

PROFESSIONAL TRAINING REPORT
at
Sathyabama Institute of Science and Technology
(Deemed to be University)

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Technology Degree in

Information Technology

By

SANGEETHA K K (38120073)



DEPARTMENT OF INFORMATION TECHNOLOGY
SCHOOL OF COMPUTING
SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY,
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI – 600119. TAMILNADU.

NOV 2021



SATHYABAMA UNIVERSITY
(Established under Section 3 of UGC Act, 1956)
Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai - 600119
www.sathyabamauniversity.ac.in



SCHOOL OF COMPUTING

BONAFIDE CERTIFICATE

This is to certify that this Professional Training Report is the bonafide work of SANGEETHA K K (38120073) who underwent the professional training in “IMAGE PROCESSING USING OPENCV” under our supervision from JUNE 2021 to NOV 2021.

Internal Guide

Dr. MATHIVANAN M.E., Ph.D.,

Head of the Department

Dr. R.SUBHASHINI M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I, SANGEETHA K K (Reg.No:3812073) hereby declare that the Professional Training Report on '**IMAGE PROCESSING USING OPENCV**' done by me under the guidance of **Dr.Mathivanan M.E.,Ph.D.**, at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Technology degree in Information Technology.

DATE: 12.11.2021

PLACE: Chennai

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T.Sasikala M.E., Ph.D, Dean**, School of Computing, **Dr.SUBHASHINI M.E., Ph.D.**, Head of the Department of Information and Technology for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide, **Dr.MATHIVANAN M.E., Ph.D.**, for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Information Technology** who were helpful in many ways for the completion of the project.

CERTIFICATE

Exposys
Data Labs



Certificate of Internship

TO WHOM IT MAY CONCERN

This is to certify that **Ms. SANGEETHA K. K** has completed internship programme on “**Web Developer**” from 14.06.2021 to 13.07.2021.

She took keen interest in the work assigned and successfully completed it. During the period of internship we found her to be punctual, hardworking and inquisitive.

We wish her luck and success in all her future endeavours.

Y Vishnuvardhan

Chief Director



hr@exposysdata.com
www.exposysdata.com

ABSTRACT

Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. It is a type of signal processing in which input is an image and output may be an optimized image or characteristics or even features associated with that image. The aim of this method is to improve the quality of image and subsequently perform features like extraction and classification. Python, due to its growing popularity as a scientific programming language and the free availability of many State of the Art Image Processing tools in its ecosystem is combined with OpenCV libraries and is executed. It is effectively used in computer vision, medical imaging, meteorology, astronomy, remote sensing and other related fields. The first process is to select an image and to work on the slicing of RGB spectrum, followed by analyzing the histograms of the image by each pixel frame. Image processing can also be used to identify different objects mostly related to medical entities. Capturing videos through webcams, image blurring, geometric implementation, object tracking and many more can be done through this technology. This process creates an ease to the emerging technologies and provides a lesser manual work.

TABLE OF COTENTS

CHANO.	TITLE	PAGE NO.
	ABSTRACT	vi
	LIST OF FIGURES	viii
1	INTRODUCTION	1
	1.1 IMAGE PROCESSING	1
	1.3 OUTLINE OF THE PROJECT	2
2.	SYSTEM DESIGN AND IMPLEMENTATION	3
	2.1 COLOR SPACES	3
	2.2 GEOMETRIC TRANSFORMATION OF IMAGES	5
	2.2.1 TRANSFORMATION	5
	2.2.2 SCALING	5
	2.2.3 TRANSLATION	6
	2.2.4 ROTATION	6
	2.3 IMAGE THRESHOLDING	7
	2.4 SOOTHING IMAGES	8
	2.4.1 IMAGE FILTERING	8
	2.4.2 IMAGE BLURRING	10
	2.5 CANNY EDGE DETECTION	13
3.	SOFTWARE	
	3.1 PYCHARM	14
	3.2 OPENCV	15
4.	CONCLUSION AND FUTURE SCOPE	16
5.	REFERENCES	17
6.	CODE AND SCREENSHOTS	20

LIST OF FIGURES

FIG	TITLE
1.	Color Spaces
2.	Transformed Image
3.	Translated Image
4.	Rotated Image
5.	Image Thresholding
6.	Image Filtering
7.	Averaging Image
8.	Gaussian Filtering
9.	Median Filtering
10.	Bilateral Filtering
11.	Canny Edge Detection

CHAPTER 1

1. INTRODUCTION

In this paper, we are going to learn how we can perform image processing using the Python language. We are not going to restrict ourselves to a single library or framework. However, there is one that we will be using the most frequently, the Open CV library. In the present days, though we have lots of improvised technologies for managing the images online as well as offline, there might be little glitches and clashes under certain circumstances. At these times, Image processing would be of great help to overcome any issues regarding the specific images. We will start off by looking into image processing and then we will move on to see different applications/scenarios where image processing can come in handy.

1.1 IMAGE PROCESSING

Image Processing is a field of knowledge that falls under Computer Vision. The premises of machine learning were first laid down by computer vision theory, applying a whole set of techniques to process and analyze imagery data to extract valuable information that computers and machines may use for a wide range of applications including stitching, morphing, 3D modelling, visual organization, facial recognition and many more. Image processing is a prior step in computer vision, where the goal is to convert an image into a form suitable for further analysis. Examples of operations such as exposure correction, color balancing, image noise reduction, or increasing image sharpness are highly important and very care demanding to achieve acceptable results in most computer vision applications like computational photography or even face recognition. The data that we collect or generate is mostly raw data, i.e. it is not fit to be used in applications directly due to a number of possible reasons. Therefore, we need to analyze it first, perform the necessary pre-processing, and then use it. For instance, let's assume that we were trying to build a cat classifier.

Our program would take an image as input and then tell us whether the image contains a cat or not. The first step for building this classifier would be to collect hundreds of cat pictures. One common issue is that all the pictures we have scraped would not be of the same size/dimensions, so before feeding them to the model for training, we would need to resize/pre-process them all to a standard size. This is just one of many reasons why image processing is essential to any computer vision application.

1.2 OUTLINE OF THE PROJECT

The main objective of this paper is to enhance images in an efficient manner to adapt to modern ecstasies. The following are some of the core operations:

- Color Spaces
- Geometric Transformation of Images
- Image thresholding
- Soothing Images
- Canny Edge Detection

All of the above operations or functions helps a particular image to develop an optimized model tracking and reduces redundancy and helps in the removal of pixelated as well as blurred images. It also helps to augment the contrasting options in a specific image. Every operation will lead to the satisfaction of any modern user with today's technologies.

CHAPTER 2

2. SYSTEM DESIGN AND IMPLEMENTATION

Firstly, one should have some basic programming knowledge in any language. Secondly, one should also know what machine learning is and the basics of how it works, as we will be using some machine learning algorithms for image processing in this paper. As a bonus, it would help us if we have had any exposure to, or basic knowledge of, Open CV before moving on with it. But this is only the prerequisites that are required. One thing that we should definitely know in order to follow this is how exactly an image is represented in memory. Each image is represented by a set of pixels i.e. a matrix of pixel values. For a grayscale image, the pixel values range from 0 to 255 and they represent the intensity of that pixel. For instance, if you have an image of 20 x 20 dimensions, it would be represented by a matrix of 20x20 (a total of 400-pixel values). If you are dealing with a colored image, you should know that it would have three channels - Red, Green, and Blue (RGB). Therefore, there would be three such matrices for a single image.

2.1 COLOR SPACES

The main goal in this operation is to convert images from one color-space to another, like $BGR \leftrightarrow \text{Gray}$, $BGR \leftrightarrow \text{HSV}$ (Hue, Saturation, Value) etc. There are more than 150 color-space conversion methods available in OpenCV. But we will look into only two which are most widely used ones, $BGR \leftrightarrow \text{Gray}$ and $BGR \leftrightarrow \text{HSV}$. For HSV, Hue range is [0,179], Saturation range is [0,255] and Value range is [0,255]. Different software use different scales. So if you are comparing OpenCV values with them, you need to normalize these ranges. For color conversion, we use the function `cv2.cvtColor(input_image, flag)` where flag determines the type of conversion. For $BGR \rightarrow \text{Gray}$ conversion we use the flags `cv2.COLOR_BGR2GRAY`.

Similarly for BGR $\rightarrow$ HSV, we use the flag cv2.COLOR_BGR2HSV. To get other flags, we can run the following commands in our Python terminal:

```
>>> import cv2
>>> flags = [i for i in dir(cv2) if i.startswith('COLOR_')]
>>> print flags
```

Now we know how to convert BGR image to HSV, we can use this to extract a colored object. In HSV, it is easier to represent a color than RGB color-space. In our application, we will try to extract a blue colored object. So here is the method:

- Take each frame of the image.
- Convert from BGR to HSV color-space
- We threshold the HSV image for a range of blue color
- Now extract the blue object alone, we can do whatever on that image we want.



Fig.1 Color Spaces

2.2 GEOMETRIC TRANSFORMATION OF IMAGES

In this module, we will learn how to apply different geometric transformation to images like translation, rotation, affine transformation etc.

2.2.1 TRANSFORMATION

OpenCV provides two functions in transformation of images which includes, `cv2.warpAffine` and `cv2.warpPerspective`, with which you can have all kinds of transformations. `cv2.warpAffine` takes a 2x3 transformation matrix while `cv2.warpPerspective` takes a 3x3 transformation matrix as input.

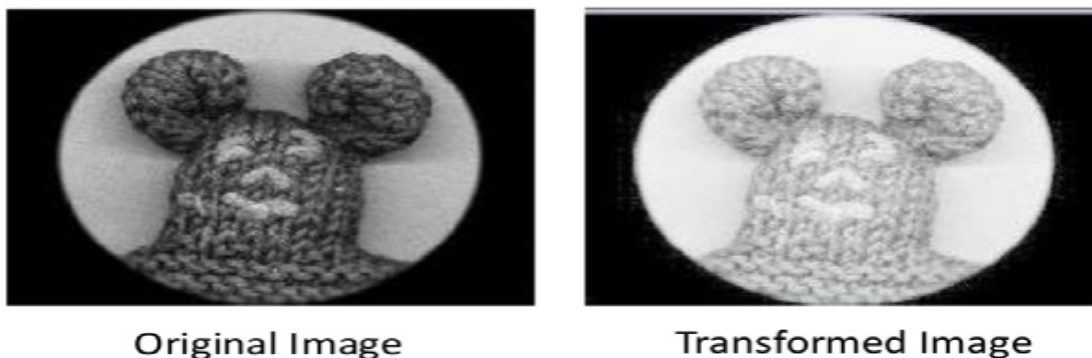


Fig.2 Transformed Image

2.2.2 SCALING

Scaling is just resizing of the image. OpenCV comes with a function `cv2.resize()` for this purpose. The size of the image can be specified manually, or you can specify the scaling factor. Different interpolation methods are used. Preferable interpolation methods are `cv2.INTER_AREA` for shrinking and `cv2.INTER_CUBIC` (slow) & `cv2.INTER_LINEAR` for zooming. By default, interpolation method used is `cv2.INTER_LINEAR` for all resizing purposes.

2.2.3 TRANSLATION

Translation is the shifting of object's location. If you know the shift in (x,y) direction, let it be (t_x, t_y) , you can create the transformation matrix M as follows:

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

You can take make it into a Numpy array of type np.float32 and pass it into cv2.warpAffine() function.



Fig.3 Translated Image

2.2.4 ROTATION

Rotation of an image for an angle θ is achieved by the transformation matrix of the form,

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

But OpenCV provides scaled rotation with adjustable center of rotation so that you can rotate at any location you prefer. Modified transformation matrix is given by,

$$\begin{bmatrix} \alpha & \beta & (1 - \alpha) \cdot center.x - \beta \cdot center.y \\ -\beta & \alpha & \beta \cdot center.x + (1 - \alpha) \cdot center.y \end{bmatrix}$$

Where,

$$\alpha = scale \cdot \cos \theta,$$

$$\beta = scale \cdot \sin \theta$$

To find this transformation matrix, OpenCV provides a function, **cv2.getRotationMatrix2D**.

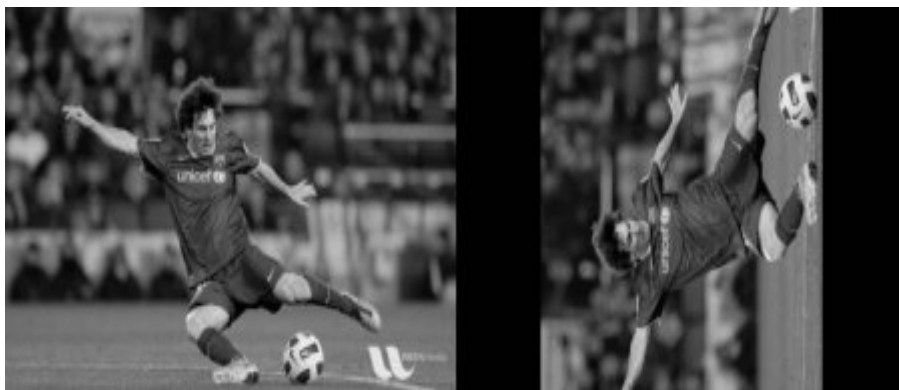


Fig.4 Rotated Image

2.3 IMAGE THRESHOLDING

The concept of thresholding is quite simple. In the image representation, pixel values can be any value from 0 to 255. Let's say we wish to convert an image into a binary image i.e. assign a pixel either a value of 0 or 1. To do this, we can perform thresholding. For instance, if the Threshold (T) value is 125, then all pixels with values greater than 125 would be assigned a value of 1, and all pixels with values lesser than or equal to that would be assigned a value of 0. The function used is **cv2.threshold**. First argument is the source image, which should be a grayscale image. Second argument is the threshold value which is used to classify the pixel values.

Third argument is the maxVal which represents the value to be given if pixel value is more than (sometimes less than) the threshold value. OpenCV provides different styles of thresholding and it is decided by the fourth parameter of the function.

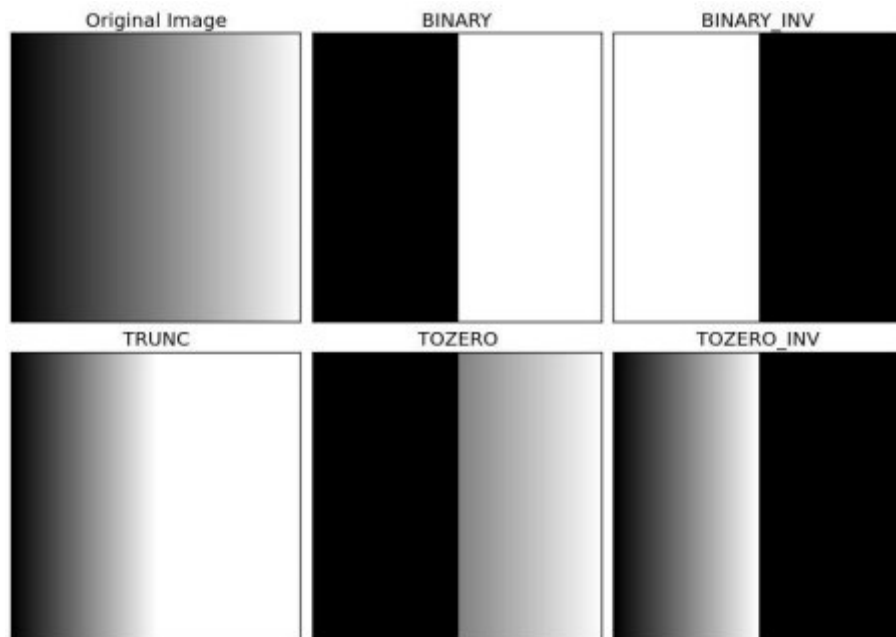


Fig.5 Image Thresholding

2.4 SOOTHING IMAGES

In this module, the following two objectives are achieved:

- Blur images with various low pass filters.
- Apply custom-made filters to images (2D convolution).

2.4.1 IMAGE FILTERING

As for one-dimensional signals, images also can be filtered with various low-pass filters (LPF), high-pass filters (HPF), etc. A LPF helps in removing noise, or blurring the image. A HPF filters helps in finding edges in an image.

OpenCV provides a function, `cv2.filter2D()`, to convolve a kernel with an image. As an example, we will try an averaging filter on an image. A 5x5 averaging filter kernel can be defined as follows:

$$K = \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Filtering with the above kernel results in the following being performed:

For each pixel, a 5x5 window is centered on this pixel, all pixels falling within this window are summed up, and the result is then divided by 25. This equates to computing the average of the pixel values inside that window. This operation is performed for all the pixels in the image to produce the output filtered image.



Fig.6 Image Filtering

2.4.2 IMAGE BLURRING

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 (e.g: noise, edges) from the image resulting in edges being blurred when this filter is applied. Well, there are blurring techniques which do not blur edges. OpenCV provides mainly four types of blurring techniques:

- **Averaging**

This is done by convolving the image with a normalized box filter. It simply takes the average of all the pixels under kernel area and replaces the central element with this average. This is done by the function `cv2.blur()` or `cv2.boxFilter()`. Check the docs for more details about the kernel.

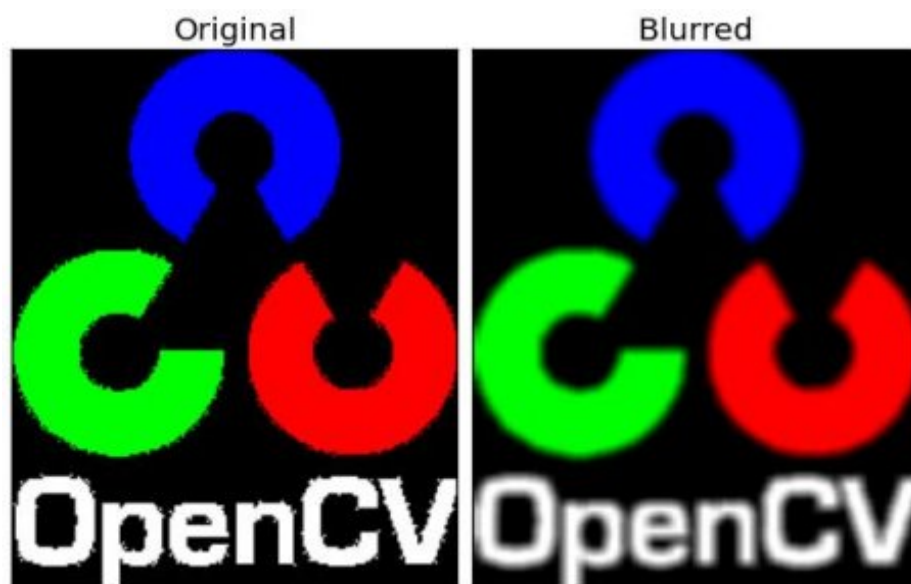


Fig.7 Averaging Image

- **Gaussian Filtering**

In this approach, instead of a box filter consisting of equal filter coefficients, a Gaussian kernel is used. It is done with the function, `cv2.GaussianBlur()`.

We should specify the width and height of the kernel which should be positive and odd. We also should specify the standard deviation in the X and Y directions, σ_x and σ_y respectively. If only σ_x is specified, σ_y is taken as equal to σ_x . If both are given as zeros, they are calculated from the kernel size. Gaussian filtering is highly effective in removing Gaussian noise from the image.

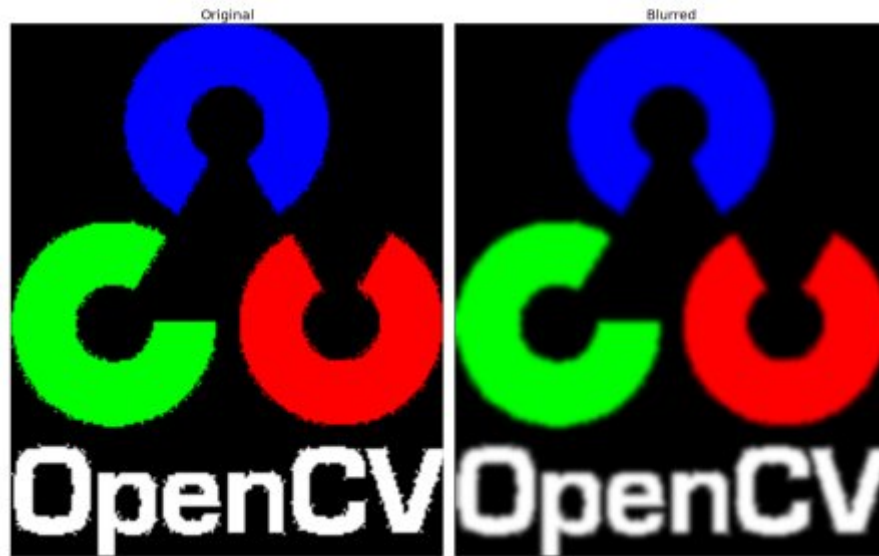


Fig.8 Gaussian Filtering

- **Median Filtering**

Here, the function `cv2.medianBlur()` computes the median of all the pixels under the kernel window and the central pixel is replaced with this median value. This is highly effective in removing salt-and-pepper noise. One interesting thing to note is that, in the Gaussian and box filters, the filtered value for the central element can be a value which may not exist in the original image. The kernel size must be an odd integer.

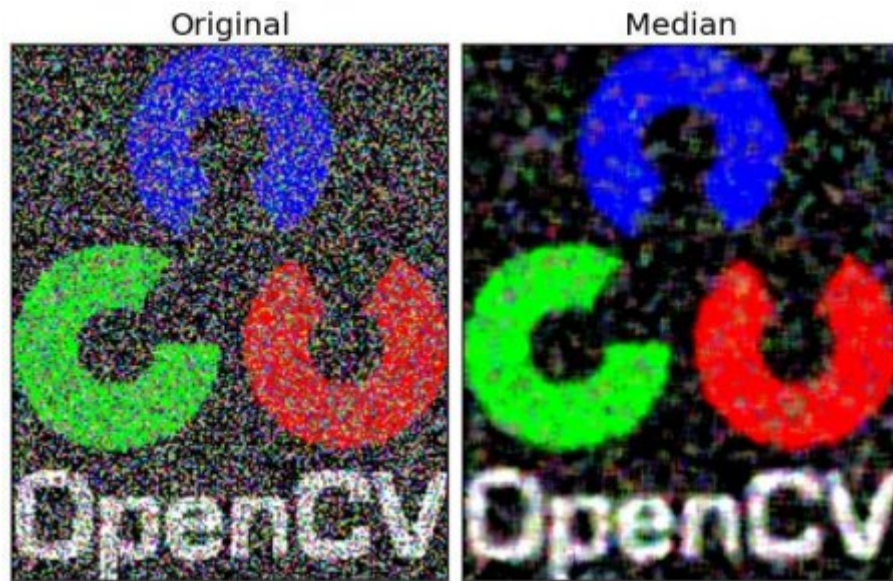


Fig.9 Median Filtering

- **Bilateral Filtering**

As we noted, the filters we presented earlier tend to blur edges. This is not the case for the bilateral filter, `cv2.bilateralFilter()`, which was defined for, and is highly effective at noise removal while preserving edges. But the operation is slower compared to other filters. We already saw that a Gaussian filter takes the a neighborhood around the pixel and finds its Gaussian weighted average. This Gaussian filter is a function of space alone, that is, nearby pixels are considered while filtering. It does not consider whether pixels have almost the same intensity value and does not consider whether the pixel lies on an edge or not. The resulting effect is that Gaussian filters tend to blur edges, which is undesirable.

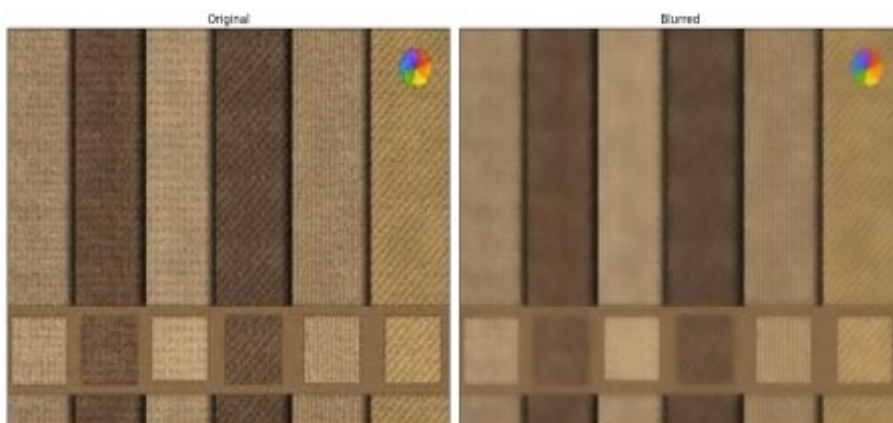


Fig.10 Bilateral Filtering

2.5 CANNY EDGE DETECTION

OpenCV puts all the above in single function, **cv2.Canny()**. We will see how to use it. First argument is our input image. Second and third arguments are our *minVal* and *maxVal* respectively. Third argument is *aperture_size*. It is the size of Sobel kernel used for find image gradients. By default it is 3. Last argument is *L2gradient* which specifies the equation for finding gradient magnitude. If it is True, it uses the equation mentioned above which is more accurate, otherwise it uses this function: By default, it is False.

$$Edge_Gradient (G) = |G_x| + |G_y|$$

By default, it is False.

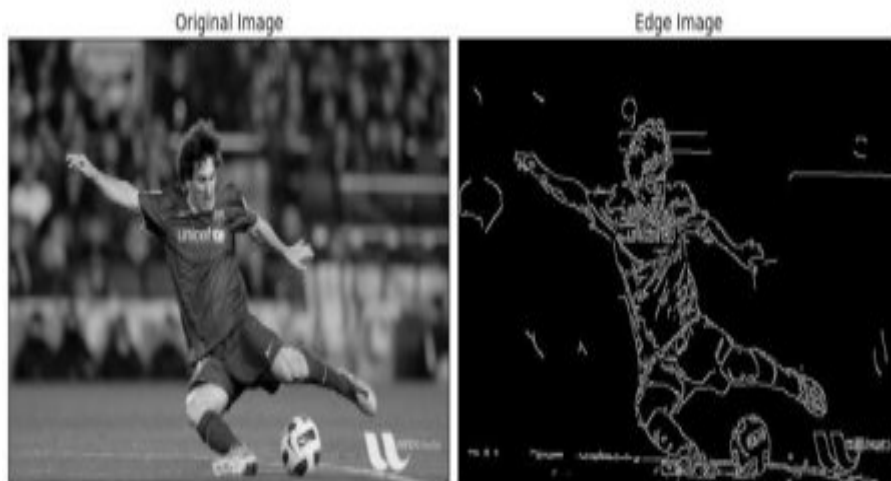


Fig.11 Canny Edge Detection

CHAPTER 3

3. SOFTWARES EMPLOYED

1. PyCharm
2. OpenCV

3.1. PYCHARM

PyCharm is an integrated development environment (IDE) used in computer programming, specifically for the Python language. It is developed by the Czech company JetBrains. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes), and supports web development with Django as well as data science with Anaconda. PyCharm is cross-platform, with Windows, macOS and Linux versions. The Community Edition is released under the Apache License, and there is also Professional Edition with extra features – released under a proprietary license. PyCharm provides an API so that developers can write their own plugins to extend its features. Several plugins from other JetBrains IDE also work with PyCharm. There are more than 1000 plugins which are compatible with this platform. Coding assistance and analysis, with code completion, syntax and error highlighting, linter integration, and quick fixes. Project and code navigation: specialized project views, file structure views and quick jumping between files, classes, methods and usages. Python refactoring: includes rename, extract method, introduce variable, introduce constant, pull up, push down and others. Support for web frameworks: Django, web2py and Flask. It consists of an Integrated Python debugger which is lined up with Integrated unit testing, with line-by-line code coverage and Google App Engine Python development. Version control integration for Mercurial, Git, Subversion, Perforce and CVS with change lists and merge. It also provides support for scientific tools like matplotlib, numpy and scipy.

3.2 OPENCV

OpenCV was started at Intel in 1999 by Gary Bradsky, and the first release came out in 2000. Vadim Pisarevsky joined Gary Bradsky to manage Intel's Russian software OpenCV team. In 2005, OpenCV was used on Stanley, the vehicle that won the 2005 DARPA Grand Challenge. Later, its active development continued under the support of Willow Garage with Gary Bradsky and Vadim Pisarevsky leading the project. OpenCV now supports a multitude of algorithms related to Computer Vision and Machine Learning and is expanding day by day. OpenCV supports a wide variety of programming languages such as C++, Python, Java, etc., and is available on different platforms including Windows, Linux, OS X, Android, and iOS. Interfaces for high-speed GPU operations based on CUDA and OpenCL are also under active development. OpenCV-Python is the Python API for OpenCV, combining the best qualities of the OpenCV C++ API and the Python language. OpenCV-Python is a library of Python bindings designed to solve computer vision problems. Python is a general purpose programming language started by Guido van Rossum that became very popular very quickly, mainly because of its simplicity and code readability. It enables the programmer to express ideas in fewer lines of code without reducing readability. Compared to languages like C/C++, Python is slower. That said, Python can be easily extended with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules. This gives us two advantages: first, the code is as fast as the original C/C++ code (since it is the actual code working in background) and second, it is easier to code in Python than C/C++. OpenCV in Python acts like a wrapper for the original OpenCV C/C++ implementation. OpenCV-Python makes use of Numpy, which is a highly optimized library for numerical operations with a MATLAB-style syntax. All the OpenCV array structures are converted to and from Numpy arrays. This also makes it easier to integrate with other libraries that use Numpy such as SciPy and Matplotlib.

CHAPTER 4

4.1 CONCLUSION AND FUTURE SCOPE

In this paper, the most popular library, OpenCV is used for image processing in Python as well as it is also verified how the preprocessing of images are done. We specifically discussed about Image Processing and its uses in the computer vision domain of Machine Learning. It also included some common types of noise and how we can remove it from our images using different filters, before using the images in our applications. Furthermore, Image processing plays an integral part in high-end applications like Object Detection or classification. Do note that this paper was just the tip of an iceberg, and Digital Image Processing has a lot more in the store. Working on all these functions would enable us to dive deeper into the advanced concepts. This technology is required for military and other types of surveillance, statistical data collection in the fields of forestry, agriculture, disaster prediction, weather prediction. In order to extract scientifically useful information, it will be necessary to develop techniques to register real-time data recorded by a variety of sensors for various applications.

The future of image processing will involve scanning the heavens for other intelligent life out in space. Also new intelligent, digital species created entirely by research scientists in various nations of the world will include advances in image processing applications. Due to advances in image processing and related technologies there will be millions and millions of robots in the world in a few decades time, transforming the way the world is managed. Advances in image processing and artificial intelligence⁶ will involve spoken commands , anticipating the information requirements of governments, translating languages, recognizing and tracking people and things, diagnosing medical conditions, performing surgery, reprogramming defects in human DNA, and automatic driving all forms of transport.

With increasing power and sophistication of modern computing, the concept of computation can go beyond the present limits and in future, image processing technology will advance and the visual system of man can be replicated. The future trend in remote sensing will be towards improved sensors that record the same scene in many spectral channels. Graphics data is becoming increasingly important in image processing applications. The future image processing applications of satellite based imaging ranges from planetary exploration to surveillance applications. Using large scale homogeneous cellular arrays of simple circuits to perform image processing tasks and to demonstrate pattern-forming phenomena is an emerging topic. The cellular neural network is an implementable alternative to fully connected neural networks and has evolved into a paradigm for future imaging techniques. The usefulness of this technique has applications in the areas of silicon retina, pattern formation and many more.

5 REFERENCES:

[Aka96]

A.Akansu, M.Smith (editors).

Subband and Wavelet transform. Design and Applications

, Kluwer Academic Publishers, 1996.

[AP922]

A Fast Precise Implementation of 8x8 Discrete Cosine Transform Using the Intel® Streaming SIMD Extensions and MMX™ Instructions

, Application Note AP922, Intel Corp. Order number 742474, 1999.

[APMF]

Fast Algorithms for Median Filtering

, Application Note, Intel Corp. Document number 79835, 2001.

[AVS]

GB/T 200090.2-2006. China Standard. Information Technology

. *Coding of Audio-Visual Objects - Part 2: Visual*

(02/2006).

[Bert01]

M.Bertalmio, A.L.Bertozzi, G.Sapiro.

Navier-Stokes, Fluid Dynamics, and Image and Video Inpainting.

Proc. ICCV 2001, pp.1335-1362, 2001.

[Bor86]

G.Borgefors.

Distance Transformations in Digital Images

. Computer Vision, Graphics, and Image Processing 34, 1986.

[Bou99]

J-Y.Bouguet.

Pyramidal Implementation of the Lucas-Kanade Feature Tracker.

OpenCV Documentation, Microprocessor Research Lab, Intel Corporation, 1999.

[Canny86]

J. Canny.

A Computational Approach to Edge Detection

, IEEE Trans. on Pattern Analysis and Machine Intelligence 8(6), 1986.

[Davis97]

J.Davis and Bobick.

The Representation and Recognition of Action Using Temporal Templates

. MIT Media Lab Technical Report 402, 1997.

[Davis99]

J.Davis and G.Bradski.

Real-Time Motion Template Gradients Using Intel(R) Computer Vision Library

. IEEE ICCV'99 FRAME-RATE WORKSHOP, 1999.

[Dalal05]

N. Dalal and B. Triggs.

Histograms of Oriented Gradients for Human Detection

. INRIA, 2005.

[DICOM]

Digital Imaging and Communications in Medicine (DICOM), published by National Electrical Manufacturers Association, 2003.

[Feig92]

E. Feig and S. Winograd.

Fast Algorithms for the Discrete Cosine Transform

, IEEE Trans. Signal Processing, vol. 40, no. 9, pp. 2174-2193, Sep. 1992.

[Felz04]

P. Felzenszwalb, D. Hattenlocher.

Distance Transforms of Sampled Functions

. Cornell Computing and Information Science Technical Report TR2004-1963, September 2004.

[Foley90]

James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes.

Computer Graphics — Principles and Practice

, Second Edition. Addison Wesley, 1990.

[Gon93]

R.C. Gonzalez and R.E.Wood.

Digital Image Processing

. Prentice Hall, 1993.

[Hir05]

K. Hirakawa, T.W. Parks,

Adaptive Homogeneity-Directed Demosaicing Algorithm

, IEEE Trans. Image Processing, March, 2005.

6. CODE AND SCREENSHOTS

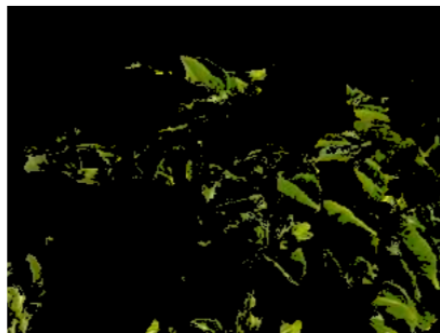
12/11/2021, 17:08

pt2_img_processing1.ipynb - Colaboratory



```
# Bitwise operations is used for image masking.
'''Syntax:
cv2.bitwise_and(src1,src2,destination,mask)
Parameters:
src1- first input image array
src2- second input image array
destination- Output array similar in dimension and type to that of input
mask- operation mask/area of interest'''

result=cv2.bitwise_and(img,img,mask=area_of_interest)
cv2_imshow(result)
```



```
lower=np.array([32,100,100])
upper=np.array([42,255,255])

# Use cv2.inRange() function to create mask over area of interest.

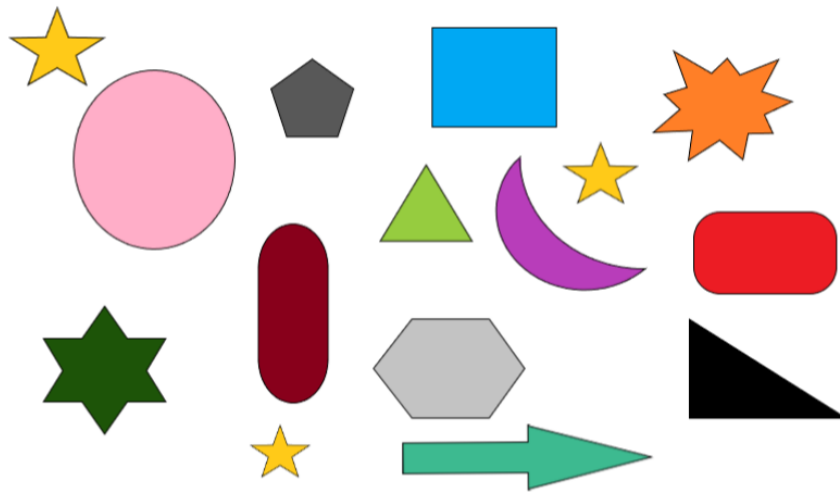
area_of_interest=cv2.inRange(final_img,lower,upper)
cv2_imshow(area_of_interest)
```

```
import cv2
from google.colab.patches import cv2_imshow
```

Load the Image

```
# The image is loaded using cv2.imread
# Syntax: cv2.imread(path, flag)
# Flag specifies how the image should be read
# We set it to 1 to indicate that we want to load the image in color

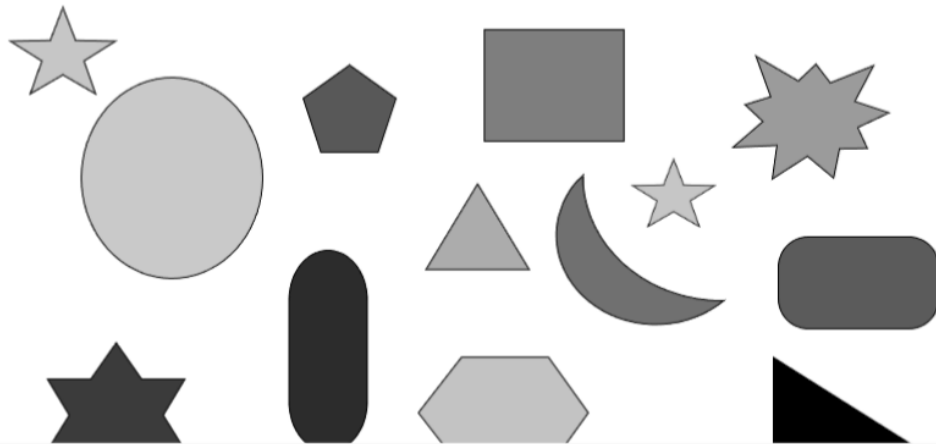
image = cv2.imread('demo1.png',1)
cv2_imshow(image)
```



Preprocessing the Image

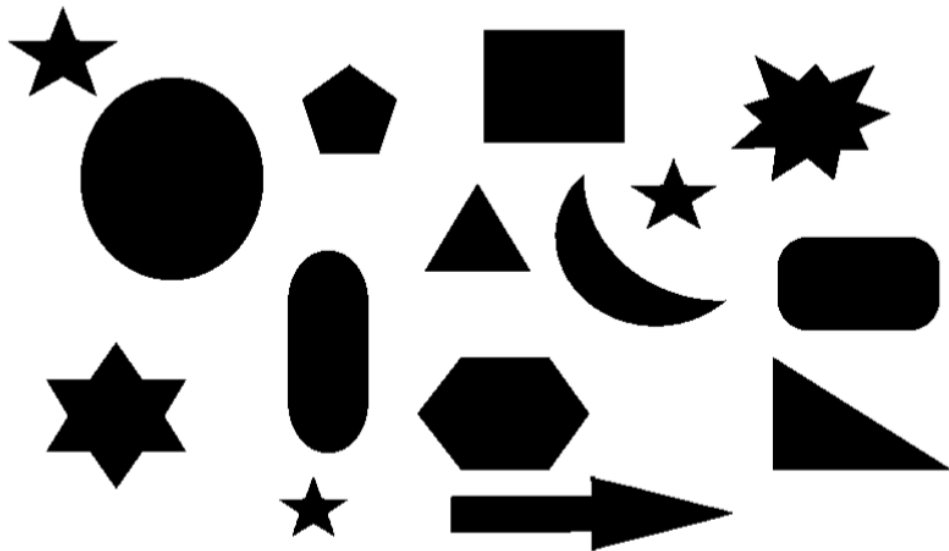
```
# The image is first converted to a gray scale image
# This is because thresholding can only be applied to gray-scale images

gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2_imshow(gray_image)
```



```
# Simple Thresholding is applied to the image.
# Applying Thresholding would give us a binary image,
# using which we could expect a better accuracy in detecting contours.
# syntax: cv2.threshold(source_image, thresh_value, max_value, threshold_type, dest)
# thresh_value can be adjusted according to the image

_, thresh = cv2.threshold(gray_image, 205, 255, 0)
cv2.imshow('thresh', thresh)
```



Contour Detection

```
# Contours are detected using cv2.findContours()
# Syntax: cv2.findContours(image, mode, method, contours=None, hierarchy=None, offset=0)
```

```
contours, heirarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMI
```

Display Contours

```
# contours are displayed on the original image by using cv2.drawContours()  
# Syntax: cv2.drawContours(image, contours, contourIndex, color, thickness)  
  
# to display all contours in the image, we set index to -1  
index = -1  
  
# color of contours is set to blue (BGR)  
color = (255,0,0)  
  
# thickness is set to 2  
thickness = 2  
  
cv2.drawContours(image, contours, index, color, thickness)  
cv2.imshow(image)
```

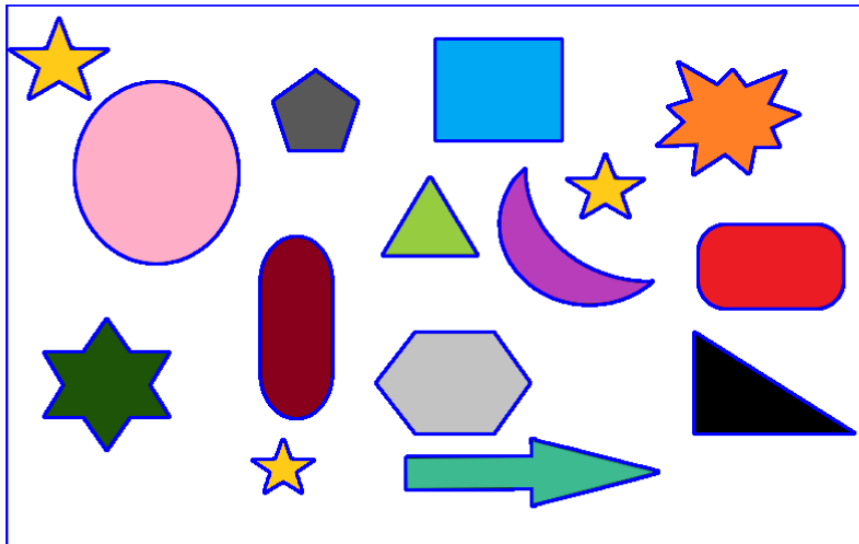
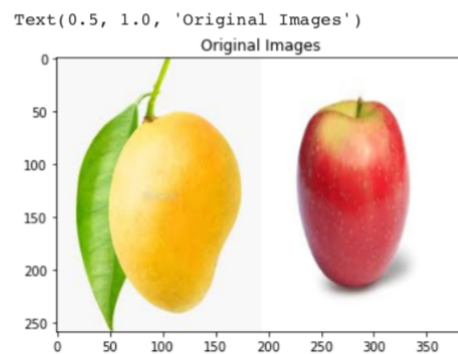


Image Blending

```
import cv2
from matplotlib import pyplot as plot
import numpy as nm
```

```
img_1 = cv2.imread("mango.jpg")[:, :, :-1]
img_2 = cv2.imread("apple.jpg")[:, :, :-1]
```

```
img_2 = cv2.resize(img_2, (img_1.shape[1], img_1.shape[0]))
disp_img = nm.hstack((img_1, img_2))
plot.imshow(disp_img)
plot.title("Original Images")
```



```
blended_img = cv2.addWeighted(img_1, 0.45, img_2, 0.55, 0)
```

```
plot.imshow(blended_img)
plot.title("Blended Images")
```

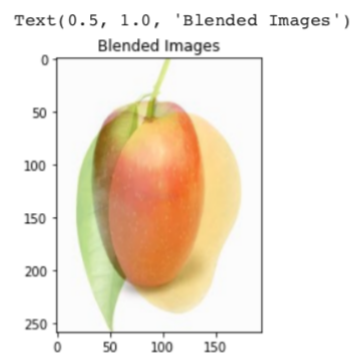


image masking

```
#import the required libraries
import cv2
from google.colab.patches import cv2_imshow
import numpy as np
#read the input image
img = cv2.imread("ppp")
#show the original image
cv2_imshow(img)
```

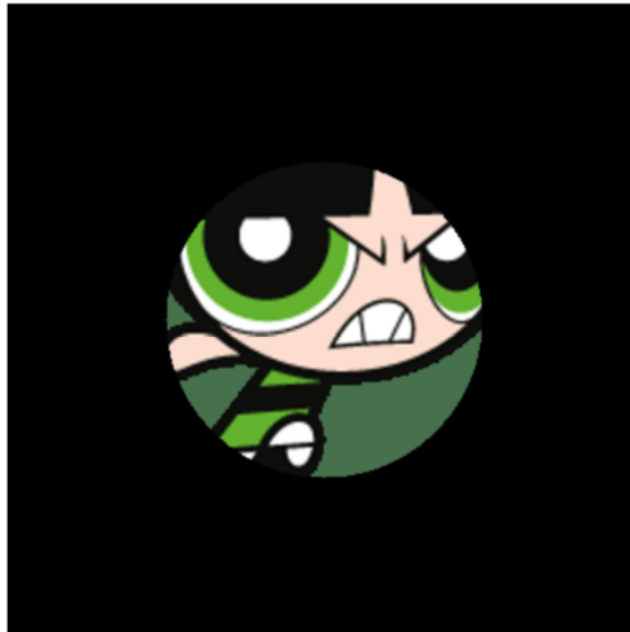


```
#create mask
mask = np.zeros(img.shape[:2], dtype = "uint8")
(cX, cY) = (img.shape[1] // 2, img.shape[0] // 2)
cv2.circle(mask, (cX, cY), 100, 255, -1)
#apply mask
masked = cv2.bitwise_and(img, img, mask = mask)
#Show masked image
cv2.imshow(masked)
```

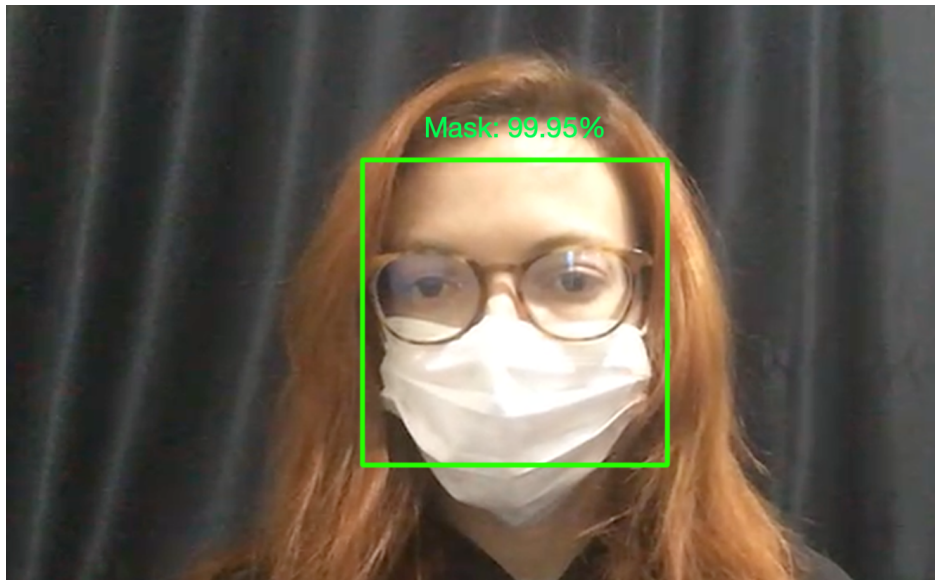
[/colab.research.google.com/drive/1k78zyiqir1B6N0TikE9kkl-o_d5yYXet#scrollTo=c7fgaL~7IKhc&printMode=true](https://colab.research.google.com/drive/1k78zyiqir1B6N0TikE9kkl-o_d5yYXet#scrollTo=c7fgaL~7IKhc&printMode=true)

/2021, 20:07

img_processing4.ipynb - Colaboratory



**FACE MASK DETECTION USING OPENCV
WITH MASK :**



WITHOUT MASK:

