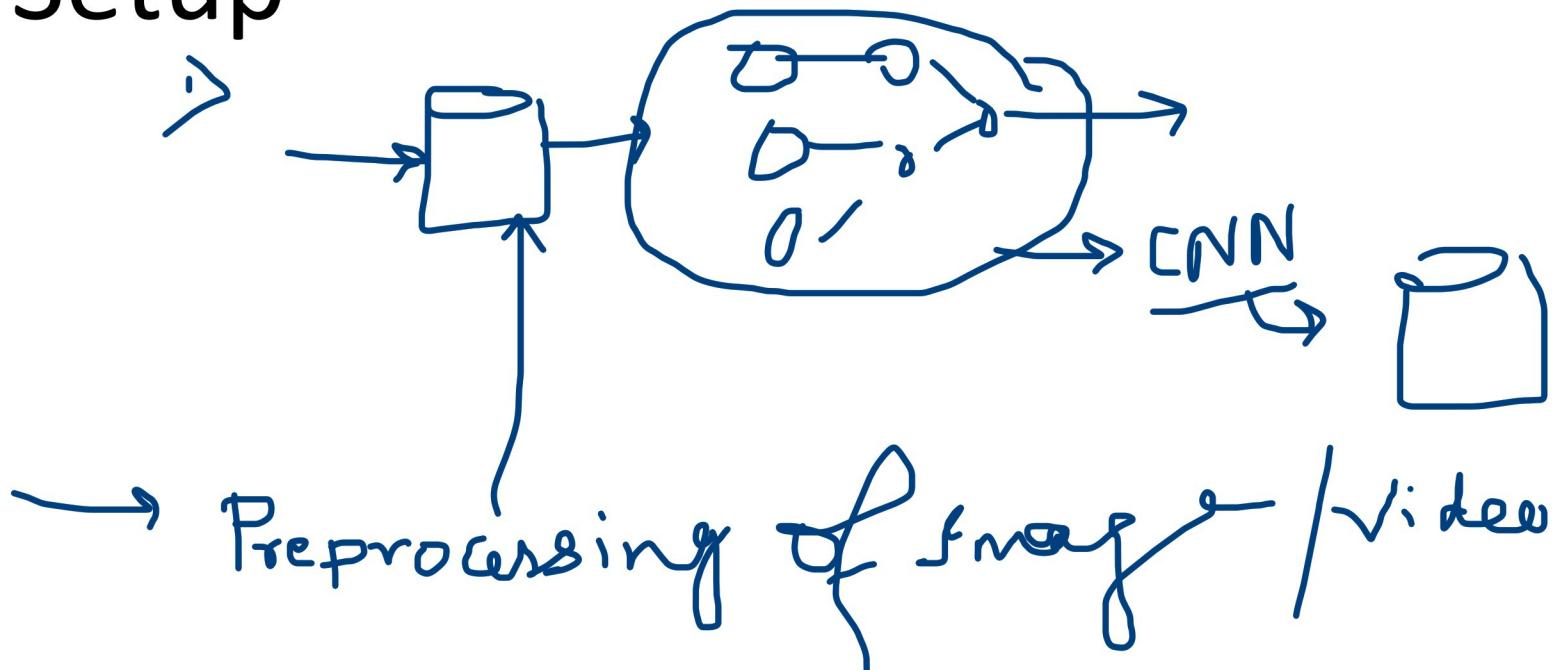




Source: pyimagesearch

Environment Setup

- Anaconda
- Open CV
- Tensorflow



Install anaconda

- Anaconda.org (appropriate python version and OS version)
- Open anaconda prompt
- Type *conda install – c conda-forge opencv*

Virtual environment

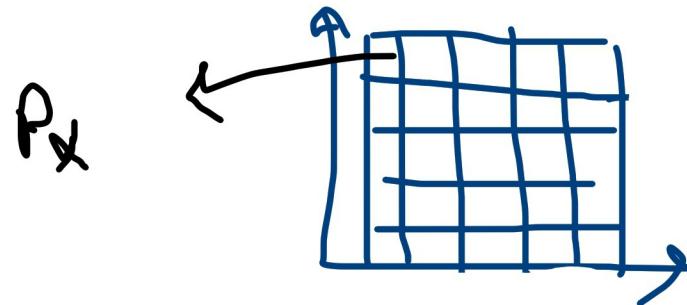
- Type *conda create -n environment_name python =version*
- *environment_name* is your env name and version is 3.6 or 3.7
- Type *source activate environment_name*
- Type *source deactivate*

Introduction to image processing

Agenda

- Images and pixels
- Resolution
- Bitmap Images
- Lossless and lossy compression
- File formats
- Color spaces

Images



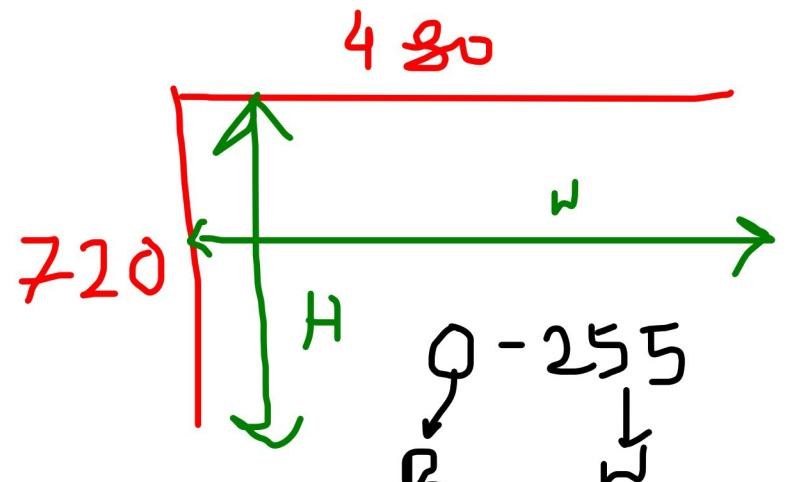
$$P_x \rightarrow 0 - 255$$

Visual representation in a 2 dimensional form is called Image.

Images are collection of pixels

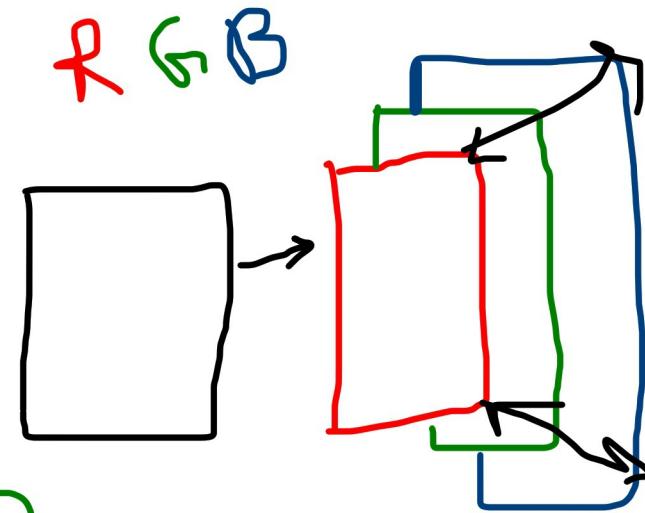
$P_x =$  $\rightarrow 720 \times 480 \text{ px}$

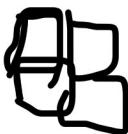
$P[\cdot], G[\cdot], B[\cdot]$



$$\Rightarrow 720 \times 480 = 345600$$

$\approx 4 \text{ MP}$





- The samples here are pixels, smallest elements in any digital space.
 - Like you zoom any photograph.
-
-

Source: www.freeimages.co.uk

Image resolution

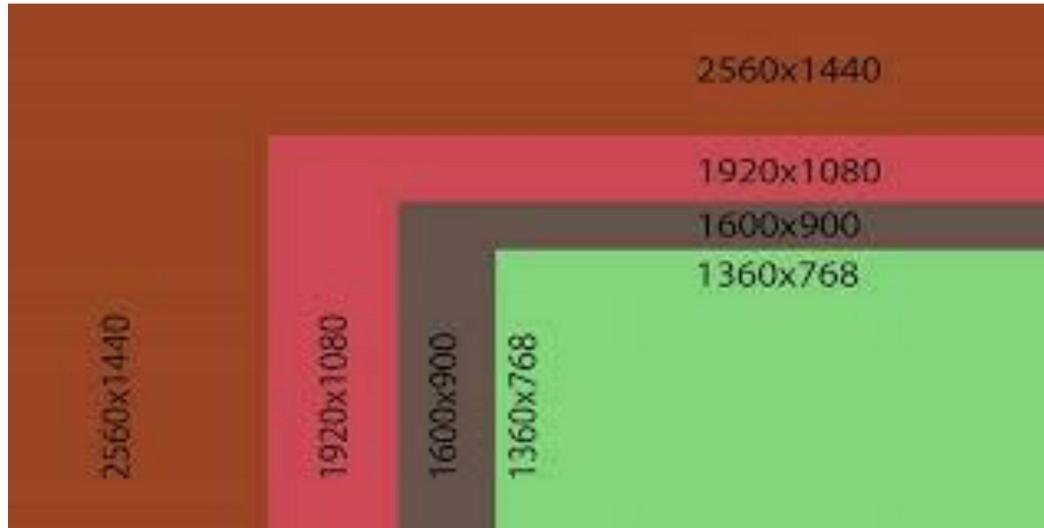


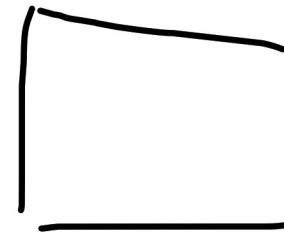
Image resolution comparison

- Image resolution is number of pixels present in an image.
- More pixels == more quality ✓
- For example if you take 1024 X 768 , that means you have 1024 pixel columns and 768 pixel rows total is 786,432 pix

Bitmap Images

$\rightarrow 0 - 255$

bit (0, 1)



- Ideally pixel values are integers, if we convert range of integers to 0s and 1s like bytes it's called a Bitmap image.
- To convert we make white as 1 and black as zero to represent the image.
- Loss less compression: ~~x~~ → fine
 - Reducing the size without losing quality is lossless compression
- Lossy compression: →
 - In lossy compression some data would be lost.

3.4 MP
 ↓
 1.1 MP

BPP = Bit per pixel

1 BPP = 2 colors

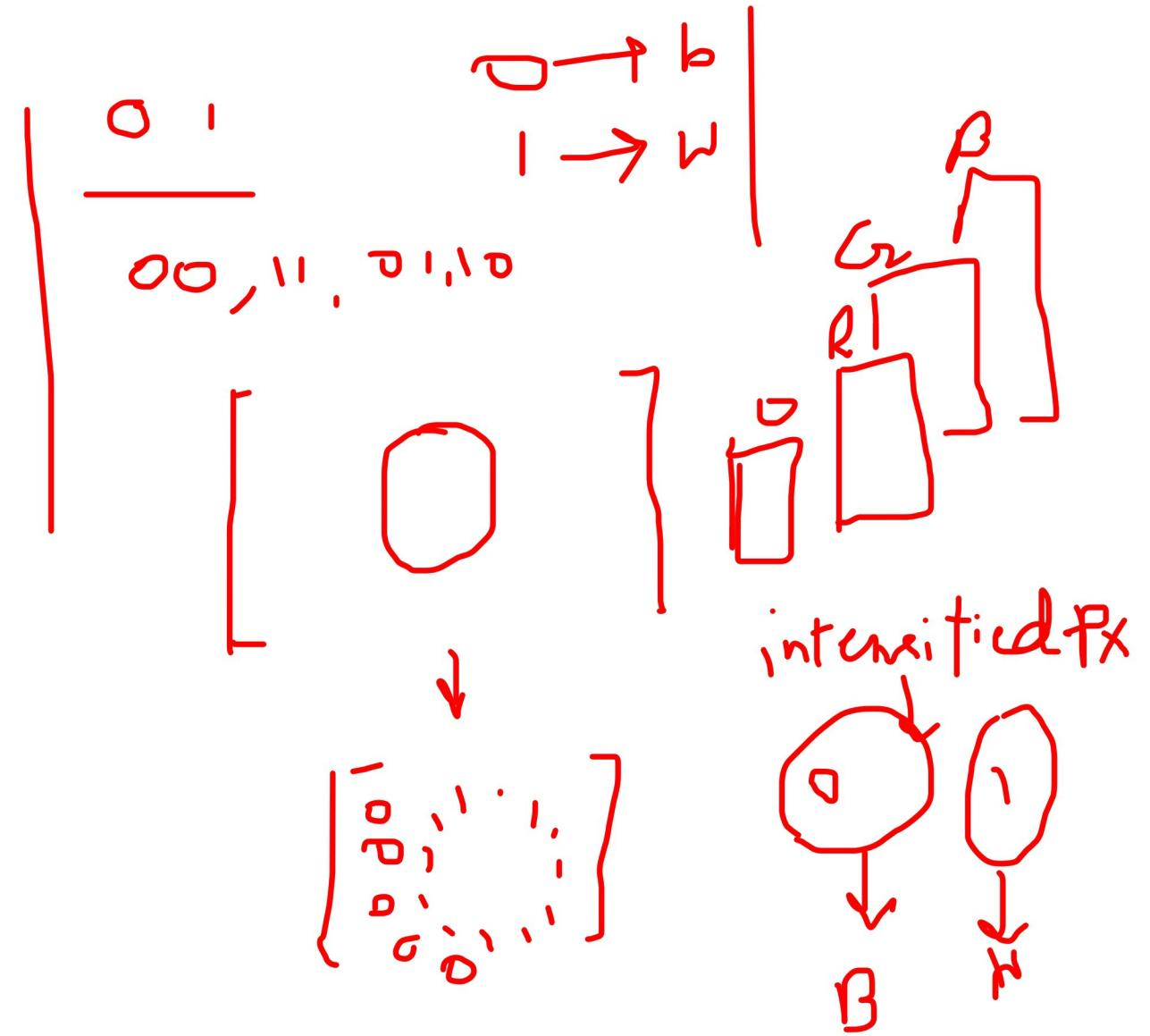
$$2^K = 2^1 \uparrow$$

$$2^2 = 4c$$

$$2^3 = 8c$$

$$2^4 = 16c$$

$$2^5 = 32c$$



R = 255, 0, 0

G = 0, 255, 0

B = 0, 0, 255

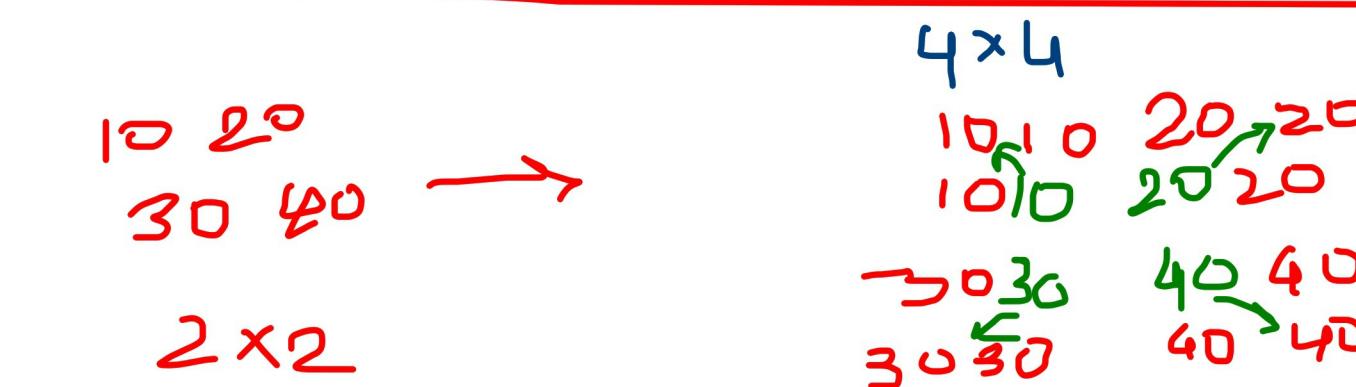
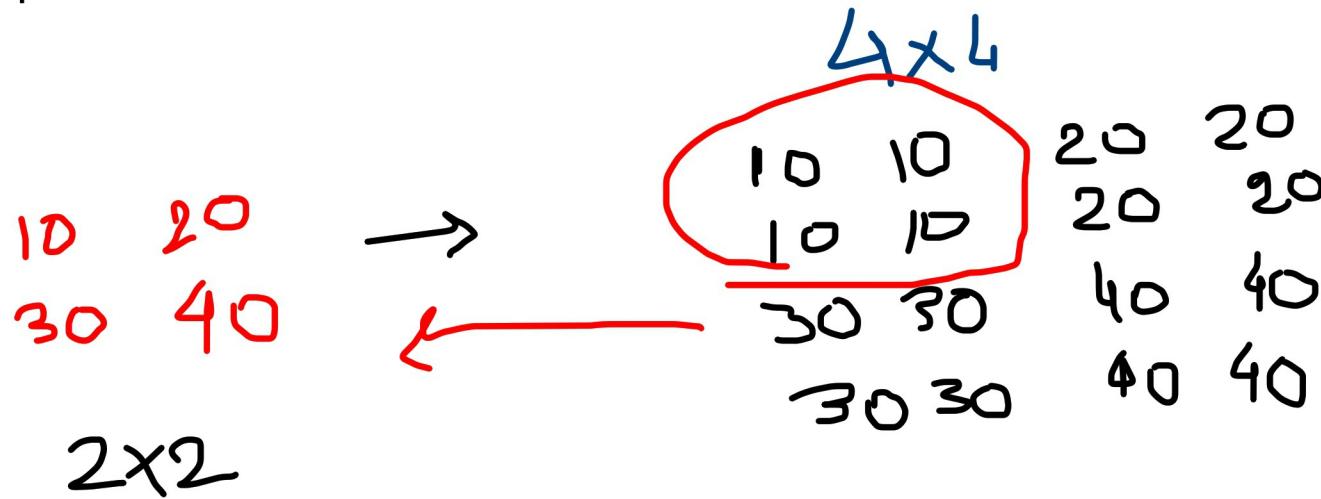
Gray = 128, 128, 128
W = 255, 255, 255
B = 0, 0, 0

$$\left| \frac{255}{2} = 128 \right.$$

Y = 255, 255, 0

Color = (0, 255, 255)

Interpolation



- Inter-nearest: nearest neighbour
 - inter-bilinear: bilinear transformation
 - inter cubic : cubic transformation
 - inter Area: computes the area and then fit the pixels
- 4
- Takes care of sharpness of image.
1. calculates the coefficient from the 2X2 and then multiply with pixels
- 10 10 → 7 10
10 10 → 3 12

Image file formats

Image Format	Description	Use
JPEG	Lossy compression of raw images	Photos and paints
JPEG2000	Optimized form of JPEG, better compression ratio. (lossless and lossy)	Surveillance
TIFF	lossless compression	Document storage
GIF	bitmap image , animations can be done and lossless compression	gaming and animation
BMP	independent of display device	windows
PNG	Lossless compression	image transfer over internet
WebP	comparable quality with JPEG	stickers in messaging apps
SVG	for interactivity and animation	website development

→ F1

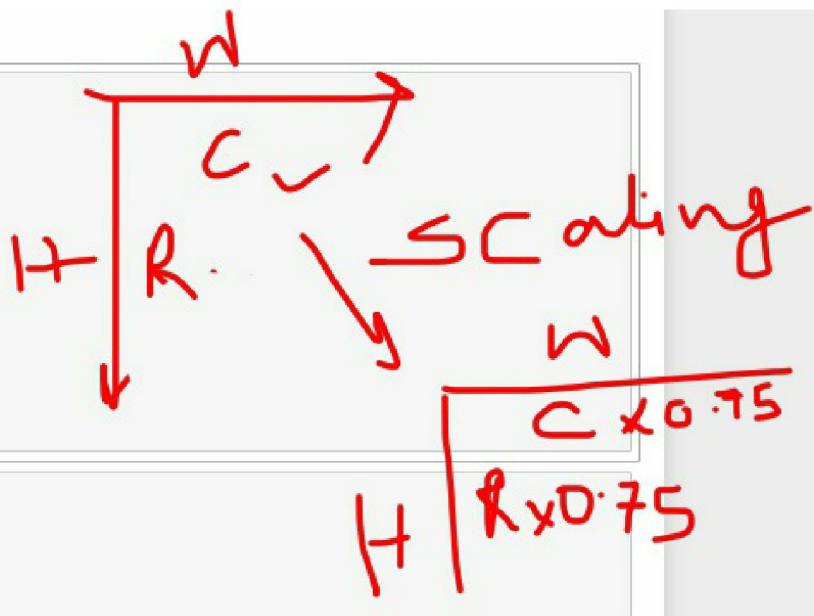
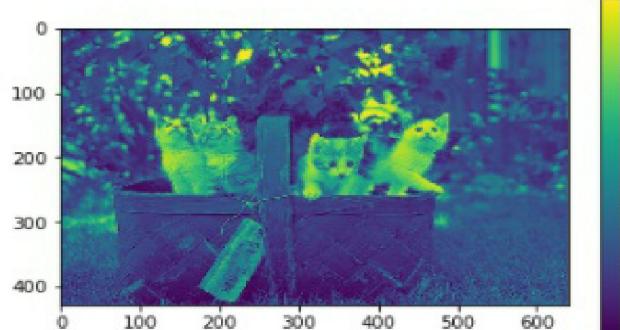
```
0 50 100 150 200 250
```

```
[2]: import cv2 as cv2
vid = cv2.VideoCapture('Resources/Videos/dog.mp4')
while(vid.isOpened()):
    ret, frame = vid.read()
    if ret:
        frameresize = Rescale_Frames(frame)
        cv2.imshow('frame', frame)
        cv2.imshow('resized frame', frameresize)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        break
vid.release()
cv2.destroyAllWindows()
```

```
[3]: def Rescale_Frames(frame, scale=0.75):
width = int(frame.shape[1]*scale)
height = int(frame.shape[0]*scale)
dimensions = (width, height)
return cv2.resize(frame, dimensions, interpolation = cv2.INTER_AREA)
```

```
[ ]:
```

```
Out[7]: <matplotlib.colorbar.Colorbar at 0x1f36cce56d8>
```



```
File Edit Insert Cell Kernel Help
```

Kernel

Smoothing Images

```
vals = np.random.rand(5,5)
# print(vals.shape)
# print(vals)
```

```
vals_ = vals*(1/5)
print(vals_)
```

```
[[0.11203741 0.02947612 0.02540384 0.0532086 0.17745157]
[0.17831626 0.01569529 0.00773126 0.12064265 0.08676238]
[0.03530667 0.10236724 0.0399299 0.03946516 0.10060309]
[0.11848442 0.11154662 0.17313083 0.11182556 0.07051929]
[0.1808257 0.05371926 0.08990013 0.1049047 0.18698858]]
```

$$K = \left(\frac{1}{25}\right) \begin{bmatrix} 5 & 8 & 1 \\ 6 & 9 & 3 \\ 7 & 2 & 4 \end{bmatrix}$$

$$K = \begin{bmatrix} 3 \times 3 \end{bmatrix}$$

jupyter Untitled Last Checkpoint: 7 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Out[32]: -1

Smoothing Images

```
In [35]: vals = np.random.rand(5,5)
# print(vals.shape)
# print(vals)
vals_ = vals*(1/5)
print(vals_)
```

```
[[0.11203741 0.02947612 0.02540384 0.0532086 0.17745157]
[0.17831626 0.01569529 0.00773126 0.12064265 0.08676238]
[0.03530667 0.10236724 0.0399299 0.03946516 0.10060309]
[0.11848442 0.11154662 0.17313083 0.11182556 0.07051929]
[0.1808257 0.05371926 0.08990013 0.1049047 0.18698858]]
```

```
In [36]: kernel = np.ones((5,5))/25
print(kernel)
```

```
[[0.04 0.04 0.04 0.04 0.04]
[0.04 0.04 0.04 0.04 0.04]
[0.04 0.04 0.04 0.04 0.04]
[0.04 0.04 0.04 0.04 0.04]
[0.04 0.04 0.04 0.04 0.04]]
```

$$k = \begin{pmatrix} 3 & 3 \end{pmatrix}^{\frac{n+1}{2}}$$

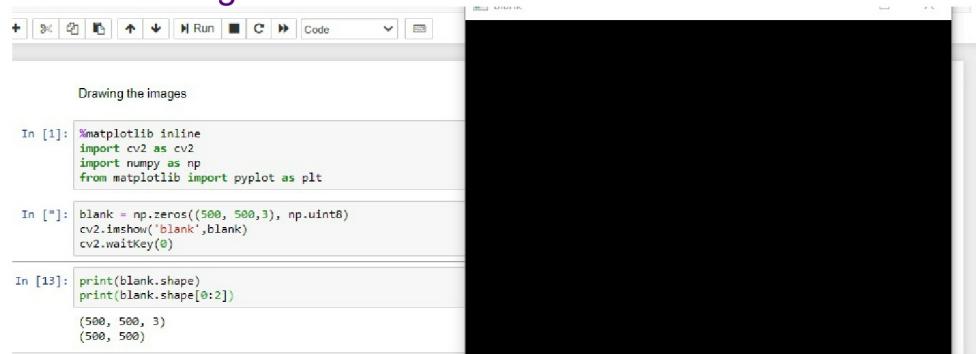
$$k = \text{odd } n/0$$

$$k = 3^{\frac{1}{2}} = 3 \times 3$$

$$\left[\begin{array}{cccc|c} & & & & 1 \\ & & & & 1 \\ & & & & 1 \\ & & & & 1 \\ \hline & & & & 1 \end{array} \right] \times \frac{1}{25} = 0.04 \times 1 = 0.04$$

Drawing -

blank Drawing



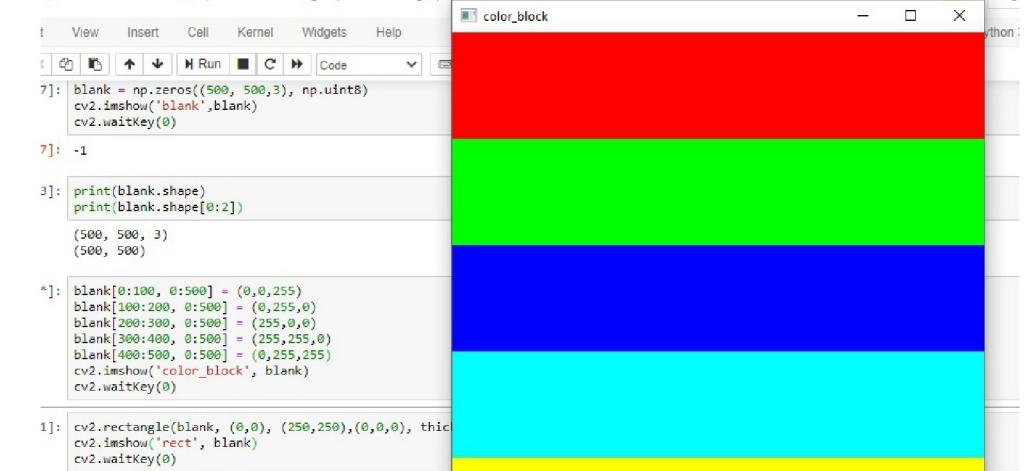
```
! Drawing the images
In [1]: %matplotlib inline
import cv2 as cv2
import numpy as np
from matplotlib import pyplot as plt

In [2]: blank = np.zeros((500, 500,3), np.uint8)
cv2.imshow('blank',blank)
cv2.waitKey(0)

In [3]: print(blank.shape)
print(blank.shape[0:2])

(500, 500, 3)
(500, 500)
```

colouring pixel



```
! colouring pixel
In [1]: blank = np.zeros((500, 500,3), np.uint8)
cv2.imshow('blank',blank)
cv2.waitKey(0)

In [2]: -1

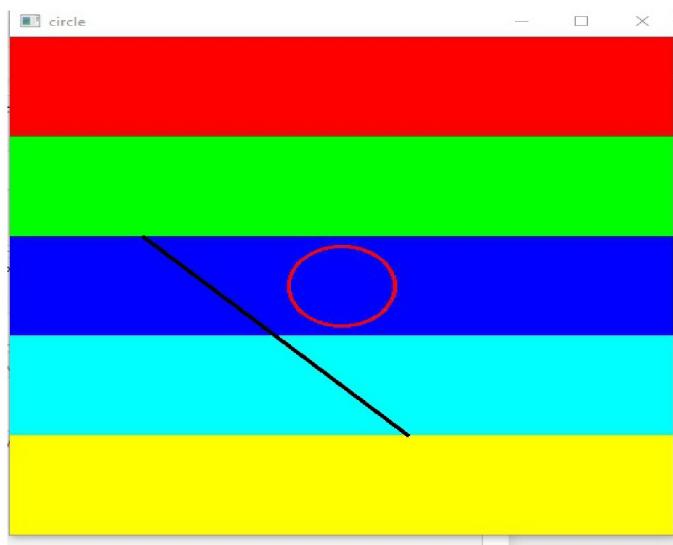
In [3]: print(blank.shape)
print(blank.shape[0:2])

(500, 500, 3)
(500, 500)

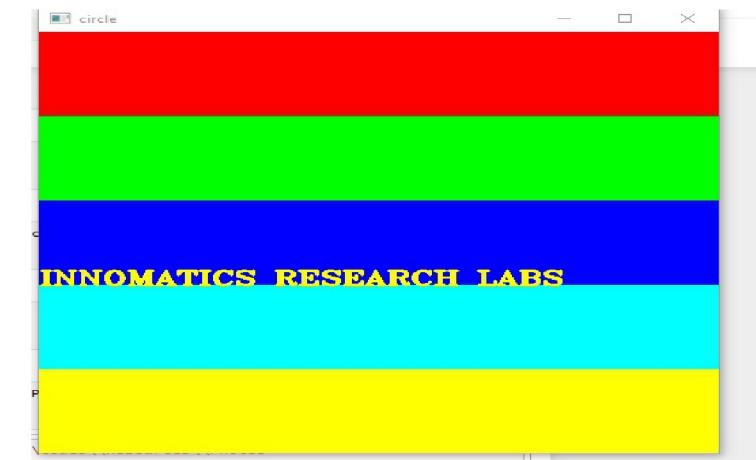
In [4]: blank[0:100, 0:500] = (0,0,255)
blank[100:200, 0:500] = (0,255,0)
blank[200:300, 0:500] = (255,0,0)
blank[300:400, 0:500] = (255,255,0)
blank[400:500, 0:500] = (0,255,255)
cv2.imshow('color_block', blank)
cv2.waitKey(0)

In [5]: cv2.rectangle(blank, (0,0), (250,250),(0,0,0), thickness=2)
cv2.imshow('rect', blank)
cv2.waitKey(0)
```

Drawing Lines, cicle

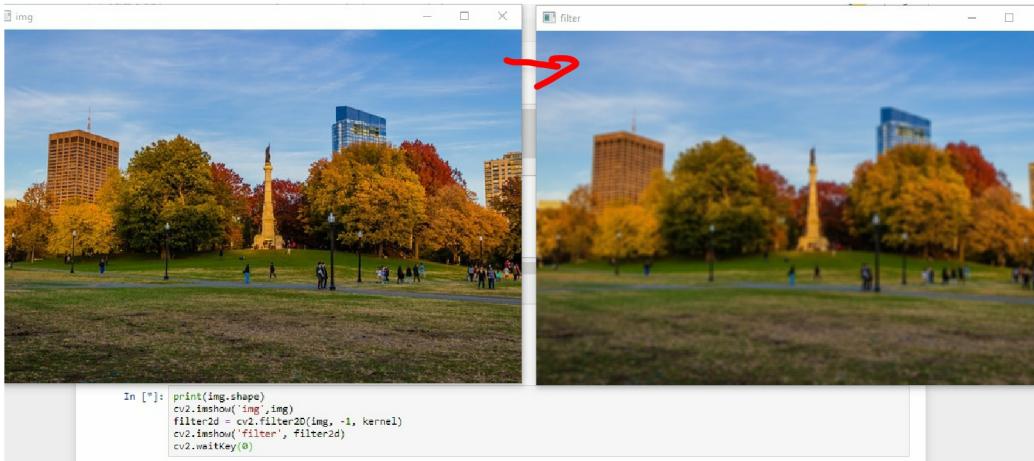


Text applications

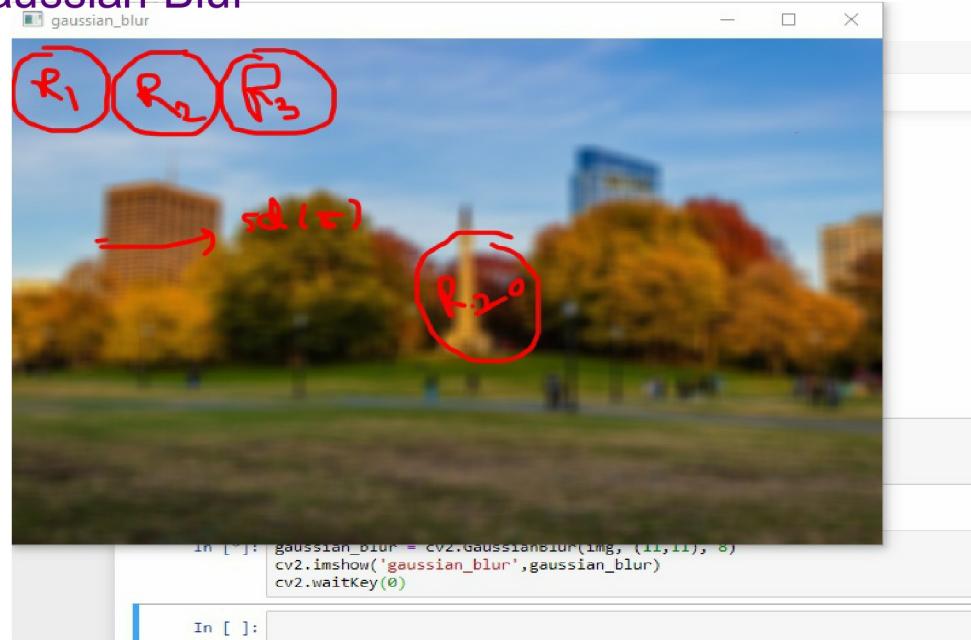


Smoothing of images

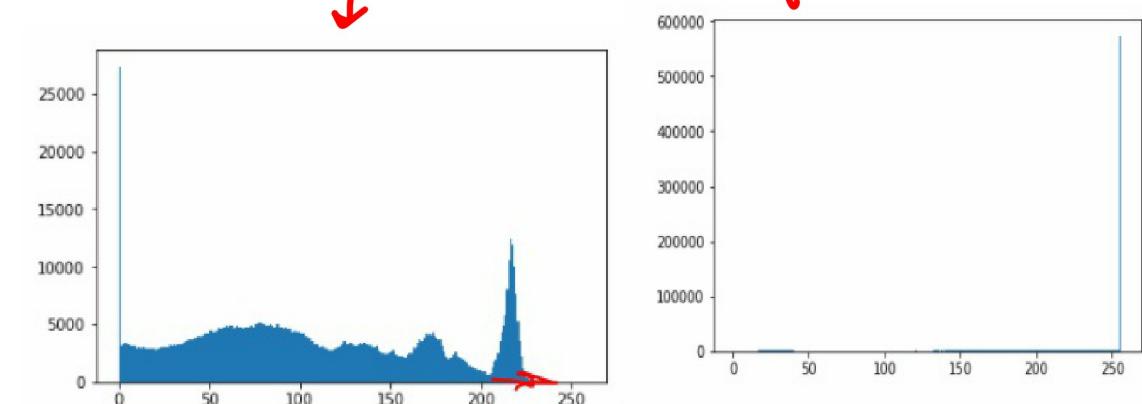
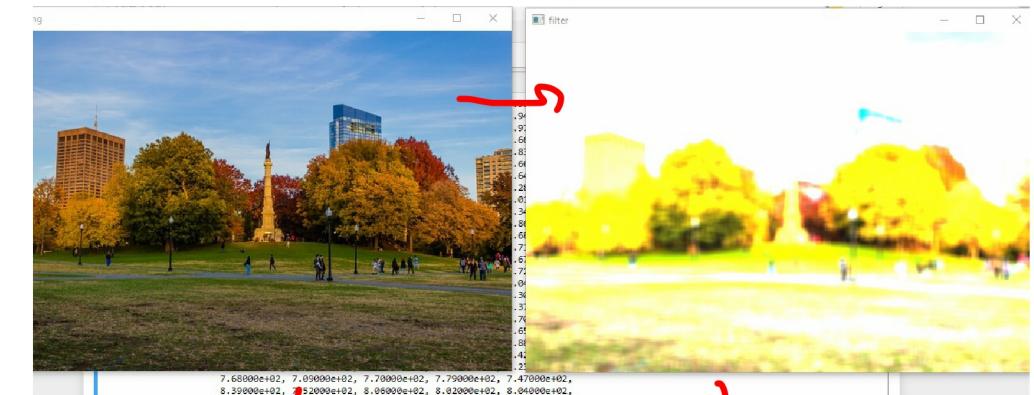
1. Filter2d - small kernel



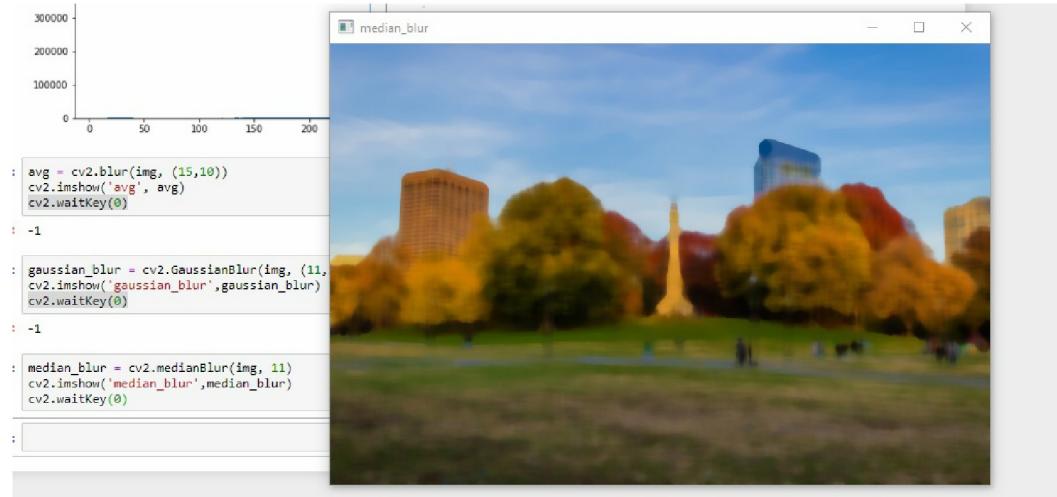
Gaussian Blur



Filter2D Technique- large kernel



Median blur



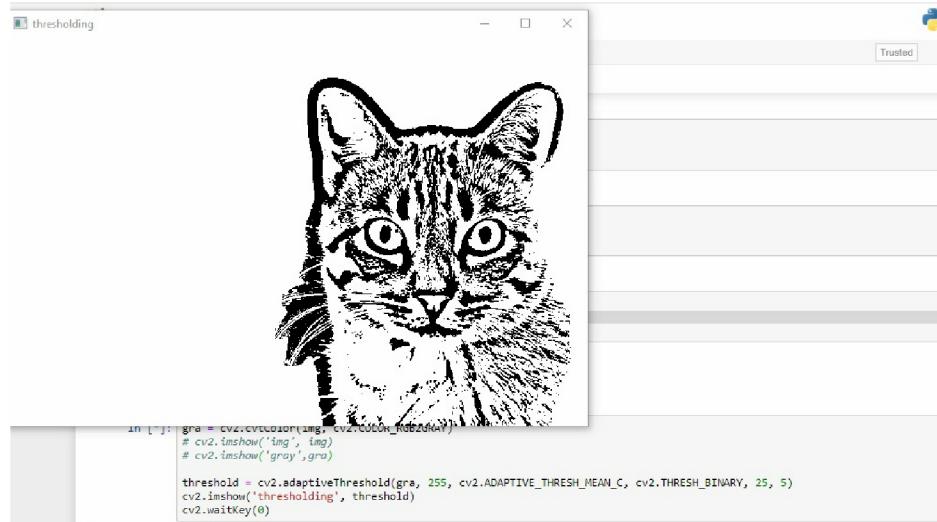
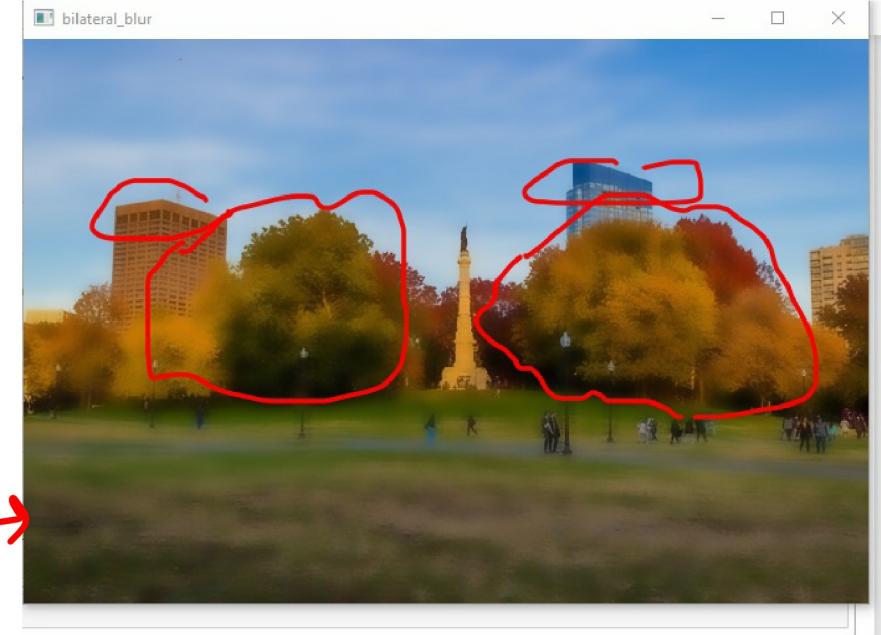
```
300000  
200000  
100000  
0  
0 50 100 150 200  
avg = cv2.blur(img, (15,10))  
cv2.imshow('avg', avg)  
cv2.waitKey(0)  
-1  
gaussian_blur = cv2.GaussianBlur(img, (11,  
cv2.imshow('gaussian_blur', gaussian_blur)  
cv2.waitKey(0)  
-1  
median_blur = cv2.medianBlur(img, 11)  
cv2.imshow('median_blur', median_blur)  
cv2.waitKey(0)  
:  
:  
median_blur
```

A screenshot of a Jupyter Notebook cell. The code uses OpenCV's `blur`, `GaussianBlur`, and `medianBlur` functions to apply different types of blurring to an image of a park with autumn trees. The resulting blurred image is displayed in a window titled "median_blur".

bilateral filter:

Gaussian blur module

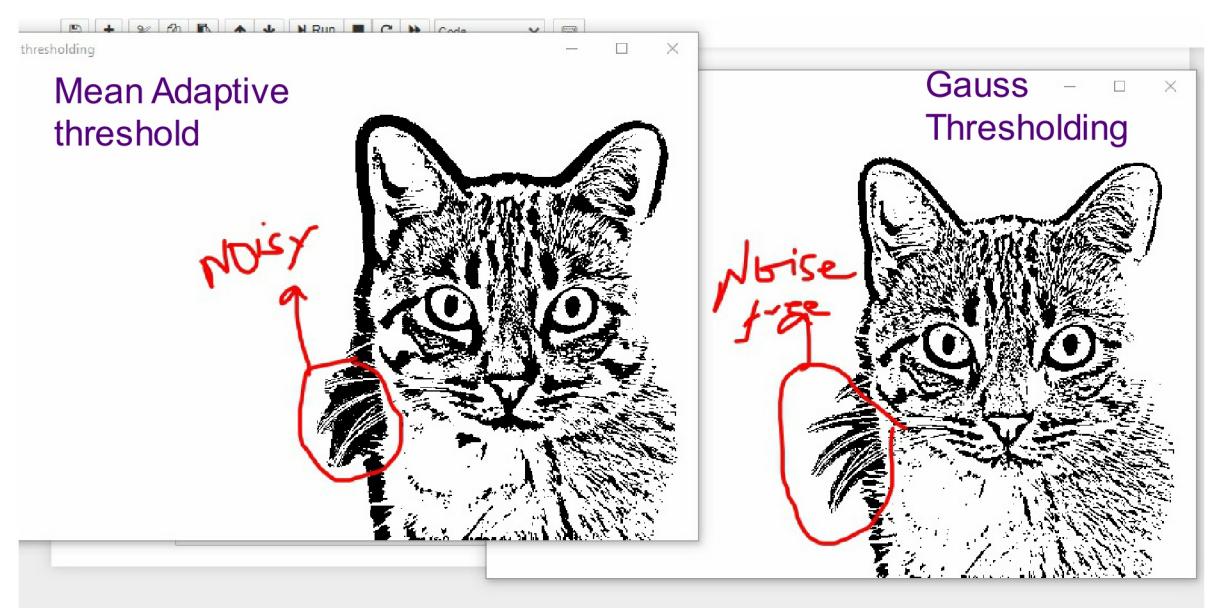
pointy edges module



```
thresholding  
In [1]: gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  
# cv2.imshow('img', img)  
# cv2.imshow('gray', gray)  
  
threshold = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 25, 5)  
cv2.imshow('thresholding', threshold)  
cv2.waitKey(0)
```

A screenshot of a Jupyter Notebook cell. The code converts an image of a cat into grayscale and then applies mean adaptive thresholding using OpenCV's `adaptiveThreshold` function. The resulting binary image is displayed in a window titled "thresholding". A red arrow points to a noisy region in the thresholded image with the label "noisy".

Mean Adaptive threshold



Thresholding

Mean Thresholding

Watch: A 2-Minute Animation Of Stephen Hawking'S Big Ideas | Co.Design

As one of the most brilliant theoretical physicists in the world, Stephen Hawking writes books that can intimidate those among us who barely passed Physics for Poets or Rocks for Jocks. In *A Brief History of Time*, he tackles questions like "How did the universe begin?" and "Does time always flow forward?" while the rest of us distract ourselves from these daunting unknowns with Internet cat videos.

The latest of the *Guardian Science's MadeSimple* animation series presents "Stephen Hawking's Big Ideas Made Simple," which condenses his grandest theories into a 2.5-minute video. In the playful cut-paper animation, a smiling cartoon Hawking rides his wheelchair like a chariot through space with his colleague, Roger Penrose. They slip into a black hole, where lots of matter—including lamps, light bulbs, and flashlights—gets crumpled up by a god-like pair of hands into a point of infinite density called a "singularity." At the edge of the black hole, pairs of particles promise, "I'll never forget you!" as one falls in and the other escapes.

"Our work is often about trying to say more with less—finding the quickest, most elegant way to communicate often very complex information," says Dan Porter, creative director at Scriberia, the animation company that designed the short. "We're trying to create visuals that will help people retain information, so sometimes the wildest ideas are the ones that are going to stick." He points out that the Hawking animation is rendered in "stuff you'd find in any elementary school." The lessons are articulated in very accessible language and tone by narrator and scriptwriter Alok Jha.

Gaussian Thresholding

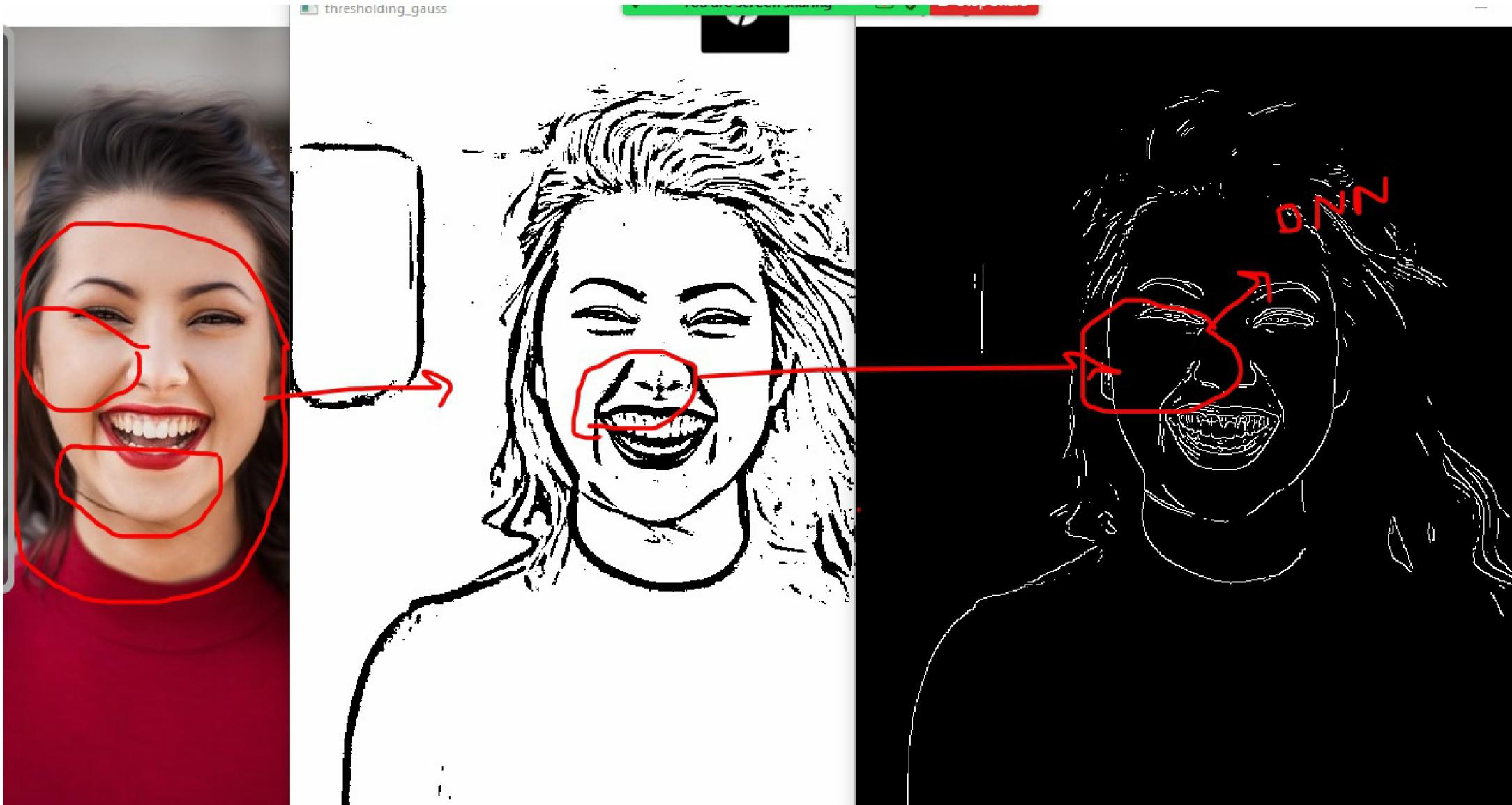
Watch: A 2-Minute Animation Of Stephen Hawking'S Big Ideas | Co.Design

As one of the most brilliant theoretical physicists in the world, Stephen Hawking writes books that can intimidate those among us who barely passed Physics for Poets or Rocks for Jocks. In *A Brief History of Time*, he tackles questions like "How did the universe begin?" and "Does time always flow forward?" while the rest of us distract ourselves from these daunting unknowns with Internet cat videos.

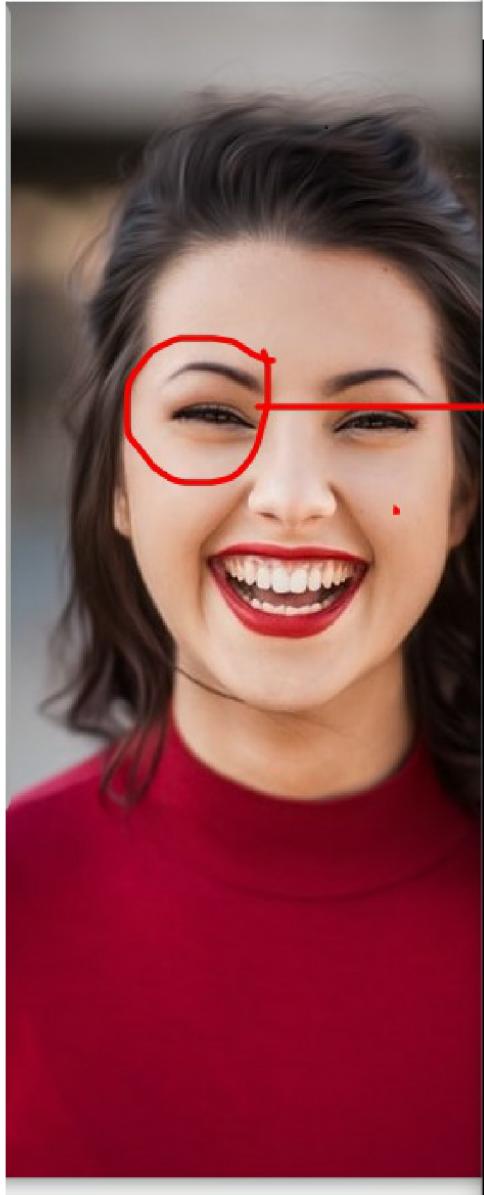
The latest of the *Guardian Science's MadeSimple* animation series presents "Stephen Hawking's Big Ideas Made Simple," which condenses his grandest theories into a 2.5-minute video. In the playful cut-paper animation, a smiling cartoon Hawking rides his wheelchair like a chariot through space with his colleague, Roger Penrose. They slip into a black hole, where lots of matter—including lamps, light bulbs, and flashlights—gets crumpled up by a god-like pair of hands into a point of infinite density called a "singularity." At the edge of the black hole, pairs of particles promise, "I'll never forget you!" as one falls in and the other escapes.

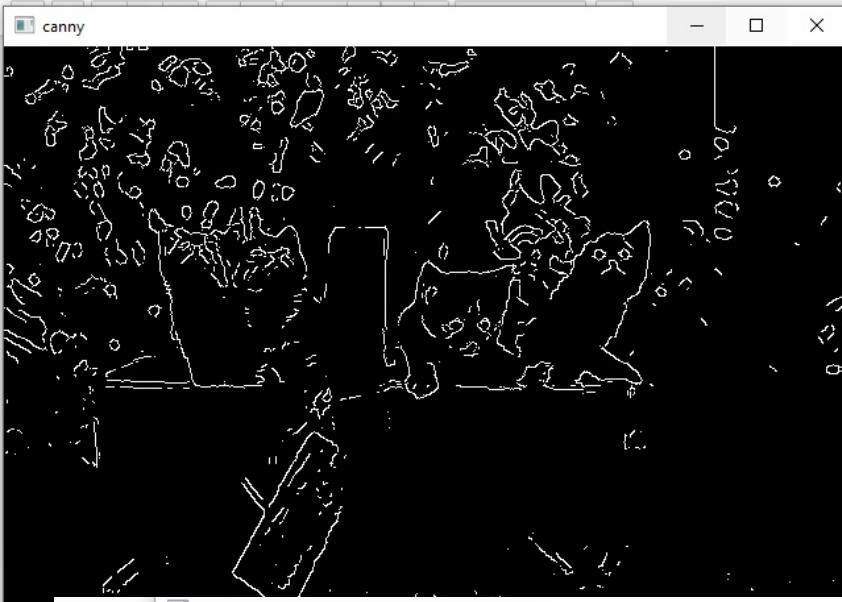
"Our work is often about trying to say more with less—finding the quickest, most elegant way to communicate often very complex information," says Dan Porter, creative director at Scriberia, the animation company that designed the short. "We're trying to create visuals that will help people retain information, so sometimes the wildest ideas are the ones that are going to stick." He points out that the Hawking animation is rendered in "stuff you'd find in any elementary school." The lessons are articulated in very accessible language and tone by narrator and scriptwriter Alok Jha.

Gaussian Filter - canny edges

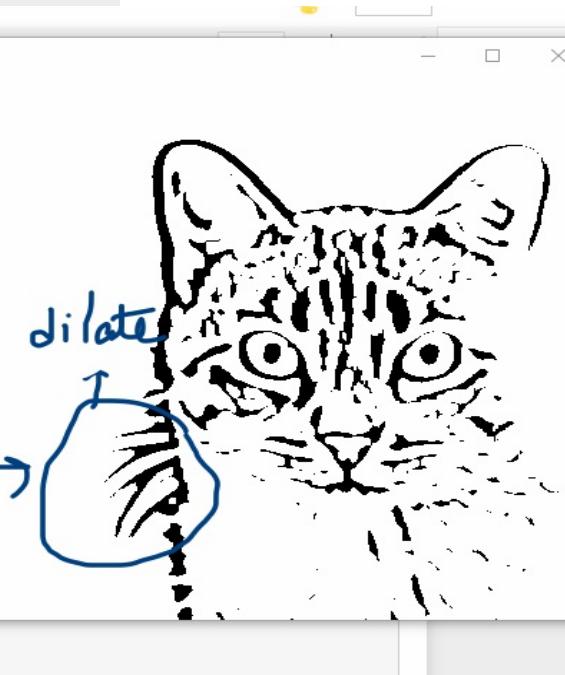
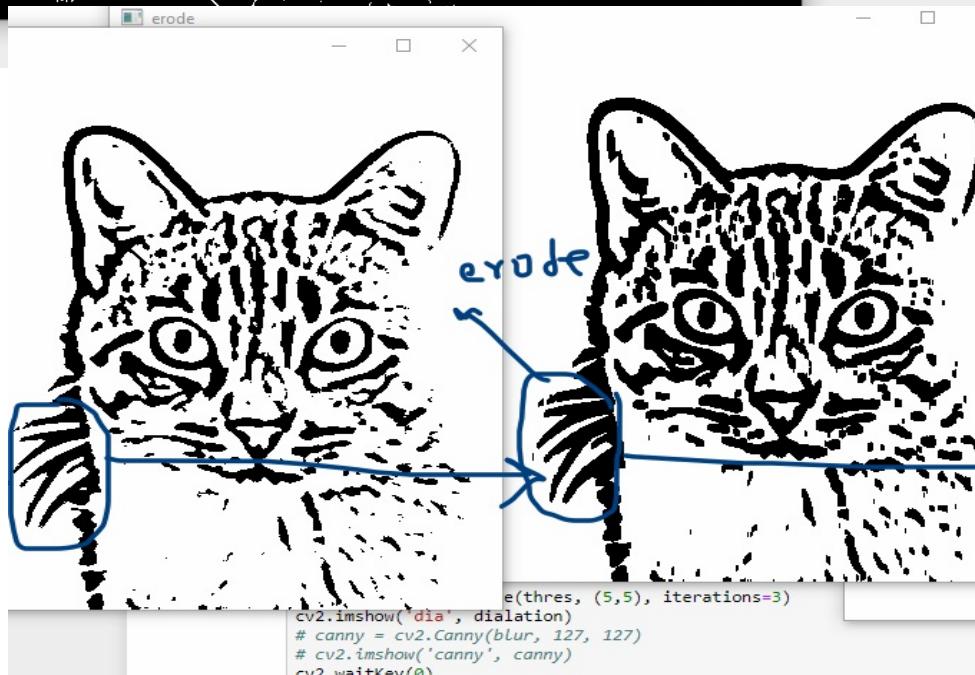
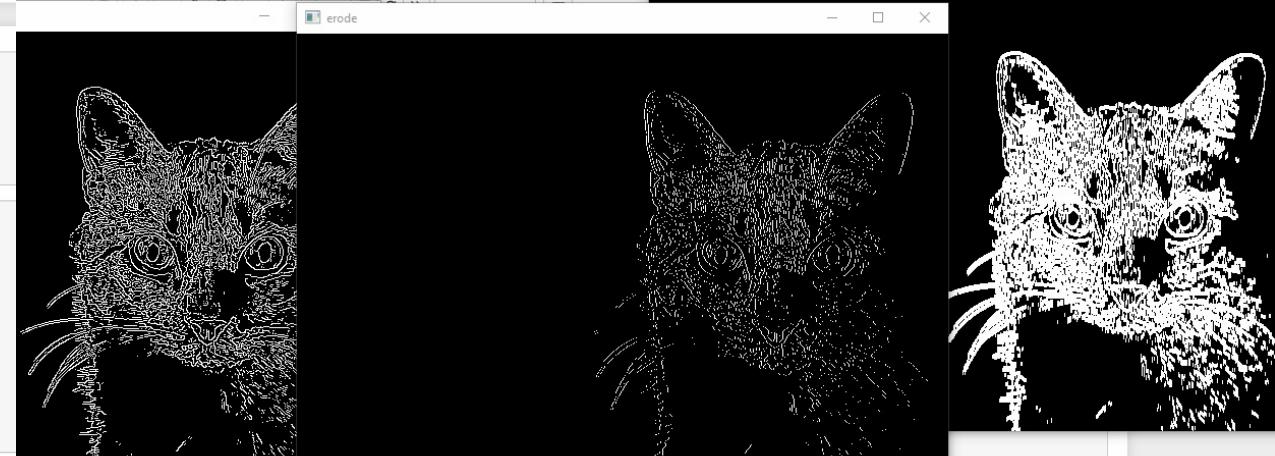
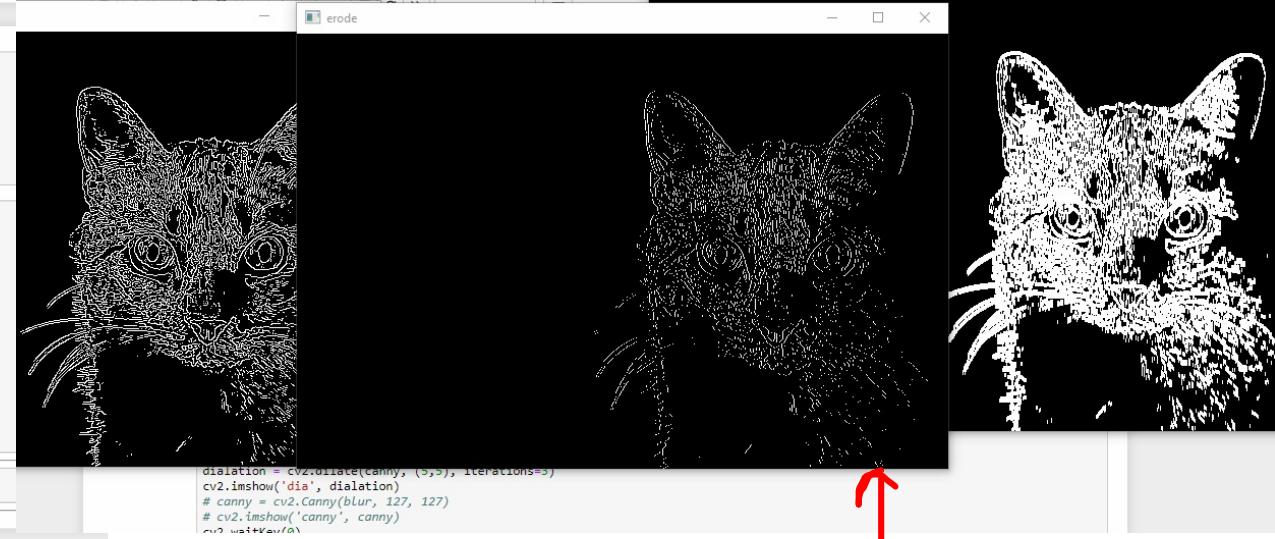


canny HPF & LPF comparison

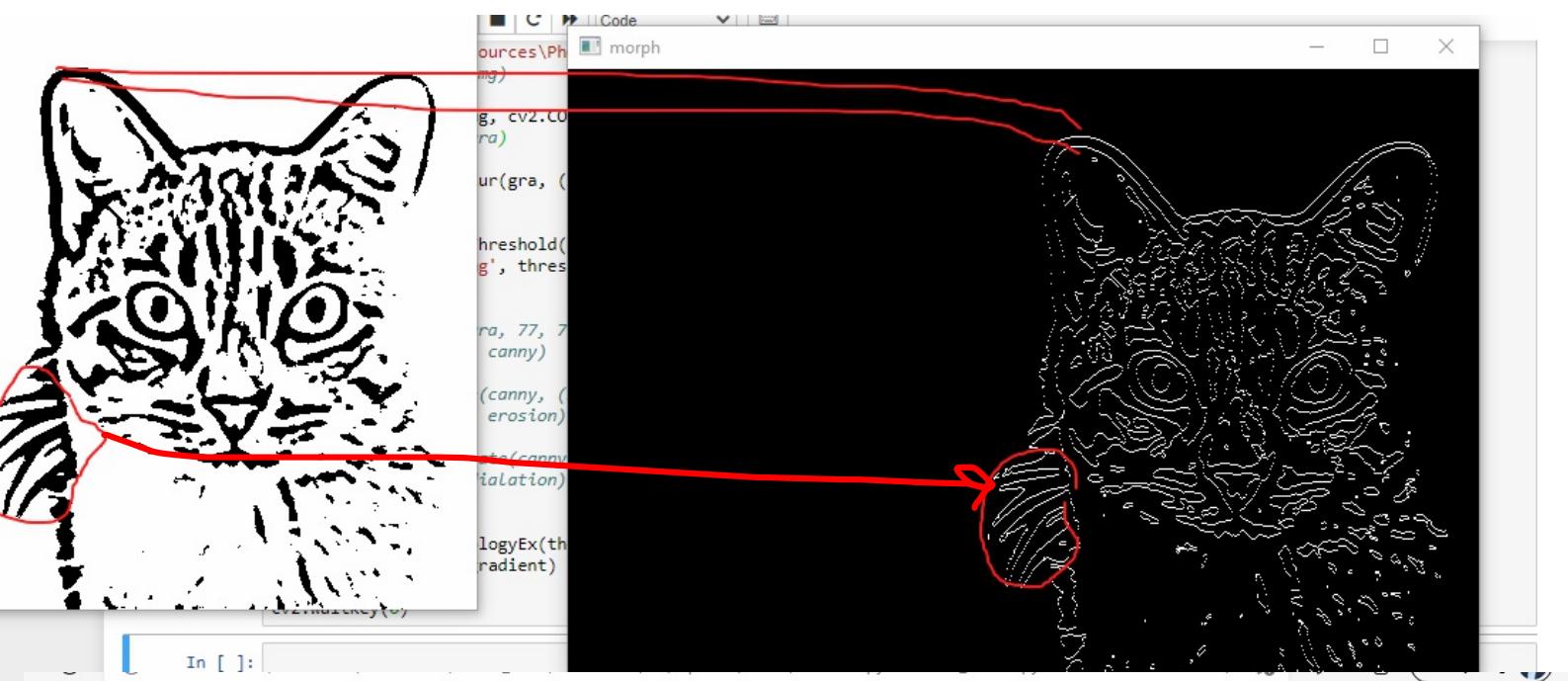




BGR
↓
GRAY
↓
GBlur
↓
Canny



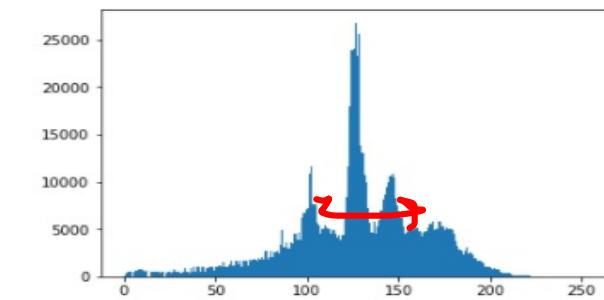
Erosion works better when there are too many thick edges are present. It is useful to remove the noise and erodes the edges hence we can use dilate to improve prominence over eroded outcomes



```

165., 166., 167., 168., 169., 170., 171., 172., 173., 17
176., 177., 178., 179., 180., 181., 182., 183., 184., 18
187., 188., 189., 190., 191., 192., 193., 194., 195., 19
198., 199., 200., 201., 202., 203., 204., 205., 206., 20
209., 210., 211., 212., 213., 214., 215., 216., 217., 21
220., 221., 222., 223., 224., 225., 226., 227., 228., 22
231., 232., 233., 234., 235., 236., 237., 238., 239., 24
242., 243., 244., 245., 246., 247., 248., 249., 250., 25
253., 254., 255., 256.]),
<a list of 256 Patch objects>

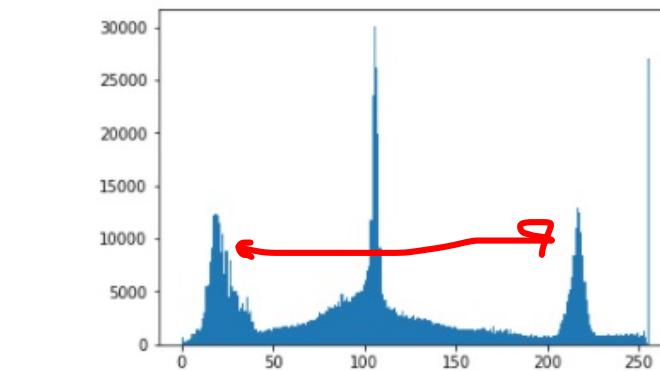
```

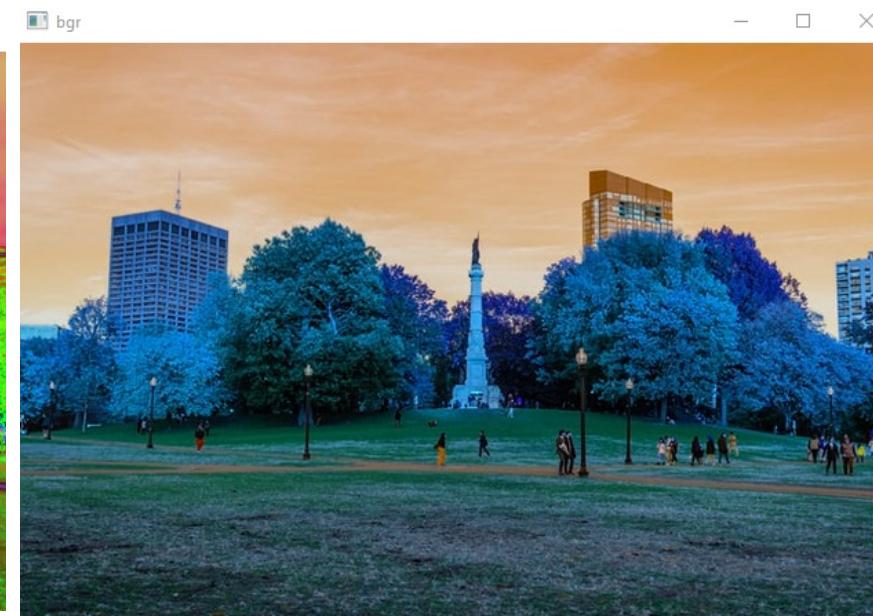
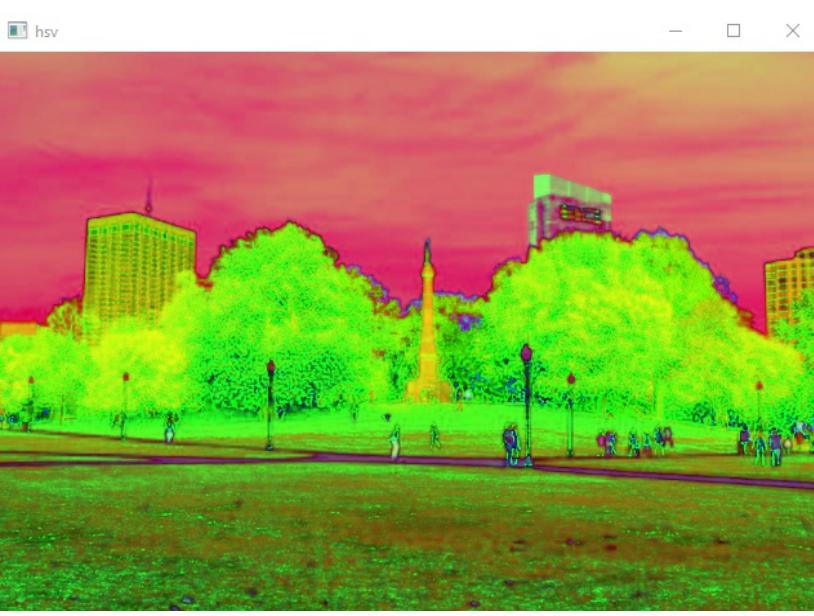
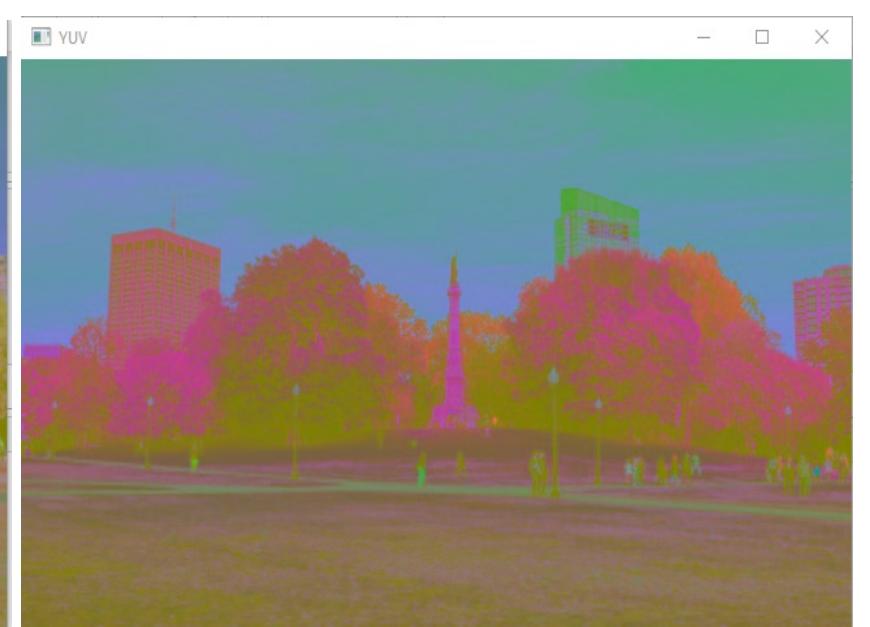
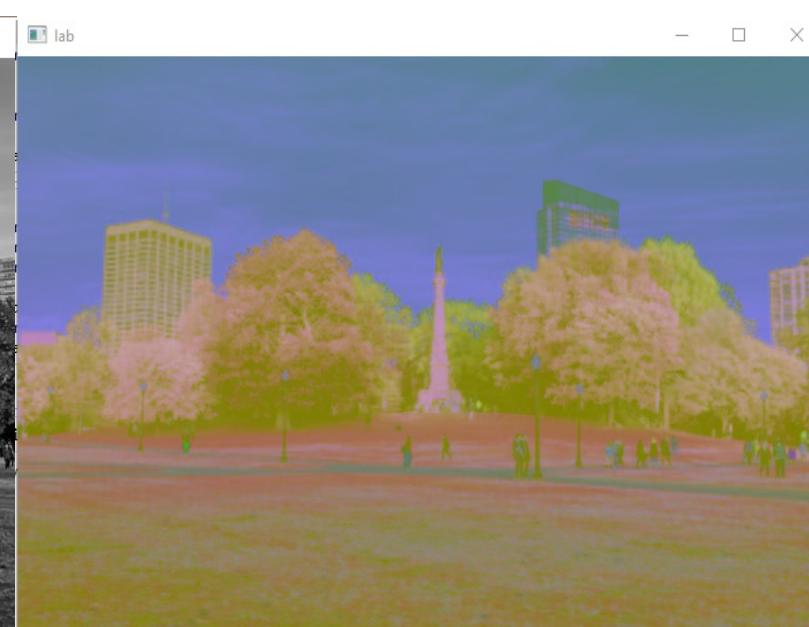
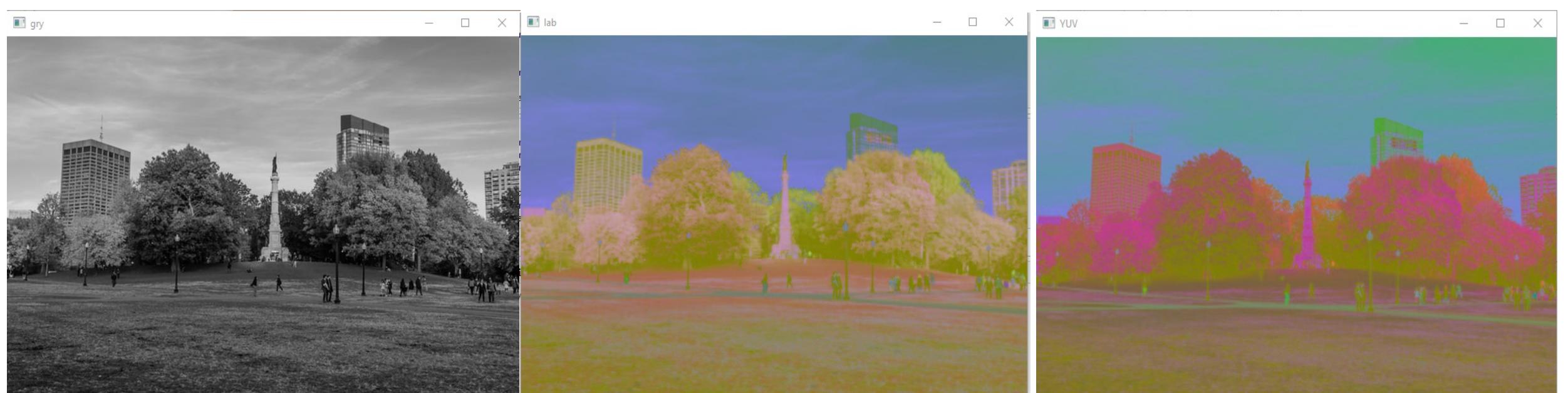


```

154., 155., 156., 157., 158., 159., 160., 161., 162., 163.,
165., 166., 167., 168., 169., 170., 171., 172., 173., 174.,
176., 177., 178., 179., 180., 181., 182., 183., 184., 185.,
187., 188., 189., 190., 191., 192., 193., 194., 195., 196.,
198., 199., 200., 201., 202., 203., 204., 205., 206., 207.,
209., 210., 211., 212., 213., 214., 215., 216., 217., 218.,
220., 221., 222., 223., 224., 225., 226., 227., 228., 229.,
231., 232., 233., 234., 235., 236., 237., 238., 239., 240.,
242., 243., 244., 245., 246., 247., 248., 249., 250., 251.,
253., 254., 255., 256.]),
<a list of 256 Patch objects>

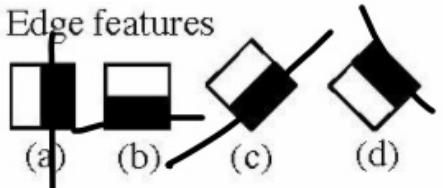
```



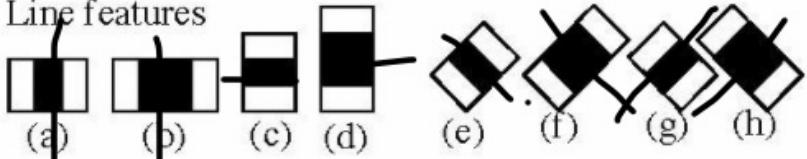


Haar Cascade

1. Edge features



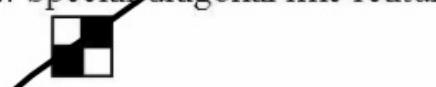
2. Line features



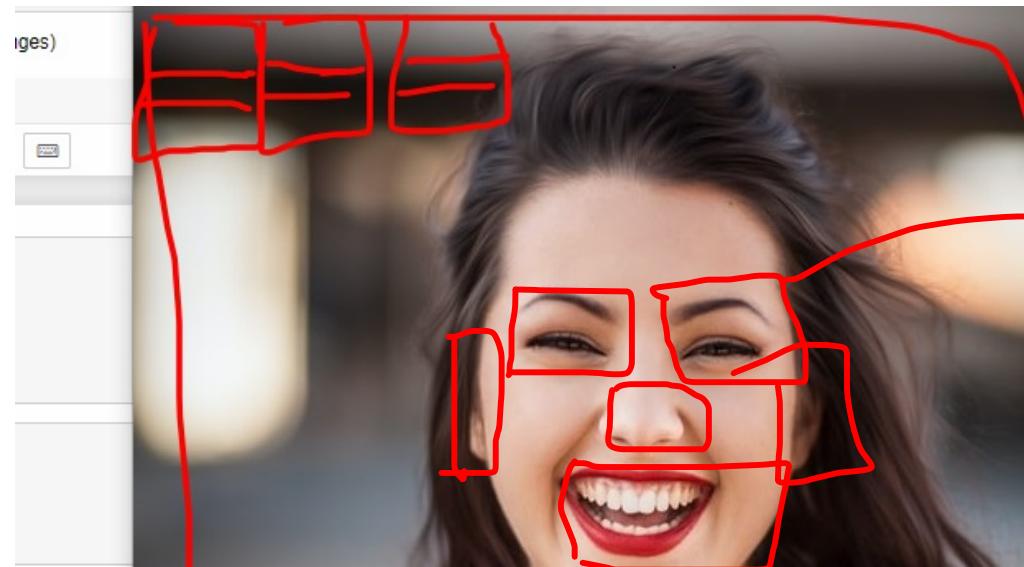
3. Center-surround features



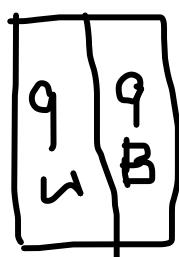
4. Special diagonal line feature used in [3,4,5]



Haar: sum of dark pixel / no of dark pixel



XML
↓
Predict

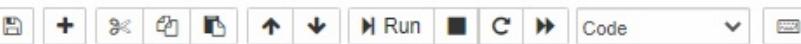


$$= \frac{0.8 + 0.7 \cdot b_2}{9} = 0.02$$

Cascading : combining the haar features = 0.7

jupyter Untitled1 Last Checkpoint: 32 minutes ago (autosaved)

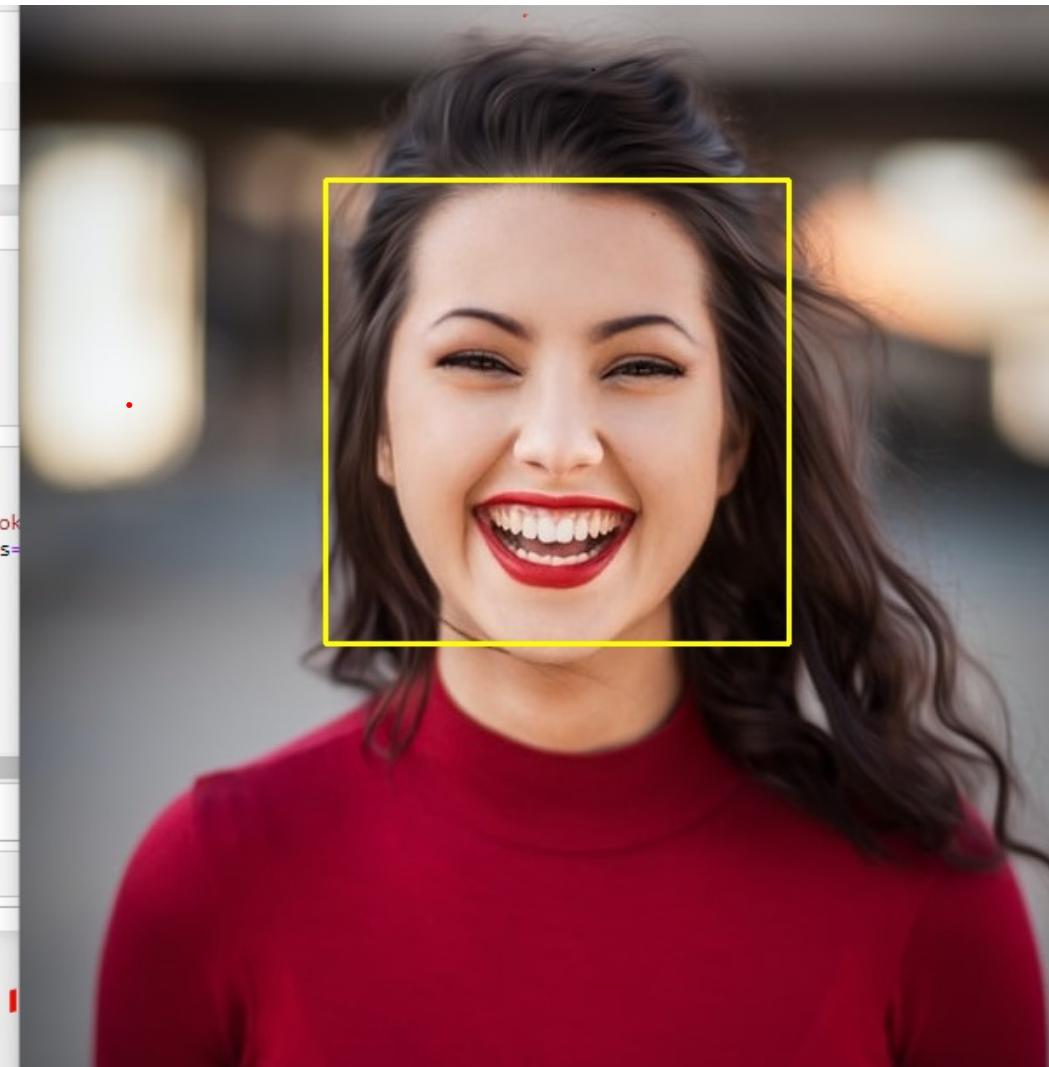
File Edit View Insert Cell Kernel Widgets Help



```
In [1]: import cv2 as cv2  
import numpy as np  
import pandas as pd  
import math as math  
from matplotlib import pyplot as plt
```

```
In [*]: imread('Resources\Photo\lady.jpg')  
.cvtColor(img, cv2.COLOR_BGR2GRAY)  
de = cv2.CascadeClassifier('C:\\\\Users\\\\R1007398\\\\Downloads\\\\Book  
haar cascade\\haarcascade\\detectMultiScale(img, scaleFactor=1.1, minNeighbors=  
> of faces : {len(deetect)})')  
  
w,h) in detect:  
rectangle(img, (x,y), (x+w, y+h), (0,255,255), thickness=2)  
  
('img', img)  
ey(0)  
  
no of faces : 1
```

```
In [ ]:
```



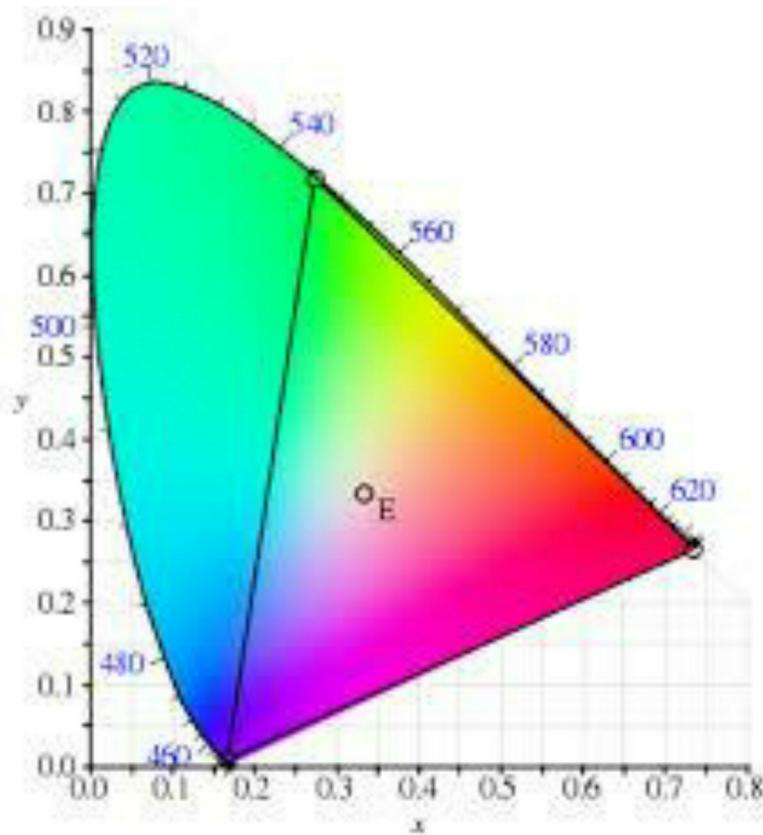
Color spaces

- The organization of the colors in an image in a specific format is called color space.
- Each and every image uses one of the following color spaces:
 - RGB : Red, Green , Blue
 - XYZ : color in the x ,y and z dimensions
 - HSV/HSL : hue , saturation and value/hue saturation and lightness
 - LAB : luminance, and green – red and blue – yellow color components
 - LCH : lightness, chroma and hue
 - YPbPr : green ,blue and red cables
 - YUV : brightness or chroma or color
 - YIQ : luminance, in-phase parameter and quadrature

RGB Color Space

- Using RGB, the colors are mixed in different ways to make different color combinations.
- Theoretically, we can form any color from these colors. Each intensity is from 0 to 255. (color depth)
- RGB has two more components:
 - White point chromaticity
 - Gamma correction curve

XYZ



https://en.wikipedia.org/wiki/CIE_1931_color_space

HSV/HSL

- This is alternate representation of RGB color space.
 - Hue
 - Saturation
 - Value
 - Lightness
- Hue explains green, red and magenta or two pure colors red and yellow
- Saturation &Lightness measures intensity of image with respect to Gray and white respectively.
- Value does the same from black.

LAB & LCH

- Luminance
- a which is a green and red color component
- b blue and yellow color component
- This is used for printing textiles
- This uses cylindrical coordinates
- LCH is also similar but this uses rectangular coordinates

YPbPr

- Y – Green cable
- Pb – blue cable
- Pr – red cable

<https://en.wikipedia.org/wiki/YPbPr>



YUV

- YUV is similar to the previous one , but this works for black and white television as well.
- Y – Represents brightness 0 to 255
- U and V represents color , if they are zeros we get a grayscale image.
- YIQ:
- NTSC television uses this.

THANK
YOU

