PRACTICAL FILE

Machine Vision and Data Understanding

(ITMDAE10)

Name – Kurakula James Paul
Roll No – 2020PMD4211
Course – M. Tech (Machine Intelligence and Data Analytics)

under the guidance of

MR. VIKAS MAHESHKAR

Department of Information Technology

Index

Sl No.	Date	Topic	Remarks
1	7/10/2021	Perform low level and mid-level image processing using open-cv	
2	14/10/2021	Demonstrate high level operation on any machine vison application	
3	21/10/2021	Implement an application for distance between two pixels using path concept.	
4	28/10/2021	Design component labelling algorithm to count number of components and implement an application on it.	
5	29/11/2021	Write a program to find line, edge and point in an image and design an application for the same.	
6	29/11/2021	Write a program to perform region-based segmentation and design an application for the same.	

Experiment1 - What do you understand by Machine Vision and study the basic image processing functionalities of the OpenCV library using low and mid level operations on image.

Machine Vision

Machine Vision is a process by which we can understand the images and videos, how they are stored and how we can manipulate and retrieve data from them. Computer Vision is the base or mostly used for Artificial Intelligence.Computer-Vision is playing a major role in self-driving cars, robotics as well as in photo correction apps.

OpenCV

OpenCV is a cross-platform library using which we can develop real-time computer vision applications. It mainly focuses on image processing, video capture, and analysis including features like face detection and object detection. In this tutorial, we explain how you can use OpenCV in your applications.

Application of openCV

There are lots of applications that are solved using OpenCV, some of them are listed below -

- Image Processing
- Face detection
- Face recognition
- Automated inspection and surveillance
- Anomaly(defect) detection in the manufacturing function
- Image/Video search and retrieval

Low Level OpenCV functions -

Changing Color Space :

Theory -

OpenCV-Python is a library of Python bindings designed to solve computer vision problems. cv2.cvtColor() method is used to convert an image from one color space to another. There are more than 150 color-space conversion methods available in OpenCV.

Syntax: cv2.cvtColor(src, code[, dst[, dstCn]])

Parameters:
src: It is the image whose color space is to be changed.
code: It is the color space conversion code.
dst: It is the output image of the same size and depth as src image. It is an optional parameter.
dstCn: It is the number of channels in the destination image. If the parameter is 0 then the number of the channels is derived automatically from src and code. It is an optional parameter.

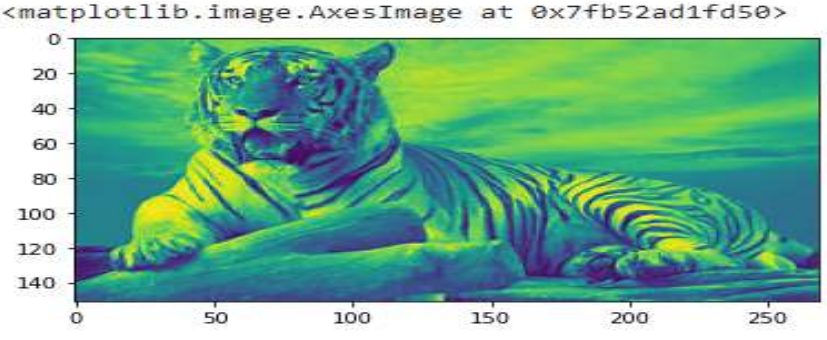
Return Value: It returns an image.

Example1 -

```
#converting image to Gray scale
gray_image =
cv2.cvtColor(cv2.imread(sample_image_path),cv2.COLOR_BGR2GRAY)
#plotting the grayscale image
plt.imshow(gray_image)
```

Output1 -

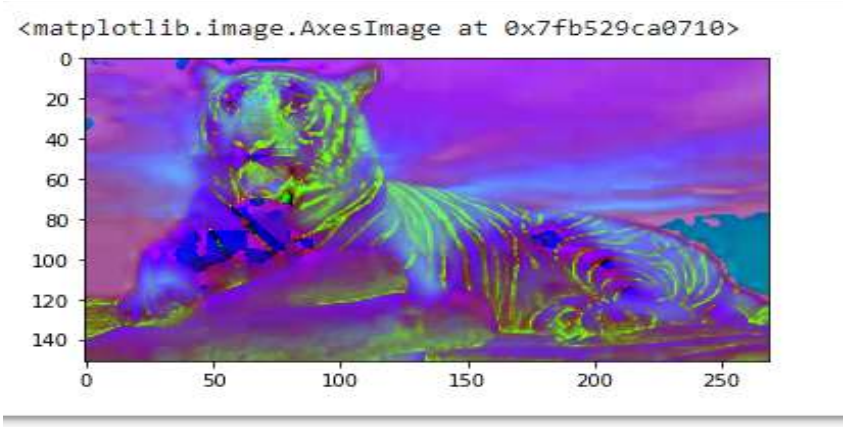
```
<matplotlib.image.AxesImage at 0x7fb52ad1fd50>
```



Example2 -

```
#converting image to HSV format
hsv_image = cv2.cvtColor(image,cv2.COLOR_BGR2HSV)
plt.imshow(hsv_image)
```

Output2 -



Uses- Image segmentation

Resizing image :

Theory -

Images can be easily scaled up and down using OpenCV.Different interpolation and downsampling methods are supported by OpenCV, which can be used by the following parameters:

- INTER_NEAREST: Nearest neighbor interpolation
- INTER_LINEAR: Bilinear interpolation
- INTER_AREA: Resampling using pixel area relation
- INTER_CUBIC: Bicubic interpolation over 4×4 pixel neighborhood
- INTER_LANCZOS4: Lanczos interpolation over 8×8 neighborhood

We can interpret an operation of resizing using a linear algebra framework.Also, another name that we use is scaling. A scaling matrix can be the following matrix the has no zero elements on the main diagonal.

$$M = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix}\vec{y}\\1\end{bmatrix} = M\begin{bmatrix}\vec{x}\\1\end{bmatrix} \quad M = \begin{bmatrix}2 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix} \quad \begin{bmatrix}\vec{x}\\1\end{bmatrix} = \begin{bmatrix}x\\y\\1\end{bmatrix}$$

$$\begin{bmatrix}2 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix} \cdot \begin{bmatrix}3\\2\\1\end{bmatrix} = \begin{bmatrix}6+0+0\\0+2+0\\0+0+1\end{bmatrix} = \begin{bmatrix}6\\2\\1\end{bmatrix}$$

Syntax: cv2.resize(src, size, fx, fy, interpolation)

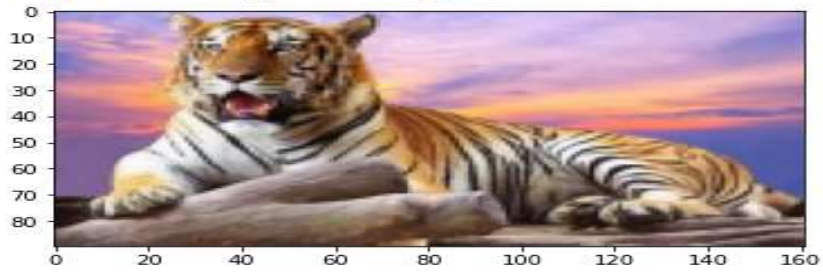
Parameters:
src: The path of the input image.
size: The required size for the output image is given as a tuple of width and height
fx:(optional) The scaling factor for the horizontal axis.
fy: (optional) The scaling factor for the vertical axis.
interpolation:It refers to the algorithm used for scaling

Example1 -

```
#downscale
scale = 60   # percent of original size
width = int(image.shape[1] * scale / 100)
height = int(image.shape[0] * scale / 100)
dim = (width, height)
smaller_image = cv2.resize(image,dim,interpolation=cv2.INTER_AREA)
plt.imshow(smaller_image)
```

Output1 -

```
<matplotlib.image.AxesImage at 0x7fb529c14490>
```



Example2 -

```
#upscale
scale = 200   # percent of original size
width = int(image.shape[1] * scale / 100)
height = int(image.shape[0] * scale / 100)
```

```
dim = (width, height)
larger_image = cv2.resize(image,dim,interpolation=cv2.INTER_AREA)
#plot the resized image
plt.imshow(larger_image)
```

Output2 -

Uses- This operation is useful for training deep learning models when we need to convert images to the model's input shape. Zooming the image.

Image Rotation :

Theory -

Images can be rotated to any degree clockwise or otherwise. We just need to define the rotation matrix listing rotation point, degree of rotation, and the scaling factor. First, we need an angle θ that will represent how many degrees we are rotating our image and coordinate at which we want to rotate.

$$M = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \vec{y} \\ 1 \end{bmatrix} = M \begin{bmatrix} \vec{x} \\ 1 \end{bmatrix} \quad M = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} \vec{x} \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} \cos 90 & -\sin 90 & 0 \\ \sin 90 & \cos 90 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0-2+0 \\ 3+0+0 \\ 0+0+1 \end{bmatrix} = \begin{bmatrix} -2 \\ 3 \\ 1 \end{bmatrix}$$

Syntax: cv2.warpAffine(src, M, dsize, dst, flags, borderMode, borderValue)

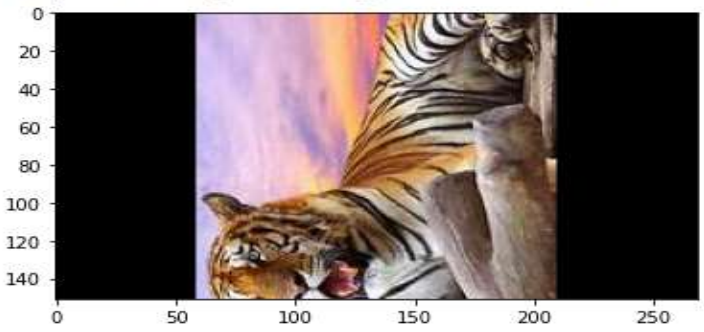
Parameters:
src: input image.
dst: output image that has the size dsize and the same type as src.
M: transformation matrix.
dsize: the size of the output image.
flags: a combination of interpolation methods and the optional flag WARP_INVERSE_MAP that means that M is the inverse transformation (dst->src).
borderMode: pixel extrapolation method; when borderMode=BORDER_TRANSPARENT, it means that the pixels in the destination image corresponding to the “outliers” in the source image are not modified by the function.
borderValue: value used in case of a constant border; by default, it is 0.

Example1 -

```
rows,cols = image.shape[:2]
#(col/2,rows/2) is the center of rotation for the image
# M is the cordinates of the center
M = cv2.getRotationMatrix2D((cols/2,rows/2),90,1)
dst = cv2.warpAffine(image,M,(cols,rows))
plt.imshow(dst)
```

Output1 -

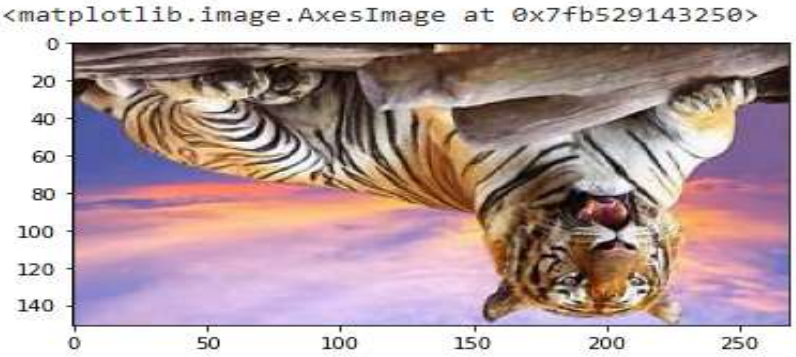
```
<matplotlib.image.AxesImage at 0x7fb5291cbd10>
```



Example2 -

```
rows,cols = image.shape[:2]
#(col/2,rows/2) is the center of rotation for the image
M = cv2.getRotationMatrix2D((cols/2,rows/2),180,1)
dst = cv2.warpAffine(image,M,(cols,rows))
plt.imshow(dst
```

Output2 -



Uses- Generating Data samples, Data augmentation technique

Drawing Function :

Theory -

Images can be rotated to any degree clockwise or otherwise. We just need to define the rotation matrix listing rotation point, degree of rotation, and the scaling factor.

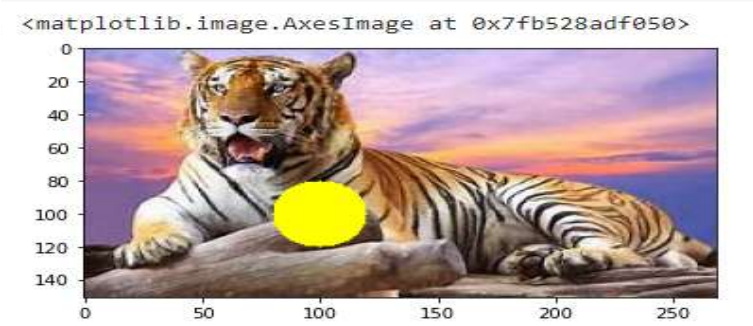
Syntax: cv2.circle(image, center_coordinates, radius, color, thickness)

Parameters:
image: It is the image on which a circle is to be drawn.
center_coordinates: It is the center coordinates of a circle. The coordinates are represented as tuples of two values i.e. (X coordinate value, Y coordinate value).
radius: It is the radius of a circle.
color: It is the color of the border line of a circle to be drawn. For BGR, we pass a tuple. eg: (255, 0, 0) for blue color.
thickness: It is the thickness of the circle border line in px. Thickness of -1 px will fill the circle shape by the specified color.
Return Value: It returns an image.

Example -

```
#circle
circle=cv2.circle(cv2.cvtColor(cv2.imread(sample_image_path),cv2.COL
OR_BGR2RGB),(100,100), 20, (255,255,0), -1)
plt.imshow(circle)
```

Output -



Uses- To highlight the area of interest of an image, To identify the particular region in the image

Blurring the image :

Theory -

Image Blurring refers to making the image less clear or distinct. It is done with the help of various low pass filter kernels.

$$G(x,y) = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$$

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Syntax: cv2.GaussianBlur(src, ksize, sigmaX[,dst[,sigmaY[borderType=BORDER_DEFAULT]]])

Parameters:
src: input image.
ksize: Gaussian Kernel Size.
sigmaX: Kernel standard deviation along X-axis.
dst: output image.

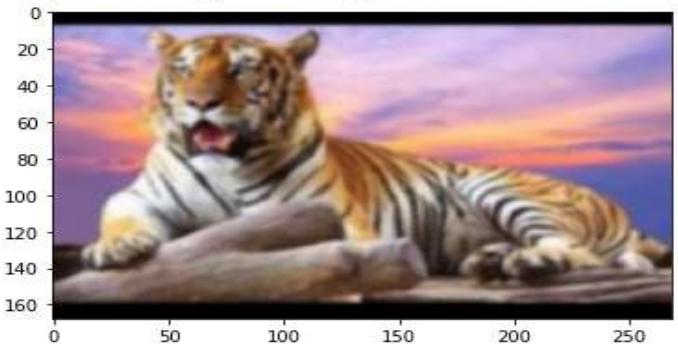
sigmaY: Kernel standard deviation along Y-axis (vertical direction). If sigmaY=0, then sigmaX value is taken for sigmaY.
borderType: Specifies image boundaries while the kernel is applied on image borders.

Example -

```
image  = cv2.imread(sample_image_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
# Gaussian Blur
gusseian = cv2.GaussianBlur(image, (5,5), 0)
plt.imshow(gusseian)
```

Output -

```
<matplotlib.image.AxesImage at 0x7f240cdd74d0>
```



Uses- Noise removal, Smoothing of image

Arithmetic operation on images :

Theory -

Arithmetic Operations like Addition, Subtraction, and Bitwise Operations(AND, OR, NOT, XOR) can be applied to the input images.

Syntax: cv2.addWeighted(img1, wt1, img2, wt2, gammaValue)

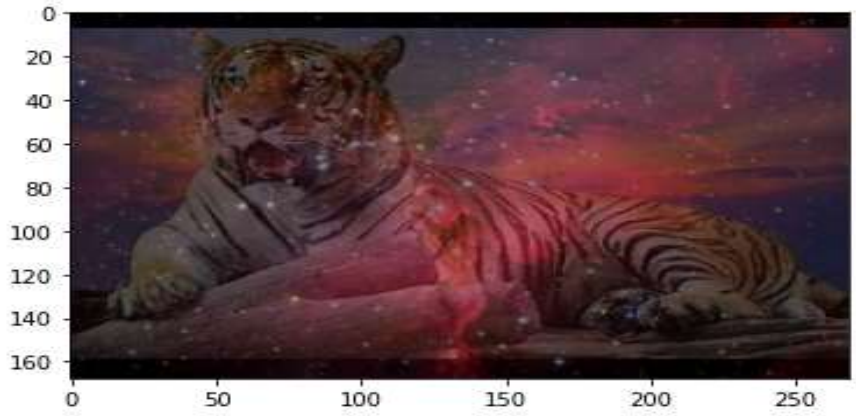
Parameters:
img1: First Input Image array.
wt1: Weight of the first input image elements to be applied to the final image.
img2: Second Input Image array.
wt2: Weight of the second input image elements to be applied to the final image.
gammaValue: Measurement of light.

Example1 -

```
# addition of two images
image1 = cv2.imread(sample_image_path)
image1 = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)
image2 = cv2.imread('/content/gdrive/MyDrive/Machine
Vision/space.jpg')
image2 = cv2.cvtColor(image2, cv2.COLOR_BGR2RGB)
weightedSum = cv2.addWeighted(image1, 0.3, image2, 0.5, 0)
plt.imshow(image1)
```

Output1 -

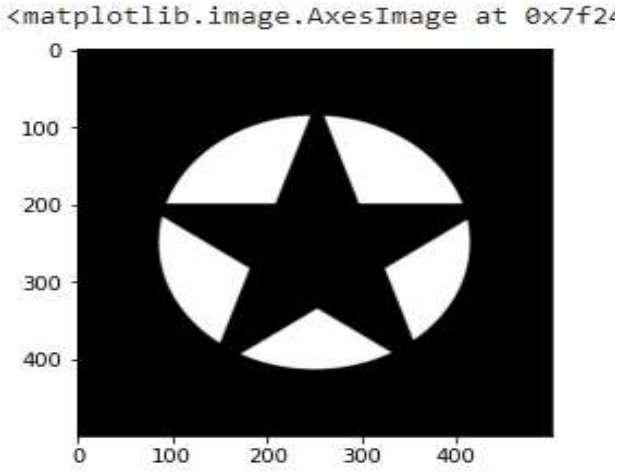
```
<matplotlib.image.AxesImage at 0x7f240d43d290>
```



Example2 -

```
image3 = cv2.imread('/content/gdrive/MyDrive/Machine
Vision/star.jpg')image3 = cv2.cvtColor(image3, cv2.COLOR_BGR2RGB)
image4 = cv2.imread('/content/gdrive/MyDrive/Machine
Vision/circle.jpg')image4 = cv2.cvtColor(image4, cv2.COLOR_BGR2RGB)
# Subtraction of two images
subractedImage = cv2.subtract(image4, image3)
plt.imshow(subractedImage)
```

Output2 -



Uses- To enhance any image.

Mid Level OpenCV functions -

Image Thresholding :

Theory -

Thresholding is a technique in OpenCV, which is the assignment of pixel values in relation to the threshold value provided. In thresholding, each pixel value is compared with the threshold value. If the pixel value is smaller than the threshold, it is set to 0, otherwise, it is set to a maximum value (generally 255).

Thresholding is a very popular segmentation technique, used for separating an object considered as a foreground from its background. A threshold is a value which has two regions on its either side i.e. below the threshold or above the threshold

Syntax: cv2.threshold(source, thresholdValue, maxVal, thresholdingTechnique)

Parameters:
source: Input Image array (must be in Grayscale).
thresholdValue: Value of Threshold below and above which pixel values will change accordingly.
maxVal: Maximum value that can be assigned to a pixel.
Thresholding techniquethe : The type of thresholding to be applied.

Example -

```
gray_image = cv2.imread(sample_image_path,0)
ret,thresh_binary =
cv2.threshold(gray_image,127,255,cv2.THRESH_BINARY)
ret,thresh_binary_inv =
cv2.threshold(gray_image,127,255,cv2.THRESH_BINARY_INV)
ret,thresh_trunc =
cv2.threshold(gray_image,127,255,cv2.THRESH_TRUNC)
ret,thresh_tozero =
cv2.threshold(gray_image,127,255,cv2.THRESH_TOZERO)
ret,thresh_tozero_inv =
cv2.threshold(gray_image,127,255,cv2.THRESH_TOZERO_INV)
names = ['Oiriginal
Image','BINARY','THRESH_BINARY_INV','THRESH_TRUNC','THRESH_TOZERO','
THRESH_TOZERO_INV']
images =
gray_image,thresh_binary,thresh_binary_inv,thresh_trunc,thresh_tozer
o,thresh_tozero_inv
for i in range(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
    plt.title(names[i])plt.xticks([]),plt.yticks([])
plt.show()
```

Output -

Uses- Image segmentation

Histogram Equalization :

Theory -

Histogram equalization is a method in image processing of contrast adjustment using the image's histogram. This method usually increases the global contrast of many images, especially when the usable data of the image is represented by close contrast values.

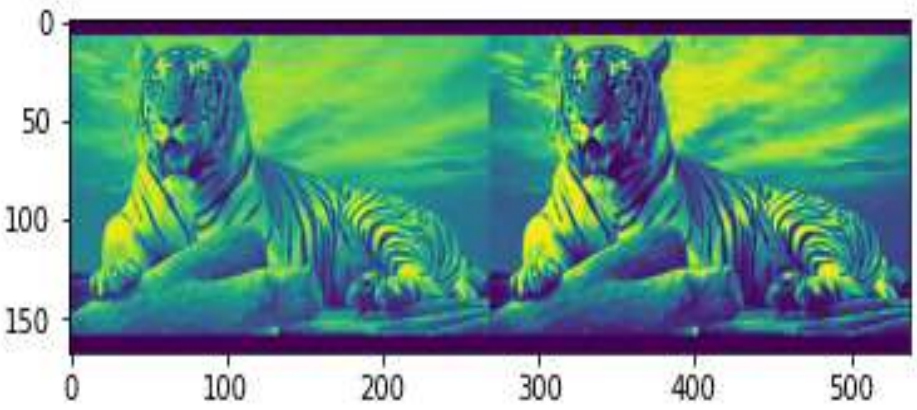
Syntax: cv2.equalizeHist(image)

Parameters:
image: An object representing the source (input image) for this operation.

Example -

```
img = cv2.imread(sample_image_path, 0)
equ = cv2.equalizeHist(img)
# stacking images side-by-side
res = np.hstack((img, equ))
plt.imshow(res)
```

Output -



Uses- Feature extraction.

Edge Detection :

Theory -

The process of image detection involves detecting sharp edges in the image.

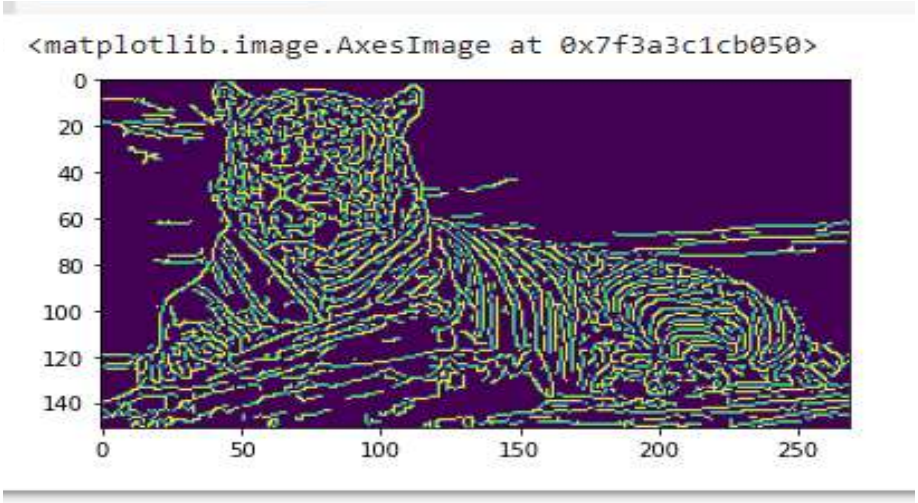
Syntax: cv2.Canny(image, edges, threshold1, threshold2)

Parameters:
image: A Mat object representing the source (input image) for this operation.
edge: A Mat object representing the destination (edges) for this operation
threshold1: A variable of the type double representing the first threshold for the hysteresis procedure.
threshold2: A variable of the type double representing the second threshold for the hysteresis procedure.

Example -

```
image = cv2.imread(sample_image_path)
#calculate the edges using Canny edge algorithm
edges = cv2.Canny(image,100,200)
plt.imshow(edges)
```

Output -



Uses- Image segmentation, Image sharpening, Feature extraction

Image Eroding:

Theory -

cv2.erode() method is used to perform erosion on the image. It is normally performed on binary images.

Syntax: cv2.erode(src, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]])

Parameters:
src: It is the image that is to be eroded.
kernel: A structuring element used for erosion. If element = Mat(), a 3 x 3 rectangular structuring element is used. Kernel can be created using getStructuringElement.
dst: It is the output image of the same size and type as src.
anchor: It is a variable of type integer representing anchor point and its default value Point is (-1, -1) which means that the anchor is at the kernel center.
borderType: It depicts what kind of border to be added. It is defined by flags like cv2.BORDER_CONSTANT, cv2.BORDER_REFLECT, etc.
iterations: It is the number of times erosion is applied.
borderValue: It is border value in case of a constant border.

Example -

```
image  = cv2.imread(sample_image_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
kernel = np.ones((5, 5), np.uint8)
image = cv2.erode(image, kernel)
plt.imshow(image)
```

Output -

```
<matplotlib.image.AxesImage at 0x7f240ccf6e90>
```

Uses- Generating Data samples, Data augmentation technique

Corner Detection :

Theory -

There are various methods available for detection of corners in an image.cv2.goodFeaturesToTrack() method finds N strongest corners in the image by Shi-Tomasi method. Note that the image should be a grayscale image..

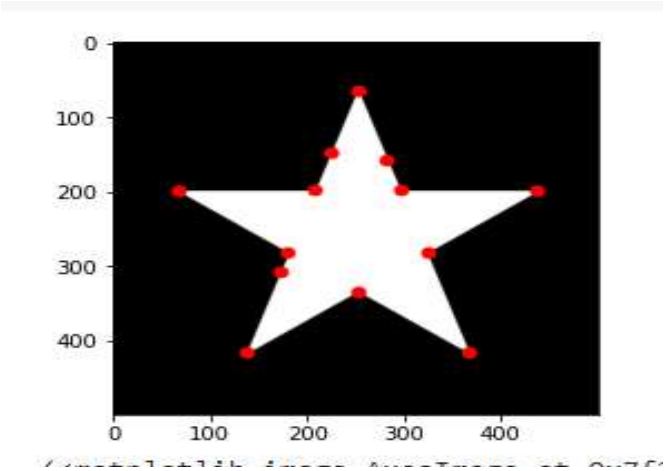
Syntax: cv2.goodFeaturesToTrack(image, maxCorners, qualityLevel, minDistance[, corners[, mask[, blockSize[, useHarrisDetector[, k]]]]])

Parameters:
image: input image.
maxCorners: Maximum number of corners
qualityLevel: Parameter characterizing the minimal accepted quality of image corners
minDistance: Minimum possible Euclidean distance between the returned corners
mask: Optional region of interest..
blockSize: Size of an average block for computing a derivative covariation matrix over each pixel neighborhood.
useHarrisDetector: Parameter indicating whether to use a Harris detector
K : Free parameter of the Harris detector.

Example -

```
img = cv2.imread('/content/gdrive/MyDrive/Machine Vision/star.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
corners = cv2.goodFeaturesToTrack(gray, 27, 0.01, 10)
corners = np.int0(corners)
for i in corners:
    x, y = i.ravel()
    cv2.circle(img, (x, y), 8, 255, -1)
plt.imshow(img), plt.show()
```

Output -



Uses- Object detection

Experiment 2 : Write a program to implement a real life application of Machine Vision.

In this project, we are going to build a deep learning model that will detect, from the X-ray tests, of a person whether they are corona positive. COVID-19 is an infectious disease caused by the coronavirus discovered in 2019.

This project aims to increase the accuracy of the detection of this virus using sophisticated deep learning.

Dataset:

We have taken the dataset from a variety of sources. The first one is taken from the corona positive X-rays and the second one is taken from the normal X-rays. The corona dataset is being updated regularly, and, at the time of this course being written, there were around 140+ corona positive images and around 140+ normal images of X-rays.

You need to combine your dataset from these two sources:

Corona positive X-Rays
Normal X-Rays

Once you have downloaded the dataset, we will jump directly into importing our modules, which is required to complete this project.

Explanation:

1.We will use keras.preprocessing to work with images.

2.We will use keras.layers to create different layers, which we have discussed in previous lessons.

3.We will use keras.models to create a Sequential model, which takes a single input and generates a single output, i.e., no branches of the nodes are created. This is the model that will be used in almost all applications.

4.We use ImageDataGenerator, which will generate batches of tensor image data with real-time data augmentation. The data will be looped over (in batches). This will help prevent over - fitting as we have a very small dataset.

5.The parameters that we passed are:

rescale - used to rescale the data values.

shear_range - specifies the shear angle counter-clockwise in degrees.

zoom_range - specifies the range of zoom for an image.

horizontal_flip - a boolean value which tells whether to flip the image horizontally or not.

6.You can see that, by using these parameters, we generate some variations of the images which will help the model to correctly classify the images, irrespective of the brightness, view angle, etc.

7.Now that we have our ImageDataGenerator object ready, we can go ahead and load our training and validation data using this object.

Model architecture:

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3), activation="relu",input_shape=(224,224,3)))
model.add(Conv2D(64, kernel_size=(3,3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, kernel_size=(3,3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(64, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(1, activation="sigmoid"))

model.compile(loss="binary_crossentropy", optimizer="adam",metrics = ["accuracy"])
model.summary()
```

Code:

```
from tensorflow.python.keras.preprocessing import image
from tensorflow.python.keras.layers import *
from tensorflow.python.keras.models import *
print('Imported Successfully!')
import glob
from matplotlib.pyplot import *
import matplotlib.pyplot as plt
```

```
Imported Successfully!
```

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

```
!nvidia-smi
```

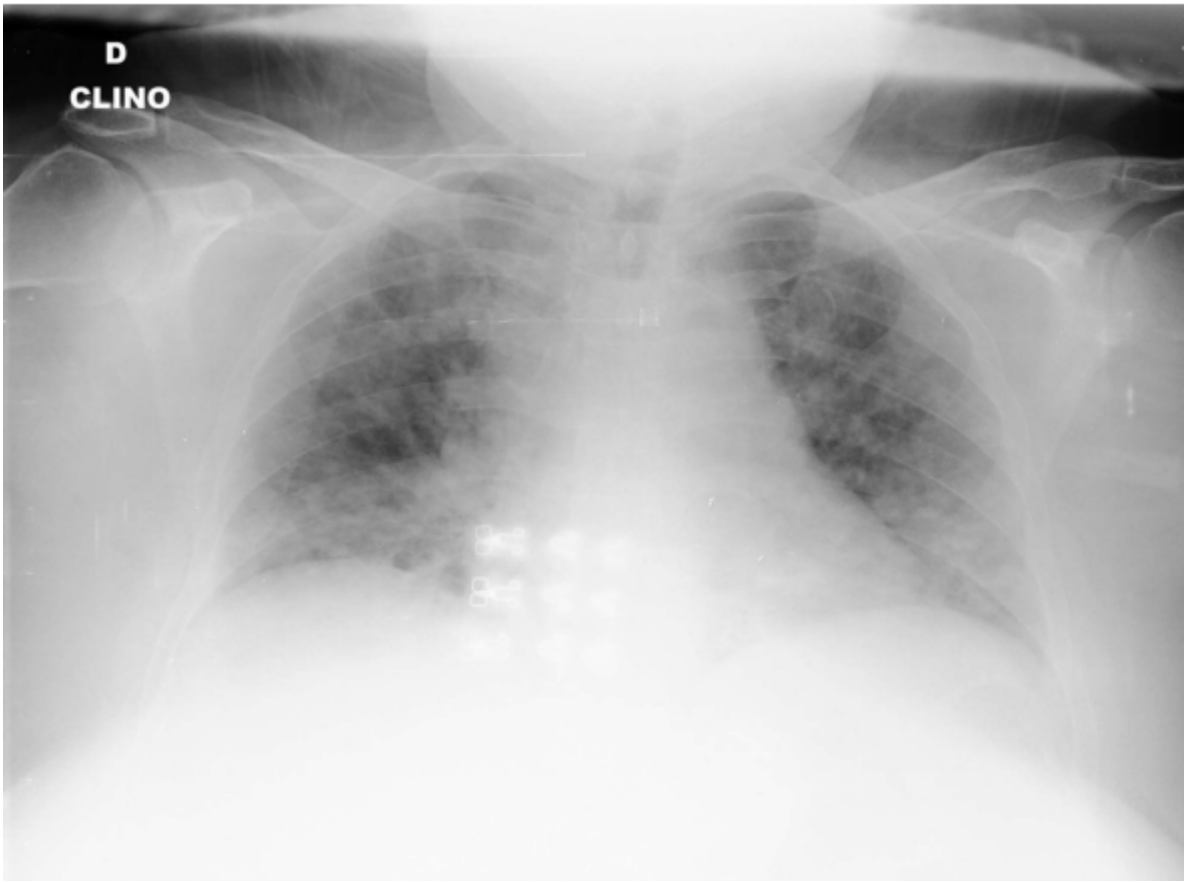
```
Mon Oct 18 02:09:41 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.74       Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   68C    P8    32W / 149W |      0MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|
```

```
train_path='/content/drive/MyDrive/CovidDataset/Train'
test_path='/content/drive/MyDrive/CovidDataset/Val'
```

```
import cv2
covid = cv2.imread("/content/drive/MyDrive/CovidDataset/Train/Covid/01E392EE-69F9-4E33-
covid
```

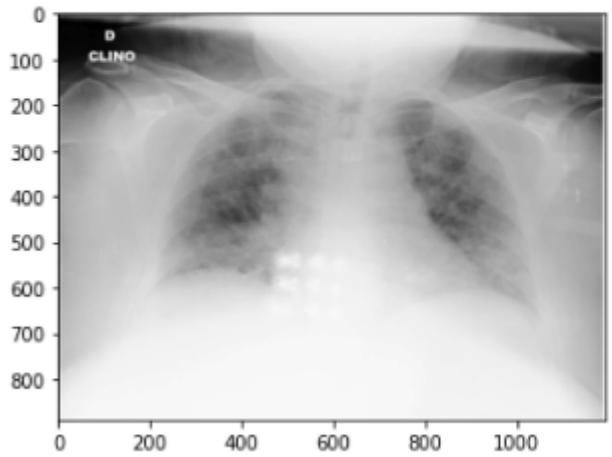
```
array([[[182, 182, 182],
        [169, 169, 169],
        [152, 152, 152],
        ...,
        [254, 254, 254],
        [254, 254, 254],
        [254, 254, 254]],

       [[165, 165, 165],
        [150, 150, 150],
        [138, 138, 138],
        ...,
        [254, 254, 254],
        [254, 254, 254],
        [254, 254, 254]],

       [[146, 146, 146],
        [134, 134, 134],
        [128, 128, 128],
        ...,
        [254, 254, 254],
        [254, 254, 254],
        [254, 254, 254]],

       ...,

       [[243, 243, 243],
        [244, 244, 244],
        [245, 245, 245],
        ...,
        [254, 254, 254],
        [254, 254, 254]
```

```
from IPython.display import Image
Image('/content/drive/MyDrive/CovidDataset/Train/Covid/01E392EE-69F9-4E33-BFCE-E5C96865
```




```
scale = 60 # percent of original size
width = int(covid.shape[1] * scale / 100)
height = int(covid.shape[0] * scale / 100)
dim = (width, height)
smaller_image = cv2.resize(covid,dim,interpolation=cv2.INTER_AREA)
plt.imshow(smaller_image)
```

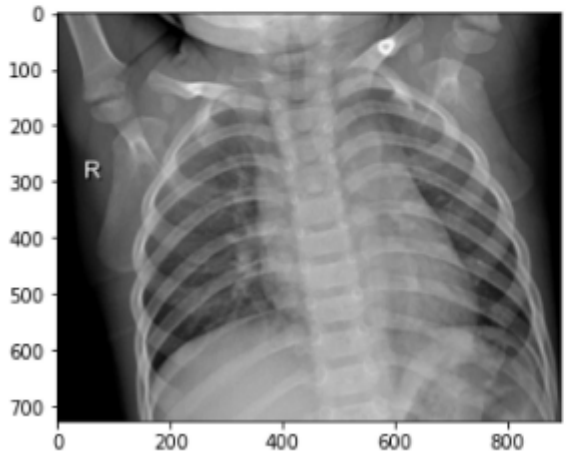
```
<matplotlib.image.AxesImage at 0x7f47f77c2e50>
```



```
non_covid=cv2.imread("/content/drive/MyDrive/CovidDataset/Train/Normal/IM-0156-0001.jpe
```

```
scale = 60 # percent of original size
width = int(non_covid.shape[1] * scale / 100)
height = int(non_covid.shape[0] * scale / 100)
dim = (width, height)
smaller_image = cv2.resize(non_covid,dim,interpolation=cv2.INTER_AREA)
plt.imshow(smaller_image)
```

```
<matplotlib.image.AxesImage at 0x7f47eb91e190>
```




```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3), activation="relu",input_shape=(224,224,3)))
model.add(Conv2D(64, kernel_size=(3,3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, kernel_size=(3,3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(64, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(1, activation="sigmoid"))

model.compile(loss="binary_crossentropy", optimizer="adam",metrics = ["accuracy"])
model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 222, 222, 32)	896
conv2d_1 (Conv2D)	(None, 220, 220, 64)	18496
max_pooling2d (MaxPooling2D)	(None, 110, 110, 64)	0
dropout (Dropout)	(None, 110, 110, 64)	0
conv2d_2 (Conv2D)	(None, 108, 108, 128)	73856

```
folders=glob.glob('/content/drive/MyDrive/CovidDataset/Train/*')
```

```
folders
```

```
['/content/drive/MyDrive/CovidDataset/Train/Normal',
 '/content/drive/MyDrive/CovidDataset/Train/Covid']
```

```
train_datagen = image.ImageDataGenerator(rescale=1./255, shear_range=0.2, zoom_range=0.
test_datagen = image.ImageDataGenerator(rescale = 1./255)

print('Created the Data Generator Objects.')
```

```
Created the Data Generator Objects.
```

```
train_generator = train_datagen.flow_from_directory('/content/drive/MyDrive/CovidDatase
```

```
Found 224 images belonging to 2 classes.
```

```
val_generator = test_datagen.flow_from_directory('/content/drive/MyDrive/CovidDataset/V
```

```
Found 60 images belonging to 2 classes.
```

```
history = model.fit(train_generator, epochs = 6, validation_data=val_generator, validat
```

```
Epoch 1/6
7/7 [==============================] - 86s 8s/step - loss: 5.5750 - accuracy: 0.6161 - v
al_loss: 0.6764 - val_accuracy: 0.5000
Epoch 2/6
7/7 [==============================] - 11s 2s/step - loss: 0.6452 - accuracy: 0.6116 - v
al_loss: 0.6699 - val_accuracy: 0.7833
Epoch 3/6
7/7 [==============================] - 11s 2s/step - loss: 0.4605 - accuracy: 0.8080 - v
al_loss: 0.3164 - val_accuracy: 0.9333
Epoch 4/6
7/7 [==============================] - 11s 2s/step - loss: 0.3931 - accuracy: 0.8616 - v
al_loss: 0.2393 - val_accuracy: 0.9333
Epoch 5/6
7/7 [==============================] - 11s 2s/step - loss: 0.4171 - accuracy: 0.8348 - v
al_loss: 0.4755 - val_accuracy: 0.9667
Epoch 6/6
7/7 [==============================] - 12s 2s/step - loss: 0.2851 - accuracy: 0.8750 - v
al_loss: 0.1523 - val_accuracy: 0.9500
```

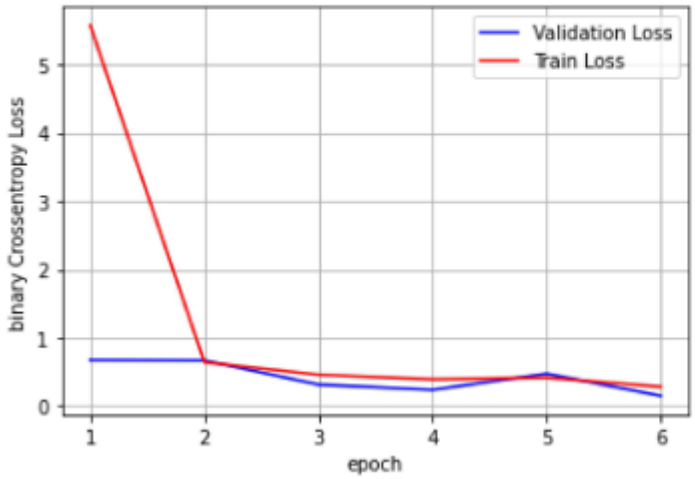
```
model_x = load_model('my_model.h5')
```

```
nb_epoch = 6
```

```
def plt_dynamic(x, vy, ty, ax, colors=['b']):
    ax.plot(x, vy, 'b', label="Validation Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

```
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('binary Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of ep

vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```



Output:

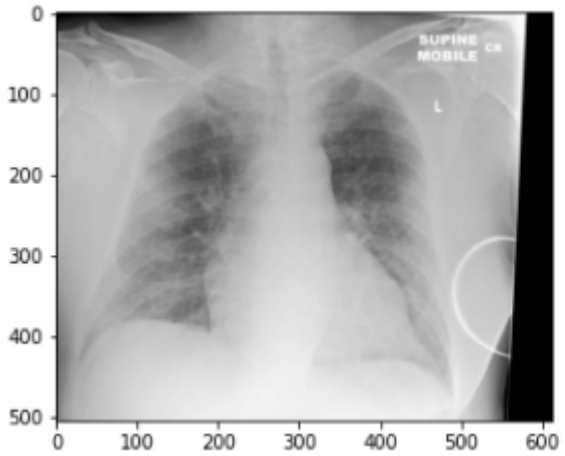
```
img = cv2.imread('/content/drive/MyDrive/xray.jpeg')
scale = 60 # percent of original size
width = int(img.shape[1] * scale / 100)
height = int(img.shape[0] * scale / 100)
dim = (width, height)
```

ER/Downloads/Covid_19_Detection_using_X_rays (1).html

Covid_19_Detection_using_X_rays (1)

```
smaller_image = cv2.resize(img,dim,interpolation=cv2.INTER_AREA)
plt.imshow(smaller_image)
```

```
<matplotlib.image.AxesImage at 0x7f46fcbbda50>
```




```
 x_test1=[]
 x_test1.append(img)
  x_test1=np.array(x_test1)
  x_test1=x_test1/255.0
  x_test1=x_test1.reshape(-1,224,224,1)
predicted_label = model_x.predict(x_test1)
if(predicted_label==1)
    print("Covid Patient")
else:
    print("Non Covid Patient")
```

```
Covid Patient
```

Experiment 3:

Consider one application and caliculate the distance between two pixels

Here we are considering a length of Spectacles in an image

1. Uploading the image:

In [3]:
```
from google.colab import files
files=files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving tst.jpg to tst.jpg
```

In this the files are uploaded to the local runtime environment of google colaboratory.These files are locally saved into the environment till the runtime and when the runtime disconnects the user need to upload the files again.

2.Importing the libraries

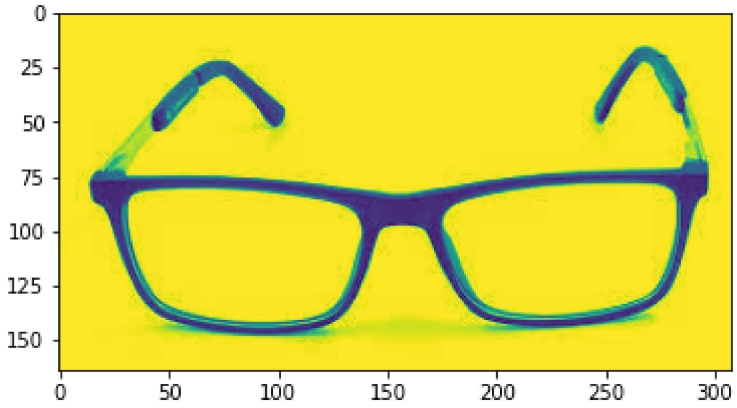
In [1]:
```
import cv2
import matplotlib.pyplot as plt
import numpy as np
```

Calculating the length needs cv2,matplotlib and numpy as the necessary libraries to be imported.Cv2 is used to perform all the image processing,Matplot is used for plotting the graphs and mentioning the measurement of the object in the image.All the pixels are stored in an numpy array of the objects of the image so numpy is imported as np

3.Reading the image:

In [4]:
```
img=cv2.imread('tst.jpg',0)
plt.imshow(img)
```

Out[4]:
```
<matplotlib.image.AxesImage at 0x7ff31bc08550>
```

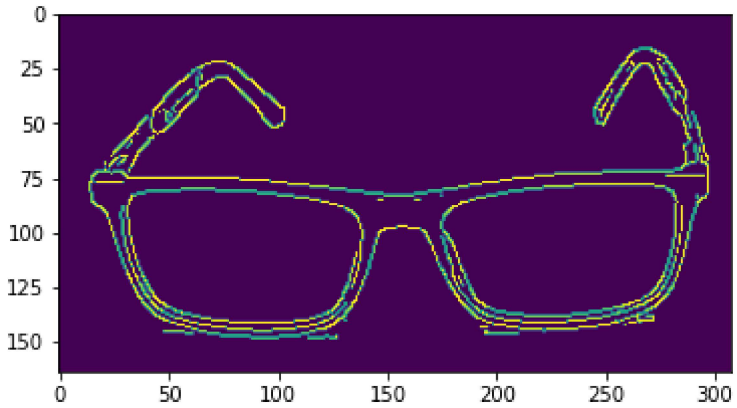


The image uploaded in the runtime in google colab is only for the runtime.To access the image in the program we must read the uploaded image using opencv.imread method in open cv is used to read the image in the google colab for program usage.

4.Edge detection:

In [5]:
```
edges = cv2.Canny(img,100,200)
plt.imshow(edges)
```

Out[5]:
```
<matplotlib.image.AxesImage at 0x7ff31bb92050>
```



For measuring the length of the image it's important to detect the edges as the edges will help in finding the starting and ending of the pixel value. Edge detection is done with the help of a canny method present in the opencv library.Detected edges along with the image is used to locat the end and stating positions of the pixels which when processed will be helpful in find the length of the object.

In [6]:
```
img.shape
```

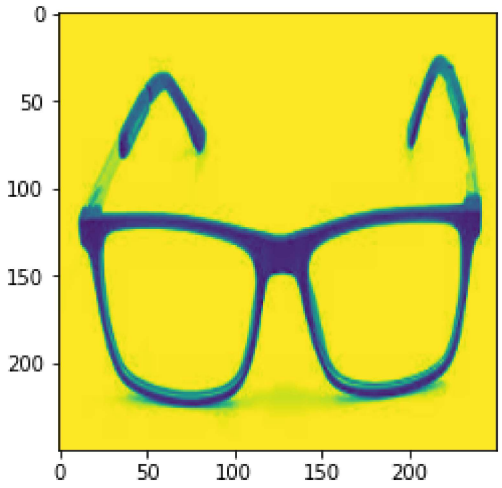
Out[6]:
```
(164, 308)
```

5. Reshaping to equal size

In [7]:
```
img=cv2.resize(img,(250,250))
```

In [8]:
```
plt.imshow(img)
```

Out[8]: <matplotlib.image.AxesImage at 0x7ff31bb06450>



5. Finding out two edge points (pixels)

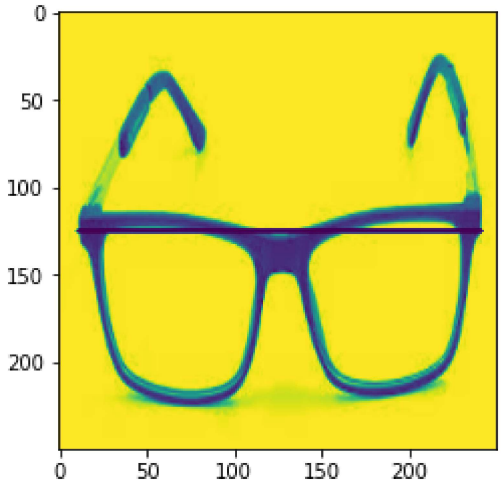
In [9]:
```
p1,p2=[],[]
for i in range(len(img)-1,-1,-1):
  #print(i,end=' ')
  for j in range(len(img[i])-1,-1,-1):
    #print(j,end=' ')
    if img[j][i]<225:
      p1=(i,j)
      break
  if not p1==[]:
    break

for i in range(len(img)):
  for j in range(len(img[i])-1,-1,-1):
    if img[j][i]<235:
      p2=(i,j)
      break
  if not p2==[]:
    break

p1,p2
```

Out[9]: ((241, 125), (11, 124))

7. Connecting two pixels

In [10]:
```
p2=(p2[0],p1[1])
cv2.line(img,p1,p2,(0,0,255),2)
plt.imshow(img)
```

Out[10]: <matplotlib.image.AxesImage at 0x7ff31baf5590>



6.Calculating the length of Spectacles in an image

In [11]:
```
inches = 30     # 1 inch = 30 pixels
pixel_count = abs(p1[0]-p2[0])
length = pixel_count/inches
length
```

Out[11]: 7.666666666666667

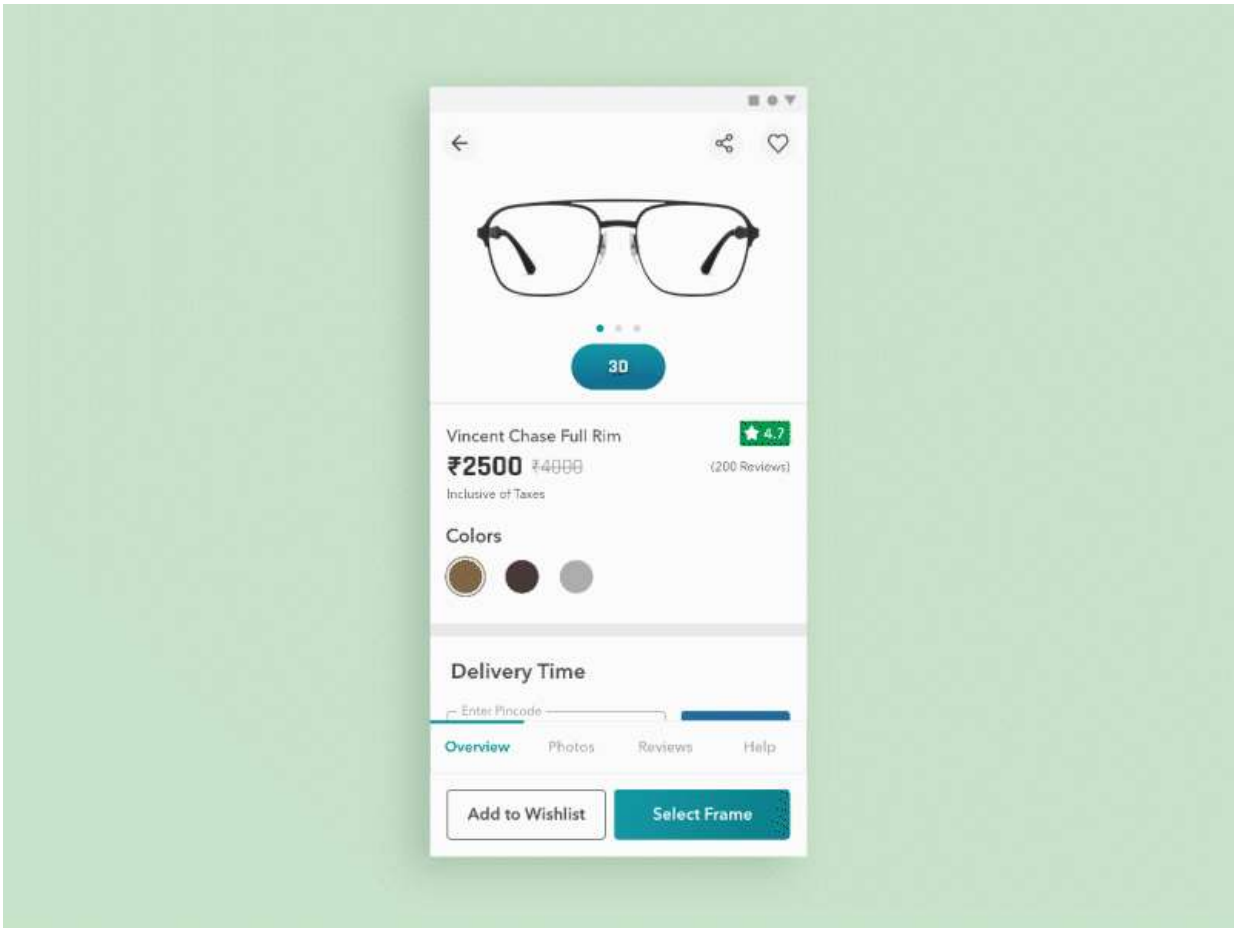
To calculate the length of the image firstly we have to define the measuring scale for the pixel.The measuring scale in this case is defined as 1 inch equals to 30pixels meaning that in every inch 30 pixels are present,so knowing stating and ending pixel location and dividing by the measuring scale the length of the object id calculated.

Real World Application:

Lenskart 3d Try on & Lenskart Frame size

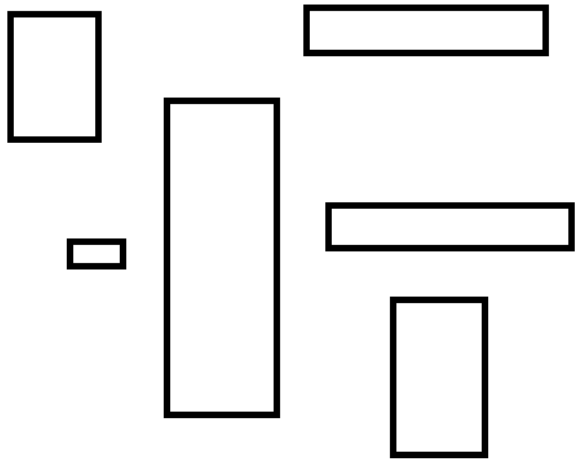
In [12]:
```
from IPython.display import Image
Image(url='https://cdn.dribbble.com/users/4189149/screenshots/14948347/dribbble_shot_2_
```

Out[12]:


3D
Vincent Chase Full Rim
4.7
₹2500 ₹4000
(200 Reviews)
Inclusive of Taxes
Colors
Delivery Time
Enter Pincode
Overview
Photos
Reviews
Help
Add to Wishlist
Select Frame

Experiment 4: Consider an image and calculate the number of components

Input: image with many rectangular boxes



Algorithm:

1.Initilize a matrix equal to the size of the image to mark and to keep track of numbers used.

2. first scan from left to right and top to bottom.

3. See If we have chance to inherit both right and above cell.

4. See If we have a chance to inherit right.

5. See If we have a chance to inherit above.

6. Else labe new value.

7. Atlast find out all possible equi pairs and form the groups.

Count the number of connected components

In [ ]:
```
import cv2
import matplotlib.pyplot as plt
import numpy as np
```

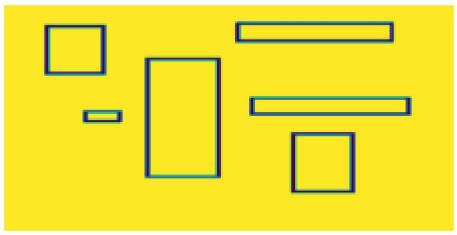
In [ ]:
```
from google.colab import files
files=files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving test6.png to test6 (1).png
```

```
        - Read Image and pre-process it
            - resize it to 100x20
            - convert it into binary threshold matrix
```

In [ ]:
```
img = cv2.imread('test6 (1).png',cv2.IMREAD_GRAYSCALE)
img = cv2.resize(img, (500, 250))
thresh = 128
img_binary = cv2.threshold(img, thresh, 1, cv2.THRESH_BINARY)[1]
plt.figure(figsize=(4,4))
plt.axis("off")
plt.imshow(img_binary)
plt.show()
```



In [ ]:
```
# Binary matrix
img_binary
```

Out[ ]:
```
array([[1, 1, 1, ..., 1, 1, 1],
       [1, 1, 1, ..., 1, 1, 1],
       [1, 1, 1, ..., 1, 1, 1],
       ...,
       [1, 1, 1, ..., 1, 1, 1],
       [1, 1, 1, ..., 1, 1, 1],
       [1, 1, 1, ..., 1, 1, 1]], dtype=uint8)
```

In [ ]:
```
img_binary.shape
```

Out[ ]:
```
(250, 500)
```

In [ ]:
```
import collections
# Eq class by disjoint set
class UnionFind:
    def __init__(self):
        self.leaders = collections.defaultdict(lambda: None)
```

```
    def find(self, x):
        # find leaders
        l = self.leaders[x]
        if l is not None:
            l = self.find(l)
            self.leaders[x] = l
            return l
        return x
    def union(self, x, y):
        # union and assign leaders
        lx, ly = self.find(x), self.find(y)
        if lx != ly:
            self.leaders[lx] = ly
    def get_groups(self):
        # eq classes
        groups = collections.defaultdict(set)
        for x in self.leaders:
            groups[self.find(x)].add(x)
        return list(groups.values())

# Algorithm
def scan(img_binary):
    # Initilize a matrix equal to size of the image to mark
    scan_one = np.zeros(img_binary.shape,dtype=np.int8)
    # To keep track of numbers used
    count = 0
    eq_pairs = set()
    # first scan
    for i in range(img_binary.shape[0]):
        for j in range(img_binary.shape[1]):
            if(img_binary[i][j] == 0):
                # If we have choice to inherit both rigth and above
                if(scan_one[max(0,i-1)][j] != 0 and scan_one[i][max(0,j-1)] != 0 ):
                        eq_pairs.add((scan_one[max(0,i-1)][j],scan_one[i][max(0,j-1)]))
                        scan_one[i][j] = scan_one[max(0,i-1)][j]
                # If we have chance to inherit right
                elif(scan_one[max(0,i-1)][j] != 0):
                    scan_one[i][j] = scan_one[max(0,i-1)][j]
                # If we have chance to inherit above
                elif(scan_one[i][max(0,j-1)] != 0):
                    scan_one[i][j] = scan_one[i][max(0,j-1)]
                # New value
                else:
                    scan_one[i][j] = count+1
                    count+=1
                    eq_pairs.add((scan_one[i][j],scan_one[i][j]))
    # eq pair generation
    uf = UnionFind()
    for (a, b) in eq_pairs:
        uf.union(a, b)
    return eq_pairs,scan_one,uf
```

In [ ]:

```
# Theory representation
test_pairs,matrix,cls = scan(img_binary)
for i in matrix:
    for j in i:
        if(j == 0):
            print('',end='  ')
        else:
```

```
            print(j,end=' ')
    print()
```

In [ ]:
```
# eq pairs
print(test_pairs)
```

```
{(1, 2), (3, 3), (5, 5), (6, 6), (4, 4), (7, 7), (2, 2), (1, 1)}
```

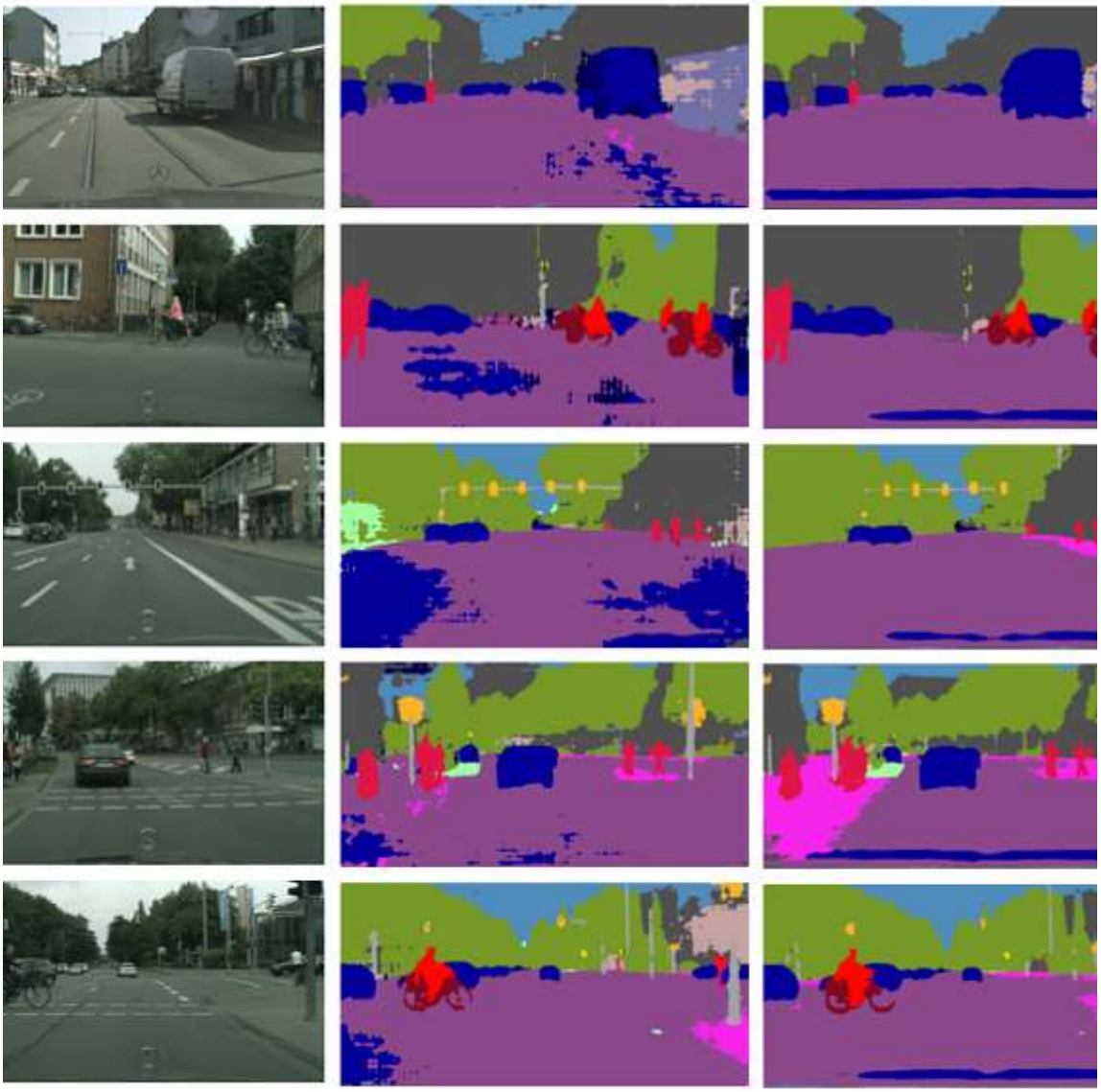
In [ ]:
```
print("Number of components : " + str(len(cls.get_groups())))
print("Components : " , cls.get_groups())
```

```
Number of components : 6
Components :  [{1, 2}, {3}, {5}, {6}, {4}, {7}]
```

Real World Application:

Sematic Segmentation:

Semantic segmentation is the process of classifying each pixel belonging to a particular label. It doesn't different across different instances of the same object. For example if there are 2 cats in an image, semantic segmentation gives same label to all the pixels of both cats



Instance Segmentation :

Instance segmentation differs from semantic segmentation in the sense that it gives a unique label to every instance of a particular object in the image. As can be seen in the image above all 3 dogs are assigned different colours i.e different labels. With semantic segmentation all of them would have been assigned the same colour.

Experiment 5

Write a program to find line, edge and point in an image and design an application for the same.

APPLICATION

1) Mark parking lanes in a parking lot (Edge and Line).
2) Convert a greyscale image to binary data structure by using thresholding.

THEORY

1. Canny edge detection
    a. Apply Gaussian filter to smooth the image in order to remove the noise
    b. Find the intensity gradients of the image

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} A, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} A$$

    c. Apply gradient magnitude thresholding or lower bound cut-off suppression to get rid of spurious response to edge detection
    d. Apply double threshold to determine potential edges
    e. Track edge by hysteresis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.
2. Hough line transform
    a. Edge detection using the Canny edge detector.
    b. Mapping of edge points to the Hough space and storage in an accumulator.
        i. Decide on the range of ρ and θ. Often, the range of θ is [0, 180] degrees and ρ is [-d, d] where d is the length of the edge image's diagonal. It is important to quantize the range of ρ and θ meaning there should be a finite number of possible values.
        ii. Create a 2D array called the accumulator representing the Hough Space with dimension (num_rhos, num_thetas) and initialize all its values to zero.
        iii. Decide on the range of ρ and θ. Often, the range of θ is [0, 180] degrees and ρ is [-d, d] where d is the length of the edge image's diagonal. It is important to quantize the range of ρ and θ meaning there should be a finite number of possible values.
        iv. Create a 2D array called the accumulator representing the Hough Space with dimension (num_rhos, num_thetas) and initialize all its values to zero.

c. Interpretation of the accumulator to yield lines of infinite length. The interpretation is done by thresholding and possibly other constraints.
d. Conversion of infinite lines to finite lines.

ALGORITHM

1) Edge:
    1.1) Read the image
    1.2) Convert it to gray scale
    1.3) Apply gaussian blur
    1.4) Apply Canny edge filter
2) Line:
    2.1) For the above generated
    2.2) Apply Hough Line transform
    2.3) Overlay the above obtained lines to actual image and print it
3) Point:
    3.1) Read the image in grayscale
    3.2) Resize the image
    3.3) Apply thresholding on above obtained image
    3.4) Print the above obtained image

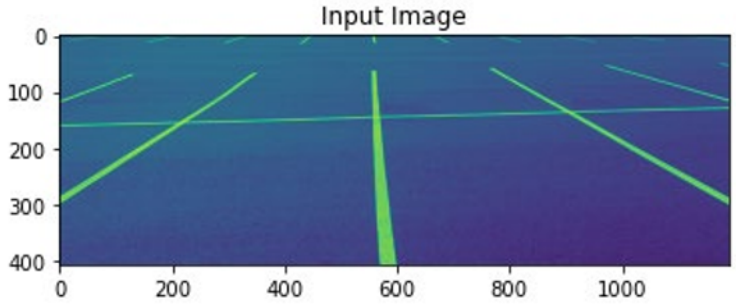
CODE

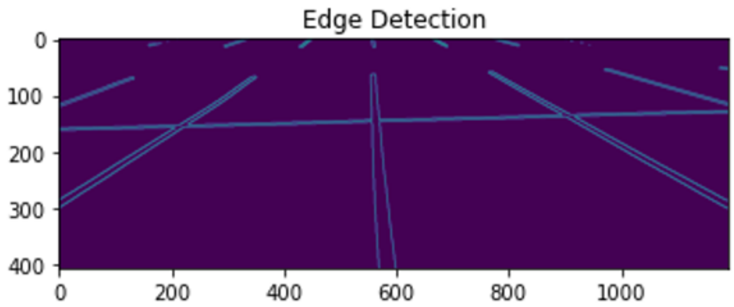
```
import cv2
import matplotlib.pyplot as plt
import numpy as np

# Line and edge detection

img = cv2.imread('line.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

kernel_size = 5
blur_gray = cv2.GaussianBlur(gray,(kernel_size, kernel_size),0)

low_threshold = 240
high_threshold = 255
edges = cv2.Canny(blur_gray, low_threshold, high_threshold)

rho = 1  # distance resolution in pixels of the Hough grid
theta = np.pi / 180  # angular resolution in radians of the Hough grid
threshold = 10  # minimum number of votes (intersections in Hough grid cell)
min_line_length = 10  # minimum number of pixels making up a line
max_line_gap = 20  # maximum gap in pixels between connectable line segments
line_image = np.copy(img) * 0  # creating a blank to draw lines on

# Run Hough on edge detected image
# Output "lines" is an array containing endpoints of detected line segments
```

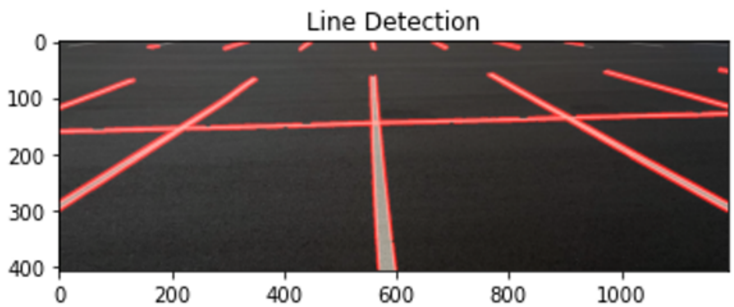
```
lines = cv2.HoughLinesP(edges, rho, theta, threshold, np.array([]),
                    min_line_length, max_line_gap)

for line in lines:
    for x1,y1,x2,y2 in line:
        cv2.line(line_image,(x1,y1),(x2,y2),(255,0,0),5)

lines_edges = cv2.addWeighted(img, 0.8, line_image, 1, 0)

plt.imshow(gray)
plt.title("Input Image")
plt.show()
plt.imshow(edges)
plt.title("Edge Detection")
plt.show()
plt.imshow(lines_edges)
plt.title("Line Detection")
plt.show()

# Line detection

img = cv2.imread('point.png',cv2.IMREAD_GRAYSCALE)
img = cv2.resize(img, (20, 20))
thresh = 128
img_binary = cv2.threshold(img, thresh, 1, cv2.THRESH_BINARY)[1]
plt.figure(figsize=(4,4))
plt.axis("off")
plt.imshow(img_binary)
plt.show()
```

OUTPUT

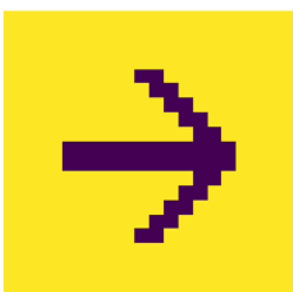
Line and edge detection :







Point Detection :



```
img_binary
```

```
array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1],
       [1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]],
      dtype=uint8)
```

Experiment 6:

Write a program to perform region-based segmentation and design an application for the same.

APPLICATION

Segment nucleus in a test sample

THEORY

Segmentation of images is crucial to our understanding of them. Consequently, much effort has been devoted to devising algorithms for this purpose. Since the sixties a variety of techniques have been proposed and tried for segmenting images by identifying regions of some common property. These can be classified into two main classes:

1. Merging algorithms: in which neighboring regions are compared and merged if they are close enough in some property.
2. Splitting Algorithms: in which large non-uniform regions are broken up into smaller areas which may be uniform.

There are algorithms which are a combination of splitting and merging. In all cases some uniformity criterion must be applied to decide if a region should be split, or two regions merged. This criterion is based on some region property which will be defined by the application, and could be one of many measurable image attributes such as mean intensity, color etc. Uniformity criteria can be defined by setting limits on the measured property, or by using statistical measures, such as standard deviation or variance.

Region Merging

Merging must start from a uniform seed region. Some work has been done in discovering a suitable seed region. One method is to divide the image into 2x2 or 4x4 blocks and check each one. Another is to divide the image into strips, and then subdivide the strips further. In the worst case the seed will be a single pixel. Once a seed has been found, its neighbors are merged until no more neighboring regions conform to the uniformity criterion. At this point the region is extracted from the image, and a further seed is used to merge another region.

There are some drawbacks which must be noted with this approach. The process is inherently sequential, and if fine detail is required in the segmentation, then the computing time will be long. Moreover, since in most cases the merging of two regions will change the value of the property being measured, the resulting area will depend on the search strategy employed among the neighbors, and the seed chosen.

Region Splitting

These algorithms begin from the whole image, and divide it up until each sub region is uniform. The usual criterion for stopping the splitting process is when the properties of a newly split pair do not differ from those of the original region by more than a threshold.

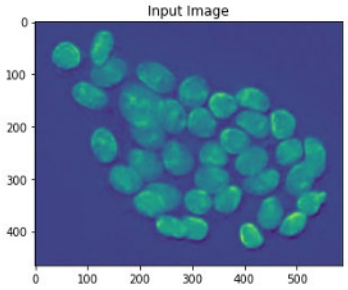
The chief problem with this type of algorithm is the difficulty of deciding where to make the partition. Early algorithms used some regular decomposition methods, and for some classes these are satisfactory, however, in most cases splitting is used as a first stage of a split/merge algorithm.

ALGORITHM

1) Read image and convert it to grayscale since water-shed algorithm requires images to be in grayscale.
2) Get elevation map by using Sobel filter
3) Define markers by specifying threshold
4) Use watershed algorithm to perform segmentation

CODE

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

# Read image and convert it to grayscale

img = cv2.imread('ws.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

# get elevation map ie:edges

from skimage.filters import sobel
elevation_map = sobel(gray)

# define markers

markers = np.zeros_like(gray)
markers[gray < 90] = 1
markers[gray > 100] = 2

# apply markers

from skimage import morphology
segmentation = morphology.watershed(elevation_map, markers)
ernel_size = 5
blur_gray = cv2.GaussianBlur(gray,(kernel_size, kernel_size),0)
```



OUTPUT