPUT:

```
Named Entity: Apple - Type: PERSON
Named Entity: Inc. - Type: ORGANIZATION
Named Entity: Steve Jobs - Type: PERSON
Named Entity: Steve Wozniak - Type: PERSON
Named Entity: Cupertino - Type: GPE
Named Entity: California - Type: GPE
```

RESULT:

Therefore text mining using chunks was implemented successfully.

EX NO: 9(d) **TEXT MINING USING POS TAGGING**

DATE:

AIM:

To implement text mining using POS tagging in python.

ALGORITHM:

- Start.
- Import the required python packages.
- Obtain a sample corpus.
- Perform basic pre-processing steps and assign PoS Tags to the entities using nltk.pos_tag method.
- Display the PoS Tags.
- Stop.

PROGRAM:

```
import nltk
from nltk.tokenize import word_tokenize
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

# Sample sentence
sentence = "The quick brown fox jumps over the lazy dog."

# Tokenize the sentence
tokens = word_tokenize(sentence)

# Perform POS tagging
pos_tags = nltk.pos_tag(tokens)
# Display the POS tags
print(pos_tags)
```

OUTPUT:

```
[('The', 'DT'), ('quick', 'JJ'), ('brown', 'NN'), ('fox', 'NN'), ('jumps', 'VBZ'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN'), ('.',
'.')]
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

('quick', 'JJ'): The word "quick" is tagged as an adjective.

('brown', 'NN'): The word "brown" is tagged as a noun.

('fox', 'NN'): The word "fox" is tagged as a noun.

('jumps', 'VBZ'): The word "jumps" is tagged as a verb in the third person singular present tense.

('over', 'IN'): The word "over" is tagged as a preposition.

('the', 'DT'): The word "the" is tagged as a determiner.

('lazy', 'JJ'): The word "lazy" is tagged as an adjective.

('dog', 'NN'): The word "dog" is tagged as a noun.

('.', '.'): The period (full stop) is tagged as a punctuation mark.

RESULT:

Therefore text mining using POS tagging was implemented successfully.

EX NO: 10 **TEXT TO SPEECH CONVERSION**

DATE:

AIM:

To implement text to speech conversion in python.

ALGORITHM:

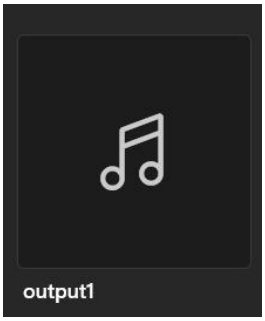
- Start.
- Import the required python packages.
- Obtain the sample text.
- Create a gTTS object and pass the sample text as text parameter and pass appropriate language parameter.
- Save the output in as an audio file and play the output file using os module.
- Stop.

PROGRAM:

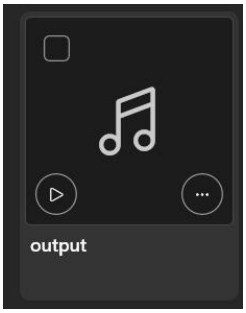
```
!pip install gtts
from gtts import gTTS
import os

# Text to be converted to speech
text = "Hello, how are you? This is a text-to-speech example."

# Create a gTTS object
tts = gTTS(text=text, lang='en')

# Save the speech as an audio file
tts.save("output1.mp3")

# Play the saved audio file
os.system("start output1.mp3")  # This works on Windows. Use appropriate command for your OS.

from gtts import gTTS
import os

text = "வணக்கம், உலகம்!"
language = "ta"  # Language code for Tamil

tts = gTTS(text=text, lang=language, slow=False)
tts.save("output.mp3")

os.system("start output.mp3")  # This will play the generated audio file
```

OUTPUT:

32512



32512



RESULT:

Therefore text to speech conversion was implemented successfully.

EX NO: 11 **SPEECH TO TEXT CONVERSION**

DATE:

AIM:

To implement speech to text conversion in python.

ALGORITHM:

- Start.
- Import the required python packages.
- Load the sample audio file and convert it into .wav format.
- Use recognizer.recognize_google method and pass the audio file as parameter.
- Display the returned text from the method.
- Stop.

PROGRAM:

```
pip install pydub
from pydub import AudioSegment

def mp3_to_wav(mp3_file, wav_file):
    # Load the MP3 file
    audio = AudioSegment.from_mp3(mp3_file)

    # Export the audio to WAV format
    audio.export(wav_file, format="wav")

if __name__ == "__main__":
    # Provide the path to your MP3 file and the desired WAV file path
    mp3_file_path = "/content/output1.mp3"
    wav_file_path = "output_wav_file.wav"

    # Convert MP3 to WAV
    mp3_to_wav(mp3_file_path, wav_file_path)
import speech_recognition as sr

def speech_to_text_from_file(audio_file_path):
    # Initialize recognizer class
    recognizer = sr.Recognizer()

    try:
        # Load audio file
        with sr.AudioFile(audio_file_path) as source:
            audio = recognizer.record(source)

        print("Recognizing...")
        # Recognize speech using Google Web Speech API
        text = recognizer.recognize_google(audio)
        print("You said:", text)
    except FileNotFoundError:
        print(f"File '{audio_file_path}' not found.")
```

```
    except sr.UnknownValueError:
        print("Sorry, could not understand audio.")
    except sr.RequestError as e:
        print("Could not request results from Google Speech Recognition service; {0}".format(e))

if __name__ == "__main__":
    # Provide the path to your audio file
    audio_file_path = "/content/output_wav_file.wav"
    speech_to_text_from_file(audio_file_path)
```

OUTPUT:

```
Recognizing...
You said: hello how are you this is a text to speech example
```

RESULT:

Therefore speech to text conversion was implemented successfully.