MAE 6291
Internet of Things for Engineers

Prof. Kartik Bulusu, CS Dept.

Week 8 [03/19/2025]	• Setting up the Edge Compute framework • Practical Introduction to Image processing and Filtering for Edge Compute Applications • Guest lecture: Designing Hardware to Improve Your Software - A Use Case in Sound by Jacob Whitton	• In-class Raspberry Pi Lab with PiNOIR camera • Practical Introduction to OpenCV library in Python on the Raspberry Pi 4B • Sobel, Laplacian, and Gaussian filtering on Raspberry Pi 4B • Edge detection using thresholding and Otsu's method on Raspberry Pi 3B+

git clone https://github.com/gwu-mae6291-iot/spring2025_codes.git

GW School of Engineering & Applied Science
THE GEORGE WASHINGTON UNIVERSITY

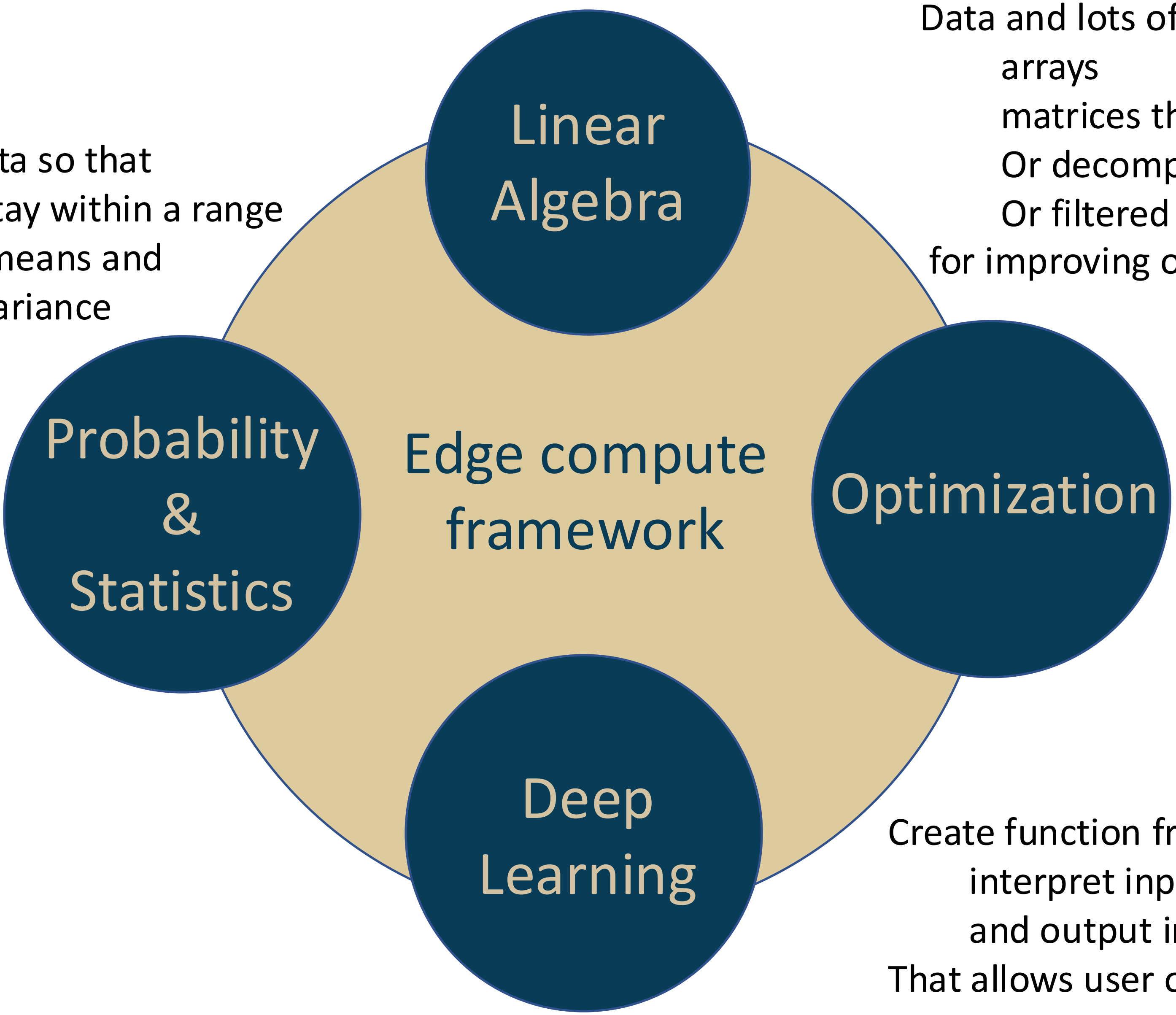
Spring 2025

Photo: Kartik Bulusu

Source:
https://youtu.be/Cx5Z-OslNWE
Strang, G., Linear Algebra and Learning from Data (2019)

Data and lots of data
arrays
matrices that are special and can be factored
Or decomposed
Or filtered
for improving our understanding of the physical phenomena

Monitor data so that
they stay within a range
means and
variance



Finding matrices
that transform data
and minimize errors
Memory intensive process that usually
happens at cloud-level

Create function from data at cloud-level
interpret input data at edge-level
and output information at edge-level
That allows user or system to take decisions

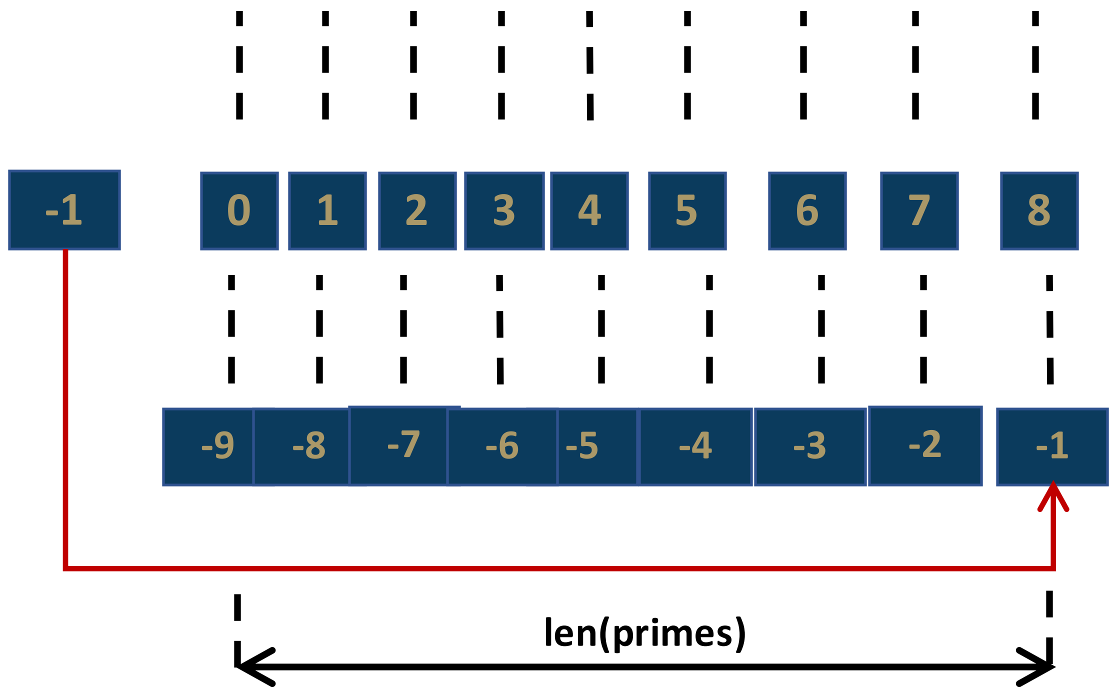
Review of the building blocks:

1. list – Python object-type
2. Matrix operations in Python

Indexing and Slicing Lists

Retrieve list-elements with a range of values

```
>>> primes = [2, 3, 5, 7, 9, 11, 13, 17, 19]
```



start stop

```
>>> primes[2:5]
[5, 7, 9]
```

start stop step

```
>>> primes[0:7:2]
[2, 5, 9, 13]
```

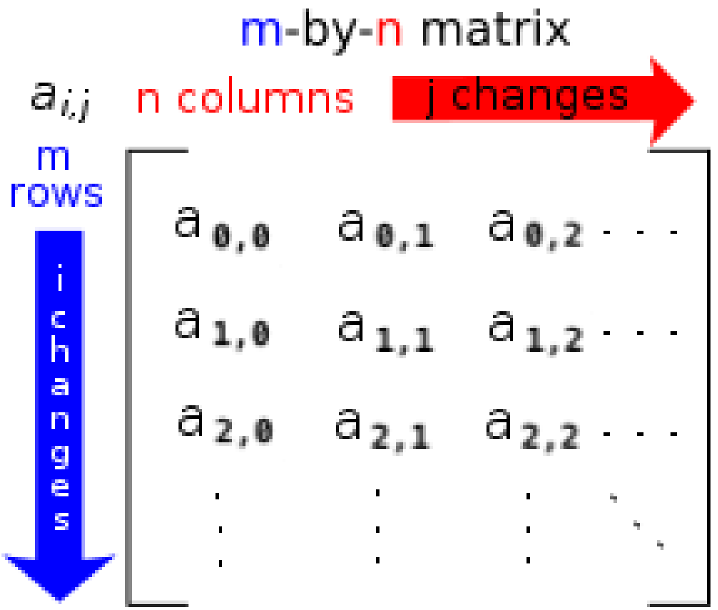
start stop step

```
>>> primes[8:2:-2]
[19, 13, 9]
```

start:	at the index value
step:	up or down at the increment value (default = 1)
stop:	at the index value but not including it

Photo: Kartik Bulusu

Bookkeeping in a Matrix



Source: http://en.wikipedia.org/wiki/Matrix_(mathematics)

The ORDER of a matrix

- $A_{m \times n}$ *is* $m \times n$
- *Read as "m-by-n"*

a_{ij} *is called an ELEMENT*

- *at the* i^{th} *row and* j^{th} *column of A*

Python:

```
>>> import numpy as np
>>> A = np.matrix([[-1, 2],[3, 4]])
>>> A[0,0]
>>> A[0,:]
>>> A[:,0]
>>> A[1,0]
```

`A[row-0:row-M,column0:columnN]`

Photo: Kartik Bulusu

Matrix scalar operations

$$A = \begin{bmatrix} -1 & 2 \\ 3 & 4 \end{bmatrix} \ \& \ s = 6$$

- Matrix, ***A*** has ***m rows*** and ***m columns***
- ***The ORDER of matrix, A ??***
- ***The ORDER of the scalar, s ??***

Scalar Multiplication and Division

- **Each element a_{ij}**
- *Is either **multiplied** with or **divided** by **s***

$$\begin{cases} \underset{(mxm)}{A} * \underset{(1x1)}{s} = \underset{(mxm)}{D} \\ \underset{(mxm)}{A} * \underset{(1x1)}{s^{-1}} = \underset{(mxm)}{F} \end{cases}$$

$$\begin{bmatrix} -1 & 2 \\ 3 & 4 \end{bmatrix} * 6 = \begin{bmatrix} -6 & 12 \\ 18 & 24 \end{bmatrix}$$

$$\begin{bmatrix} -1 & 2 \\ 3 & 4 \end{bmatrix} * \left(\frac{1}{6}\right) = \begin{bmatrix} -\frac{1}{6} & \frac{1}{3} \\ \frac{1}{2} & \frac{2}{3} \end{bmatrix}$$

Python:

```
>>> import numpy as np
>>> A = np.matrix([[-1, 2],[3, 4]])
>>> B1 = A * 6
>>> B2 = A * (1/6)
>>> len(B1)
>>> np.shape(B2)
```

python™

Think of
Array, A as an image
Scalar, s as brightness

Created by Guilherme Appolinário from the Noun Project

Matrix-matrix operations

$$\left\{ \underset{(mxn)}{A} \; x \; \underset{(nxp)}{B} = \underset{(mxp)}{C} \right.$$

- Matrix, *A* has *m rows* and *n columns*
- Matrix, *A* has *n rows* and *p columns*
- *The ORDER of matrix, A ??*
- *The ORDER of matrix, B ??*

Matrix Multiplication

$$\underset{(2x2)}{\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}} \underset{(2x1)}{\begin{bmatrix} b_{11} \\ b_{21} \end{bmatrix}} = \underset{(2x1)}{\begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} \\ a_{21}b_{11} + a_{22}b_{21} \end{bmatrix}}$$

$$\begin{bmatrix} -1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 4 \\ -2 \end{bmatrix} = \begin{bmatrix} -4-4 \\ 12-8 \end{bmatrix} = \begin{bmatrix} -8 \\ 4 \end{bmatrix}$$

Python:

```
>>> import numpy as np
>>> A = np.matrix([[-1, 2],[3, 4]])
>>> B = np.matrix([[4], [-2]])
>>> C = np.dot(A, B)
>>> len(C)
>>> np.shape(C)
```

Think of

Array, A as an image

Array, B as a transformation

Created by Guilherme Appolinário from the Noun Project

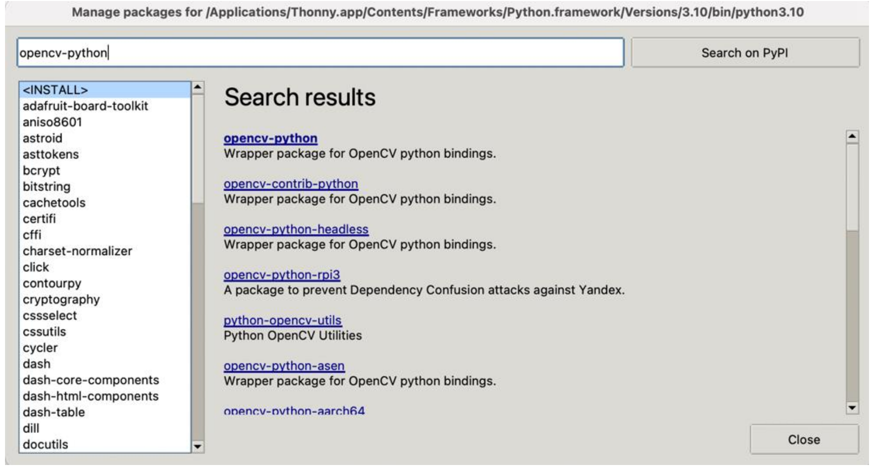
Source: https://en.wikipedia.org/wiki/OpenCV

Explore Image Processing with OpenCV - Python library

OpenCV

OpenCV (**Open Source Computer Vision Library**) is a library of programming functions mainly for real-timecomputer vision.[1]

Originally developed by Intel, it was later supported by Willow Garage, then Itseez (which was later acquired by Intel[2]). The library is cross-platform and licensed as free and open-source softwareunder Apache License 2. Starting in 2011, OpenCV features GPU acceleration for real-time operations.[3]



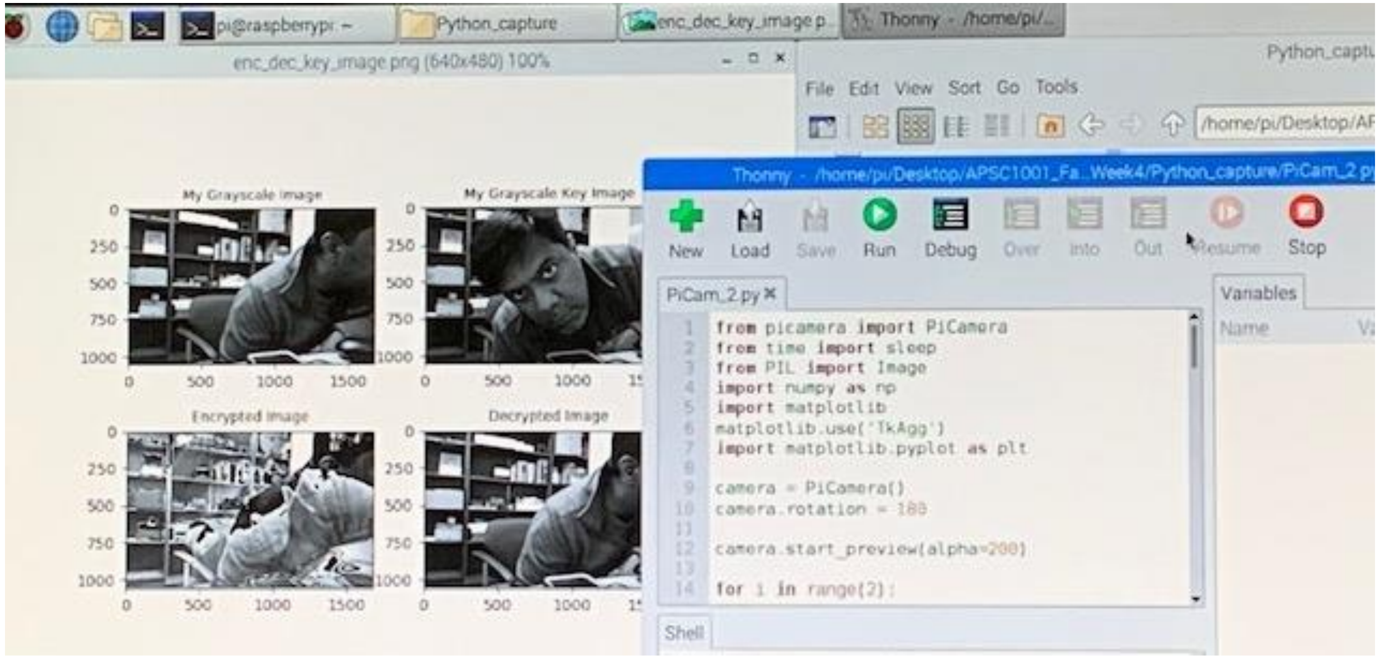
>>> sudo pip install opencv-python

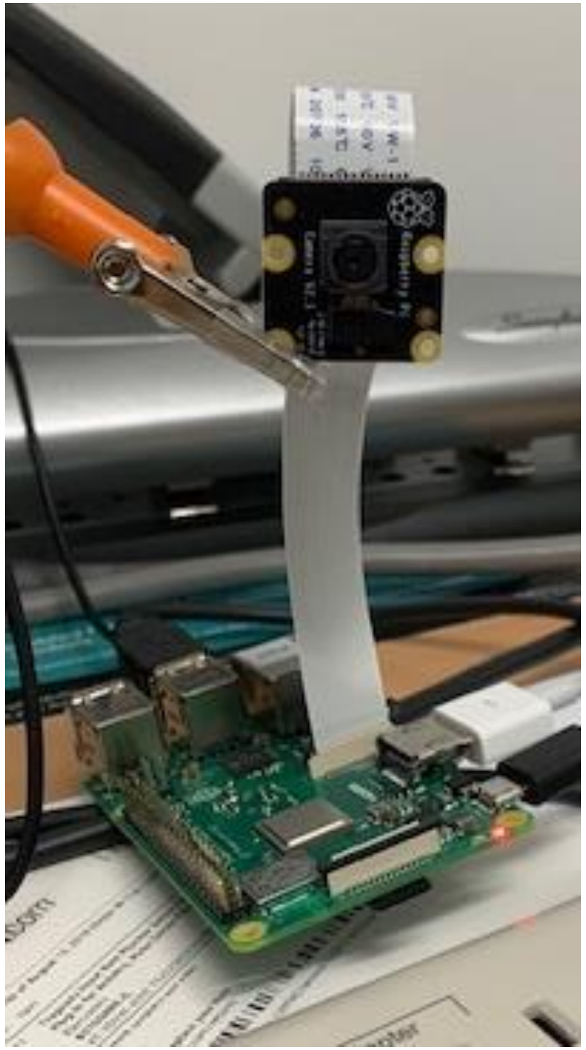
Let’s mess with the PiCamera

Graded in-class lab
Download codes from shared-drive and demonstrate
[10 points]

Use Raspberry Pi and PiCamera

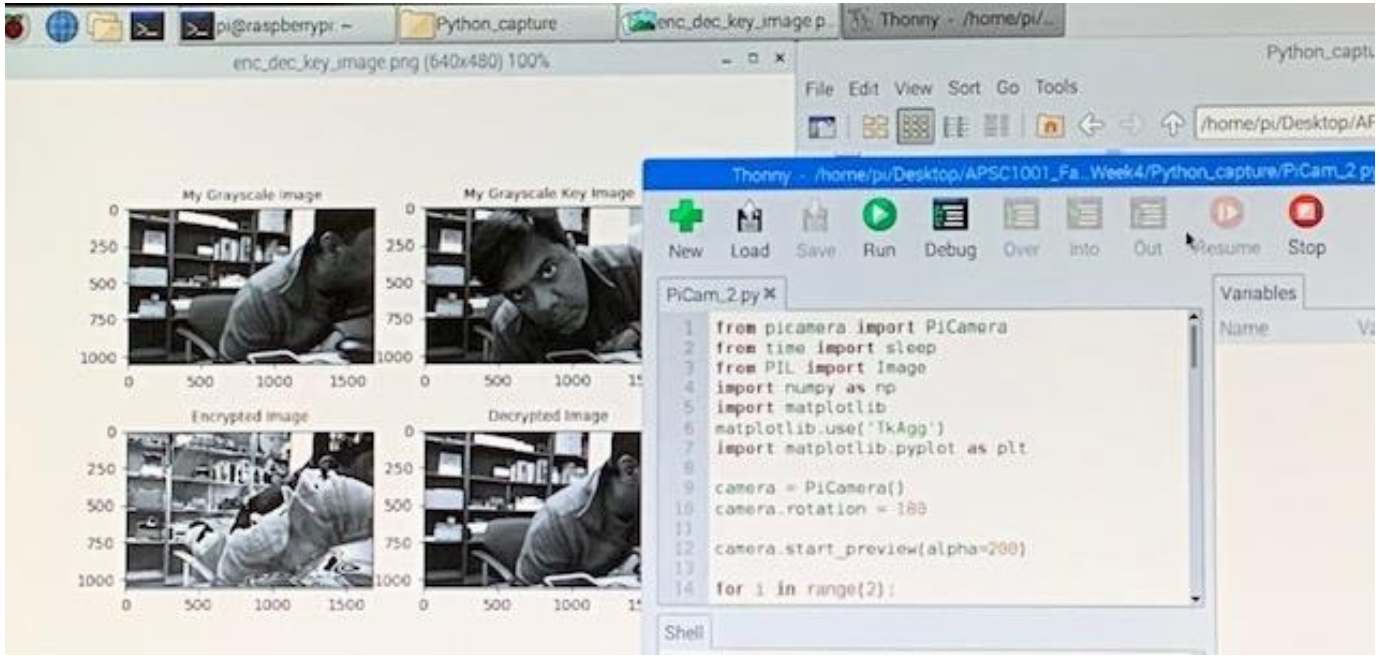
1. To acquire images
2. To filter and transform image data
3. To detect objects using low-level ML functions

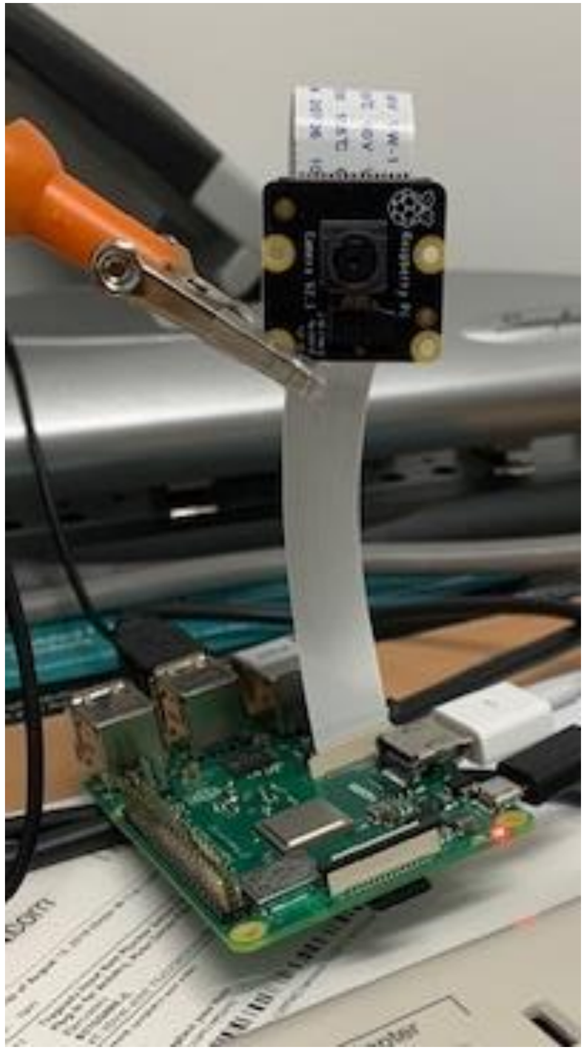




Use Raspberry Pi and PiCamera

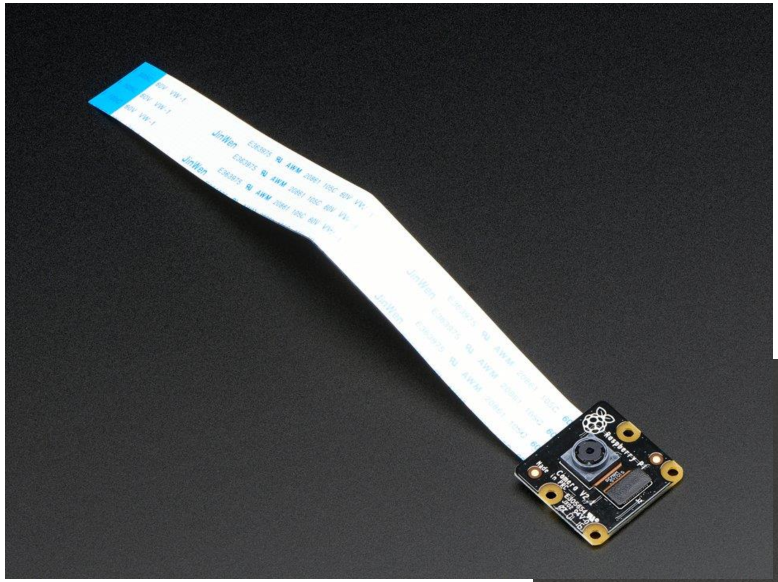
1. To acquire images
2. To perform on-board encryption-decryption
3. To filter and transform image data
4. To detect objects using low-level ML functions

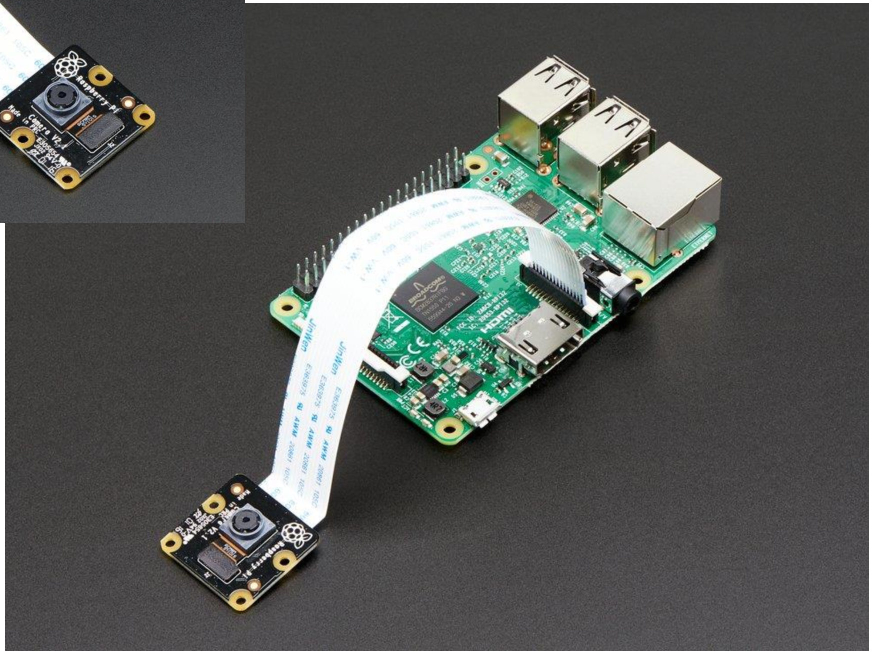




Broad specs of the Pi NoIR Camera

Source:
https://www.adafruit.com/product/3100#description





- 8 megapixel native resolution high quality Sony IMX219 image sensor
- 3280 x 2464 pixel static images
- Capture video at
  - 1920 x 1080 p30
  - 1280 x 720 p60
  - 640 x 480 p90 resolutions
- No Infrared (NoIR) filter
  - Infrared photographs or photographing objects in low light (twilight) conditions

Picamera2

OpenCV

Raspberry Pi hardware and Camera hardware interactions

Low-level image acquisition and processing

Computer vision on the Raspberry Pi

Skeleton of the OpenCV library-based Python program

Picamera2

```
# import the necessary packages

from picamera2 import Picamera2
import time
import cv2
```

Picamera2

```
# initialize the camera and grab a reference
# to the raw camera capture

camera = Picamera2()
camera.resolution = (320, 240)
rawCapture = camera.capture_array("main")
```

```
# allow the camera to warmup
time.sleep(0.1)
```

Picamera2

```
# grab an image from the camera

camera.capture(rawCapture, format="bgr")
image = rawCapture.array
```

```
#start the camera and display a preview
camera.start(show_preview=True)

# grab an image from the camera
camera.capture(rawCapture, format="bgr")
image = rawCapture.array
```

```
# display the image on screen
# and wait for a keypress

cv2.imshow("Image", image)
cv2.imwrite("savedImage.png", image)
cv2.waitKey(0)

cv2.destroyAllWindows()
camera.stop()
exit()
```

Goal of the lab segment

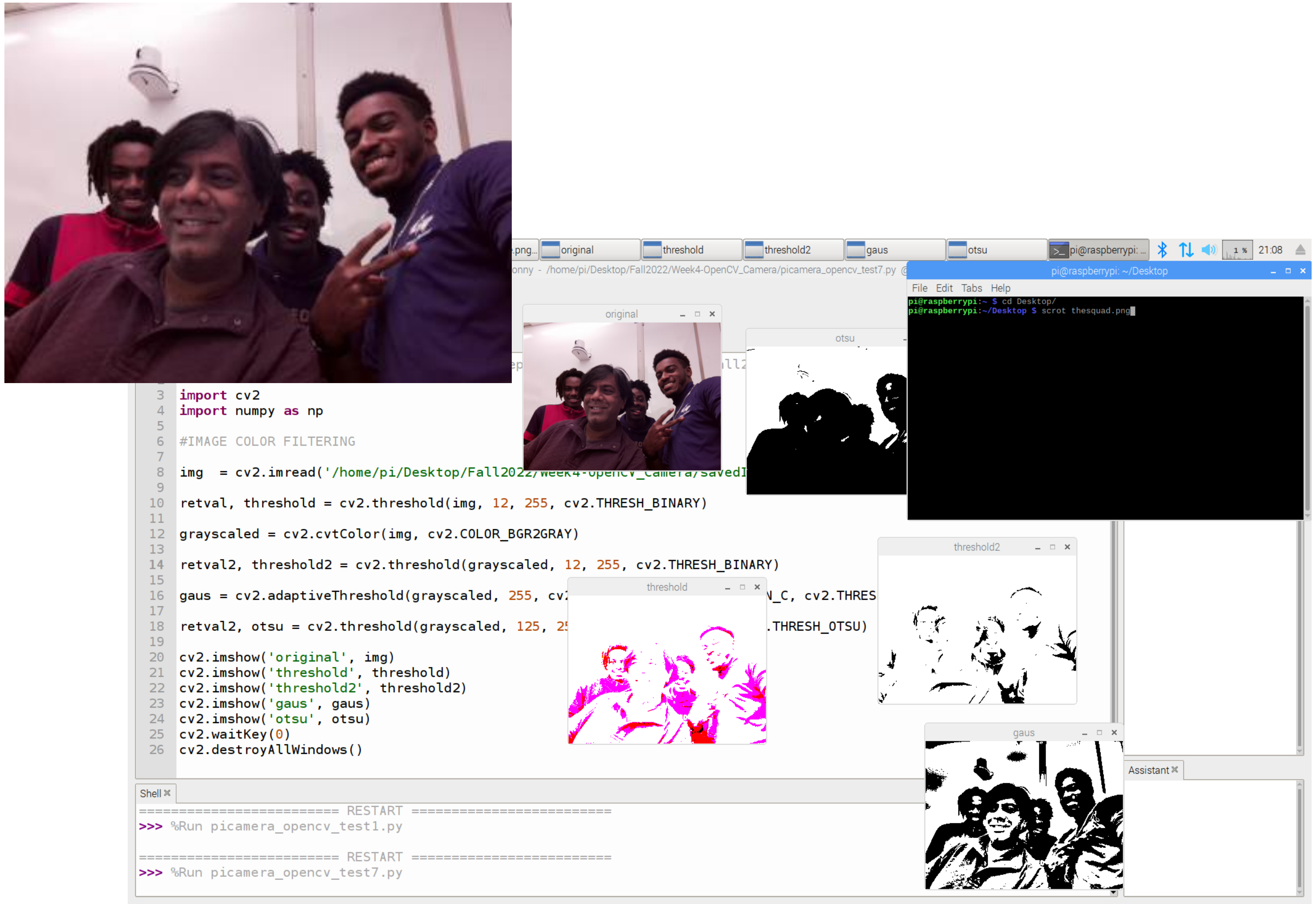
Co-work

- Observe, ask and try in groups

Make

- Build-a-hack
- Use Pi NoIR Camera to acquire an images
- import OpenCV library

Perform basic image processing functions using OpenCV



Very basics of data encryption-decryption and applications

Encryption is the **transformation of data** into some **unreadable form**.

Decryption is the reverse of encryption; it is the **transformation of encrypted data** back into some **intelligible form**.

Image encryption - process of encoding image with the help of an encryption algorithm in such a way that unauthorized users can't access it.

Authorization entails a "key".

Created by Round Icons
from the Noun Project

Building up the vocabulary: Shift cipher

Assign each letter in the alphabet a number

- Start from 0
- I have row matrix: A_{1x26}

Message (X) = T E A C H

- Convert the letter into the number that matches its order in the alphabet starting from 0
- I now have a row matrix: X_{1x5} = [19 4 0 2 7]

To encrypt assign a shift key (K) = 4

- Must be an integer from 0 to 25
- Map each letter to a different letter using the shift key
- Y = (X+KJ) **where *J is a vector-of-ones i.e., [1 1 1 1 ….]***
- I have a new row matrix: Y_{1x5} = [23 8 4 6 11]

Encrypted message (Y) = X I E G L

To decrypt apply the same shift key (K) = 4

- Map each encrypted letter to a different letter using the shift key
- X = (Y-KJ) **where *J is a vector-of-ones i.e., [1 1 1 1 ….]***
- I have a new row matrix: X_{1x5} = [19 4 0 2 7]

Decrypted Message (X) = T E A C H

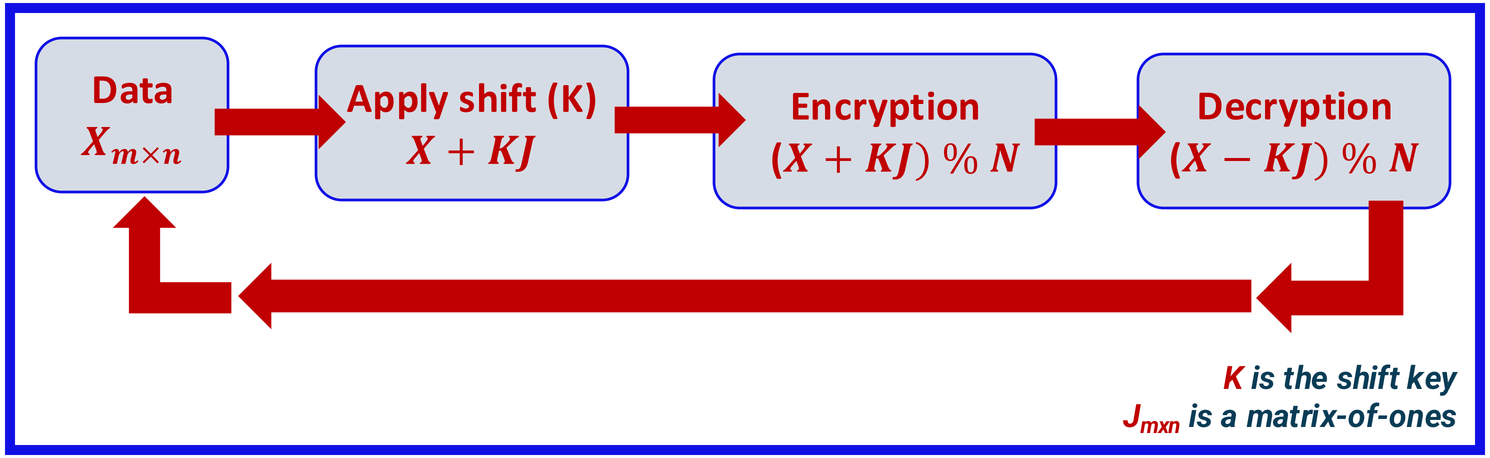
encrypt

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

decrypt

Building up the vocabulary: Modulo

In computing, the **modulo operation** finds the **remainder after division** of one number by another.



Encryption

	T	E	A	C	H
X	19	4	0	2	7
K	20	20	20	20	20
X + K	39	24	20	22	27
(X + K)%N	13	24	20	22	1
	N	Y	U	W	B

Decryption

	N	Y	U	W	B
Y	13	24	20	22	1
K	20	20	20	20	20
Y - K	-7	4	0	2	-19
(Y - K)%N	19	4	0	2	7
	T	E	A	C	H

The in-class programming exercise will demonstrate these operations on images using Python

Apply Python to process the image

import cv2

import PIL

Perform encryption and decryption

Source:
https://www.geeksforgeeks.org/python-thresholding-techniques-using-opencv-set-1-simple-thresholding/

Apply OpenCV + Python to filter the image

Simple Thresholding

The basic Thresholding technique is Binary Thresholding.
For every pixel, the same threshold value is applied.
If the pixel value is smaller than the threshold, it is set to 0, otherwise, it is set to a maximum value.

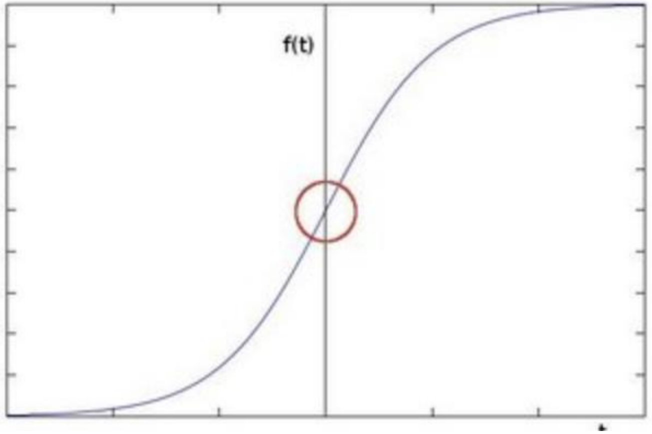
cv2.threshold(*source*, *thresholdValue*, *maxVal*, *thresholdingTechnique*)

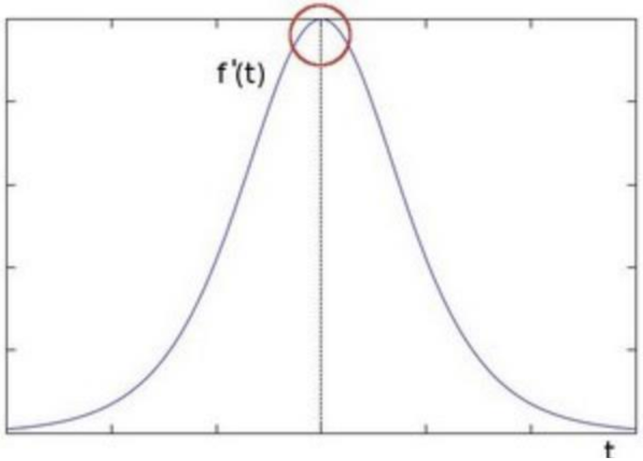
Parameters:

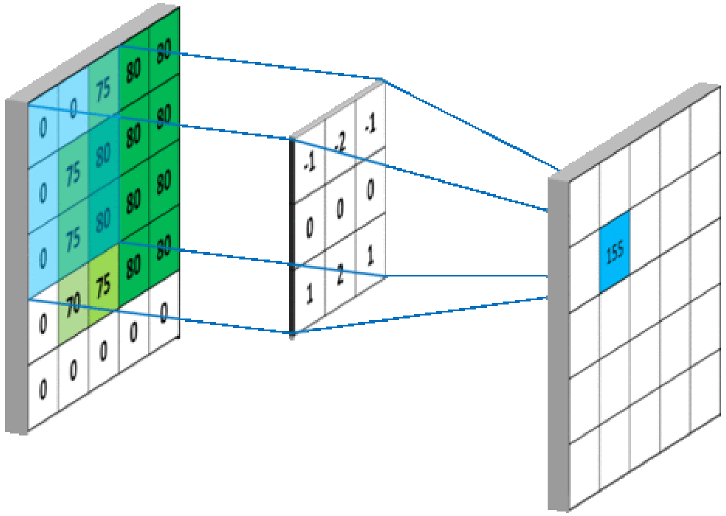
- **source**: Input Image array (must be in Grayscale).
- **thresholdValue**: Value of Threshold below and above which pixel values will change accordingly.
- **maxVal**: Maximum value that can be assigned to a pixel.
- **thresholdingTechnique**: The type of thresholding to be applied.

Apply OpenCV + Python to detect edges

Source:
https://mlnotebook.github.io/post/CNN1/
https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html







```
laplacian = cv2.Laplacian(frame, cv2.CV_64F)
sobelx = cv2.Sobel(frame, cv2.CV_64F, 1, 0, ksize=5)
sobely = cv2.Sobel(frame, cv2.CV_64F, 0, 1, ksize=5)
edges = cv2.Canny(frame, 100, 200)
```

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

$$G = \sqrt{G_x^2 + G_y^2}$$

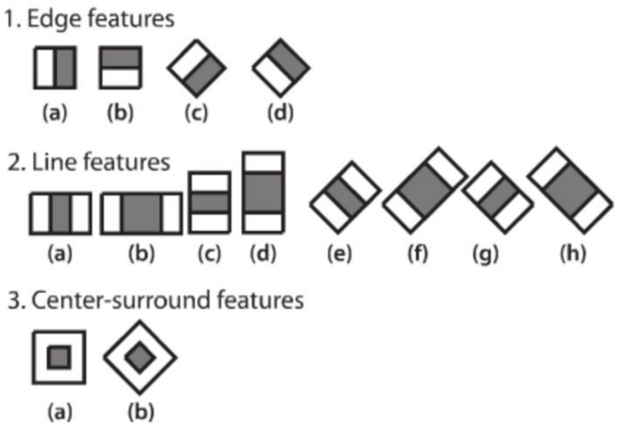
Source:
https://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_Object_Detection_Face_Detection_Haar_Cascade_Classifiers.php
https://docs.opencv.org/3.4/d0/de3/citelist.html#CITEREF_Dalal2005

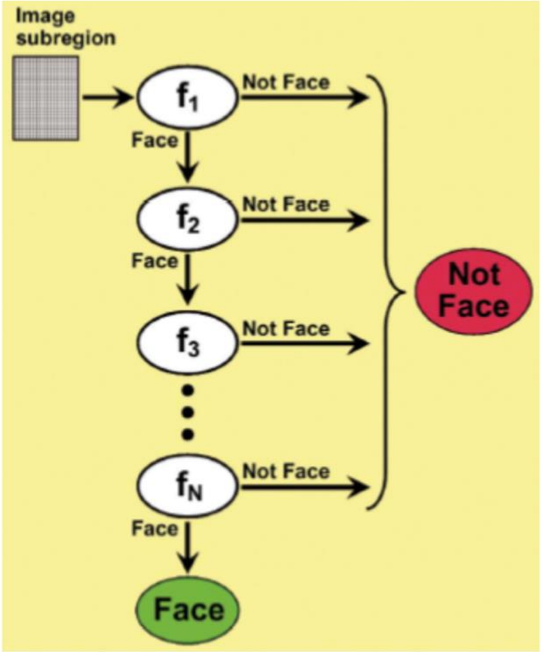
Apply OpenCV + Python to detect objects

face_cascade = **cv2.CascadeClassifier**("/home/pi/opencv-3.4.1/data/haarcascades/**haarcascade_frontalface_default.xml**")

Haar features

OpenCV's algorithm is currently using the following Haar-like features which are the input to the basic classifiers:





OpenCV's pre-trained classifiers

OpenCV already contains many pre-trained classifiers for face, eyes, smile etc. Those XML files are stored in **opencv/data/haarcascades/** folder:

```
~/OpenCV/opencv/data/haarcascades$ ls

haarcascade_eye_tree_eyeglasses.xml   haarcascade_mcs_leftear.xml
haarcascade_eye.xml                   haarcascade_mcs_lefteye.xml
haarcascade_frontalface_alt2.xml      haarcascade_mcs_mouth.xml
haarcascade_frontalface_alt_tree.xml  haarcascade_mcs_nose.xml
haarcascade_frontalface_alt.xml       haarcascade_mcs_rightear.xml
haarcascade_frontalface_default.xml   haarcascade_mcs_righteye.xml
haarcascade_fullbody.xml              haarcascade_mcs_upperbody.xml
haarcascade_lefteye_2splits.xml       haarcascade_profileface.xml
haarcascade_lowerbody.xml             haarcascade_righteye_2splits.xml
haarcascade_mcs_eyepair_big.xml       haarcascade_smile.xml
haarcascade_mcs_eyepair_small.xml     haarcascade_upperbody.xml
```

Source:
https://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_Object_Detection_Face_Detection_Haar_Cascade_Classifiers.php
Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.
https://www.intel.com/content/www/us/en/docs/ipp/developer-reference/2021-7/histogram-of-oriented-gradients-hog-descriptor.html

Apply OpenCV + Python to detect humans

```
hog = cv2.HOGDescriptor()
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())
```

Implementation of HOG (Histogram of Oriented Gradients) descriptor and object detector.

Histogram of oriented gradients (HOG) is a feature descriptor

- used to detect objects in computer vision and image processing.

The HOG descriptor technique counts

- occurrences of gradient orientation in localized portions of an image - detection window, or region of interest (ROI).