ANNAMALAI UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

B.E.(CSE) – ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

V – SEMESTER

22AICP508 - IMAGE AND SPEECH PROCESSING LAB

Name : _____

Reg. No.: _____

ANNAMALAI UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

B.E.(CSE) – ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

V – SEMESTER

22AICP508 - IMAGE AND SPEECH PROCESSING LAB

Certified that this is the bona-fide record of work done by Mr./Ms._____ Reg. No._____ of **B.E. Computer Science and Engineering (Artificial Intelligence and Machine Learning)** in the **22AICP508 – Image and Speech Processing Lab** during the odd semester of the academic year 2024 – 2025.

Place: Annamalainagar
Date: / / 2024

Staff in-charge

Internal Examiner

External Examiner

Annamalai University
Department of Computer Science and Engineering

VISION

To provide a congenial ambience for individuals to develop and blossom as academically superior,socially conscious and nationally responsible citizens.

MISSION

- Impart high quality computer knowledge to the students through a dynamic scholastic environment wherein they learn to develop technical, communication and leadership skills to bloom as a versatile professional.
- Develop life-long learning ability that allows them to be adaptive and responsive to the changes in career, society, technology, and environment.
- Build student community with high ethical standards to undertake innovative research and developmentin thrust areas of national and international needs.
- Expose the students to the emerging technological advancements for meeting the demands of the industry.

Program Educational Objectives

PEOs	PEO Statements
PEO1	To equip the graduates with fundamental concepts and impart problem solving skills that will help them to pursue professional careers in Computer Science and Engineering.
PEO2	To provide the graduates with the requisite knowledge to pursue higher education and carry out research in the field of Computer Science and Engineering.
PEO3	To equip the graduates with the ability to acquire new skills in emerging areas of Computer Science and Engineering such as artificial intelligence, machine learning and data science and enable them to become successful professionals and entrepreneurs.
PEO4	To ensure that the graduates exhibit high levels of ethical and moral behavior as computer engineers in addressing societal problems and providing sustainable development for the betterment of society.

Programme Specific outcomes

PSO1: Ability to investigate challenging problems in various domains and exhibit programming skills to build automation solutions.

PSO2: Ability to apply the algorithms and computational techniques to construct robust and resilient computer systems and applications.

PSO3: Ability to comprehend and implement the contemporary trends in industry and research environment paving the way for innovative solutions to existing and emerging problems.

COURSE OBJECTIVE

- To illustrate the image processing concepts through actual processing of images using python.
- To analyze simple Image enhancement techniques in spatial domain.
- To study various concepts in speech processing through various signal processing techniques

LIST OF EXPERIMENTS

Cycle 1: Image Processing

1. Smoothening and Sharpening filters in spatial domain.
2. Implementation of Edge detection methods.
3. Write a program to find the histogram equalization
   a) For full image.
   b) For part of the image.
4. Write a program to find the Fourier transform of a given image.
5. Displaying individual color components(R,G,B,Cr,CB,H,S,I) of a color image.
6. Segmentation of Image using K-Means algorithm.
7. Implementation of morphological dilation and erosion operations for a given image.
8. Write programs to extract SIFT and ORB features for given input image samples.
9. Implementation of Mouse as a paint Brush.

Cycle 2: Speech Processing

10. Write a program to find pitch, short time energy and zero crossing rate for a given speech signal.
11. Write a program to perform convolution and correlation of speech signals.
12. Write a program to perform simple low pass filtering and high pass filtering of speech signal.
13. Write a program to extract MFCC feature from sample speech signal.

COURSE OUTCOME

At the end of this course, students will be able to:

1. Work with Digital Image and Speech fundamentals using python.
2. Analyse how Image Enhancement techniques in spatial domain used in processing of images.
3. Demonstrate an ability to listen and answer the viva questions related to programming skills needed for solving real-world problems in Computer Science and Engineering.

Mapping of Course Outcomes with Programme Outcomes												
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**
CO1	3	3	3	3	2	-	-	-	-	-	-	-
CO2	3	3	2	2	2	-	-	-	-	-	-	-
CO3	2	2	-	-	-	-	-	-	-	-	-	2

Rubric for CO3 in Laboratory Courses				
Rubric	**Distribution of 10 Marks for CIE/SEE Evaluation Out of 40/60 Marks**			
	Up To 2.5 Marks	**Up To 5 Marks**	**Up To 7.5 Marks**	**Up To 10 marks**
Demonstrate an ability to listen and answer the Viva Questions related to programming skills needed for solving real-world problems in Computer Science and Engineering.	Poor listening and communication skills. Failed to relate the programming skills needed for solving the problem.	Showed better communication skill by relating the problem with the programming skills acquired but the description showed serious errors.	Demonstrated good communication skills by relating the problem with the programming skills acquired with few errors.	Demonstrated excellent communication skills by relating the problem with the programming skills acquired and have been successful in tailoring the description.

CONTENTS

Image Processing

Basic Concepts

Digital Image: Digital image is an image composed of picture elements, also known as pixels, each with finite, discrete quantities of numeric representation for its intensity or grey level. So the computer sees an image as numerical values of these pixels and in order to recognise a certain image, it has to recognise the patterns and regularities in this numerical data.

Here is a hypothetical example of how pixels form an image. The darker pixels are represented by a number closer to the zero and lighter pixels are represented by numbers approaching one. All other colours are represented by the numbers between 0 and 1.

Any colour image, there are 3 primary channels – Red, green and blue and the value of each channel varies from 0-255. In more simpler terms we can say that a digital image is actually formed by the combination of three basic colour channels Red, green, and blue whereas for a grayscale image we have only one channel whose values also vary from 0-255.

OpenCV : OpenCV (Open Source Computer Vision Library) is an open source software library for computer vision and machine learning. OpenCV was created to provide a shared infrastructure for applications for computer vision and to speed up the use of machine perception in consumer products. OpenCV, as a BSD-licensed software, makes it simple for companies to use and change the code. There are some predefined packages and libraries that make our life simple and OpenCV is one of them

This library is based on optimised C / C++ and supports Java and Python along with C++ through an interface. The library has more than 2500 optimised algorithms, including an extensive collection of computer vision and machine learning algorithms, both classic and state-of-the-art.Using OpenCV it becomes easy to do complex tasks such as identify and recognise faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D object models, generate 3D point clouds from stereo cameras, stitch images together to generate an entire scene with a high resolution image and many more. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation.

OpenCV installation

Install using Anaconda:

Anaconda is a conditional free and open-source distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment. You can download it from https://www.anaconda.com/products/individual and install it.

After successfully installing anaconda, just go to the anaconda prompt and use this command to install OpenCV

conda install -c conda-forge opencv

After this command is successfully executed, OpenCV will be available on your computer.

For Windows:

You can use pip to install OpenCV on windows. Pip is a de facto standard package-management system used to install and manage software packages written in Python and it usually comes in installed when you install Python. Use this command in the command prompt to install OpenCV

pip install opencv-python

After installing it, do check if it is installed successfully. For that just go to the command prompt and type ‘python’ and hit enter. Next type import cv2 and if there is no error then it is installed successfully.

For Mac:

You can use homebrew to install OpenCV as it makes it really easy and you just have to use this command for installing:

brew install opencv

Now that you have installed the OpenCV onto your system.

Read & Save Images

Now for OpenCV to work on any image, it must be able to read it. Here we will see how to read a file and save it after we are done with it.

Imread function in OpenCV

We use the imread function to read images,here is the syntax of this function

cv2.imread(path, flag)

The path parameter takes a string representing the path of the image to be read.The file should be in the working directory or we must give the full path to the image.The other parameter is the flag which is used to specify how our image should be read. Here are possible values that it takes and their working:

cv2.IMREAD_COLOR: It specifies to convert the image to the 3 channel BGR colour image. Any transparency of image will be neglected. It is the default flag. Alternatively, we can passinteger value 1 for this flag.

cv2.IMREAD_GRAYSCALE: It specifies to convert an image to the single channel grayscale image. Alternatively, we can pass integer value 0 for this flag.

cv2.IMREAD_UNCHANGED: It specifies to load an image as such including alpha channel.Alternatively, we can pass integer value -1 for this flag.

Usually the method imread() returns an image that is loaded from the specified file but in case the image cannot be read because of unsupported file format, missing file, unsupported or invalid format, it just returns a matrix

```
#importing the opencv module
import cv2
# using imread('path') and 1 denotes read as  color image
img = cv2.imread('beads.jpg',1)
#This is using for display the image
cv2.imshow('image',img)
cv2.waitKey() # This is necessary to be required so that the image doesn't close immediately.
#It will run continuously until the key press.
cv2.destroyAllWindows()
```

Imwrite function in OpenCV

We can use OpenCV's imwrite() function to save an image in a storage device and the file extension defines the file format.

cv2.imwrite(filename, image)

Parameters:

filename: A string representing the file name. The filename must include image format.

image: It is the image that is to be saved.

```
import cv2
# read image
img = cv2.imread('butterfly.jpeg', 1)
# save image
status = cv2.imwrite(r'C:\Users\Mirza\dog.jpeg',img)
print("Image written sucess? : ", status)
```

Basic Operation On images

Basic operations that we can do on the images once we have successfully read them.

Access pixel values and modify them

To access particular pixel values

```
import numpy as np
import cv2 as cv
img = cv.imread('beads.jpeg')
px = img[100,100]
print( px )
```

Output: [157 166 200]

OpenCV stores the color image as BGR color image,so the first value in the list is the value of the blue channel of this particular pixel, and the rest are values for green and red channels.

To access only one of the channels

```
# accessing only blue pixel
blue = img[100,100,0]
```

print(blue) **Output : 157**

To access all the B,G,R values, need to call array.item() separately for each value:

```
# accessing RED value
img.item(10,10,2)
>>59
# modifying RED value
img.itemset((10,10,2),100)
img.item(10,10,2)
>>100
```

Splitting and Merging Image Channels

```
b,g,r = cv.split(img)
img = cv.merge((b,g,r))
b = img[:,:,0]
g = img[:,:,1]
r = img[:,:,2]
```

Resize Image

When working on images, often need to resize the images according to certain requirements. Mostly done such operation in Machine learning and deep learning as it reduces the time of training of a neural network

cv2.resize(s, size,fx,fy,interpolation)

Parameters:

s - input image (required).

size - desired size for the output image after resizing (required)

fx - Scale factor along the horizontal axis.(optional)

fy - Scale factor along the vertical axis.

Interpolation(optional)

```
import cv2
import numpy as np
 #importing the opencv module
import cv2
# using imread('path') and 1 denotes read as  color image
```

```
img = cv2.imread('dog.jpg',1)
print(img.shape)
img_resized=cv2.resize(img, (780, 540),
        interpolation = cv2.INTER_NEAREST)
cv2.imshow("Resized",img_resized)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Ex. No. 1 Smoothening and Sharpening of Image
Date:

a) Smoothening of Image

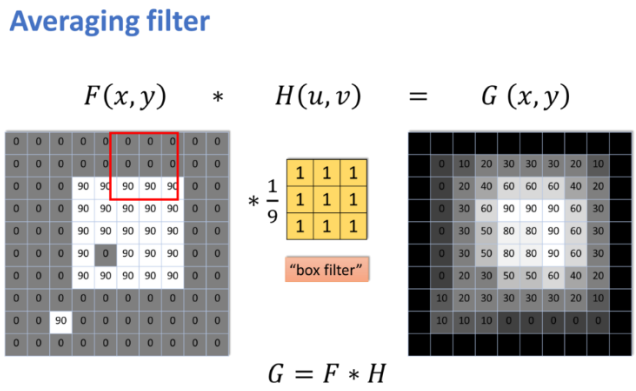
Aim

To smoothening the given input gray scale image using filters in spatial domain.

Theoretical Concept

Smoothening of images: Image blurring is achieved by convolving the image with a low-pass filter kernel. It is useful for removing noise. It actually removes high frequency content (eg: noise, edges) from the image. So edges are blurred a little bit in this operation (there are also blurring techniques which don't blur the edges.

This is done by convolving an image with a normalized box filter. It simply takes the average of all the pixels under the kernel area and replaces the central element. This is done by the function **cv.blur()** . We should specify the width and height of the kernel. A 3x3 normalized box filter would look like the below:



Implementation

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('test.jpg')
blur = cv2.blur(img,(5,5))
plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title('Blurred')
```

plt.xticks([]), plt.yticks([])

plt.show()

Sample Input and Output

The above code can be modified for Gaussian and Bilateral blurring:

blur = cv.GaussianBlur(img,(5,5),0)
blur = cv.bilateralFilter(img,9,75,75)

Result

Thus, a program has been written to smoothen the given input image using python.

b) Sharpening of images

Aim

To sharpening the given input gray scale image using filters in spatial domain.

Theoretical Concept

The process of sharpening is usually used to enhance edges in an image. There are many filters that we can use but one that can sharpen our image is represented in a matrix below.

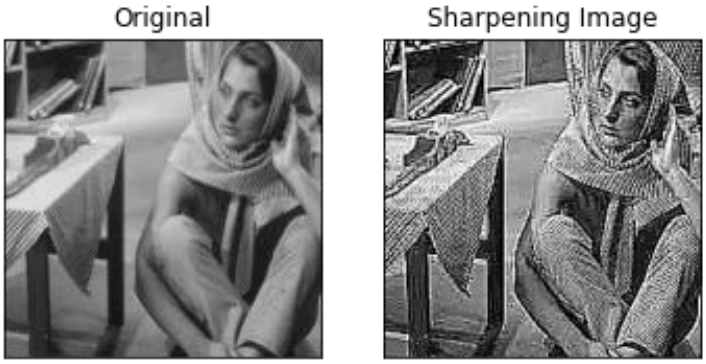
-1	-1	-1
-1	9	-1
-1	-1	-1

This filter has a positive 9 in a center, whereas it has -1 at all other places. For this particular filter we don't have an implemented OpenCV function. Therefore, we will use a function **cv2.filter2D()** which will process our image with an arbitrary filter. This filter is often used for sharpening colored images.

Implementation

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('sharp.jpg',1)
# Creating our sharpening filter
filter = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])
# Applying cv2.filter2D function on our Cybertruck image
sharpen_img=cv2.filter2D(img,-1,filter)
plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(sharpen_img),plt.title('Shapening Image')
plt.xticks([]), plt.yticks([])
plt.show()
```

Sample input and Output



Result

Thus, a program has been written to sharpen the given input image using python.

Ex. No. 2 Implementation of Edge Detection Methods

Date:

Aim:

To Implement Edge Detection for the given input image using Sobel edge detection Method.

Theoretical Concept:

Edge detection is one of the fundamental operations when we perform image processing. It helps us reduce the amount of data (pixels) to process and maintains the structural aspect of the image.

Sobel edge detector is a gradient based method based on the first order derivatives. It calculates the first derivatives of the image separately for the X and Y axes. The operator uses two 3X3 kernels which are convolved with the original image to calculate approximations of the derivatives - one for horizontal changes, and one for vertical. The matrix below shows Sobel Kernels in x-dir and y-dir:

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

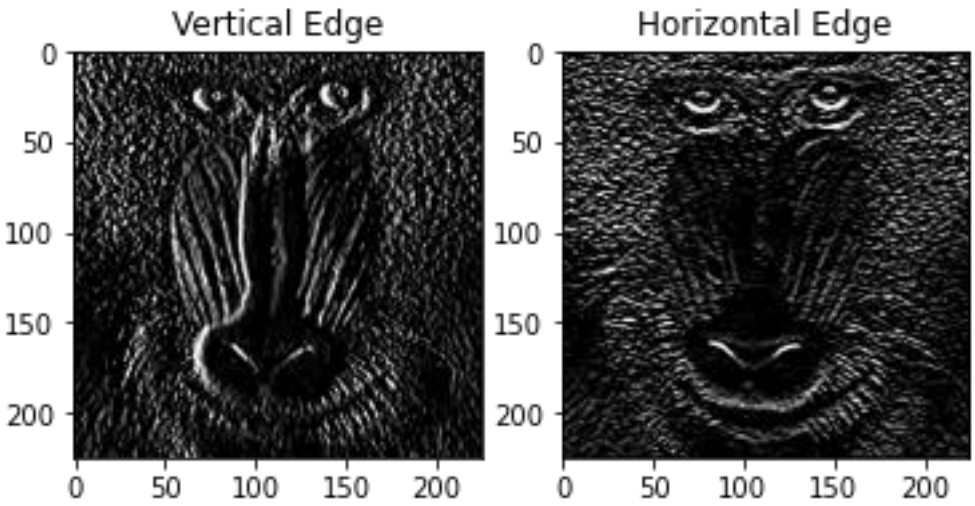
$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Implementation

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
filter = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
 image = cv2.imread("baboon.jpg")
#Filter the image using filter2D, which has inputs: (grayscale image, bit-depth, kernel)
sobel_verti_image = cv2.filter2D(image,-1,filter)
sobel_hori_image = cv2.filter2D(image,-1,np.flip(filter.T, axis=0))
plt.subplot(121)
plt.imshow(sobel_verti_image, cmap='gray')
```

```
plt.title("Vertical Edge")
plt.subplot(122)
plt.imshow(sobel_hori_image, cmap='gray')
plt.title("Horizontal Edge")
plt.show()
```

Sample Input and Output



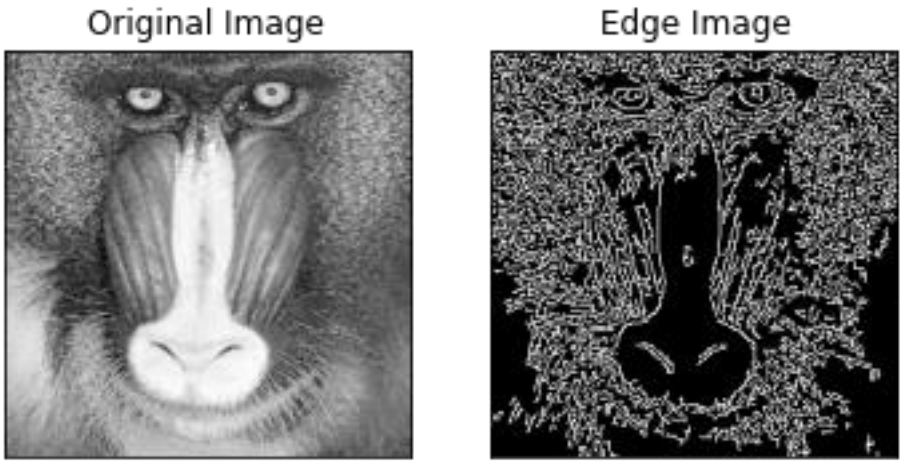
Result

Thus, a program has been written to detect the edges of the given input image using Sobel Edge Detection Algorithm.

The Edge detection can be done with Canny edge Detection method. Try with following code.

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('baboon.jpg',0)
edges = cv.Canny(img,100,200)
plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
plt.show()
```

Sample Input and Output



Ex. No. 3 Histogram Equalization of an Image

Date:

Aim

To adjust the contrast of an image, histogram equalization technique to be applied for full image and Part of an image.

Theoretical Concept

Histogram as a graph or plot, which gives you an overall idea about the intensity distribution of an image. It is a plot with pixel values (ranging from 0 to 255, not always) in X-axis and corresponding number of pixels in the image on Y-axis.

Consider an image whose pixel values are confined to some specific range of values only. For eg, brighter image will have all pixels confined to high values. But a good image will have pixels from all regions of the image. So you need to stretch this histogram to either ends (as given in below image, from wikipedia) and that is what Histogram Equalization does (in simple words). This normally improves the contrast of the image.

Numpy provides a function, **np.histogram()** for creating histogram

histogram, bin_edges = np.histogram(image, bins=256, range=(0, 1))

The parameter **bins** determines the histogram size, or the number of "bins" to use for the histogram. Bins= 256 because the pixel count for each of the 256 possible values in the grayscale image. The bins method of calculation is: 0-0.99, 1-1.99, 2-2.99, etc. So the last range is 255-255.99.

The parameter **range** is the range of values each of the pixels in the image can have. 0 and 1, which is the value range of our input image after transforming it to grayscale.0 to 256 for color image.

The first output of the **np.histogram** function is a one-dimensional NumPy array, with 256 rows and one column, representing the number of pixels with the color value corresponding to the index. I.e., the first number in the array is the number of pixels found with color value 0, and the final number in the array is the number of pixels found with color value 255. The second output of np.histogram is an array with the bin edges and one column and 257 rows (one more than the histogram itself). There are no gaps between the bins, which means that the end of the first bin, is the start of the second and so on. For the last bin, the array also has to contain the stop, so it has one more element, than the histogram.

OpenCV has a function to do this, cv.equalizeHist(). Its input is color image and output is our histogram equalized image.

Implementation

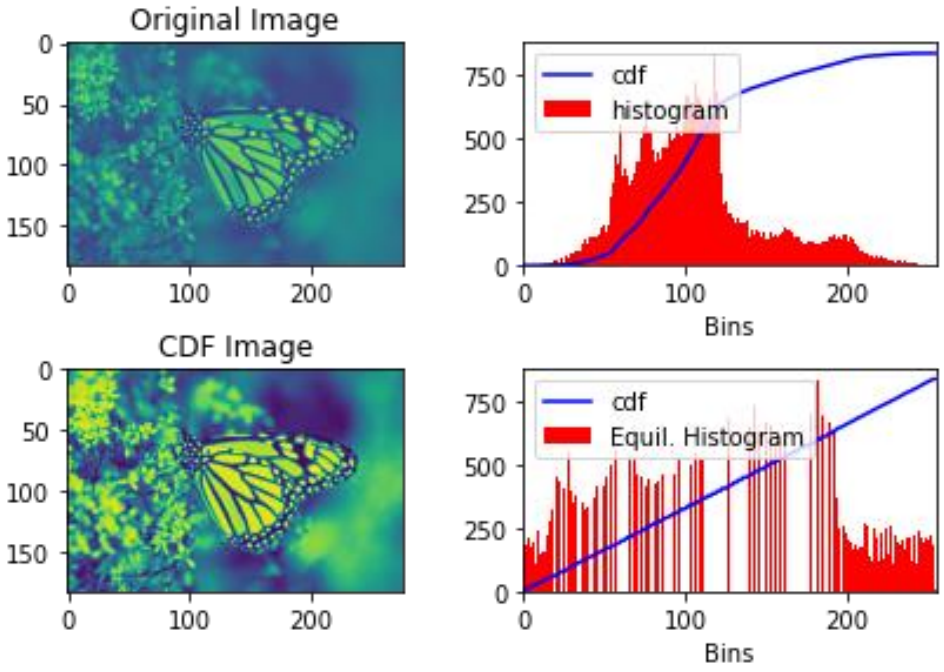
a. Histogram equalization for full image

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('butterfly.jpg',0)
#subplot with 2 rows and 2 columns
plt.subplot(2,2,1)
#To display original image
plt.title('Original Image')
plt.imshow(img)

# get image histogram
hist,bins = np.histogram(img.flatten(),256,[0,256])
# cumulative distribution function
cdf = hist.cumsum()
# normalize
cdf_normalized = cdf * float(hist.max()) / cdf.max()
#subplot with 2 rows and 2 columns
plt.subplot(2,2,2)
#plotting cdf and histogram
plt.plot(cdf_normalized, color = 'b')
plt.hist(img.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.legend(('cdf','histogram'), loc = 'upper left')
plt.xlabel('Bins')
cdf_m = np.ma.masked_equal(cdf,0)
cdf_m = (cdf_m - cdf_m.min())*255/(cdf_m.max()-cdf_m.min())
```

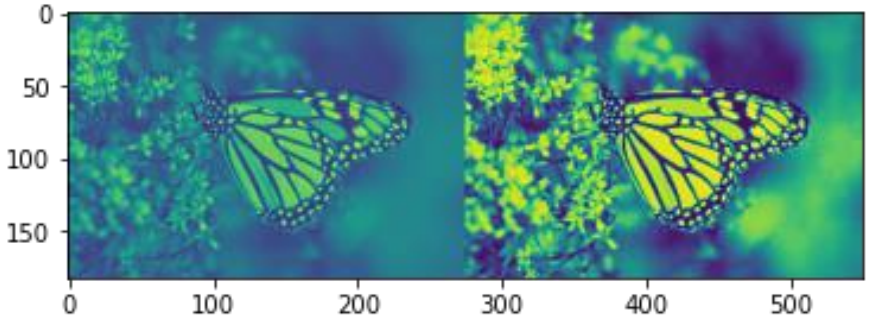
```
cdf = np.ma.filled(cdf_m,0).astype('uint8')
# normalized image
img2 = cdf[img]
plt.subplot(2,2,3)
plt.title('CDF Image')
#To display Cdf image
plt.imshow(img2)
#histogram Equilization
hist,bins = np.histogram(img2.flatten(),256,[0,256])
cdf = hist.cumsum()
cdf_normalized = cdf * float(hist.max()) / cdf.max()
plt.subplot(2,2,4)
#plotting cdf and H.Equilization
plt.plot(cdf_normalized, color = 'b')
plt.hist(img2.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.legend(('cdf','Equil. Histogram'), loc = 'upper left')
plt.tight_layout()
plt.xlabel('Bins')
plt.show()

#OpenCV Implementation
img = cv.imread('butterfly.jpg',0)
equ = cv.equalizeHist(img)
res = np.hstack((img,equ)) #stacking images side-by-side
cv.imwrite('res.png',res)
img = cv.imread('res.png',0)
plt.imshow(img)
plt.show()
```

Sample Input and Output

Numpy Implementation



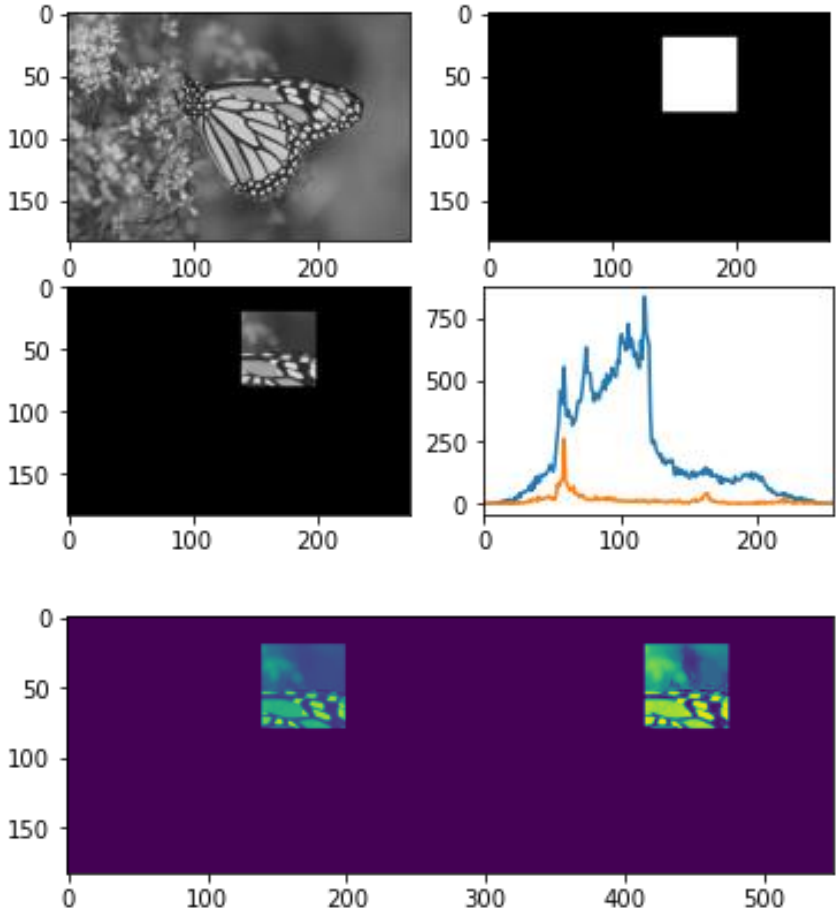
Opencv Implementation



b. Histogram equalization for part of an Image

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('home.jpg',0)
# create a mask
mask = np.zeros(img.shape[:2], np.uint8)
mask[20:80, 140:200] = 255
masked_img = cv.bitwise_and(img,img,mask = mask)
# Calculate histogram with mask and without mask
# Check third argument for mask
hist_full = cv.calcHist([img],[0],None,[256],[0,256])
hist_mask = cv.calcHist([img],[0],mask,[256],[0,256])
plt.subplot(221), plt.imshow(img, 'gray')
plt.subplot(222), plt.imshow(mask,'gray')
plt.subplot(223), plt.imshow(masked_img, 'gray')
plt.subplot(224), plt.plot(hist_full), plt.plot(hist_mask)
plt.xlim([0,256])
plt.show()
equ = cv.equalizeHist(masked_img)
res = np.hstack((masked_img,equ)) #stacking images side-by-side
cv.imwrite('result.png',res)
img = cv.imread('result.png',0)
plt.imshow(img)
plt.show()
```

Sample Input and Output



Result

Thus, a program has been written to adjust the contrast of the given input image using Histogram Equalization technique.

Ex. No. 4 Fast Fourier Transform
Date:

Aim

To write a program that converts an image from spatial domain to frequency domain.

Theoretical Concept

Fast Fourier Transform is applied to convert an image from the image (spatial) domain to the frequency domain. Applying filters to images in frequency domain is computationally faster than to do the same in the image domain. Once the image is transformed into the frequency domain, filters can be applied to the image by convolutions. FFT turns the complicated convolution operations into simple multiplications. An inverse transform is then applied in the frequency domain to get the result of the convolution.

Step 1: Compute the 2-dimensional Fast Fourier Transform

The result from FFT process is a complex number array which is very difficult to visualize directly. Therefore, we have to transform it into 2-dimension space. FFT can visualized in terms of Spectrum. The white area in the spectrum image show the high power of frequency. The corners in the spectrum image represent low frequencies. Therefore, combining two points, the white area on the corner indicates that there is high energy in low/zero frequencies which is a very normal situation for most images.

Step 2: Shift the zero-frequency component to the center of the spectrum.

2-D FFT has translation and rotation properties, so we can shift frequency without losing any piece of information. zero-frequency component to the center of the spectrum which makes the spectrum image more visible for human. Moreover, this translation could help us implement high/low-pass filter easily

.

Step 3 :Shift the zero-frequency component back to original location

Step 4 :Compute the 2-dimensional inverse Fast Fourier Transform

The processes of step 3 and step 4 are converting the information from spectrum back to gray scale image. It could be done by applying inverse shifting and inverse FFT operation.

Functions

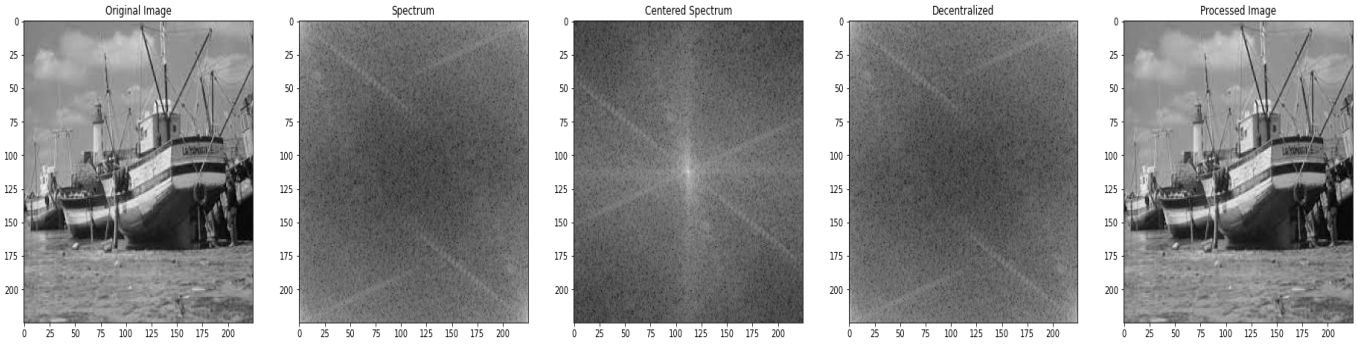
fft.fft2( ) :Compute the 2-dimensional discrete Fourier Transform by means of the Fast Fourier Transform (FFT).

fft.fftshift( ) :Shift the zero-frequency component to the center of the spectrum.

Implementation

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img_c1 = cv2.imread("ship.jpg", 0)
img_c2 = np.fft.fft2(img_c1)
img_c3 = np.fft.fftshift(img_c2)
img_c4 = np.fft.ifftshift(img_c3)
img_c5 = np.fft.ifft2(img_c4)
plt.figure(figsize=(6.4*5, 4.8*5), constrained_layout=False)
plt.subplot(151)
plt.imshow(img_c1, "gray")
plt.title("Original Image")
plt.subplot(152)
plt.imshow(np.log(1+np.abs(img_c2)), "gray")
plt.title("Spectrum")
plt.subplot(153)
plt.imshow(np.log(1+np.abs(img_c3)), "gray")
plt.title("Centered Spectrum")
plt.subplot(154)
plt.imshow(np.log(1+np.abs(img_c4)), "gray")
plt.title("Decentralized")
plt.subplot(155)
plt.imshow(np.abs(img_c5), "gray")
plt.title("Processed Image")
plt.show()
```

Sample Input and Output



Result:

Thus, a program has been written to convert an image from spatial domain to frequency domain using Fast Fourier Transform.

Ex. No. 5 Individual Color Components

Date:

Aim

To extract and display the individual color components (R, G, B) and derived color spaces (Chromium Red, Chromium Blue, Hue, Saturation, Intensity) from a color image.

Theoretical Concept

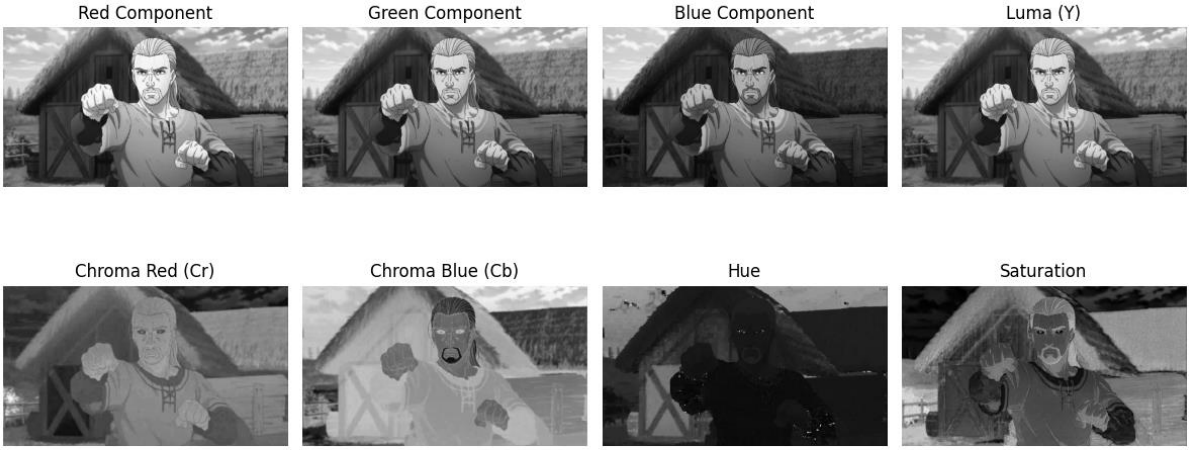
Color components represent the fundamental elements of color in various color models. In the RGB model, colors are formed by combining Red, Green, and Blue intensities. The YCrCb model separates brightness (Luma, Y) from color information (Chroma: Cr for red, Cb for blue). The HSI model decomposes color into Hue (color type), Saturation (color purity), and Intensity (brightness). These models enable different manipulations and interpretations of color in digital images and video systems.

Implementation

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

# Load an image
image_path = 'image.jpg'  # replace with your image path
img = cv2.imread(image_path)

# Convert from BGR (OpenCV default) to RGB
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Split the RGB channels
R, G, B = cv2.split(img_rgb)

# Convert RGB to YCrCb (Luma and Chroma components)
img_ycrcb = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2YCrCb)
Y, Cr, Cb = cv2.split(img_ycrcb)

# Convert RGB to HSI (Hue, Saturation, Intensity)
img_hsv = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2HSV)
H, S, I = cv2.split(img_hsv)

# Display the individual channels
fig, axs = plt.subplots(2, 4, figsize=(12, 6))
```

```
# RGB Components
axs[0, 0].imshow(R, cmap='gray')
axs[0, 0].set_title('Red Component')
axs[0, 0].axis('off')

axs[0, 1].imshow(G, cmap='gray')
axs[0, 1].set_title('Green Component')
axs[0, 1].axis('off')

axs[0, 2].imshow(B, cmap='gray')
axs[0, 2].set_title('Blue Component')
axs[0, 2].axis('off')

# YCrCb Components
axs[0, 3].imshow(Y, cmap='gray')
axs[0, 3].set_title('Luma (Y)')
axs[0, 3].axis('off')

axs[1, 0].imshow(Cr, cmap='gray')
axs[1, 0].set_title('Chroma Red (Cr)')
axs[1, 0].axis('off')

axs[1, 1].imshow(Cb, cmap='gray')
axs[1, 1].set_title('Chroma Blue (Cb)')
axs[1, 1].axis('off')

# HSI Components
axs[1, 2].imshow(H, cmap='gray')
axs[1, 2].set_title('Hue')
axs[1, 2].axis('off')

axs[1, 3].imshow(S, cmap='gray')
axs[1, 3].set_title('Saturation')
axs[1, 3].axis('off')

plt.tight_layout()
plt.show()
```

Sample Input and Output



Result

Thus a python program to display each color components of an image has been written and verified successfully.

Ex. No. 6 Image Segmentation

Date:

Aim

To segment the given input image using Cluster based segmentation algorithm.

Theoretical Concept

Image segmentation is a branch of digital image processing which focuses on partitioning an image into different parts according to their features and properties. The primary goal of image segmentation is to simplify the image for easier analysis. In image segmentation, you divide an image into various parts that have similar attributes. The parts in which you divide the image are called Image Objects.

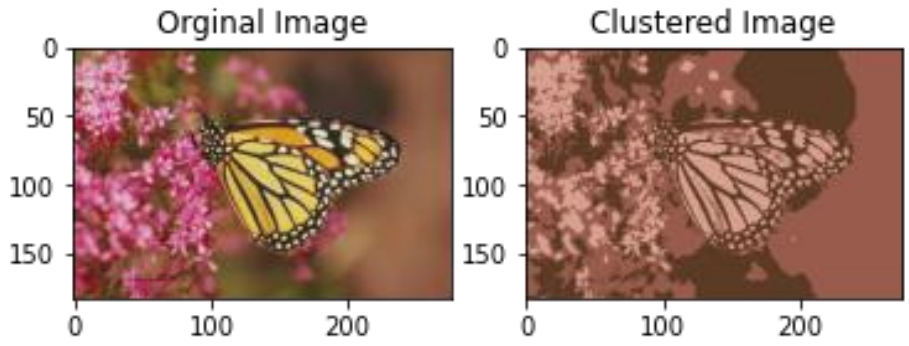
Clustering and similar machine learning algorithms use this method to detect unknown features and attributes. K-means is a simple unsupervised machine learning algorithm. It classifies an image through a specific number of clusters. It starts the process by dividing the image space into k pixels that represent k group centroids. Then they assign each object to the group based on the distance between them and the centroid. When the algorithm has assigned all pixels to all the clusters, it can move and reassign the centroids.

Implementation

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
%matplotlib inline

# Read in the image
image = plt.imread('butterfly.jpg')

# Change color to RGB (from BGR)
print(image.shape)
plt.subplot(121)
plt.imshow(image)

plt.title("Orginal Image")
# Reshaping the image into a 2D array of pixels and 3 color values (RGB)
pixel_vals = image.reshape((-1,3))

# Convert to float type
pixel_vals = np.float32(pixel_vals)
#the below line of code defines the criteria for the algorithm to stop running,#which will happen is 100 iterations are run or the epsilon (which is the required accuracy) #becomes 85%
```

```
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.85)

# then perform k-means clustering with number of clusters defined as 3
#also random centres are initially chosed for k-means clustering k = 3

retval, labels, centers = cv2.kmeans(pixel_vals, k, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)

# convert data into 8-bit values
centers = np.uint8(centers)
segmented_data = centers[labels.flatten()]

# reshape data into the original image dimensions
segmented_image = segmented_data.reshape((image.shape))
print(segmented_image.shape)
plt.subplot(122)

plt.imshow(segmented_image)
plt.title("Clustered Image")
```

Sample Input and Output



Result

Thus, a program has been written to segment an image using K-means algorithm.

Ex. No. 7 Morphological Operations

Date:

Aim

Implementation of morphological dilation and erosion operations for a given input image.

Theoretical Concept:

Erosion: Erodes away the boundaries of the foreground object

Used to diminish the features of an image.

Working of erosion

1. A kernel (a matrix of odd size (3,5,7) is convolved with the image.
2. A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel are 1, otherwise, it is eroded (made to zero).
3. Thus, all the pixels near the boundary will be discarded depending upon the size of the kernel.
4. So, the thickness or size of the foreground object decreases or simply the white region decreases in the image.

Dilation: Increases the object area

Used to accentuate features

Working of dilation

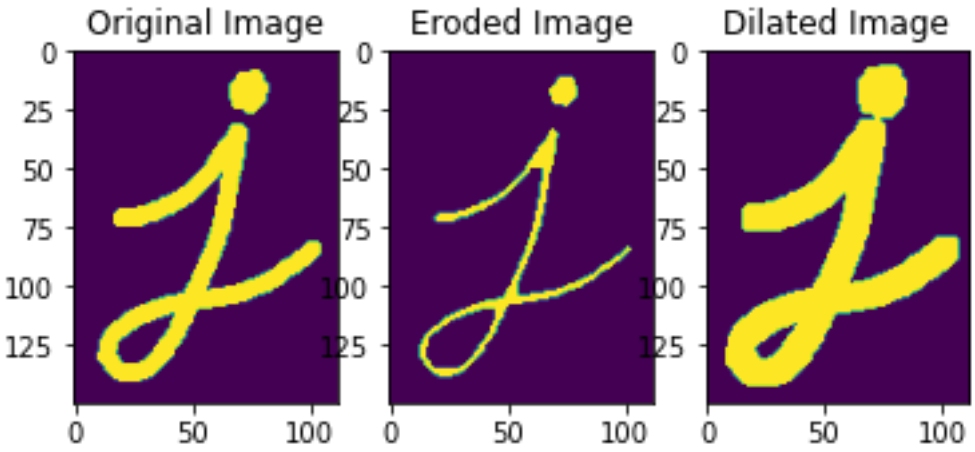
1. A kernel (a matrix of odd size (3,5,7) is convolved with the image
2. A pixel element in the original image is '1' if at least one pixel under the kernel is '1'.
3. It increases the white region in the image or the size of the foreground object increases

Implementation

```
import cv2

import numpy as np

import matplotlib.pyplot as plt

img = cv2.imread('j.png',0)

# use a 5x5 kernel with full of ones

kernel = np.ones((5,5),np.uint8)

# The first parameter is the original image,kernel is the matrix with which image is # convolved and third #parameter is the number # of iterations, which will determine how much you want to erode/dilate a #given image.

erosion = cv2.erode(img,kernel,iterations = 1)
```

```
dilation = cv2.dilate(img,kernel,iterations = 1)
plt.subplot(131)
plt.imshow(img)
plt.title("Original Image")
plt.subplot(132)
plt.imshow(erosion)
plt.title("Eroded Image")
plt.subplot(133)
plt.imshow(dilation)
plt.title("Dilated Image")
```

Sample Input and Output



Result

Thus, a program has been written to implement morphological dilation and erosion operations for a given input image.

Ex. No. 8 Feature Extraction Techniques

Date:

Aim

To write the programs to extract SIFT and ORB features for given input image samples

Theoretical Concept

SIFT: The scale-invariant feature transform (SIFT) is a feature detection algorithm in computer vision to detect and describe local features in images. SIFT keypoints of objects are first extracted from a set of reference images and stored in a database.

ORB: ORB (Oriented FAST and Rotated BRIEF) is a fusion of FAST keypoint detector and BRIEF descriptor with some added features to improve the performance. FAST is Features from Accelerated Segment Test used to detect features from the provided image. It also uses a pyramid to produce multiscale-features.

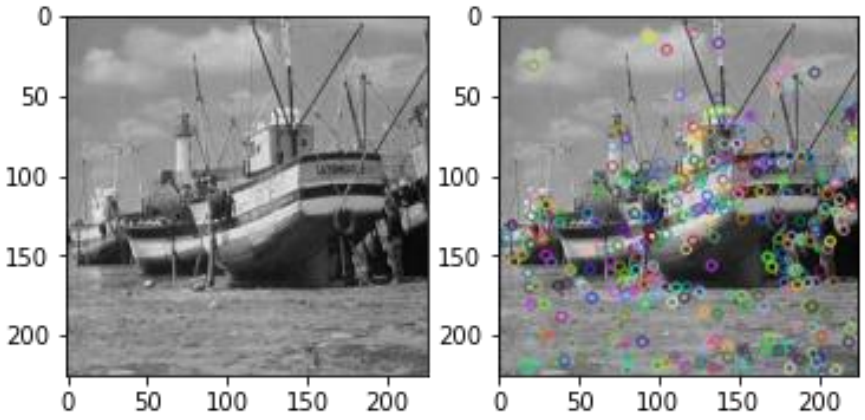
a. Scale-Invariant Feature Transform (SIFT)

Implementation

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('ship.jpg')
#plt.subplot(1, 2, 1)
plt.subplot(121)
plt.imshow(img)
# convert to greyscale
gray= cv.cvtColor(img,cv.COLOR_BGR2GRAY)
# create SIFT feature extractor
sift = cv.SIFT_create()
# detect features from the image
kp = sift.detect(gray,None)
# draw the detected key points
img=cv.drawKeypoints(gray,kp,img)
plt.subplot(122)
```

```
plt.imshow(img)
plt.show()
```

Sample Input and Output

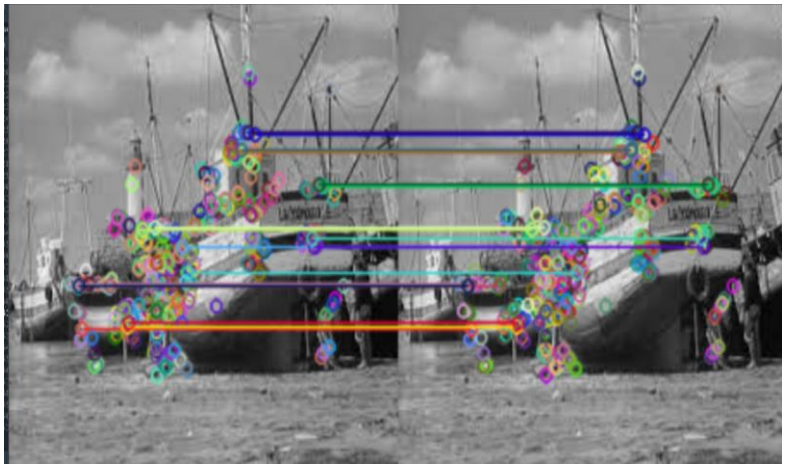


b. ORB (Oriented FAST and Rotated BRIEF)

Implementation

```
import numpy as np
import cv2

# Read the query image as query_img and train image This query image is what you need to
find in train image # Save it in the same directory  with the name image.jpg

query_img = cv2.imread('ship.jpg')
train_img = cv2.imread('ship.jpg')

# Convert it to grayscale
query_img_bw = cv2.cvtColor(query_img,cv2.COLOR_BGR2GRAY)
train_img_bw = cv2.cvtColor(train_img, cv2.COLOR_BGR2GRAY)

# Initialize the ORB detector algorithm
orb = cv2.ORB_create()

# Now detect the keypoints and compute the descriptors for the query image  and train image
queryKeypoints, queryDescriptors = orb.detectAndCompute(query_img_bw,None)
trainKeypoints, trainDescriptors = orb.detectAndCompute(train_img_bw,None)

# Initialize the Matcher for matching the keypoints and then match the # keypoints
matcher = cv2.BFMatcher()
matches = matcher.match(queryDescriptors,trainDescriptors)

# draw the matches to the final image containing both the images the drawMatches() function
takes both images and keypoints and outputs the matched query image with its train image
final_img = cv2.drawMatches(query_img, queryKeypoints,
train_img, trainKeypoints, matches[:20],None)

final_img = cv2.resize(final_img, (1000,650))

# Show the final image
while(1):
   cv2.imshow("Matches", final_img)
   if cv2.waitKey(20) & 0xFF == 27:
      break
cv2.destroyAllWindows()
```

Sample Output :



Result

Thus, a program has been written to extract the features from a given input image for further processing of images.

Ex. No. 9 Mouse as a Paint Brush
Date:

Aim

To implement a program to use the mouse as a paint brush using double-click.

Theoretical Concept

OpenCV provides a facility to use the mouse as a paint brush or a drawing tool. Whenever any mouse event occurs on the window screen, it can draw anything. Mouse events can be left-button down, left-button up, double-click, etc. It gives us the coordinates (x,y) for every mouse event. By using these coordinates, we can draw whatever we want.

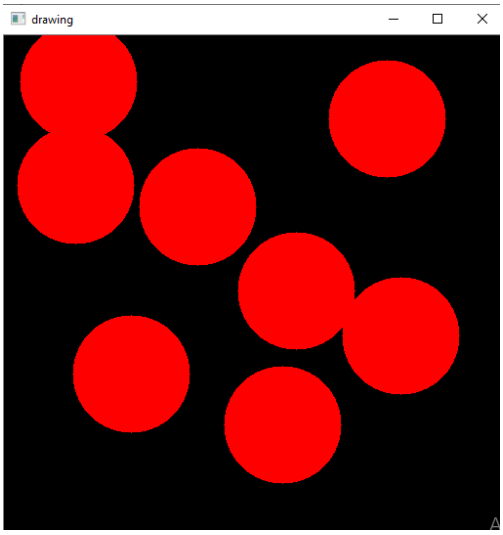
To Draw Circle

To draw a circle on the window screen, we first need to create a mouse callback function by using the cv2.setMouseCallback() function. The mouse callback function is facilitated by drawing a circle using double-click.

Implementation

```
import cv2
import numpy as np
# Creating mouse callback function
def draw_circle(event,x,y,flags,param):
   if(event == cv2.EVENT_LBUTTONDBLCLK):
        cv2.circle(img,(x,y),100,(255,255, 0),-1)
# Creating a black image, a window and bind the function to window
img = np.zeros((512,512,3), np.uint8)
cv2.namedWindow('image')
cv2.setMouseCallback('image',draw_circle)
while(1):
   cv2.imshow('image',img)
   if cv2.waitKey(20) & 0xFF == 27:
      break
cv2.destroyAllWindows()
```

Sample Input and Output



Result

The program has been written to use the mouse as a paint brush using double-click.

SPEECH PROCESSING
BASIC CONCEPTS

LIBROSA

- A Python package for audio and music signal processing.
- Audio processing includes digital signal processing, machine learning, information retrieval, and musicology.
- Provides implementations of a variety of common functions used in the field of audio signal processing.

Reading and plotting of audio files and samples

One option to read audio is to use LIBROSA's function **librosa.load**.

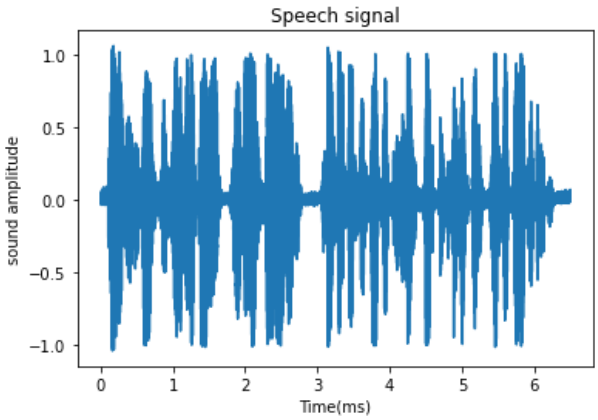
- Per default, librosa.load resamples the audio to 22050 Hz . Setting sr=None keeps the native sampling rate.
- The loaded audio is converted to a float with amplitude values lying in the range of [−1,1].
- librosa.load is essentially a wrapper that uses either PySoundFile or audioread.
- When reading audio, librosa.load first tries to use PySoundFile. This works for many formats, such as WAV, FLAC, and OGG. However, MP3 is not supported. When PySoundFile fails to read the audio file (e.g., for MP3), a warning is issued, and librosa.load falls back to another library called audioread. When ffmpeg is available, this library can read MP3 files.

 # Playing audio file

```
import librosa

import IPython

import matplotlib.pyplot as plt

#path of the audio file

audio_data = 'F:\speech.wav'

#This returns an audio time series as a numpy array with a default sampling rate(sr) of 22KHZ

x = librosa.load(audio_data, sr=None)

#We can change this behavior by resampling at sr=44.1KHz.

x = librosa.load(audio_data, sr=44000)

IPython.display.Audio(audio_data)
```

#plotting audio file

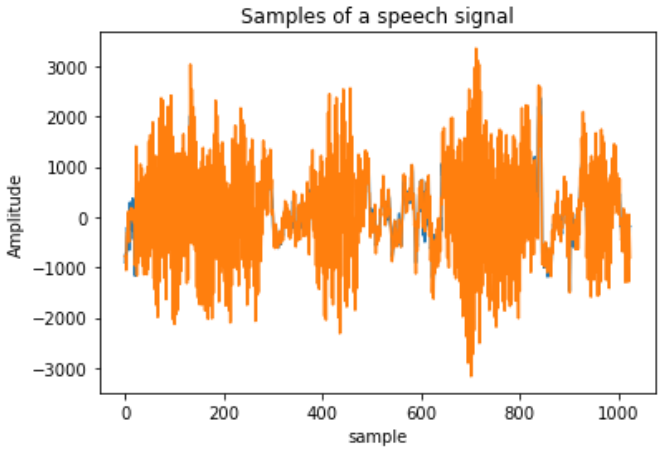
```
from scipy.io.wavfile import read

import matplotlib.pyplot as plt

import librosa as lr

import IPython

import numpy as np

audio='speech'

#This returns an audio time series as a numpy array with a default sampling rate(sr) of 22KHZ

y, sr = lr.load('./{}.wav'.format(audio))

time = np.arange(0,len(y))/sr

fig, ax = plt.subplots() #returns single array of objects, Create just a figure and only one subplot

ax.plot(time,y)

plt.title("Speech signal")

ax.set(xlabel='Time(ms)',ylabel='sound amplitude')

plt.show()
```



Reading and plotting audio samples

```
from scipy.io.wavfile import read

import matplotlib.pyplot as plt

# read audio samples

input_data = read("F:\speech.wav")

audio = input_data[1]
```

```
# plot the first 1024 samples
plt.plot(audio[0:1024])
# label the axes
plt.ylabel("Amplitude")
plt.xlabel("sample")
# set the title
plt.title("Samples of a speech signal")
# display the plot
plt.show()
```



Resampling at sr=44.1KHz.

```
x = lr.load(audio_data, sr=44000)
print(x)
IPython.display.Audio(audio_data)
```

Output:

```
(array([-0.02462068, -0.02384715, -0.02957482, ...,  0.07869781,
        0.06410387,  0.04024108], dtype=float32), 44000)
```

Ex. No. 10 Pitch, Energy and Zero Crossing Rate of speech signal
Date:

Aim:

To find the pitch, Energy and Zero crossing rate of given input speech signal.

Theoretical Concepts

Pitch

Pitch, in speech, the relative highness or lowness of a tone as perceived by the ear, which depends on the number of vibrations per second produced by the vocal cords. Pitch is the main acoustic correlate of tone and intonation.

Energy

Sum of squares of all the samples in the frame.

$$E_l = \sum_{n=0}^{N-1} \tilde{x}_l^2(n)$$

Zero crossing

Zero crossing is said to occur if successive samples have different algebraic signs. Simple measure of the frequency content of a signal.

$$Z_i = \frac{1}{N} \sum_{n=0}^{N-1} \frac{1}{2} \left| sgn\left(\tilde{x}_i(n)\right) - sgn\left(\tilde{x}_i(n+1)\right) \right|$$

Where,

$$sgn\left(\tilde{x}_i(n)\right) = \begin{cases} 1, & x \geq 0 \\ -1, & x < 0 \end{cases}$$

Implementation

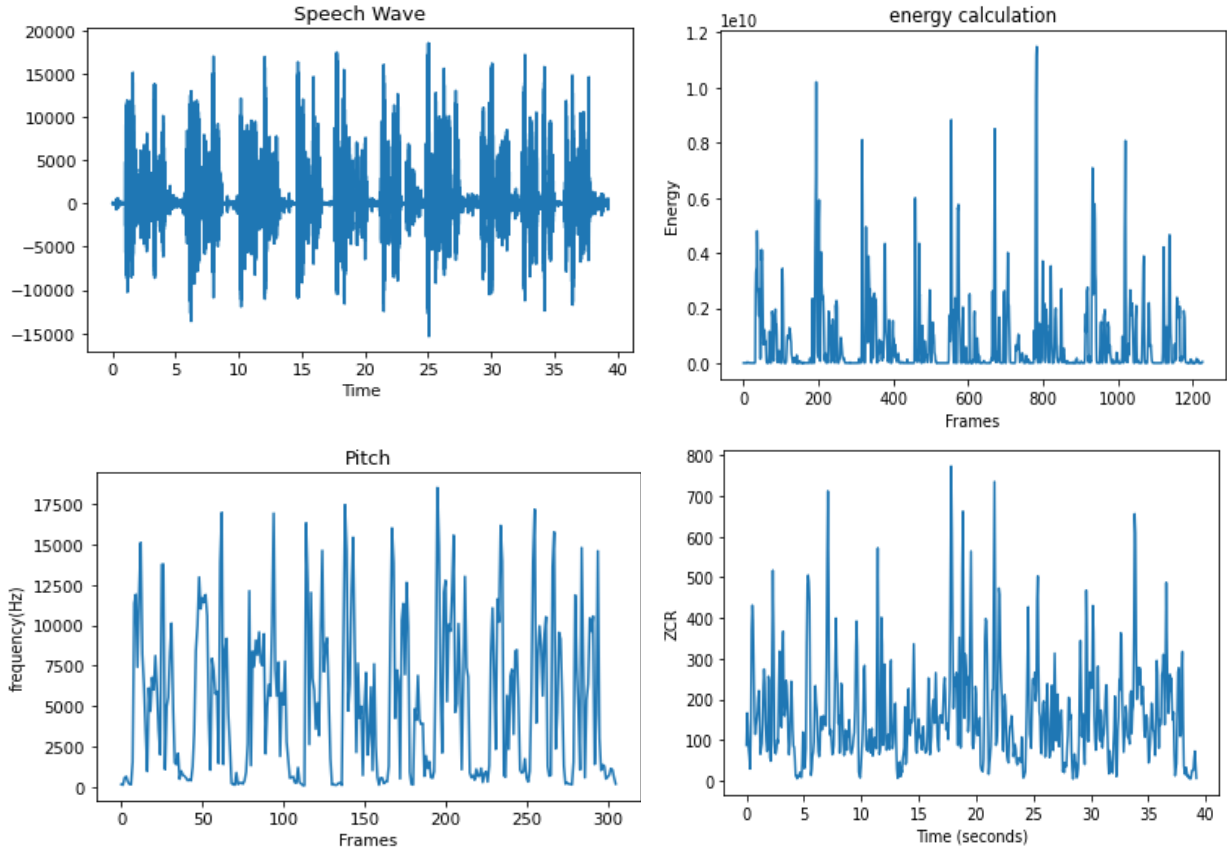
```
import wave
import numpy as np
import matplotlib.pyplot as plt
from scipy.io.wavfile import read, write
import math
# Calculate the energy of each frame 256 samples are one frame
def calEnergy(wave_data):
```

```
    energy = []
    sum = 0
    for i in range(len(wave_data)):
        sum = sum + (int(wave_data[i]) * int(wave_data[i]))
        if (i + 1) % 256 == 0:
            energy.append(sum)
            sum = 0
        elif i == len(wave_data) - 1:
            energy.append(sum)
    return energy
# read the wav file and parameters
f = wave.open("sample1.wav" ,"rb")
# getparams() returns the format information of all WAV files at once
params = f.getparams()
# nframes Number of sampling points
nchannels, sampwidth, framerate, nframes = params[:4]
# readframes() Read data according to the sampling point
Str_data = f.readframes(nframes) # str_data is a binary string
# gets the frame rate
Str_data = np.frombuffer(Str_data, dtype ="int16")
f_rate = f.getframerate()
# Convert to a two-byte array form (two bytes per sample point)
wave_data = np.fromstring(Str_data, dtype = np.short)
print( "Number of sample points:" + str(len(wave_data))) #output should be the number of
sample point
# to Plot the x-axis in seconds
    # you need get the frame rate
    # and divide by size of your signal
    # to create a Time Vector
    # spaced linearly with the size
    # of the audio file
```

```
time = np.linspace(0, # start
        len(Str_data) / f_rate,
        num = len(Str_data)  )
plt.figure(1)
plt.title("Speech Wave")
plt.xlabel("Time")
plt.plot(time, Str_data)
plt.show()
energy = calEnergy(wave_data)
plt.figure()
plt.plot(energy)
plt.axis('tight')
plt.xlabel('Frames')
plt.ylabel('Energy')
plt.title('energy calculation')
plt.show()
# Pitch Calculation
FRAME_SIZE = 1024
def ProcessFrame(frame, Fs):
    freq = max(frame)
    return freq
Fs, data = read('sample1.wav')
numFrames = int(len(data) / FRAME_SIZE)
frequencies = np.zeros(numFrames)
for i in range(numFrames):
    frame = data[i * FRAME_SIZE : (i + 1) * FRAME_SIZE]
    frequencies[i] = ProcessFrame(frame.astype(float), Fs)
plt.figure()
plt.plot(frequencies)
plt.axis('tight')
```

```
plt.xlabel('Frames')
plt.ylabel('frequency(Hz)')
plt.title('Pitch')
plt.show()
# Zero Crossing Rate Calculation
def ZeroCR(waveData,frameSize,overlap):
    wlen = len(waveData)
    step = frameSize - overlap
    frameNum = math.ceil(wlen/step)
    zcr = np.zeros((frameNum,1))
    for i in range(frameNum):
        curFrame = waveData[np.arange(i*step,min(i*step+frameSize,wlen))]
        #To avoid DC bias, usually we need to perform mean subtraction on each frame
        curFrame = curFrame - np.mean(curFrame) # zero-justified
        zcr[i] = sum(curFrame[0:-1]*curFrame[1::]<=0)
    return zcr
overlap = 512
#Read wav file and parameters from energy programm
wave_data.shape = -1, 1
zcr = ZeroCR(wave_data,FRAME_SIZE,overlap)
time2 = np.arange(0, len(zcr)) * (len(wave_data)/len(zcr) / f_rate)
plt.plot(time2, zcr)
plt.ylabel('ZCR')
plt.xlabel('Time (seconds)')
plt.show()
f.close()
```

Sample Input and Output



Result

Thus, the program has been written to find pitch, energy and zero crossing rate from the given speech signal.

Ex. No. 11 Convolution and Correlation of speech signals

Date:

(a) Convolution of speech signals

Aim

Write a program to perform Convolution of two different speech signals.

Theoretical Concept

Convolution is a mathematical way of combining two signals to form a third signal. Convolution is a formal mathematical operation, just as multiplication and addition. The convolution is the same operation as multiplying the polynomials whose coefficients are the elements of two signals. The convolution of two signals u(n) and v(n) is given by,

$$w(k) = \sum_{j=-\infty}^{\infty} u(j)v(k+1-j)$$

The output sample is simply a sum of products involving simple arithmetic operations such as additions, multiplications and delays. But practically u(n) and v(n) are finite in length. If the lengths of the two sequences being convolved are m and n, then the resulting sequence after convolution is of length m+n-1 and is given by,

$$w(k) = \sum_{j=\max(1,k+1-n)}^{\min(k,m)} u(j)v(k+1-j)$$

When m = n, this gives

w (1) = u(1)v(1)
w (2) = u(1)v(2) + u(2)v(1)
w (3) = u(1)v(3) + u(2)v(2) + u(3)v(1)
...
W (n) = u(1)v(n) + u(2)v(n − 1) + ... + u(n)v(1)
...
W (2n − 1) = u(n)v(n)

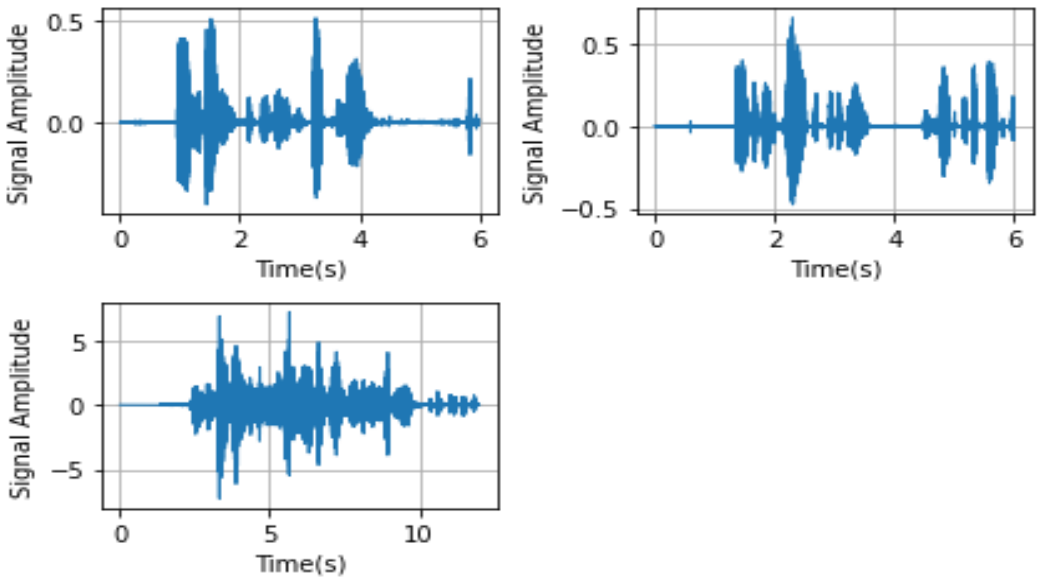
Implementation

```
#convolution of two audio signals
from scipy import signal
import numpy as np
import librosa
import matplotlib.pyplot as plt
d_file = "set1.wav"
```

```
d1_file="set2.wav"
#load audio files with librosa
d, sr = librosa.load(d_file,8000)
#print(sr)
d1, sr = librosa.load(d1_file,8000)
plt.subplot(2,2,1)
t=np.linspace(0,len(d)/sr,len(d))
plt.plot(t,d, lw=1)
#plt.xlabel('Time(s) ')
#plt.ylabel('Amplitude')

#Convolve two N-dimensional arrays.Convolve in1 and in2.discrete, linear convolution
y=signal.convolve(d,d1)
#plotting subplot nrows,ncols,nsubplot
#plt.plot(d,'r')
plt.grid(True)

#Naming the x-axis, y-axis
plt.xlabel('Time(s)')
plt.ylabel('Amplitude')

#plotting subplot nrows,ncols,nsubplot
plt.subplot(2,2,2)
t=np.linspace(0,len(d1)/sr,len(d1))
plt.plot(t,d1, lw=1)
#plt.plot(d1,'g')
plt.grid(True)
#Naming the x-axis, y-axis
plt.xlabel('Time(s)')
plt.ylabel(' Amplitude')

#plotting subplot nrows,ncols,nsubplot
plt.subplot(2,2,3)
```

```
t=np.linspace(0,len(y)/sr,len(y))
plt.plot(t,y, lw=1)
#plt.plot(y,'b')
plt.grid(True)
#Naming the x-axis, y-axis
plt.title('Covolution of signals')
plt.xlabel('Time(s)')
plt.ylabel(' Amplitude')
#Adds space between different subplots
plt.tight_layout()
plt.grid(True)
#To load the display window
plt.show()
plt.grid(True)
#To load the display window
plt.show()
```

Sample Input and Output



Result

Thus, the program has been written to compute the convolution of two input speech signals.

(b) Correlation of Speech signals

Aim

Write a program to perform Correlation of speech signals.

Theoretical Concept

Correlation is basically used to compare two signals. Correlation measures the similarity between two signals. The correlation process is essentially the convolution of two sequences in which one of the sequences has been reversed. It is classified into two types namely auto correlation and cross correlation.

Auto correlation

The autocorrelation of a signal describes the similarity of a signal against a time-shifted version of itself. The autocorrelation is useful for finding repeated patterns in a signal. For example, at short lags, the autocorrelation can tell us something about the signal's fundamental frequency. For longer lags, the autocorrelation may tell us something about the tempo of a signal.

$$r(k) = \sum_{n} x(n)x(n-k)$$

In above equation, k is often called the **lag** parameter. r(k) is maximized at k=0 and is symmetric about k.

Cross correlation

Cross correlation between a pair of signals x(n) and y(n) is given by

$$r_{xy}(l) = \sum_{n=-\infty}^{\infty} x(n)y(n-1) \qquad l = 0, \pm 1, \pm 2, \ldots$$

The parameter l called lag, indicates the time-shift between the pair. The time sequence y(n)is said to be shifted by l samples with respect to the reference sequence x(n) to the right for the positive values of l, and shifted by l samples to the left for negative values of l.

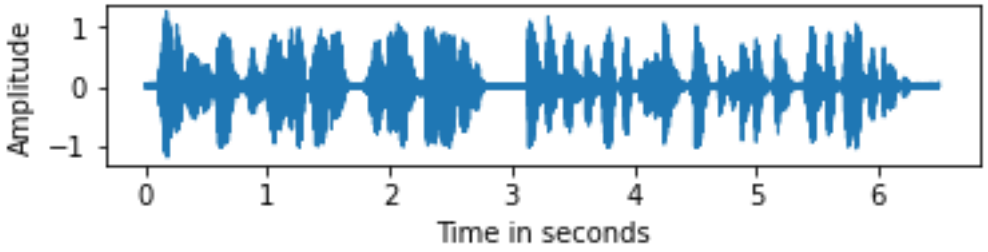
Implementation

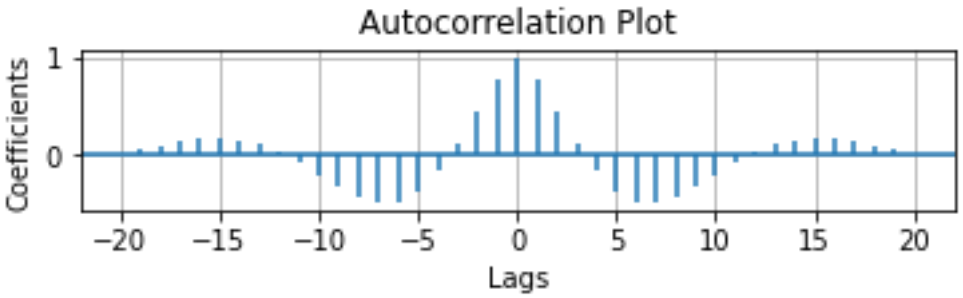
Auto correlation

```
# Importing the libraries.

import numpy as np

import librosa

import matplotlib.pyplot as plt

d_file = "speech.wav"
```

```
d, sr = librosa.load(d_file,8000)
plt.subplot(2,1,1)
t=np.linspace(0,len(d)/sr,len(d))
plt.plot(t,d, lw=1)
plt.xlabel('Time in seconds')
plt.ylabel('Amplitude')
#computing autocorrelation using librosa
r = librosa.autocorrelate(d, max_size=10000)
plt.subplot(2,1,2)
print(r.shape)
# Adding plot title
plt.title("Autocorrelation Plot")
# Providing x-axis name.
plt.xlabel("Lags")
plt.ylabel('Coefficients')
# Plotting the Autocorreleation plot
plt.acorr(r, maxlags = 20)
# Displaying the plot
print("The Autocorrelation plot for the data is:")
plt.grid(True)
plt.tight_layout(pad=3.0)
#plt.tight_layout()
plt.show()
```

Sample Input and Output





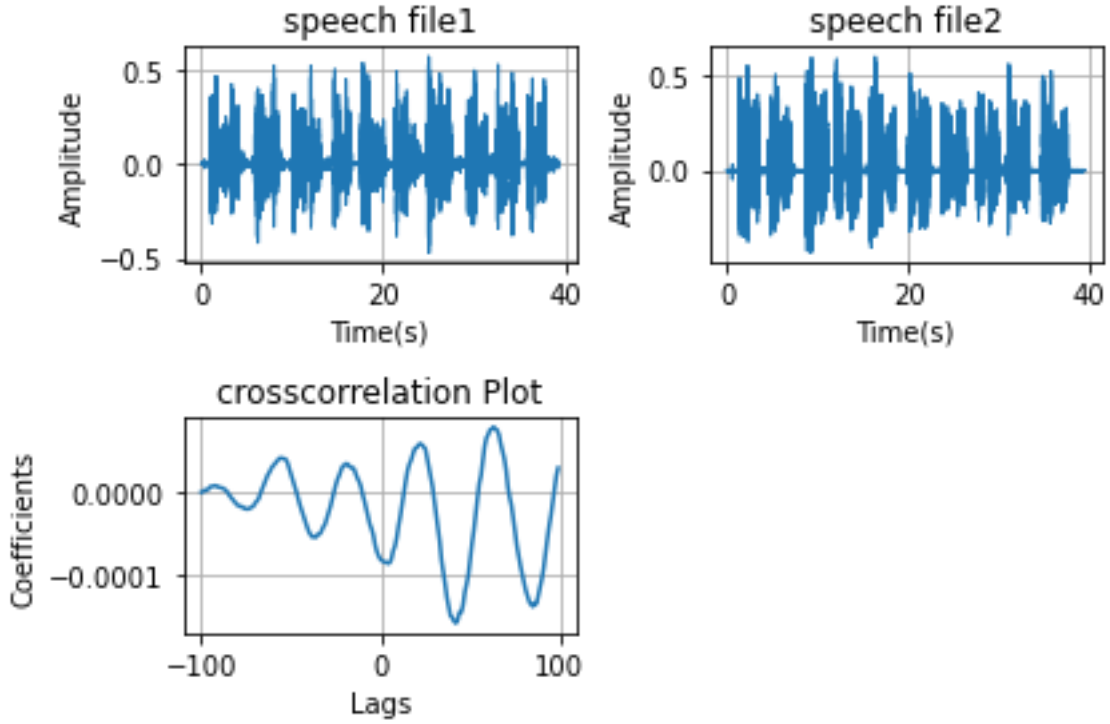
Cross correlation

```
import librosa
import matplotlib.pyplot as plt
from scipy import signal
import numpy as np
d_file = "sample1.wav"
d1_file="sample2.wav"
#load audio files with librosa
d, sr = librosa.load(d_file,8000)
d1, sr = librosa.load(d1_file,8000)
#t=np.linspace(0,len(d)/sr,len(d))
#plt.plot(t,d, lw=1)
#print array values of first .wav file
print(d)
#print array values of second .wav file
print(d1)
#cross correlation of two arrays of same size i.e. two .wav files of same size
```

```
correlation = np.correlate(d, d1, mode="full")
lags = np.arange(-100, 100)
print(correlation)
#plotting subplot nrows,ncols,nsubplot
plt.subplot(2,2,1)
plt.title('speech file1')
t=np.linspace(0,len(d)/sr,len(d))
plt.plot(t,d, lw=1)
#plotting first .wav file
#plt.plot(d,'g')
plt.grid(True)
plt.xlabel('Time(s)')
plt.ylabel('Amplitude')
#Naming the x-axis, y-axis
#plotting subplot nrows,ncols,nsubplot
plt.subplot(2,2,2)
plt.title('speech file2')
t=np.linspace(0,len(d1)/sr,len(d1))
plt.plot(t,d1, lw=1)
#plotting second .wav file
#plt.plot(d,'r')
#grid display
plt.grid(True)
plt.xlabel('Time(s)')
plt.ylabel('Amplitude')
#plotting subplot nrows,ncols,nsubplot
# Plotting the crosscorreleation plot.
plt.subplot(2,2,3)
plt.plot(lags,correlation[0:200])
# Adding plot title.
```

```
plt.title("crosscorrelation Plot")
# Providing x-axis name.
plt.xlabel("Lags")
plt.ylabel('Coefficients')
#grid display
plt.grid(True)
  #Adds space between different subplots
plt.tight_layout()
plt.show()
```

Sample Input and Output



Result

Thus, the program has been written to find the Correlation coefficients of given input speech signal.

Ex. No. 12 Low pass Filter and High Pass Filter to Speech Signal

Date:

Aim

To apply Low pass filter and High pass filter in the given input speech signal

Theoretical Concept

Low pass filter

A low-pass filter is a filter that passes signals with a frequency lower than a selected cutoff frequency and attenuates signals with frequencies higher than the cutoff frequency. The exact frequency response of the filter depends on the filter design. A low-pass filter is the complement of a high-pass filter.

High-pass filter

A high-pass filter (HPF) is an electronic filter that passes signals with a frequency higher than a certain cutoff frequency and attenuates signals with frequencies lower than the cutoff frequency. The amount of attenuation for each frequency depends on the filter design

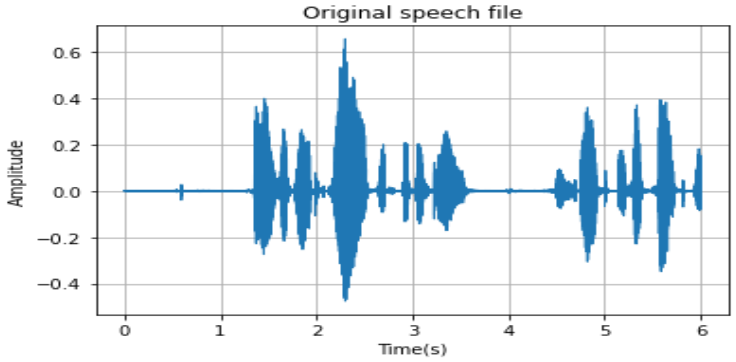
Implementation

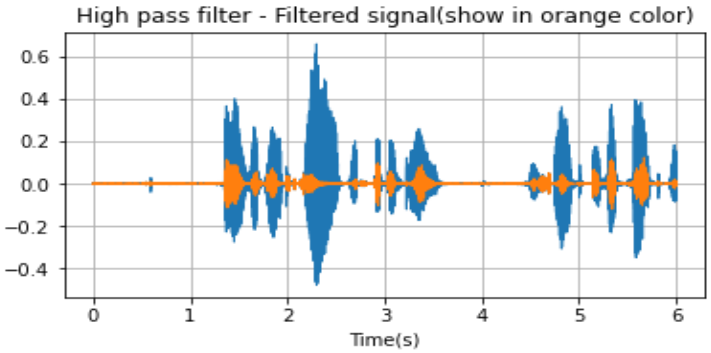
High Pass Filter

```
import numpy as np
import scipy.signal as sg
import matplotlib.pyplot as plt
import librosa
from scipy.signal import butter,filtfilt
d_file = "set2.wav"
d, sr = librosa.load(d_file,8000)
t=np.linspace(0,len(d)/sr,len(d))
plt.plot(t,d, lw=1)
plt.xlabel('Time(s)')
plt.ylabel('Amplitude')
plt.title('Original speech file')
plt.grid(True)
plt.show()
#high-pass filter (2000 Hz cutoff frequency)
```

```
#butter(n,Wn)returns the transfer function coefficients
#of an nth-order lowpass digital Butterworth filter with normalized cutoff frequency Wn.
#butter(n,Wn,ftype) designs a lowpass, highpass, bandpass, or bandstop Butterworth filter,
#depending on the value of ftype and the number of elements of Wn.
b, a = sg.butter(4, 2000. / (sr / 2.), 'high')
#performs zero-phase digital filtering by processing the input data, x, in both the forward and reverse directions.
#After filtering the data in the forward direction, filtfilt reverses the filtered sequence and runs it back through the filter.
d_fil = sg.filtfilt(b, a, d)
#plotting signal
fig, ax = plt.subplots(1, 1, figsize=(6, 3))
#equal sized arrays
t = np.linspace(0., len(d) / sr, len(d))
#plotting signal with line width =1
ax.plot(t, d, lw=1)
#plotting filtered signal with line width=1
ax.plot(t, d_fil, lw=1)
plt.xlabel('Time(s)')
plt.title('High pass filter - Filtered signal(show in orange color)')
plt.grid(True)
plt.show()
```

Sample Input Output



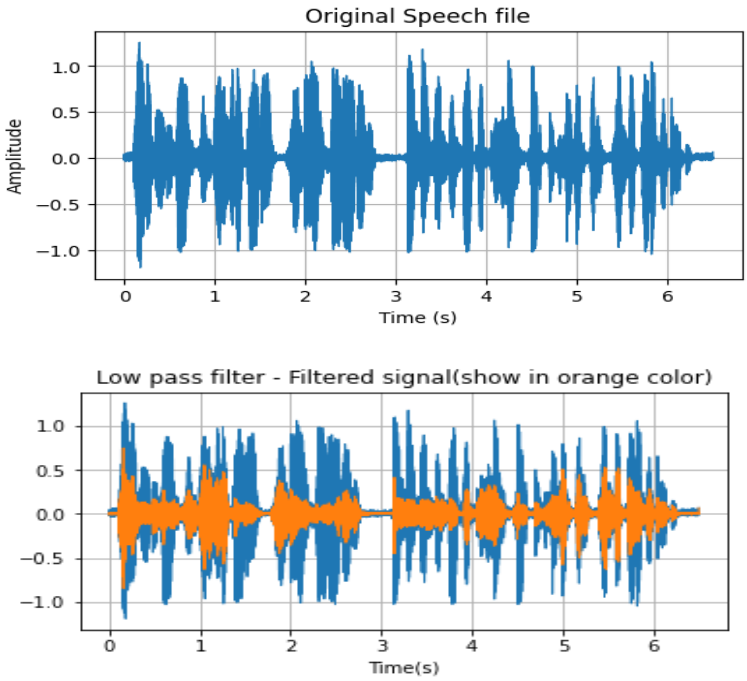


Low Pass Filter

```
import numpy as np
import scipy.signal as sg
import matplotlib.pyplot as plt
import librosa
from scipy.signal import butter,filtfilt
d_file = "F:\speech.wav"
d, sr = librosa.load(d_file,8000)
t=np.linspace(0,len(d)/sr,len(d))
plt.plot(t,d, lw=1)
plt.xlabel('Time(s)')
plt.ylabel('Amplitude')
plt.title('Original speech file')
plt.grid(True)
plt.show()
#Butterworth low-pass filter applied to this sound (500 Hz cutoff frequency)
#butter(n,Wn)returns the transfer function coefficients
#of an nth-order lowpass digital Butterworth filter with normalized cutoff frequency Wn.
#butter(n,Wn,ftype) designs a lowpass, highpass, bandpass, or bandstop Butterworth filter,
#depending on the value of ftype and the number of elements of Wn.
b, a = sg.butter(4, 500. / (sr / 2.), 'low')
#performs zero-phase digital filtering by processing the input data, x, in both the forward and reverse directions.
#After filtering the data in the forward direction, filtfilt reverses the filtered sequence and runs it back through the filter.
```

```
d_fil = sg.filtfilt(b, a, d)
#plotting signal
fig, ax = plt.subplots(1, 1, figsize=(6, 3))
#equal sized arrays
t = np.linspace(0., len(d) / sr, len(d))
#plotting signal with line width =1
ax.plot(t, d, lw=1)
#plotting filtered signal with line width=1
ax.plot(t, d_fil, lw=1)
plt.xlabel('Time(s)')
plt.title('Low pass filter - Filtered signal(show in orange color)')
plt.grid(True)
plt.show()
```

Sample Input and Output



Result

Thus, the program has been implemented to apply low pass filter and high pass filter to the given input speech signal.

Ex. No. 13 Mel Frequency Cepstral Coefficients

Date:

Aim

To write a program to extract MFCC feature from sample speech signal.

Theoretical concept

Mel-frequency cepstral coefficients (MFCCs) are coefficients that collectively make up an MFC.[1] They are derived from a type of cepstral representation of the audio clip (a nonlinear "spectrum-of-a-spectrum"). The difference between the cepstrum and the mel-frequency cepstrum is that in the MFC, the frequency bands are equally spaced on the mel scale, which approximates the human auditory system's response more closely than the linearly-spaced frequency bands used in the normal spectrum. This frequency warping can allow for better representation of sound, for example, in audio compression.

MFCCs are commonly derived as follows:

1. Take the Fourier transform of (a windowed excerpt of) a signal.
2. Map the powers of the spectrum obtained above onto the mel scale, using triangular overlapping windows or alternatively, cosine overlapping windows.
3. Take the logs of the powers at each of the mel frequencies.
4. Take the discrete cosine transform of the list of mel log powers, as if it were a signal.

The MFCCs are the amplitudes of the resulting spectrum.

librosa.feature.mfcc(y=None, sr=22050, S=None, n_mfcc=20, dct_type=2, norm='ortho', lifter=0, **kwargs)

y = audio time series

sr = number > 0 [scalar] , sampling rate of y

S = np.ndarray [shape=(d, t)] or None,log-power Mel spectrogram

n_mfcc = int > 0 [scalar], number of MFCCs to return

dct_type= {1, 2, 3},Discrete cosine transform (DCT) type. By default, DCT type-2 is used.

Norm = None or ‘ortho’ :If dct_type is 2 or 3, setting norm='ortho' uses an ortho-normal DCT basis.

Kwarg = sadditional keyword arguments to melspectrogram, if operating on time series input

Implentation

```
import librosa
y,sr=librosa.load('speech.wav')
print(y)
print("Sample rate: {0}Hz".format(sr))
print("Audio duration: {0}s".format(len(y) / sr))
print(librosa.feature.mfcc(y=y,sr=sr,n_mfcc=13))
```

Sample Input and Output

```
Sample rate: 22050Hz
Audio duration: 6.5s
 [-0.01738697 -0.02810948 -0.00662852 ...  0.07072017  0.07115451   0.06516534]
[[-219.37381    -215.54312    -217.26982    ... -245.7114  -245.99171    -244.60828]
 [  55.12384     58.72249     61.233887   ...   66.5266    65.32631     66.8613]
 [ -30.448849    -27.89038    -22.451464   ...  -19.271458   -23.097046   -18.843634]
 ...
 [  -6.342108    -9.058981    -9.702562   ...  -11.525118    -9.8483515   -6.803407]
 [  -2.7326827   -2.249867     1.7608168  ...    1.4896486    0.93037903   2.8084617]
 [ -13.274951    -11.341682    -6.6493864...   -2.445509    -5.0610194   -2.5841856]]
```

Result

Thus, the program has been written to extract Mel Frequency Cepstral Coefficients from the given input speech signal.

ADDITIONAL EXERCISES – IMAGE PROCESSING

Ex. No. 1 Huffman Encoding and Decoding

Date:

Aim

To write a program to encode and decode an image using Huffman algorithm to perform lossless compression.

Theoretical concept

Huffman encoding is a lossless data compression algorithm developed by David A. Huffman in 1952. It minimizes the total number of bits needed to represent a dataset by assigning variable-length codes to each symbol, with shorter codes assigned to more frequent symbols and longer codes to less frequent ones. This approach reduces the average number of bits used to store the data.

The process begins by calculating the frequency of each symbol in the dataset. Then, a binary tree (Huffman Tree) is built by merging the two least frequent symbols into a parent node, repeatedly combining nodes until only one remains. Each left branch of the tree is assigned a '0' and each right branch a '1', resulting in unique binary codes for each symbol. More frequent symbols are closer to the root of the tree, leading to shorter codes.

Huffman encoding is widely used in compression algorithms like ZIP, JPEG, and MP3. It ensures that no code is a prefix of another (prefix-free property), enabling unambiguous decoding. Although optimal for single-pass compression of a fixed set of symbols, Huffman encoding is not suitable for dynamic data compression, as it requires knowing symbol frequencies in advance.

Implentation

```
import cv2
import numpy as np
from collections import Counter
import heapq


# Function to build Huffman Tree
def build_huffman_tree(frequencies):
    heap = [[weight, [symbol, ""]] for symbol, weight in frequencies.items()]
```

```
    heapq.heapify(heap)

    while len(heap) > 1:
        lo = heapq.heappop(heap)
        hi = heapq.heappop(heap)
        for pair in lo[1:]:
            pair[1] = '0' + pair[1]
        for pair in hi[1:]:
            pair[1] = '1' + pair[1]
        heapq.heappush(heap, [lo[0] + hi[0]] + lo[1:] + hi[1:])

    return sorted(heapq.heappop(heap)[1:], key=lambda p: (len(p[-1]), p))

# Function to encode image using Huffman codes
def huffman_encode_image(image, huffman_code):
    encoded_image = ''.join([huffman_code[pixel] for pixel in image.flatten()])
    return encoded_image

# Function to decode encoded image
def huffman_decode_image(encoded_data, huffman_code, shape):
    reverse_code = {v: k for k, v in huffman_code.items()}
    decoded_image = []

    code = ''
    for bit in encoded_data:
        code += bit
        if code in reverse_code:
            decoded_image.append(reverse_code[code])
            code = ''

    return np.array(decoded_image).reshape(shape)
```

```
# Example usage
image_path = 'image.png'  # Replace with your image path
img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Step 1: Calculate frequency of each pixel value
freq = Counter(img.flatten())

# Step 2: Build Huffman Tree and codes
huffman_tree = build_huffman_tree(freq)
huffman_code = {item[0]: item[1] for item in huffman_tree}

# Step 3: Encode the image
encoded_image = huffman_encode_image(img, huffman_code)
print(f"Original image size: {img.size * 8} bits")
print(f"Encoded image size: {len(encoded_image)} bits")

# Step 4: Decode the image back
decoded_image = huffman_decode_image(encoded_image, huffman_code, img.shape)

# Show the original and decoded image
cv2.imshow("Original Image", img)
cv2.imshow("Decoded Image", decoded_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Sample Input and Output

Original image size: 8392704 bits

Encoded image size: 7200445 bits

Result

Thus a program to perform lossless compression using Huffman coding algorithm is implemented successfully.

Ex. No. 2 Segmentation using Watershed Algorithm

Date:

Aim

To write a program to implement segmentation for the given image using watershed algorithm in python.

Theoretical concept

The watershed algorithm is a popular image segmentation technique used to separate overlapping or touching objects within an image. It treats the image like a topographic surface, where pixel intensity values represent elevations. The algorithm floods basins (low-intensity areas) from the markers, which are predefined seed points. As the basins flood and meet, watershed lines are created to segment the image.

The segmentation of image using watershed algorithm takes the following steps,

1. Apply a threshold or edge detection to distinguish foreground objects.
2. Use distance transformation and thresholding to identify the sure foreground region.
3. Find unknown regions (possible background) and sure background.
4. Define markers for watershed.
5. Apply the watershed algorithm.
6. Boundaries between segments are marked.

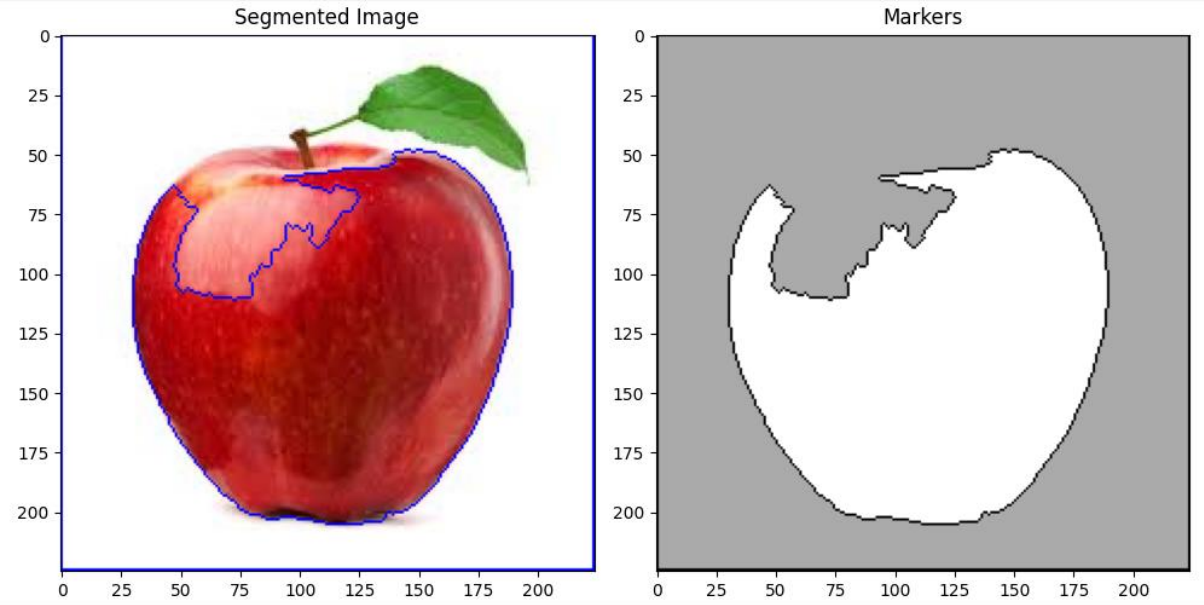
Implentation

```
import cv2

import numpy as np

from matplotlib import pyplot as plt


# Load the image

image_path = 'apple.jpeg'  # replace with your image path

img = cv2.imread(image_path)

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


# Step 1: Apply Otsu's thresholding to convert the image to binary

_, binary = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)


# Step 2: Remove noise using morphological operations (opening)
```

```
kernel = np.ones((3, 3), np.uint8)
opening = cv2.morphologyEx(binary, cv2.MORPH_OPEN, kernel, iterations=2)

# Step 3: Dilate the binary image to get sure background
sure_bg = cv2.dilate(opening, kernel, iterations=3)

# Step 4: Distance transform to find sure foreground regions
dist_transform = cv2.distanceTransform(opening, cv2.DIST_L2, 5)
_, sure_fg = cv2.threshold(dist_transform, 0.7 * dist_transform.max(), 255, 0)

# Step 5: Find unknown region by subtracting sure foreground from sure background
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg, sure_fg)

# Step 6: Label markers
_, markers = cv2.connectedComponents(sure_fg)

# Step 7: Add 1 to all markers to distinguish between background and other regions
markers = markers + 1

# Mark the region of unknown (background) with 0
markers[unknown == 255] = 0

# Step 8: Apply watershed algorithm
markers = cv2.watershed(img, markers)

# Step 9: Boundaries are marked with -1 in the marker image, so highlight the
boundaries
img[markers == -1] = [255, 0, 0]  # marking watershed boundaries with red color

# Display results
plt.figure(figsize=(10, 5))
```

```
plt.subplot(121), plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)), plt.title('Segmented Image')
plt.subplot(122), plt.imshow(markers, cmap='gray'), plt.title('Markers')
plt.tight_layout()
plt.show()
```

Sample Input/Output



Result

Thus a python program to segment the given image using the Watershed Algorithm has been implemented successfully.

ADDITIONAL EXERCISES – SPEECH PROCESSING

Ex. No. 1 Noise Detection Program

Date:

Aim

To write a program to detect noise by computing the Short-Time Fourier Transform (STFT), and detecting high-energy noise components in the frequency domain of the given speech signal.

Theoretical concept

Noise detection in audio processing involves identifying unwanted or irrelevant sounds within an audio signal, such as background hum, hiss, or random disturbances. These noises degrade the quality and intelligibility of the audio, making noise detection a crucial step in many applications like speech recognition, audio compression, and communication systems.

Noise detection generally relies on distinguishing between the desired signal and noise based on statistical, spectral, or temporal differences. Common techniques include:

1. **Spectral Analysis:** Noises often occupy specific frequency bands. By analyzing the frequency spectrum, high-energy spikes or noise patterns (e.g., white noise or hums) can be detected and separated from the signal.
2. **Thresholding:** Based on amplitude or power levels, parts of the audio signal below a certain threshold may be classified as noise.
3. **Machine Learning:** Advanced methods involve training models on labeled noisy and clean data to automatically identify noise.
4. **Time-Frequency Methods:** Short-Time Fourier Transform (STFT) or wavelet transform can be used to analyze the signal in both time and frequency domains, detecting noise across various timescales.

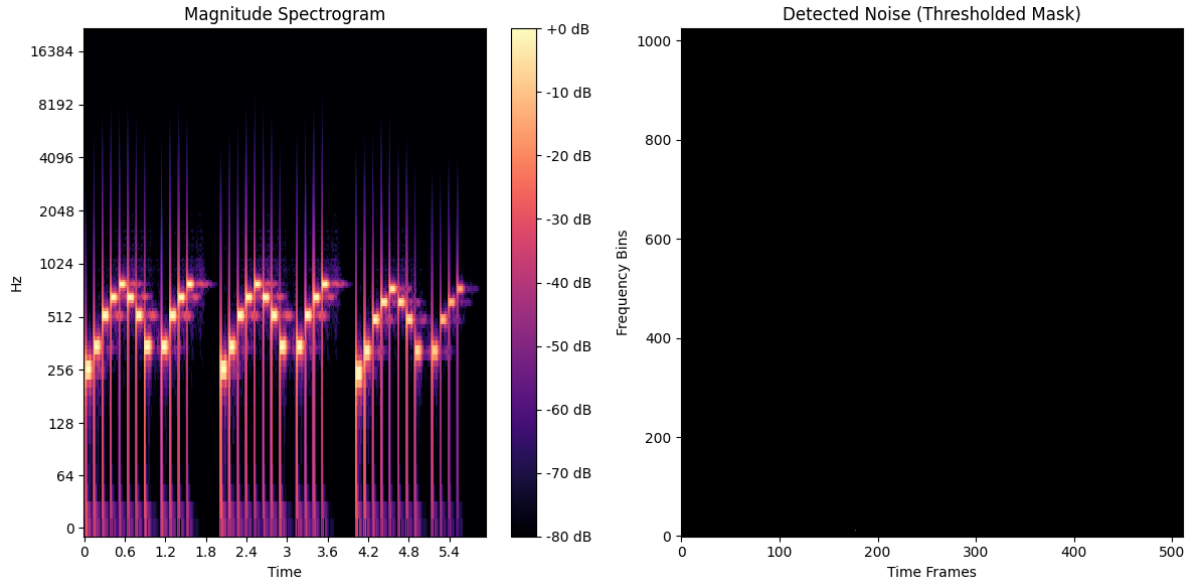
Noise detection is often followed by noise reduction techniques such as spectral subtraction or adaptive filtering to improve audio quality.

Implentation

```
import numpy as np

import librosa

import librosa.display

import matplotlib.pyplot as plt
```

```
audio_path = 'audio_file.wav'
noise_threshold = 0.5

audio, sr = librosa.load(audio_path, sr=None)
stft_audio = np.abs(librosa.stft(audio))
magnitude_spectrogram = librosa.amplitude_to_db(stft_audio, ref=np.max)
noise_mask = magnitude_spectrogram < (np.max(magnitude_spectrogram) *
noise_threshold)

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
librosa.display.specshow(magnitude_spectrogram, sr=sr, x_axis='time', y_axis='log')
plt.colorbar(format='%+2.0f dB')
plt.title('Magnitude Spectrogram')

plt.subplot(1, 2, 2)
plt.imshow(noise_mask, aspect='auto', cmap='gray_r', origin='lower')
plt.title('Detected Noise (Thresholded Mask)')
plt.xlabel('Time Frames')
plt.ylabel('Frequency Bins')

plt.tight_layout()
plt.show()
```

Sample Input/Output



Result

Thus a python program to detect noise in the given audio has been implemented and verified successfully.

Ex. No. 2 Voice Activity Detection

Date:

Aim

To detect and distinguish voice activity from silence or noise in an audio signal using energy-based methods.

Theoretical concept

Voice Activity Detection (VAD) is a technique used to identify the presence of speech in an audio signal. It helps distinguish between speech segments and non-speech segments such as silence, background noise, or music. VAD is essential in applications like speech recognition, telecommunication systems, and noise reduction, where processing only the speech parts improves efficiency and accuracy.

In energy-based VAD, the audio signal is divided into short overlapping frames. For each frame, the Short-Time Energy (STE) is computed by summing the squared amplitude of the samples within that frame. This energy reflects the intensity of the sound during that period. Speech typically has higher energy compared to silence or background noise.

A threshold is applied to the energy values to classify each frame. Frames with energy above the threshold are considered to contain speech, while frames below the threshold are classified as silence or noise. This approach is simple and effective for clean audio but may struggle in noisy environments where background noise energy can be comparable to speech.

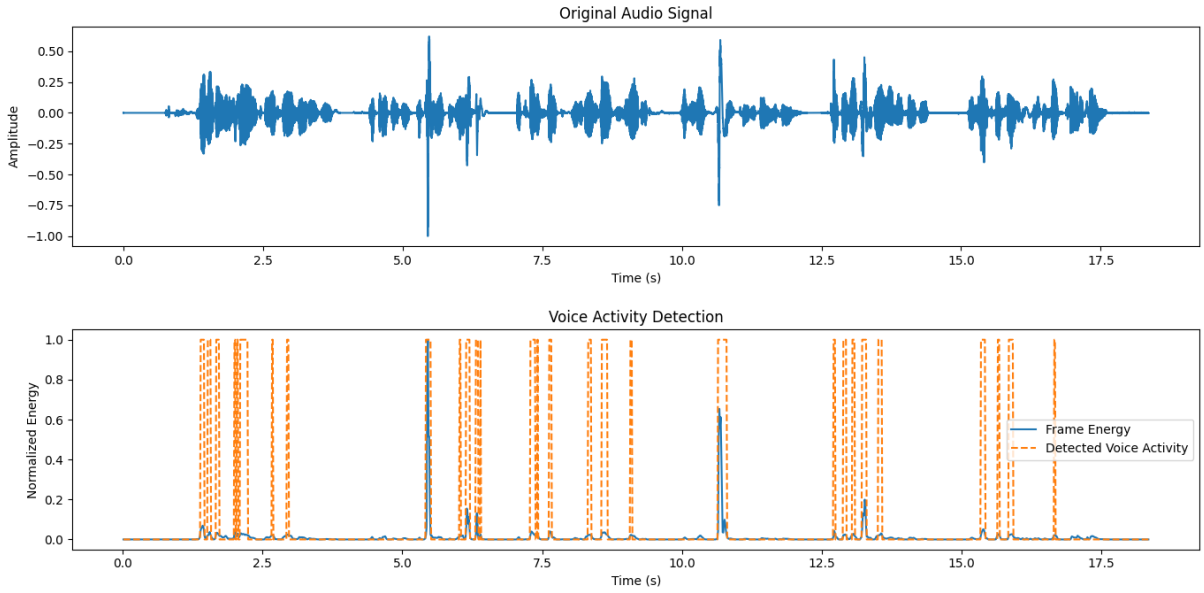
Advanced VAD systems may use spectral features, statistical models, or machine learning techniques to improve detection accuracy in more challenging audio environments, such as noisy or low-quality recordings.

Implentation

```
import numpy as np

import librosa

import matplotlib.pyplot as plt


audio_path = 'speech_sample.wav'

frame_length=1024

hop_length=512

energy_threshold=0.02
```

```
audio, sr = librosa.load(audio_path, sr=None)


frames = librosa.util.frame(audio, frame_length=frame_length,
hop_length=hop_length)
frame_energy = np.sum(frames**2, axis=0) / frame_length
frame_energy = frame_energy / np.max(frame_energy)
voice_activity = frame_energy > energy_threshold


time = np.arange(len(audio)) / sr
frame_time = librosa.frames_to_time(np.arange(len(frame_energy)), sr=sr,
hop_length=hop_length, n_fft=frame_length)


plt.figure(figsize=(14, 6))


plt.subplot(2, 1, 1)
plt.plot(time, audio, label="Audio Signal")
plt.title('Original Audio Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')


plt.subplot(2, 1, 2)
plt.plot(frame_time, frame_energy, label="Frame Energy")
plt.plot(frame_time, voice_activity * np.max(frame_energy), label="Detected Voice
Activity", linestyle='--')
plt.title('Voice Activity Detection')
plt.xlabel('Time (s)')
plt.ylabel('Normalized Energy')
plt.legend()


plt.tight_layout()
plt.show()
```

Sample Input/Output



Result

Thus a python program to implement voice activity detection has been implemented successfully.

ANNEXURE

Programme Outcomes

Sl. No.	Program Outcomes
PO1	**Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, andan engineering specialization to the solution of complex engineering problems.
PO2	**Problem Analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.
PO3	**Design/Development of Solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	**Conduct Investigations of Complex Problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	**Modern Tool Usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	**The Engineer and Society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	**Environment and Sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	**Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and Team Work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10	**Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	**Project Management and Finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	**Life-long Learning:** Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change.