

Mastering Computer Vision

with **OpenCV** in **Python**

Learn all core basics, do almost coding 50 exercises & completing 12 fun mini projects





Computer Vision is a cutting edge field of Computer Science that aims to **enable computers** to understand what is **being seen** in an **image**



The Artificial Intelligence Dream

Machines that can see and understand the world around them



Source: Terminator 2: Judgement Day

However, Computer Vision is a Challenging!

Why is it so hard? So Many Reasons!

Camera sensor & lens limitations



Why is it so hard? So Many Reasons!

Viewpoint variations



Why is it so hard? So Many Reasons!

Changing Lighting

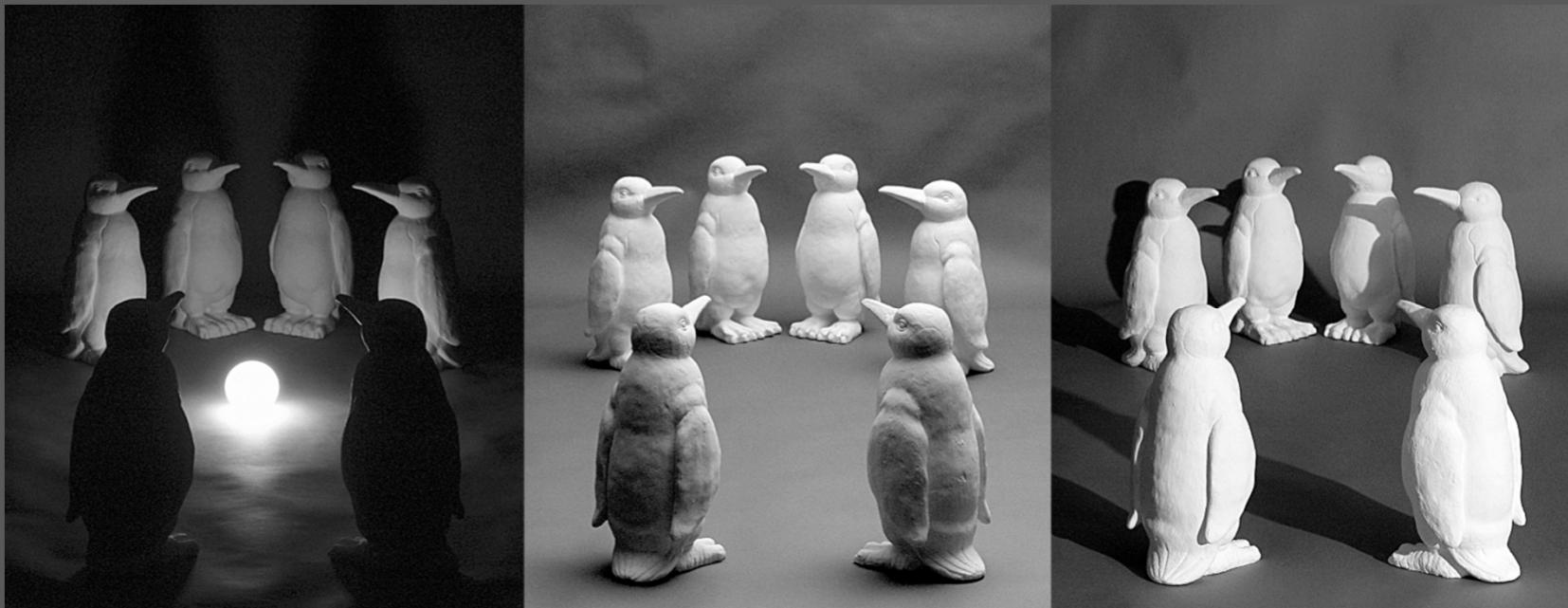


image credit: J. Koenderink

Why is it so hard? So Many Reasons!

Scaling



Why is it so hard? So Many Reasons!

Non-rigid deformations



image credit: Xu, Beihong

Why is it so hard? So Many Reasons!

Occlusion



image credit: Magritte, 1957

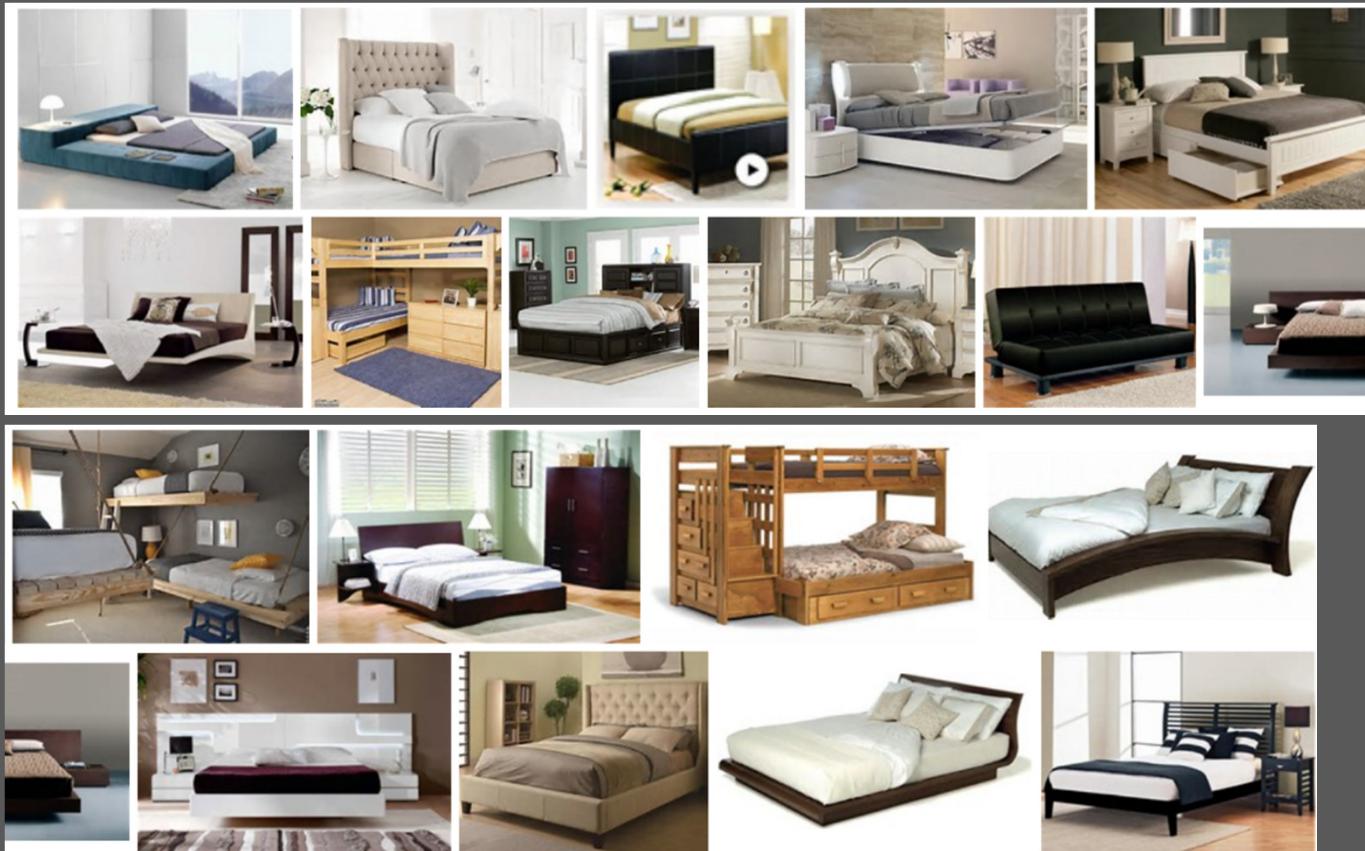
Why is it so hard? So Many Reasons!

Clutter



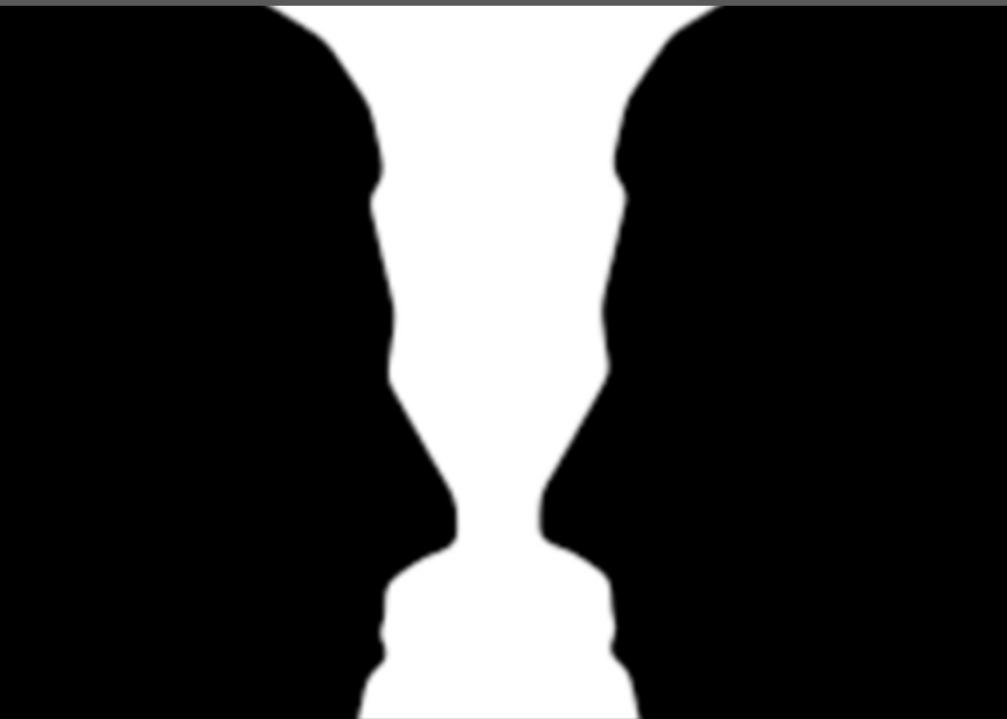
Why is it so hard? So Many Reasons!

Object class variations



Why is it so hard? So Many Reasons!

Ambiguous Images/Optical Illusions



Despite the difficulty, Computer Vision scientists have had many success stories!

Robotic Navigation – Self Driving Cars

Face Detection & Recognition

Search Engine Image Search

License Plate Reading

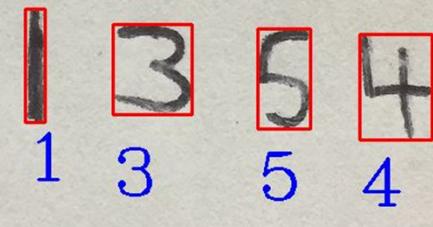
Handwriting Recognition

Snapchat & MSQRD Face Filters

Object Recognition

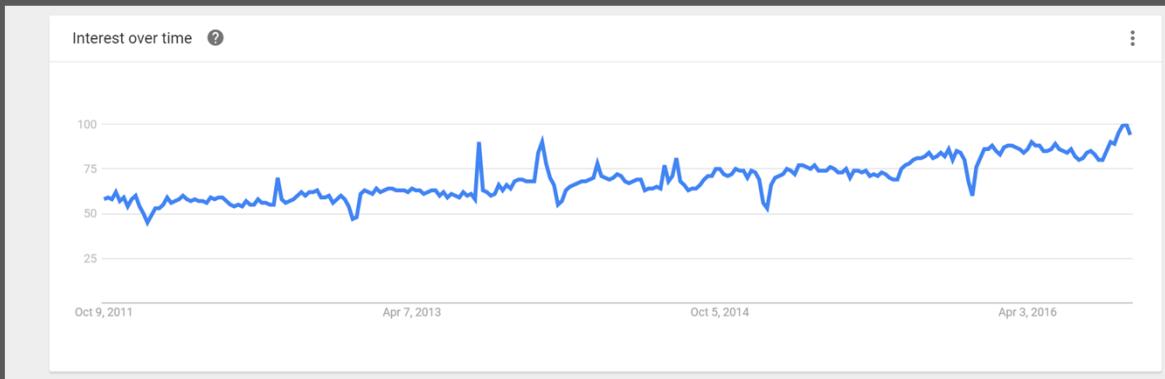
Ball & Player Tracking in Sports

And many more!



So why OpenCV in Python?

Python is one of the **easiest** languages for beginners



It is **extremely powerful** for data science and machine learning applications

It stores images in **numpy arrays** which allows us to do some very powerful operations quite easily

What You'll be Learning!

Introduction to all key areas of **Computer Vision Theory**

- Course Introduction & Setup
- Basics of Computer Vision & OpenCV
- Image Manipulations
- Image Segmentation
- Object Detection
- Face, People & Car Detection
- Face Analysis and Filters
- Machine Learning in Computer Vision
- Motion Analysis & Object Tracking
- Bonus - Computational Photography
- Course Summary
 - Become an expert!
 - Explore latest research
 - Startup Ideas

Coding along with almost 50 exercises

Develop **12 Fun Mini Projects!**

How this course is taught?

More than half your time will be spent **coding**

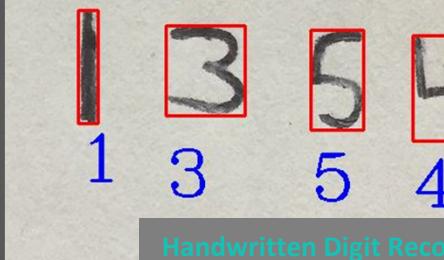
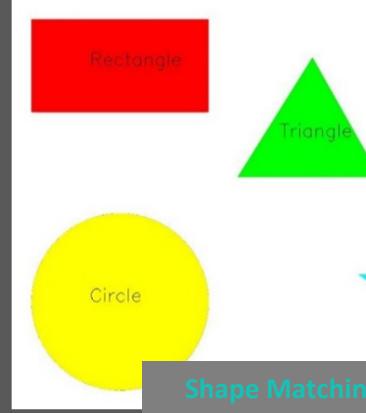
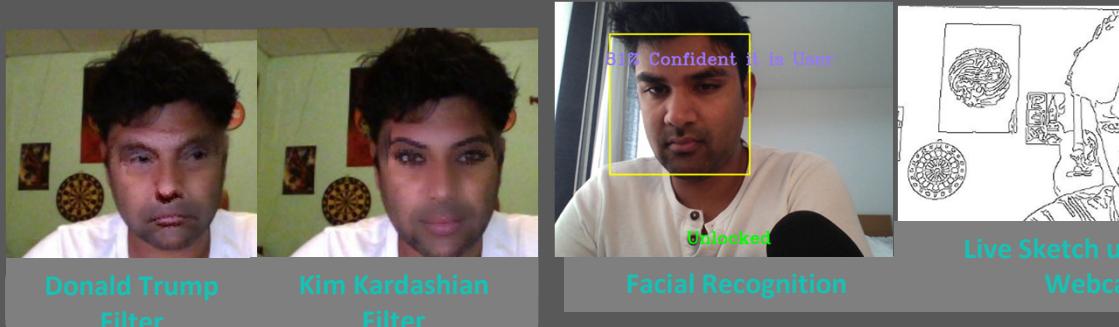
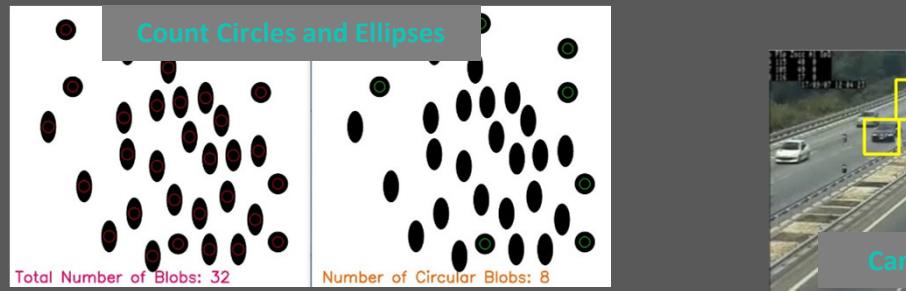
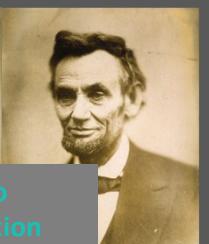
Overview of the theory first then we **code**

Almost all **code is explained line by line**

12 Amazing Mini Projects

What are these Mini Projects?

Making a Live Drawing Sketch of yourself
Shape Matching
Counting Circles and Ellipses
Finding Waldo
Object Identification
Face, Pedestrian & Car Detection
Live Face Swaps - Become Donald Trump or Kim Kardashian (just like MSQRD & Snapchat)
Face Reader - Detect and count yawns
Handwritten Digit Recognition
Facial Recognition
Ball Tracking
Photo Restoration



Requirements?

Basic programming is useful, but not needed I'll walk you through most of the code. Exposure to Numpy would be helpful.

High School Level Math

A Webcam

Python 2.7 or 3.6

- Anaconda Package is preferred but not required. I use Python 2.7, but all code works in 3.6
- I use Python Notebooks which are excellent for teaching

OpenCV 2.4.6 & or 3.4.X

- Ideally we'd like to use the latest version OpenCV 3.4.6, however it is missing some important functions (SIFT, SURF etc.) that are included in 2.4.16. There are ways around it however, so don't despair.

Who is this course for?

Beginners interest in Computer Vision

Software developers & engineers looking strengthen their skills for job promo

College/University students looking to get a head start in computer vision projects and research are ideal.

Startup founders who wish to utilize computer vision

Hobbyist who want to build a fun computer vision project. E.g. Raspberry Pi projects.

•-----•
**Let's begin our exciting journey into
The world of Computer Vision using
OpenCV in Python!**



Python & OpenCV Windows Installation

STEP 1 – Download & Install Anaconda Python Package

Go to: <https://www.anaconda.com/download>

Select appropriate version - either Python 2.7 or 3.6

To Test – Go to windows command prompt and type:

- jupyter notebook

STEP 2 – Download and Run OpenCV

Go to: <https://opencv.org/releases.html>

Download either OpenCV 2.4.16 or 3.4.3

- **NOTE:** A small number of sections in this course require 2.4.16 and some require 3.4.X
- ~~You can download both and switch them easily~~ No longer the case. Choose OpenCV 2.4.16 as most parts of this course are compatible with that version.

NOTE: This works for only Python 2.7 (Python 3.6 on next slide)

Python & OpenCV Installation

STEP 3

Find your `cv2.py` file

Your CV2 file should be found here - `C:\opencv\build\python\2.7\x64`

STEP 4

Copy this `cv2.py` file and place it in the following directory (main directory will be where your python.exe is located):

- `C:\Users\Anaconda2\Lib\site-packages`

And that's it, you're ready to get started!

Python 3.6 Installation

Python & OpenCV Installation

STEP 3

Open command prompt and type:

- pip install opencv-python

And that's it, you're ready to get started!

Section 2

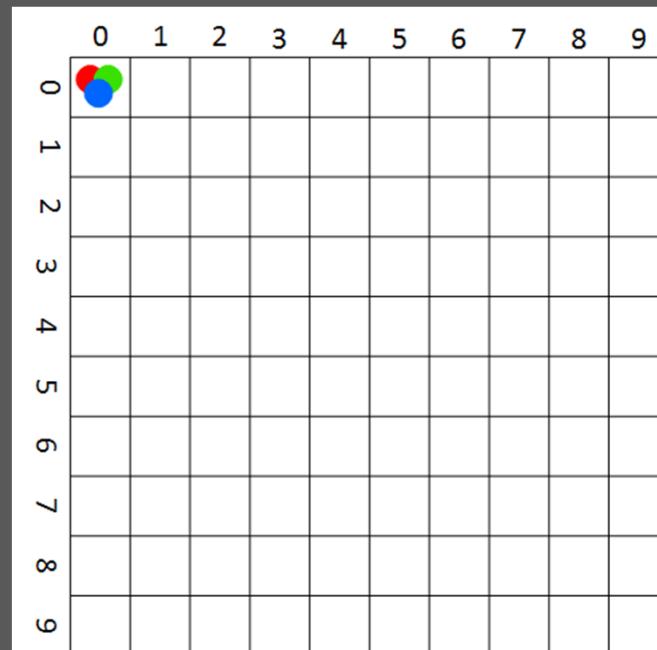
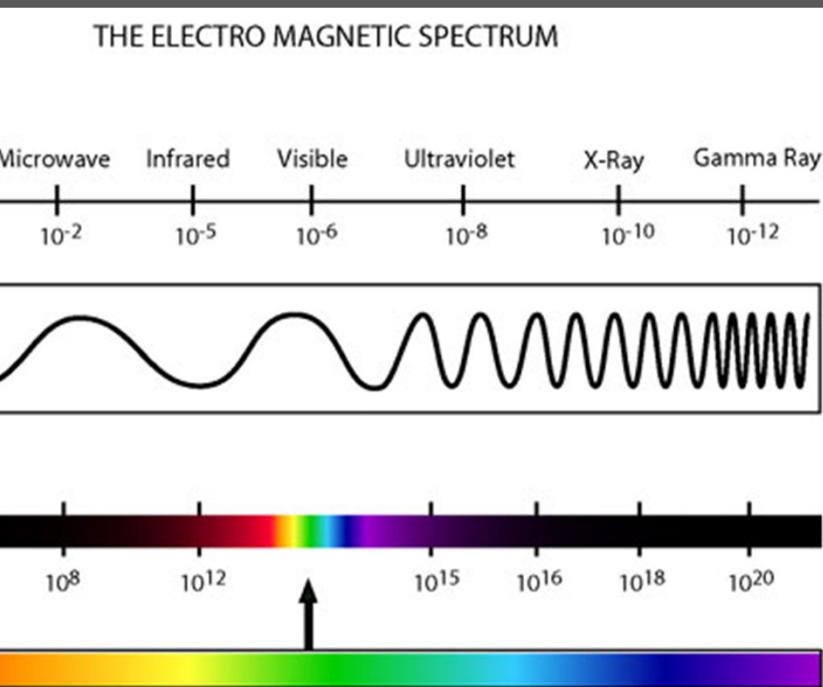
Basics of Computer Vision & OpenCV

Basics of Computer Vision & OpenCV

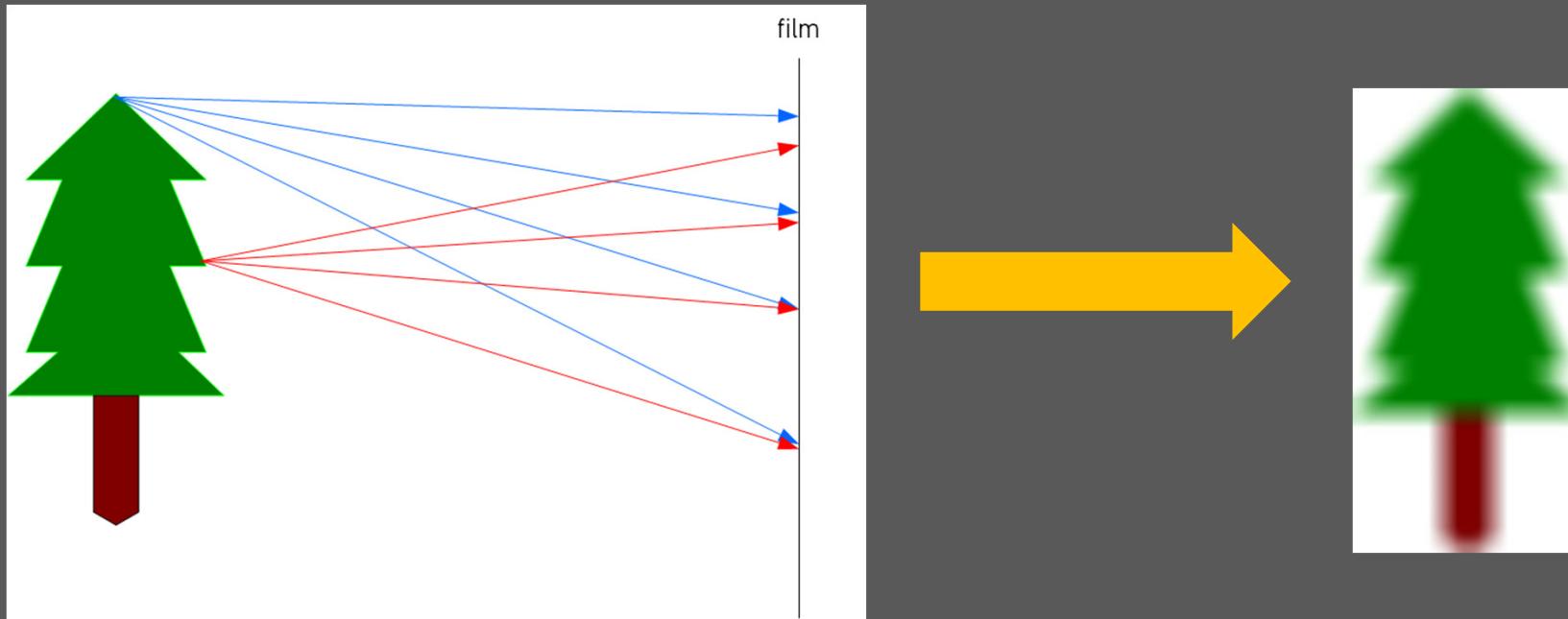
- . What are Images?
- . Image Formation
- . Storing images on computers
- . Getting Started with OpenCV: reading, writing and displaying images
- . Gray scaling
- . Color Spaces
- . Histograms
- . Drawing images

What are Images?

2-Dimensional representation of the [visible light spectrum](#)

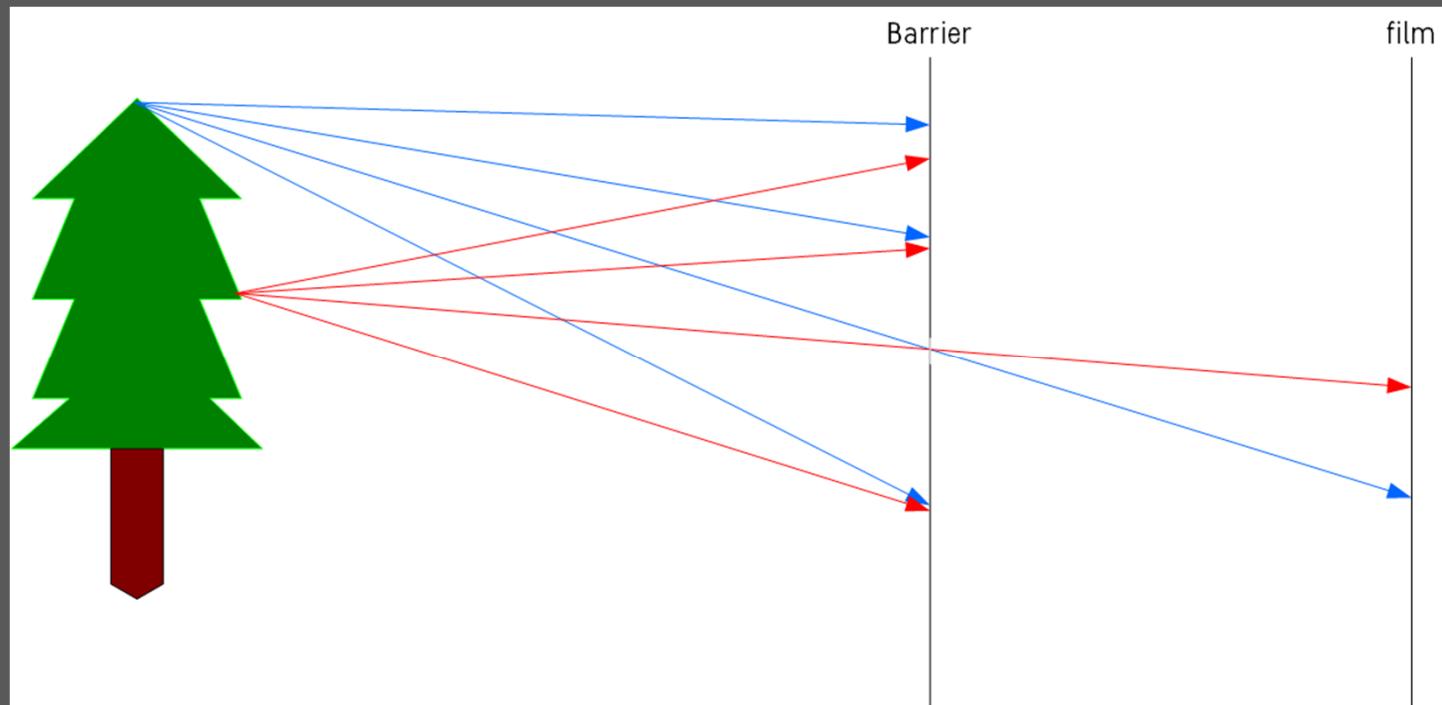


How are images formed



- When light reflects off an object onto a film, sensor or retina
- In the above example the image here will be blurred

Image Formation

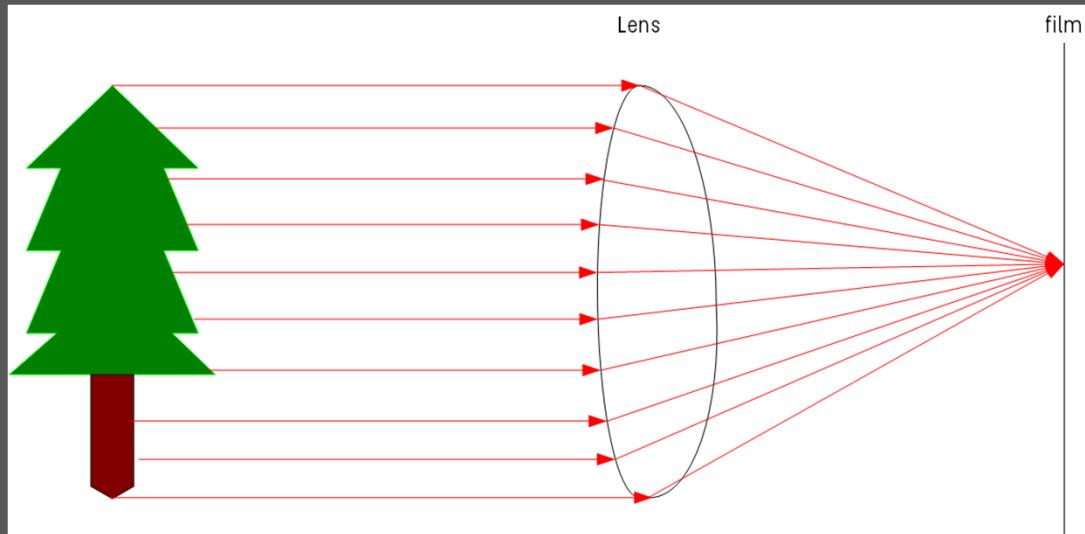


- Using a small opening in the barrier (called aperture), we block off most of the rays of light reducing blurring on the film or sensor
- This is the pinhole camera model

Controlling Image Formation with a Lens

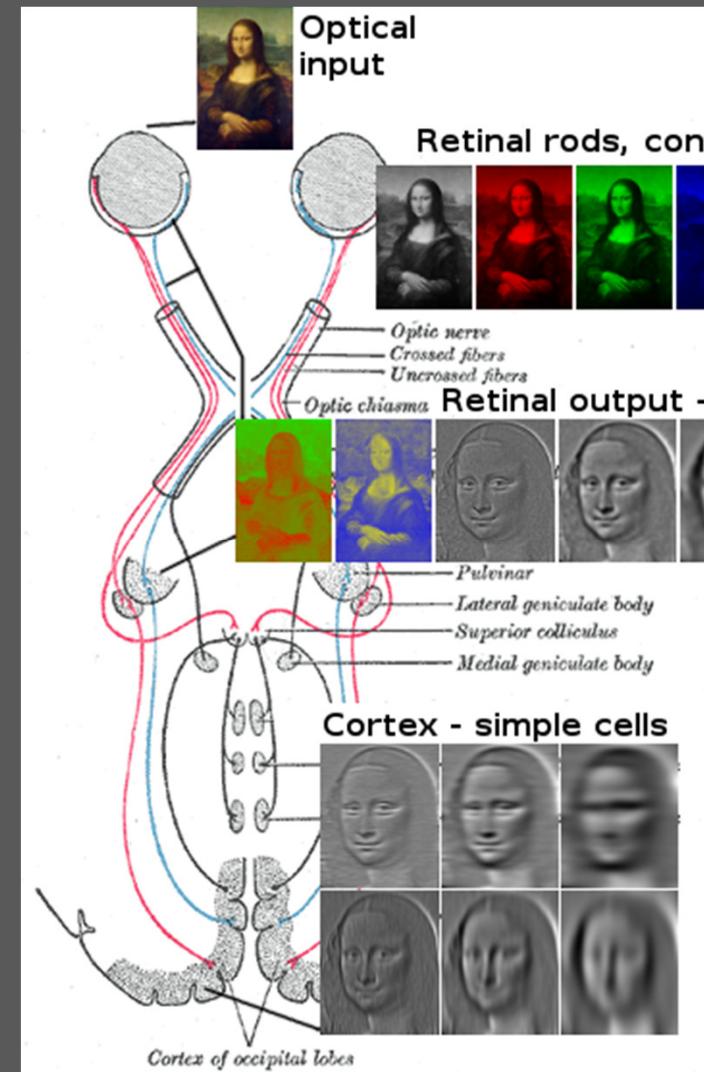
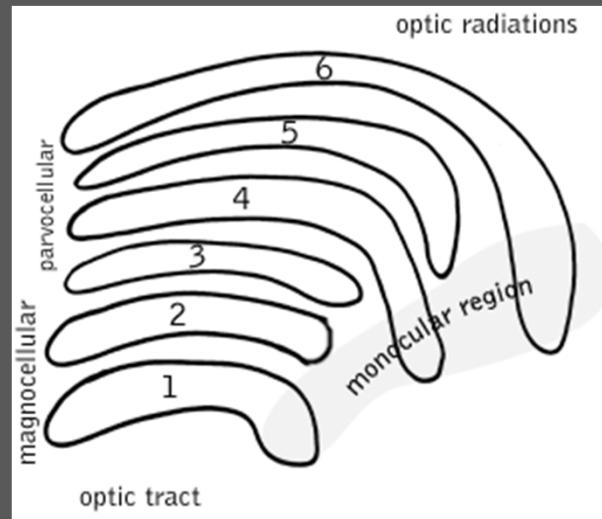
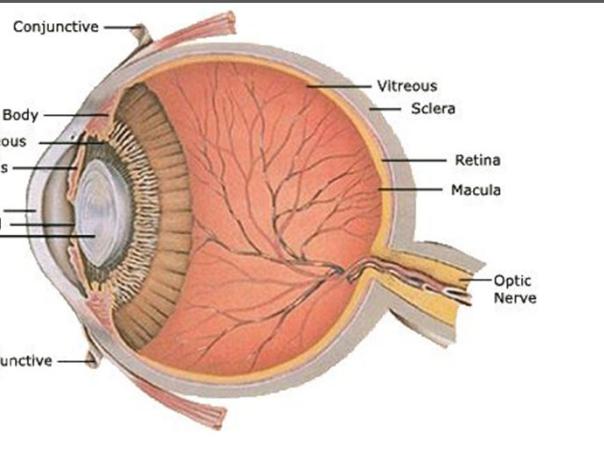
Both our [eyes](#) and [cameras](#) use an [adaptive lens](#) to control many aspects of image formation such as:

- [Aperture Size](#)
 - Controls the amount of light allowed through (f-stops in cameras)
 - Depth of Field (Bokeh)
- [Lens width](#) - Adjusts focus distance (near or far)



How Humans See

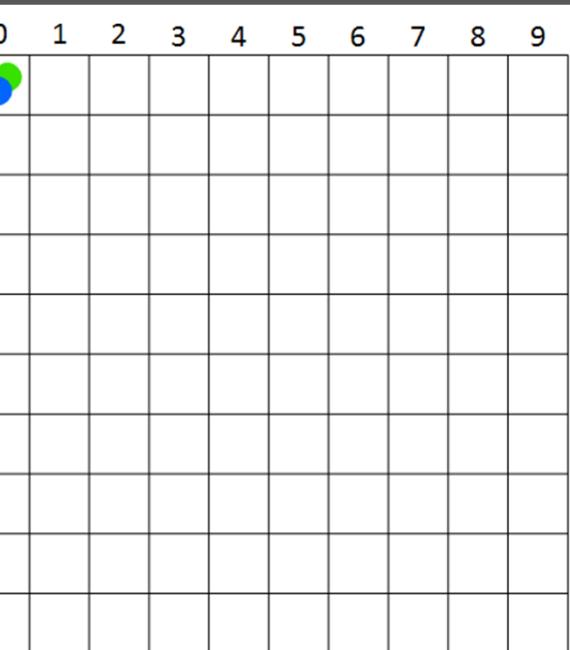
The human visual system (eye & visual cortex) is incredibly good at image processing



Check out - https://en.wikipedia.org/wiki/Visual_system

How do Computers store images?

OpenCV uses **RGB** color space by default.

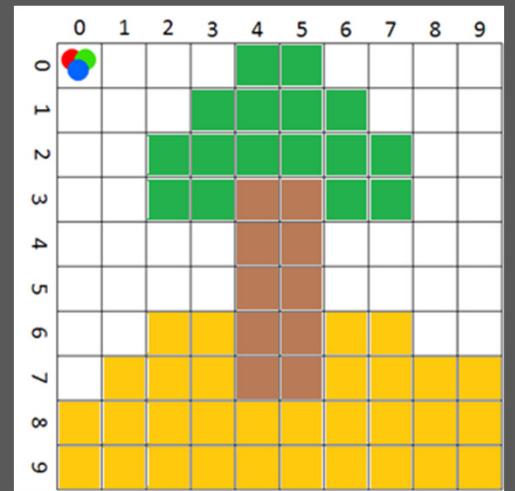


Each pixel coordinate (x, y) contains **3** values ranging for intensities of 0 to 255 (8-bit).

- Red
- Green
- Blue

Mixing different intensities of each color gives us the full color spectrum, example **Yellow**:

- Red – 255
- Green – 255
- Blue - 0



How does it look when stored on a computer?

Images are stored in multi-dimensional arrays

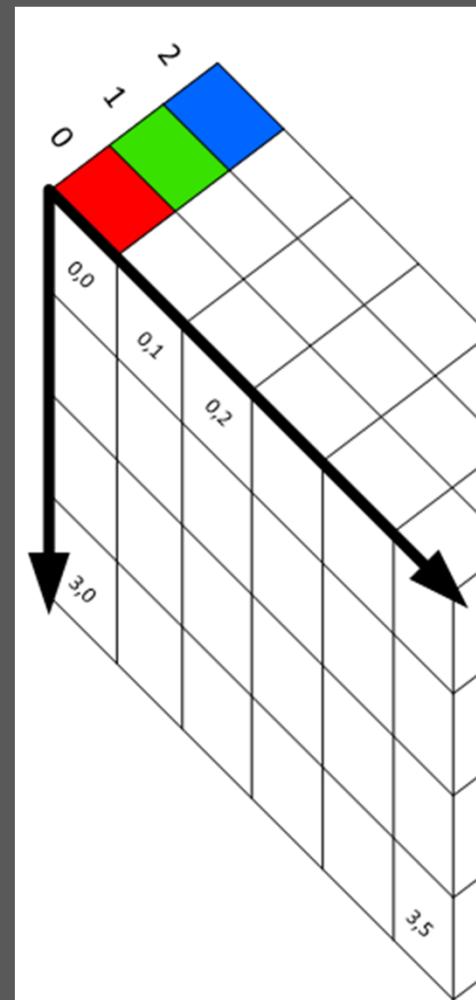
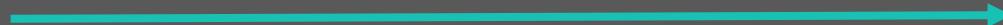
Think of an array as table. A **1-dimensional** arrays looks like this:

0	1	2	3	4	5	6	7

- A **2-dimensional** array looks like this:

0	1	2	3	4	5	6	7

3-dimensional array



Black and White or Greyscale

Black and White images are stored in
2-Dimensional arrays

There are two types of B&W images

- **Grayscale** – Ranges of shades of grey
 - **Binary** – Pixels are either black or white

What exactly is OpenCV

Officially launched in 1999, OpenCV ([Open Source Computer Vision](#)) from an Intel initiative.

OpenCV is written in C++.

First major release 1.0 was in 2006, second in 2009 and third in 2015.

OpenCV [2.4.13](#) is latest stable release for OpenCV 2.X.

Open 3.X is very similar, and has some benefits and new functions, however, it also removed some of the important algorithms (due to patents) such as SIFT & SURF.

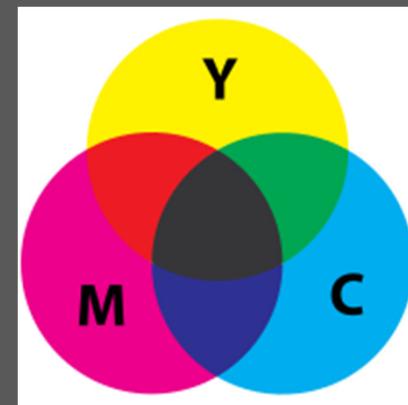
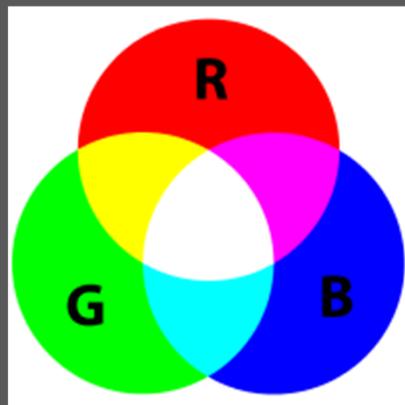
Visit - [opencv.org](#) or [wikipedia.org/wiki/OpenCV](#) to learn more

Let's now get started with
using **OpenCV** in Python.

Color Spaces

ever heard of the terms [RGB](#), [HSV](#) or [CMYK](#)?

These are color spaces, [which is simply a way to represent color](#).



RGB....wait BGR Color Space

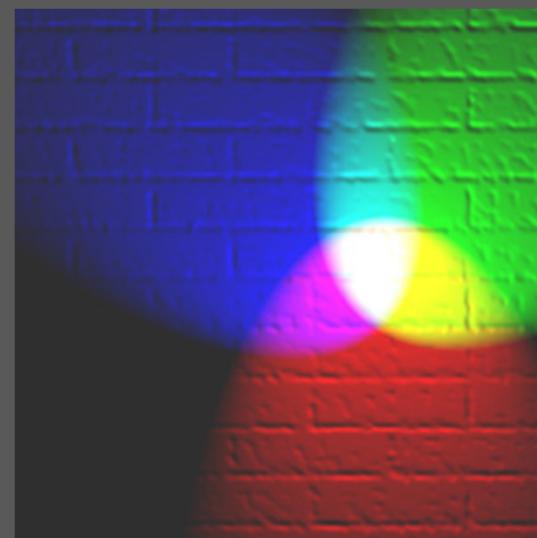
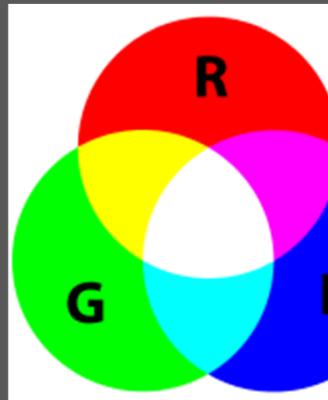
OpenCV's default color space is RGB.

RGB is an **additive color** model that generates colors by combining blue, green and red and different intensities/brightness. In OpenCV we have 8-bit color depths

- Red
- Green
- Blue

However, OpenCV actually stores color in the **BGR format**.

COOL FACT - We use BGR order on computers due to how unsigned 32-bit integers are stored in memory, it still ends up being stored as RGB. The integer representing a color e.g. 0x00BBGGRR will be stored as 0xRRGGBB00.



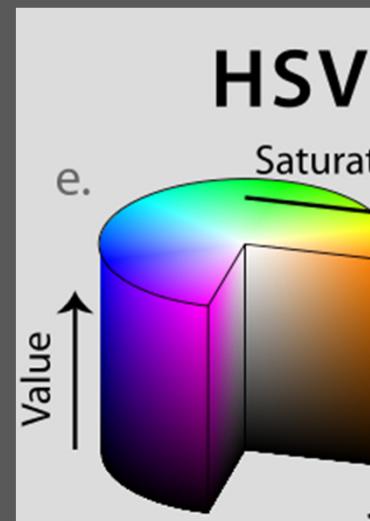
HSV Color Space

HSV (Hue, Saturation & Value/Brightness) is a color space that attempts to represent colors the way humans perceive it. It stores color information in cylindrical representation of RGB color points.

Hue – Color Value (0 – 179)

Saturation – Vibrancy of color (0-255)

Value – Brightness or intensity (0-255)



It's useful in computer vision for color segmentation. In RGB, filtering specific colors isn't easy, however, HSV makes it much easier to set color ranges to filter specific colors as we perceive them.

Visit these links to learn more:

[wikipedia.org/wiki/HSL_and_HSV](https://en.wikipedia.org/wiki/HSL_and_HSV)

http://coecsl.ece.illinois.edu/ge423/spring05/group8/FinalProject/HSV_writeup.pdf

Color Filtering

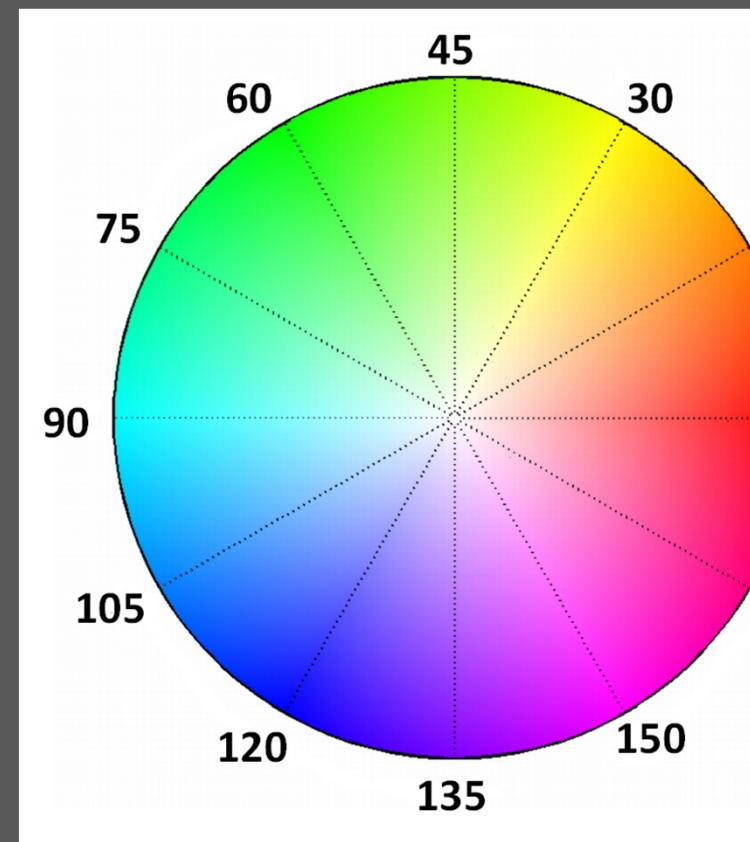
The Hue (Hue color range, goes from 0 to 180, not 360) and
is mapped differently than standard

Color Range Filters:

Red – 165 to 15

Green – 45 to 75

Blue – 90 to 120



Section 3

Image Manipulations

Image Manipulations

- . Transformations, affine and non affine
- . Translations
- . Rotations
- . Scaling, re-sizing and interpolations
- . Image Pyramids
- . Cropping
- . Arithmetic Operations
- . Bitwise Operations and Masking
- . Convolutions & Blurring
- 0. Sharpening
- 1. Thresholding and Binarization
- 2. Dilation, erosion, opening and closing
- 3. Edge Detection & Image Gradients
- 4. Perspective & Affine Transforms
- 5. **Mini Project # 1 – Make a Live Sketch of Yourself!**

Transformations

transformations – are geometric distortions enacted upon an image.

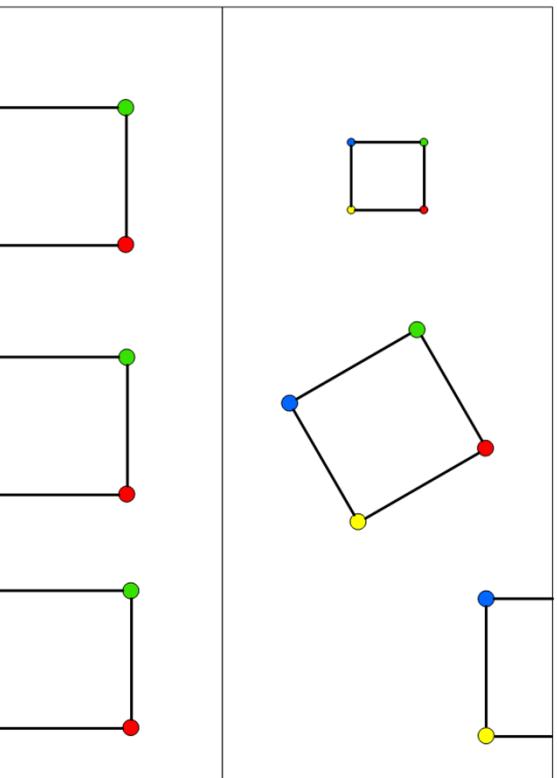
We use transformations to correct distortions or perspective issues from rising from the point of view an image was captured.

ypes:

Affine

Non-Affine

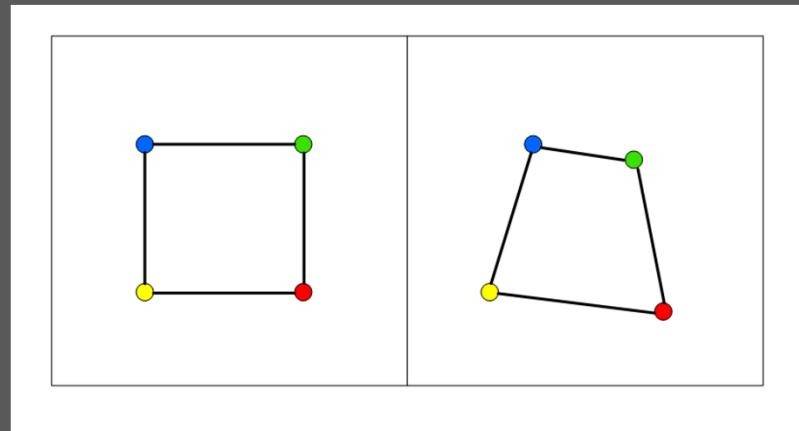
Affine vs Non Affine Theory



Scaling

Rotation

Translation



The non-affine or projective transformation does not preserve parallelism, length, and angle. It does however preserve collinearity and incidence.

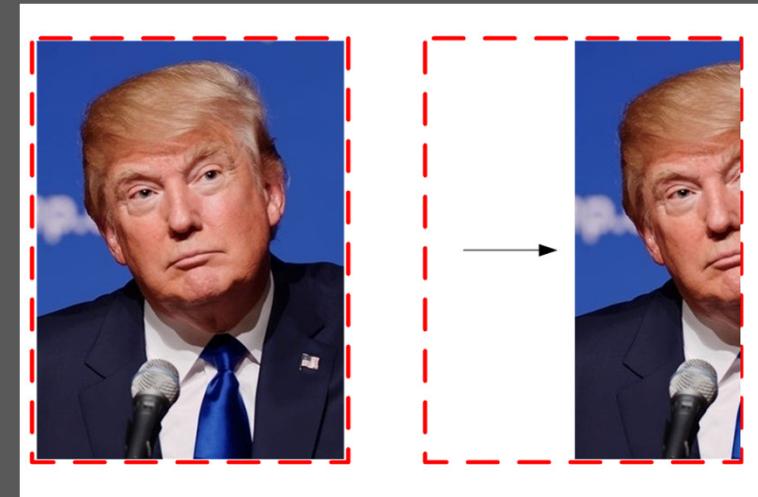
Non-Affine or Projective Transformations
also called Homogeneous Transformations

Translations

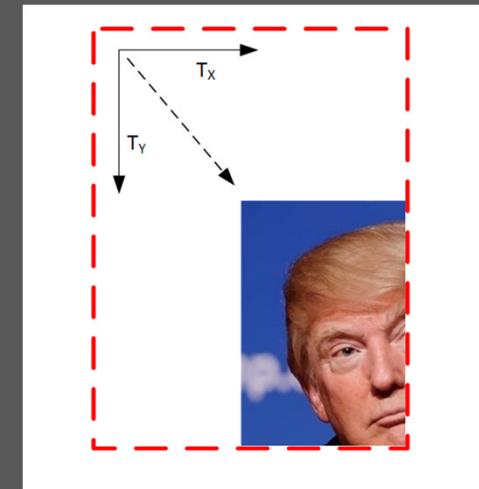
translation Matrix

$$= \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \end{bmatrix}$$

- Represents the shift along the x-axis (horizontal)
- Represents the shift along the y-axis (vertical)



We use the OpenCV function cv2.warpAffine to implement these translations

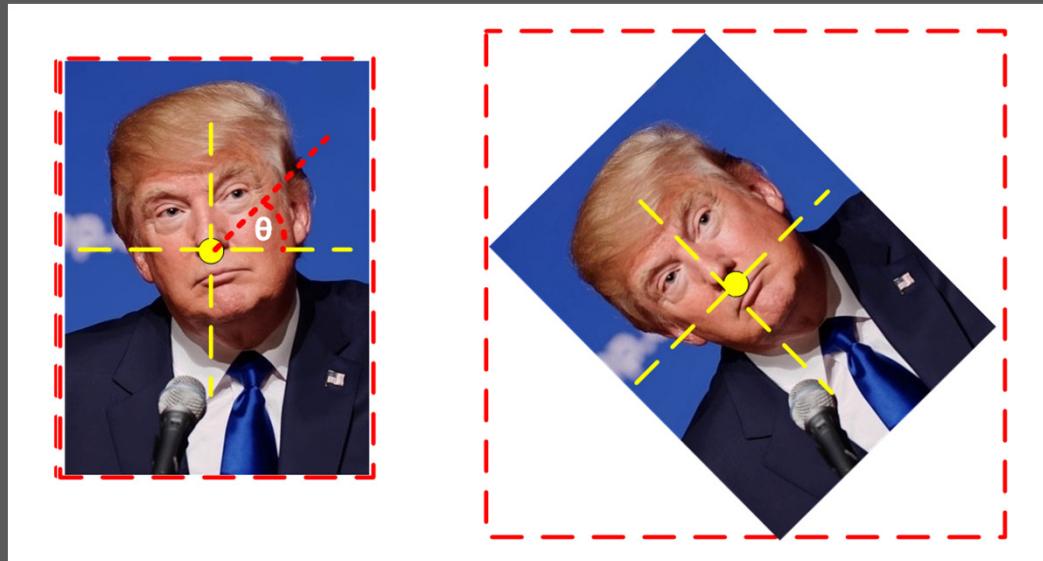


Rotations

Rotation Matrix

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

θ – the angle of rotation



OpenCV allows you to scale and rotate at the same time using the function:

```
v2.getRotationMatrix2D(rotation_center_x, rotation_center_y, angle of rotation, scale)
```

Re-sizing, Scaling and Interpolation

What is interpolation?

Interpolation is a method of constructing new data points within the range of a discrete set of known data points

v2.INTER_AREA - Good for shrinking or down sampling

v2.INTER_NEAREST - Fastest

v2.INTER_LINEAR - Good for zooming or up sampling (default)

v2.INTER_CUBIC - Better

v2.INTER_LANCZOS4 - Best

Good comparison of interpolation methods:

<http://tanbakuchi.com/posts/comparison-of-opencv-interpolation-algorithms/>

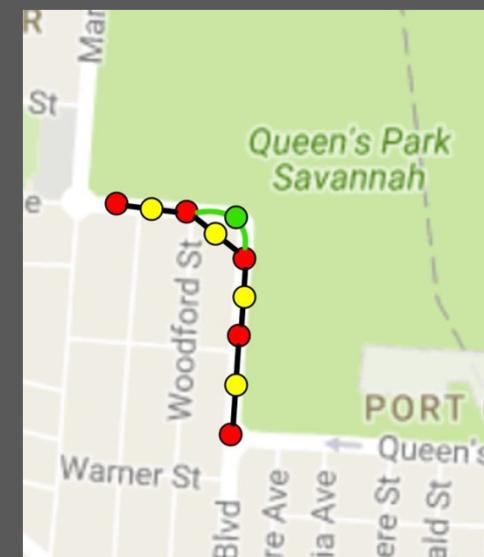


Image Pyramids

A pyramiding image refers to either upscaling (enlarging) and downscaling (shrinking) images.

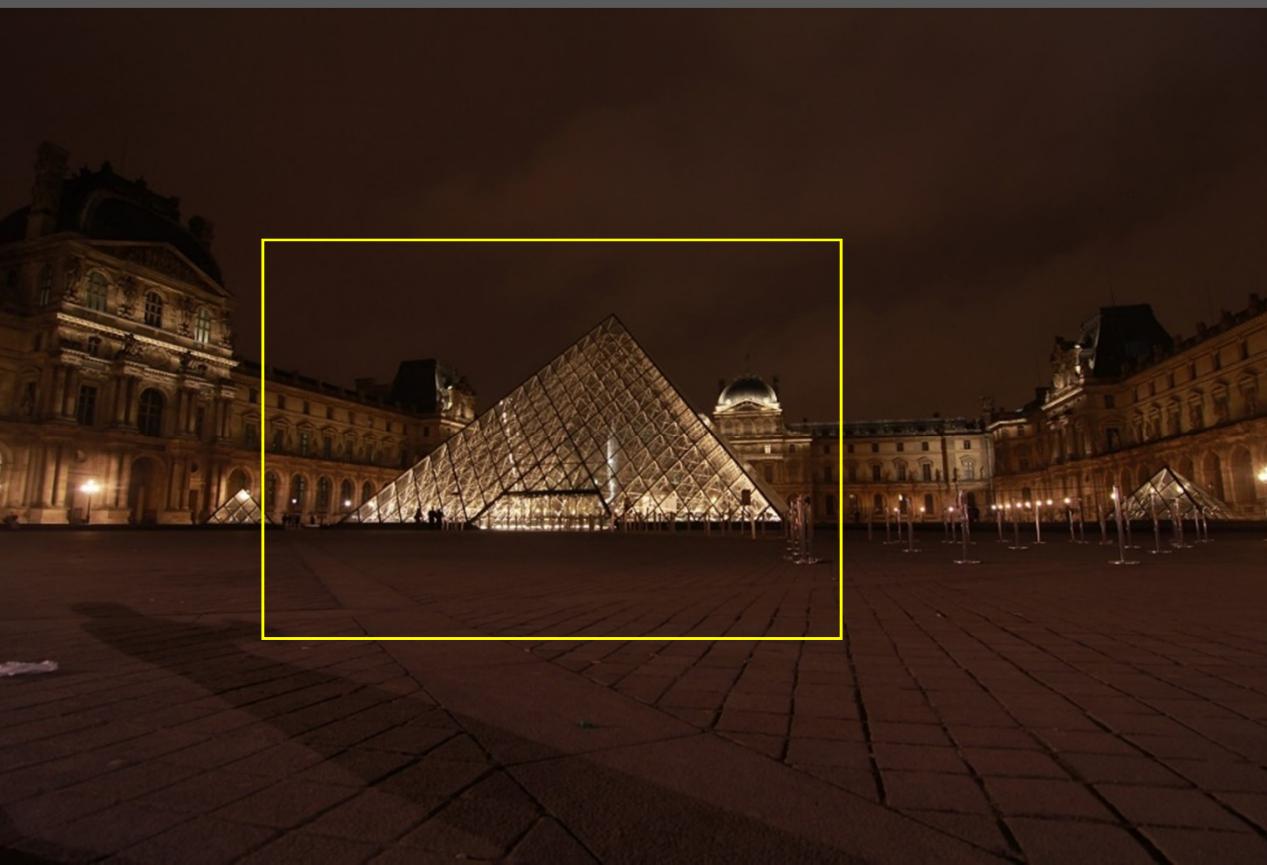
It's simply a different way of re-sizing that allows us to easily and quickly scale images. Scaling down reduces the height and width of the new image by half.

This comes in useful when making object detectors that scale images each time it looks for an object.



Cropping Images

Cropping images refers to extracting a segment a of that image.

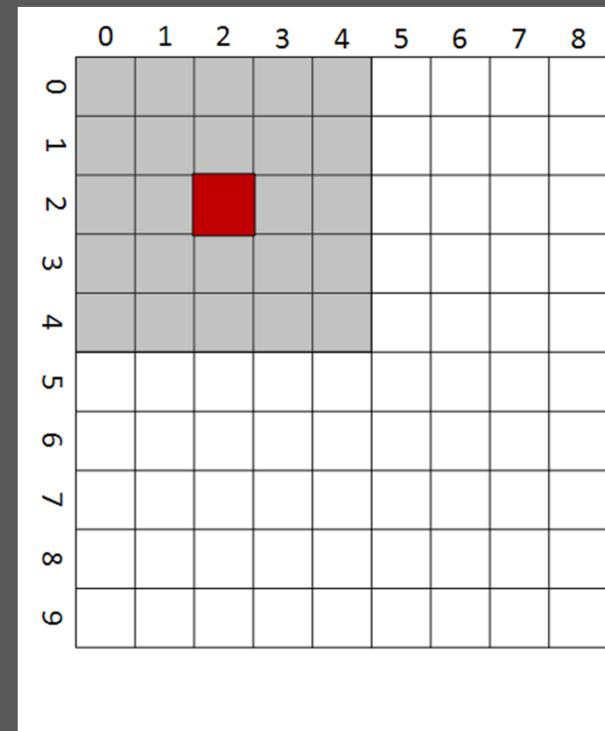


Convolutions & Blurring

Convolution is a mathematical operation performed on two functions producing a third function which is typically a modified version of one of the original functions.

$$\text{Output Image} = \text{Image} \otimes \text{Function}_{\text{Kernel Size}}$$

In Computer Vision we use kernels to specify the size over which we run our manipulating function over our image.



5 x 5 Kernel over our image

Blurring

Blurring is an operation where we average the pixels within a region (kernel).

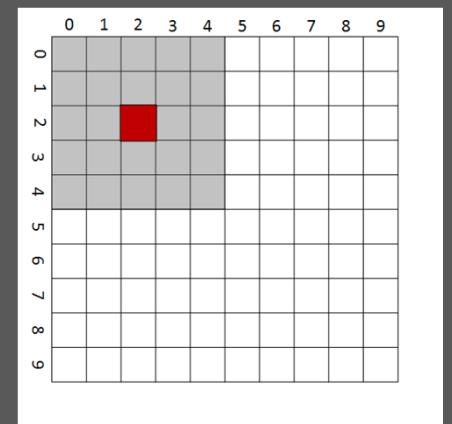
$$Kernel_ = \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$



The above is a 5×5 kernel.

We multiply by $1/25$ to normalize i.e. sum to 1, otherwise we'd be increasing intensity.

```
cv2.filter2D(image, -1, kernel)
```



5×5 Kernel over our image

Other Types of Blurring

`cv2.blur` – Averages values over a specified window

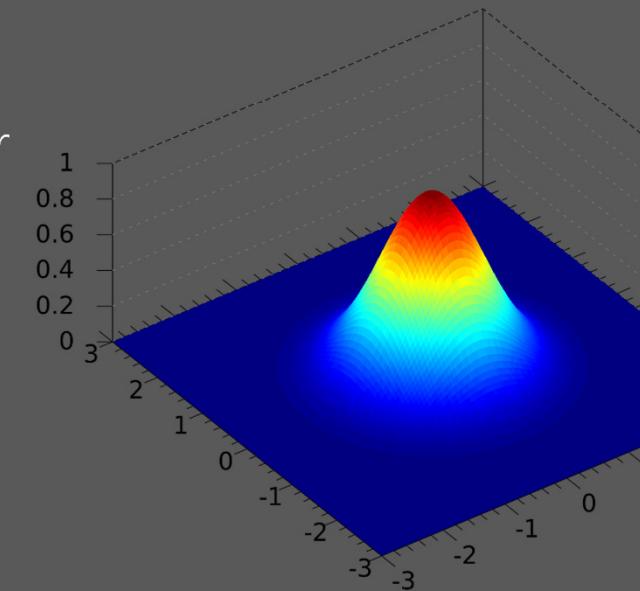
`cv2.GaussianBlur` – Similar, but uses a Gaussian window (more emphasis or weighting on points around the center)

`cv2.medianBlur` – Uses median of all elements in the window

`cv2.bilateralFilter` – Blur while keeping edges sharp (slower). It also takes a Gaussian filter in space, but one more Gaussian filter which is a function of pixel difference. The pixel difference function makes sure only those pixels with similar intensity to central pixel is considered for blurring. So it preserves the edges since pixels at edges will have large intensity variation.

1	4	7	4	
4	16	26	16	
7	26	41	26	
4	16	26	16	
1	4	7	4	

$\frac{1}{273}$

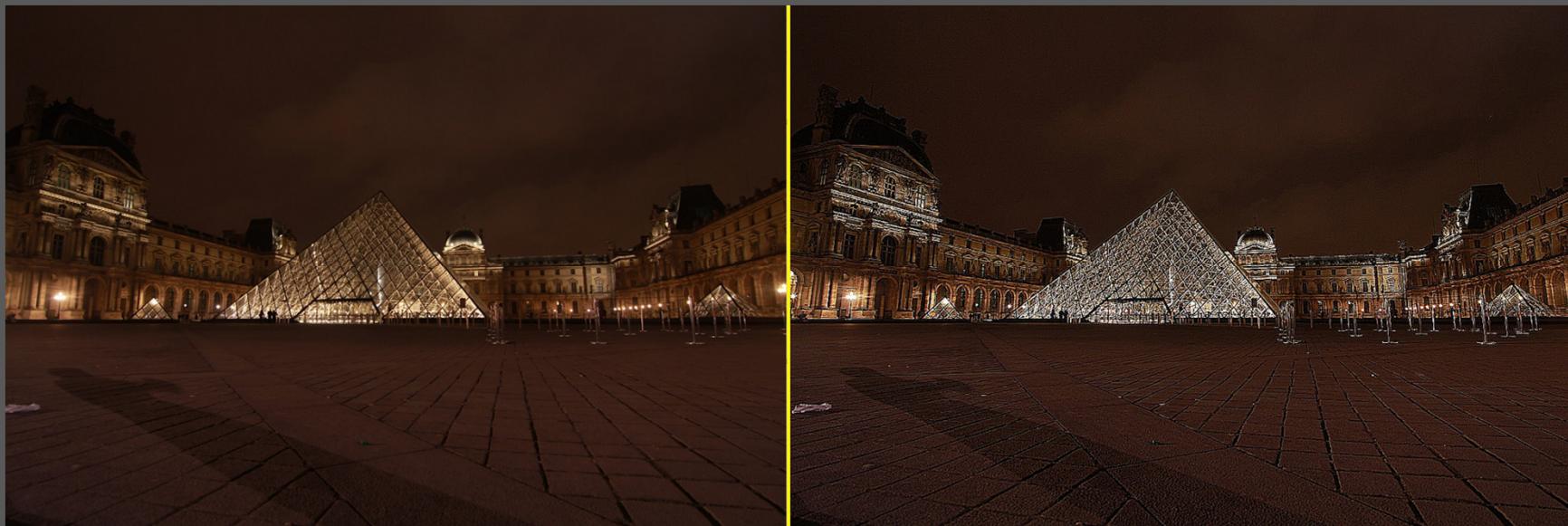


Sharpening

Sharpening is the opposite of blurring, it strengthens or emphasizes edges in an image

$$\text{Kernel}_s = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Our kernel matrix sums to one, so there is no need to normalize (i.e. multiply by a factor to retain the same brightness of the original)



Thresholding, Binarization & Adaptive Thresholding

Thresholding is act of converting an image to a binary form.

```
cv2.threshold(image, Threshold Value, Max Value, Threshold Type)
```

threshold Types:

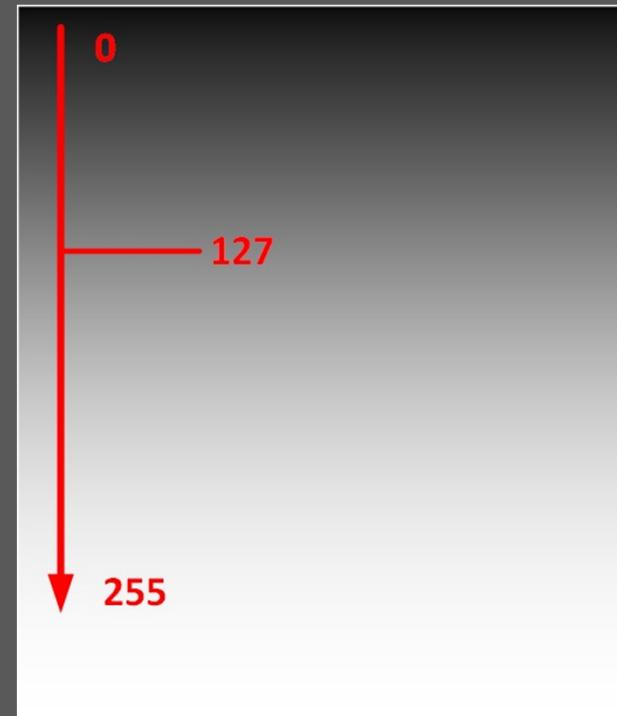
cv2.THRESH_BINARY – Most common

cv2.THRESH_BINARY_INV – Most common

cv2.THRESH_TRUNC

cv2.THRESH_TOZERO

cv2.THRESH_TOZERO_INV



NOTE: Image need to be converted to greyscale before thresholding.

Adaptive Thresholding

Simple threshold methods require us to provide the threshold value.

Adaptive threshold methods take that uncertainty away

`cv2.adaptiveThreshold`(image, Max Value, Adaptive type, Threshold Type, Block size, Constant that is subtracted from mean)

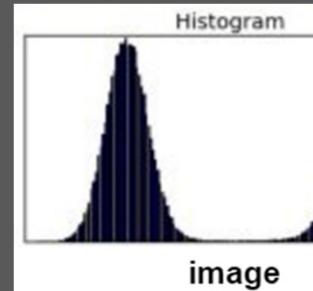
OTE: Block sizes need to be odd numbers!

Adaptive Threshold Types:

`ADAPTIVE_THRESH_MEAN_C` – based on mean of the neighborhood of pixels

`ADAPTIVE_THRESH_GAUSSIAN_C` – weighted sum of neighborhood pixels under the Gaussian window

`THRESH_OTSU` ([uses cv2.threshold function](#)) – Clever algorithm assumes there are two peaks in the gray scale histogram of the image and then tries to find an optimal value to separate these two peaks to find T.



Dilation and Erosion

These are operations in the field of mathematical morphology:

Dilation – Adds pixels to the boundaries of objects in an image

erosion – Removes pixels at the boundaries of objects in an image

Opening - Erosion followed by dilation

Closing - Dilation followed by erosion

Dilation

Confusion with Dilation and Erosion

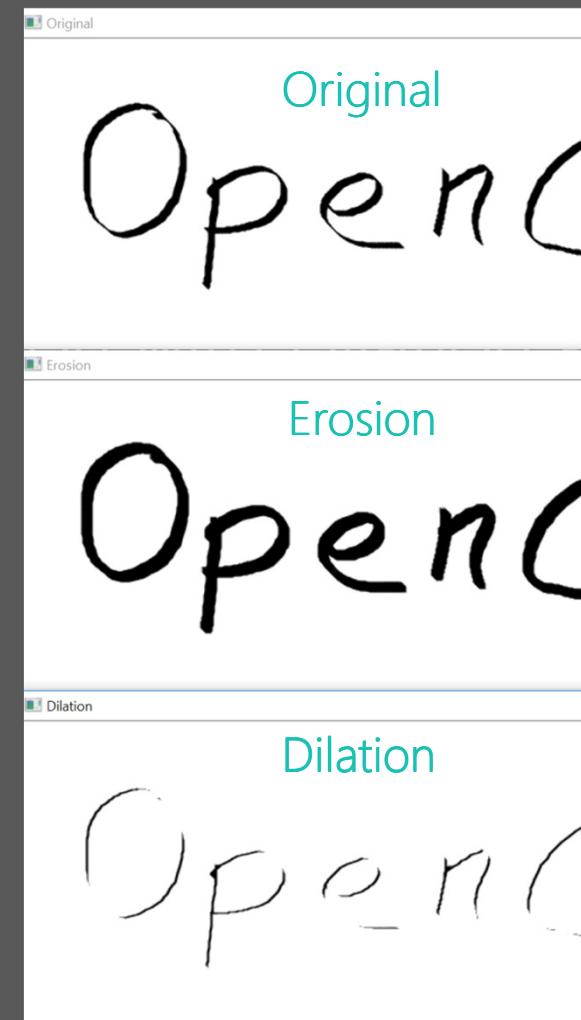
Common StackOverflow question:

"Why is dilation and erosion doing the reverse of what I expect?"

remember:

Dilation – Adds pixels to the boundaries of objects in an image

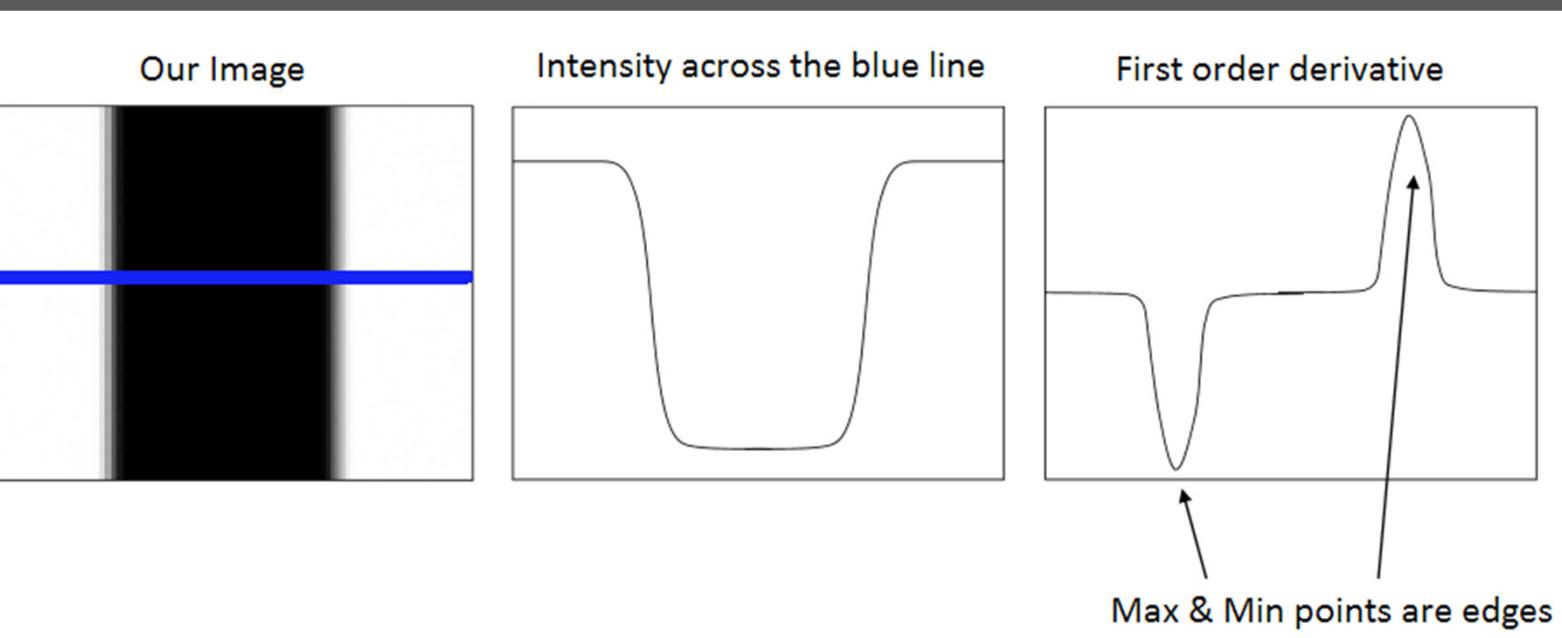
Erosion – Removes pixels at the boundaries of objects in an image



Edge Detection & Image Gradients

Edge Detection is a very important area in Computer Vision, especially when dealing with contours (you'll learn this later soon).

Edges can be defined as sudden changes (discontinuities) in an image and they can encode just as much information as pixels.



Edge Detection Algorithms

here are three main types of Edge Detection:

Sobel – to emphasize vertical or horizontal edges

Laplacian – Gets all orientations

Canny – Optimal due to low error rate, well defined edges and accurate detection.

[Canny Edge Detection Algorithm](#) (developed by John F. Canny in 1986)

Applies Gaussian blurring

Finds intensity gradient of the image

Applied non-maximum suppression (i.e. removes pixels that are not edges)

Hysteresis – Applies thresholds (i.e. if pixel is within the upper and lower thresholds, it is considered an edge)

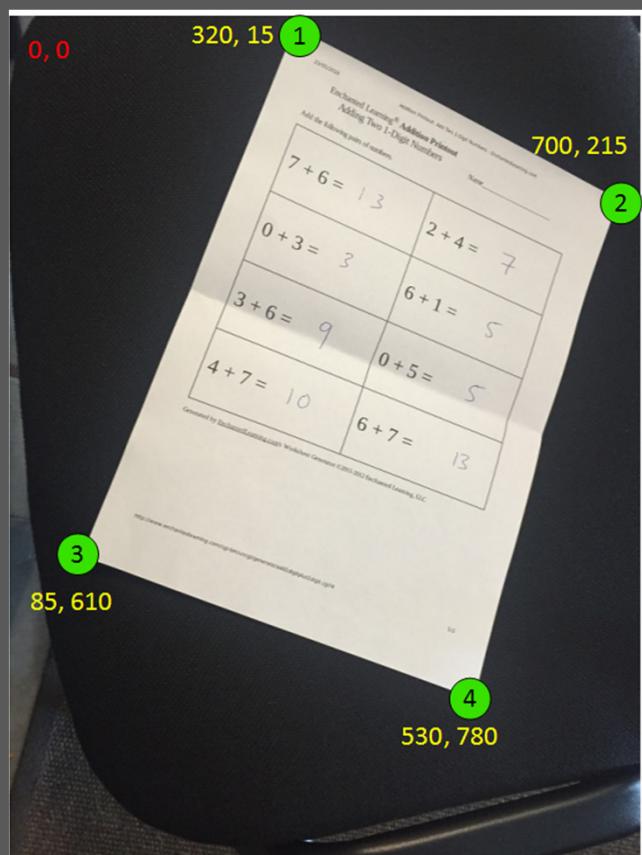


Learn more here – https://en.wikipedia.org/wiki/Canny_edge_detector

Comparison of Edge Detection Techniques - <http://www.ijcsi.org/papers/IJCSI-9-5-1-269-276.pdf>

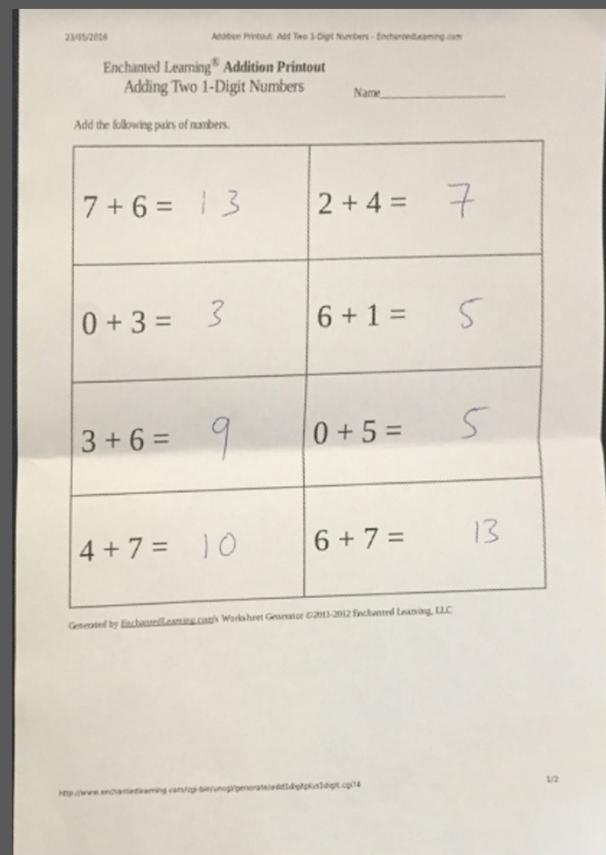
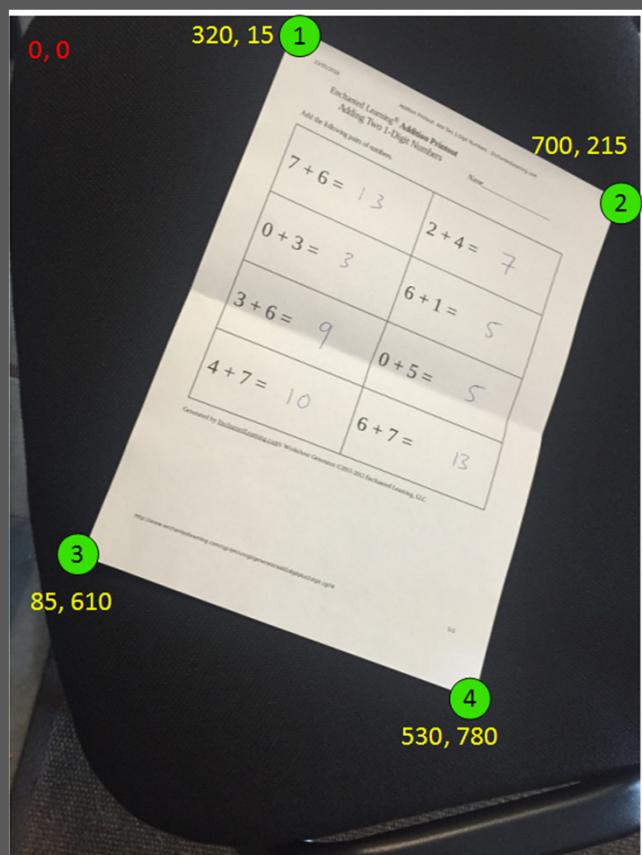
Obtaining the Perspective of Non-Affine Transforms

What if we had a mapping of points between images?

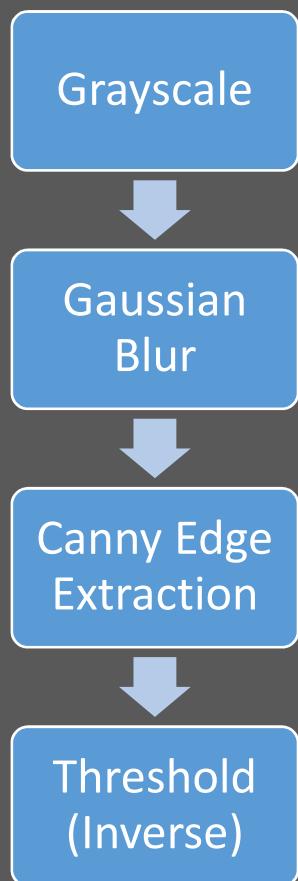


Obtaining the Perspective of Affine Transforms

What if we had a mapping of points between images?



Mini Project # 1 – Live Sketch Using Webcam



Section 4

Image Segmentation

Segmentation - Partitioning images into different regions

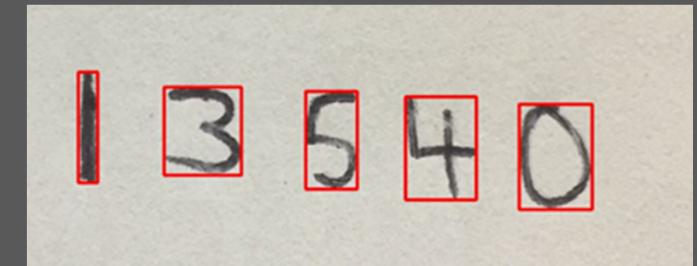
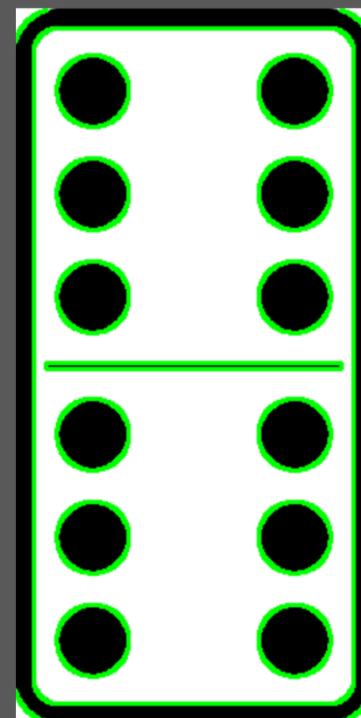
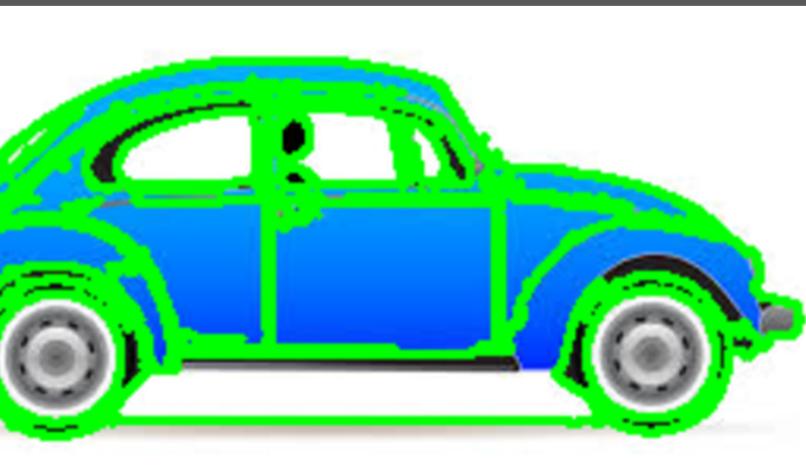


Image Segmentation

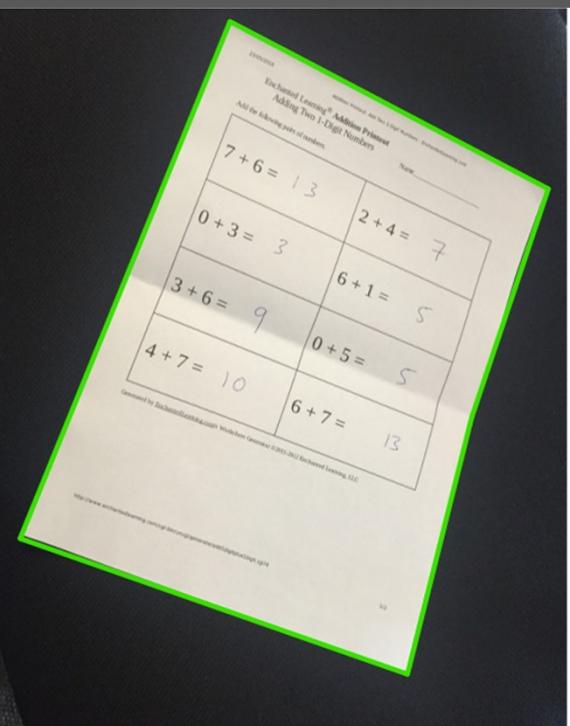
- . Understanding contours
- . Sorting contours by size or left to right
- . Approximating contours & finding their convex hull
- . Matching Contour Shapes
- . **Mini Project # 2 – Identifying Shapes**
- . Line Detection
- . Circle Detection
- . Blob Detection
- . **Mini Project # 3 – Counting Circles and Ellipses**

Contours

Contours are continuous lines or curves that bound or cover the full boundary of an object in an image.

Contours are very important in:

- Object Detection
- Shape Analysis



Contours

OpenCV stores Contours in a list of lists.

Contours in OpenCV

```
cv2.findContours (image, Retrieval Mode, Approximation Method)
```

- **Returns** -> contours, hierarchy
- **Contours** are stored as a numpy array of (x,y) points that form the contour
- **Hierarchy** describes the child-parent relationships between contours (i.e. contours within contours)

Retrieval Mode:

- **cv2.CHAIN_APPROX_NONE** – Stores all the points along the line (inefficient!)
- **cv2.CHAIN_APPROX_SIMPLE** – Stores the end points of each line

NOTE: Differences between OpenCV 2.4 and OpenCV 3.X:

cv2.findContours – returns 3 arguments in OpenCV3, which is the modified image with contours overlaid

Drawing Contours

```
v2.drawContours(image, contours, specific contour, color,  
thickness)
```

- The 'contours' parameter is the output from findContours
- Specific contour relates to which contour you wish to draw e.g. Contour[0], Contour[1].
- If you wish to draw all contours, use '-1'

Hierarchy in Contours

Hierarchy Types (the first two are the most useful)

cv2.RETR_LIST – Retrieves all contours

cv2.RETR_EXTERNAL - Retrieves external or outer contours only

cv2.RETR_COMP - Retrieves all in a 2-level hierarchy

cv2.RETR_TREE - Retrieves all in full hierarchy

Hierarchy is stored in the following format: [Next, Previous, First Child, Parent]

NOTE - Contour Hierarchy is a quite lengthy to explain, if you're interested read here:

http://opencv-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_contours_hierarchy/py_contours_hierarchy.html

Sorting Contours

Sorting contours is quite useful when doing image processing.

Sorting by Area can assist in Object Recognition (using contour area)

- Eliminate small contours that may be noise
- Extract the largest contour

Sorting by spatial position (using the contour centroid)

- Sort characters left to right
- Process images in specific order

Contour Moments

Moments are a set of scalers that are an aggregate of set of vectors

Used in Mechanics and Statistics quite frequently and has now been adopted in Computer Vision

Without getting to heavy into the mathematics behind it think of it as a measure of image intensities

An image with pixel intensities $I(x,y)$, moments are given by:

$$M_{ij} = \sum_x \sum_y I(x, y)$$

Approximating Contours

Approximating contours is useful when correcting slight distortions your contour.

We can use approxPolyDP to achieve this:

`v2.approxPolyDP (contour, Approximation Accuracy, Closed)`

- `Contour` – is the individual contour we wish to approximate
- `Approximation Accuracy` – Important parameter is determining the accuracy of the approximation. Small values give precise approximations, large values give more generic approximation. A good rule of thumb is less than 5% of the contour perimeter
- `Closed` – a Boolean value that states whether the approximate contour should be open or closed

Using OpenCV's matchShapes function

cv2.matchShapes(contour template, contour, method, method parameter)

- Output – match value (lower values means a closer match)

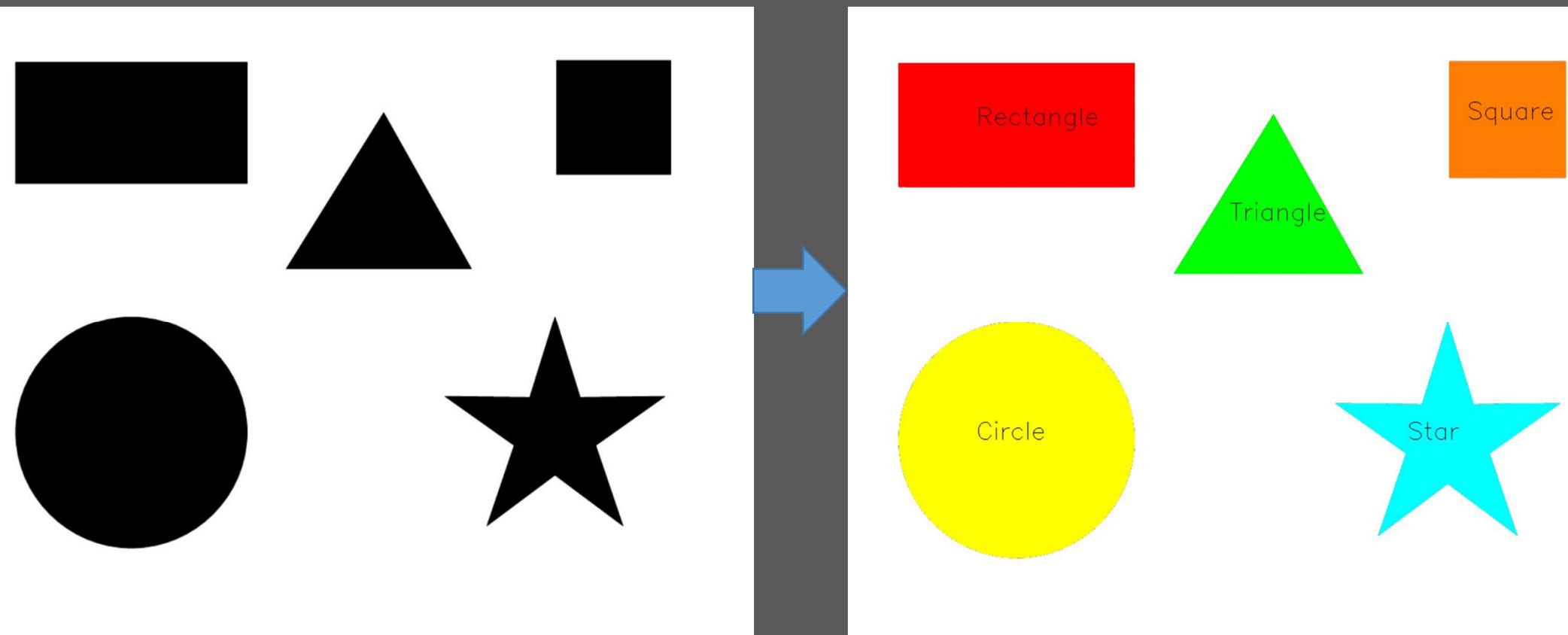
Contour Template – This is our reference contour that we're trying to find in the new image

Contour – The individual contour we are checking against

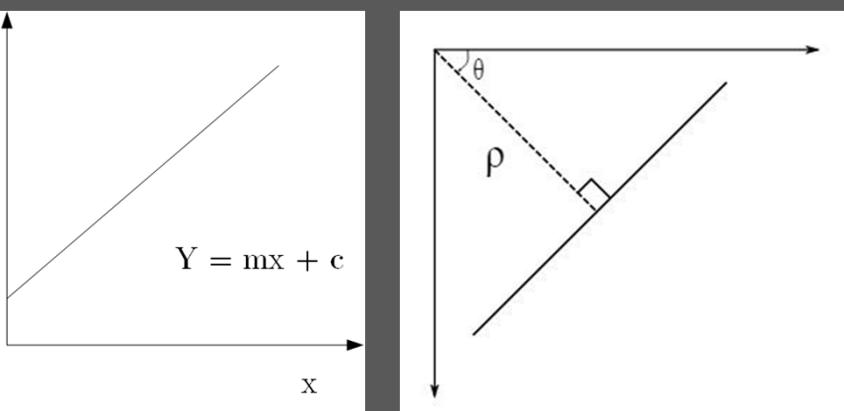
Method – Type of contour matching (1, 2, 3)

Method Parameter – leave alone as 0.0 (not fully utilized in python OpenCV)

Mini Project # 2 – Shape Matching

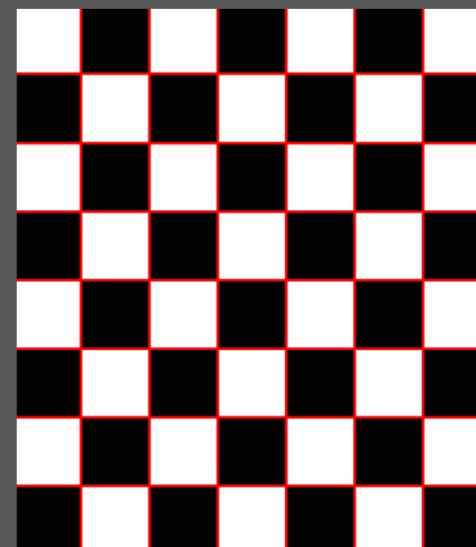


Line Detection – Hough Lines & Probabilistic Hough Lines



$$\rho = x\cos\theta + y\sin\theta$$

ρ is the perpendicular distance from origin
 θ is the angle formed by the normal of this line to the origin (measured in radians)



`v2.HoughLines(binarized image, rho accuracy, theta accuracy, threshold)`

Threshold here is the minimum vote for it to be considered a line

Probabilistic Hough Lines (<http://cmp.felk.cvut.cz/~matas/papers/matas-bmvc98.pdf>)

Idea is that it takes only a random subset of points sufficient enough for line detection

Also returns the start and end points of the line unlike the previous function

`v2.HoughLinesP(binarized image, rho accuracy, theta accuracy, threshold, min line length, max line gap)`

Circle Detection

```
cv2.HoughCircles(image, method, dp, MinDist, param1, param2, minRadius, MaxRadius)
```

Method - currently only cv2.HOUGH_GRADIENT available

dp - Inverse ratio of accumulator resolution

MinDist - the minimum distance between the center of detected circles

param1 - Gradient value used in the edge detection

param2 - Accumulator threshold for the HOUGH_GRADIENT method,

- Lower allows more circles to be detected (false positives)

minRadius - limits the smallest circle to this size (via radius)

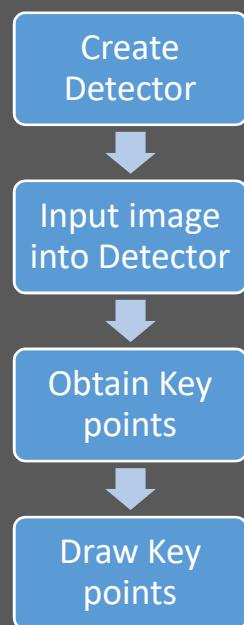
MaxRadius - similarly sets the limit for the largest circles



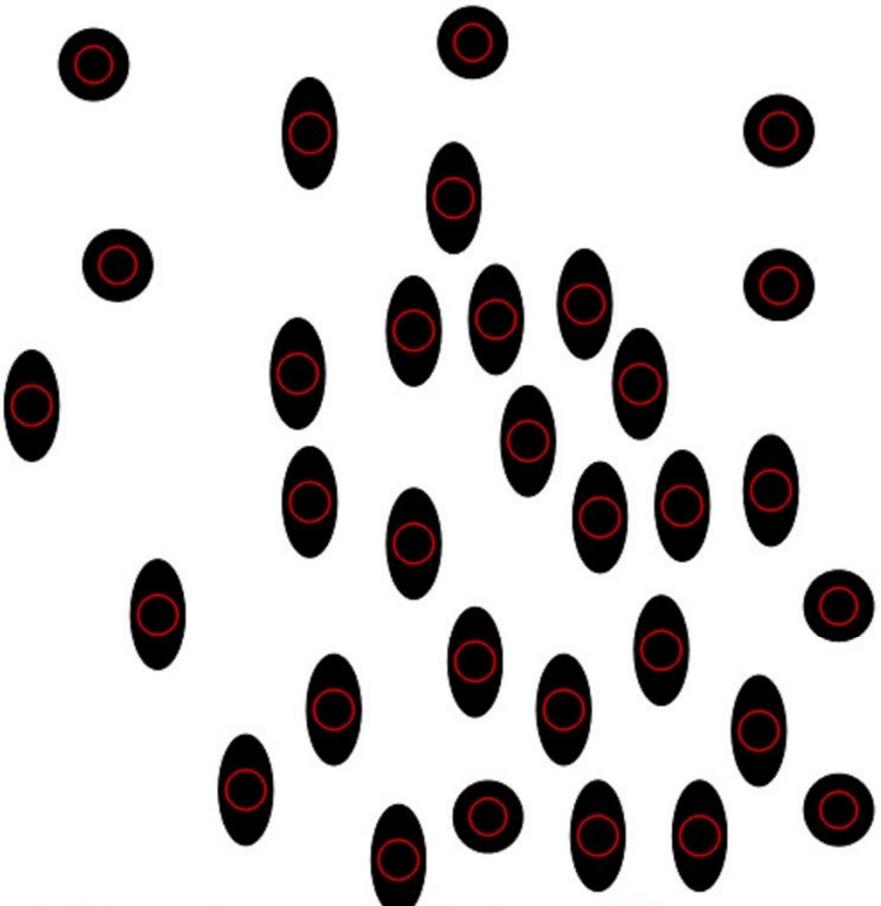
Blob Detection

Blobs can be described as groups of connected pixels that all share a common property.

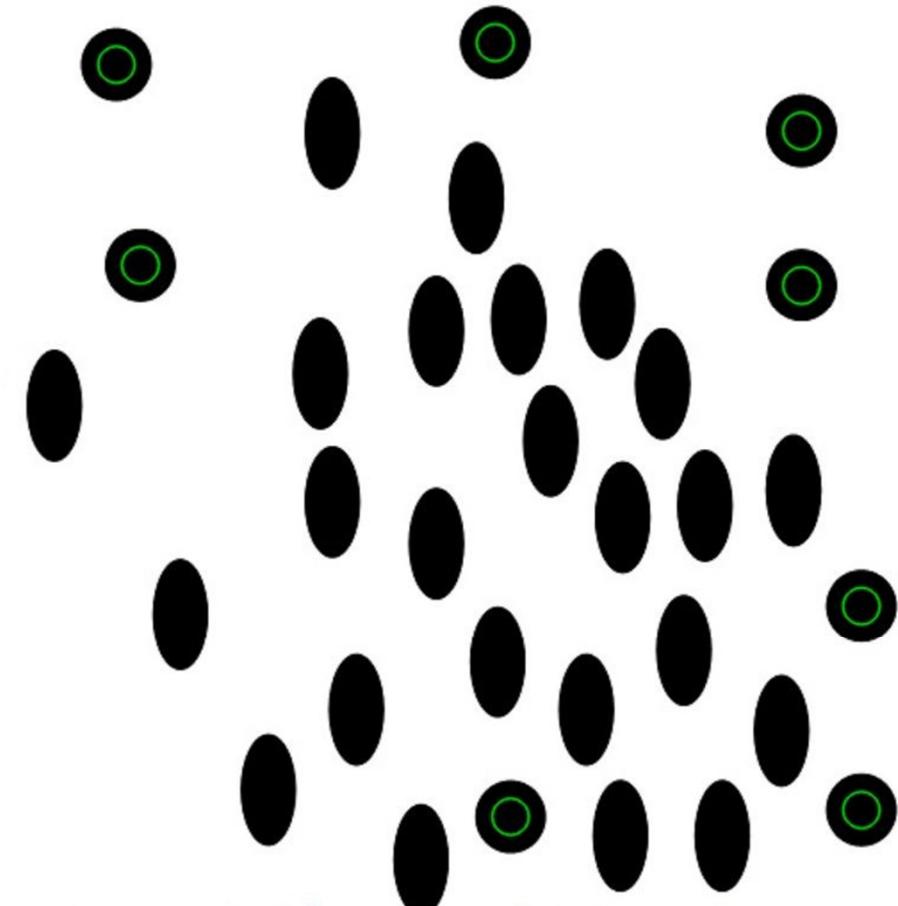
How to use OpenCV's simpleBlobDetector?



Mini Project # 3 – Counting Circles and Ellipses



Total Number of Blobs: 32



Number of Circular Blobs: 8

Blob Filtering – Shape & Size

cv2.SimpleBlobDetector_Params()

area

params.filterByArea = True/False
params.minArea = pixels
params.maxArea = pixels

circularity

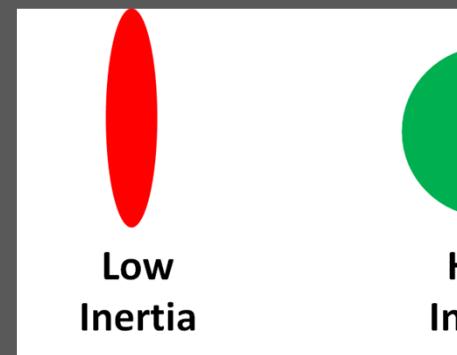
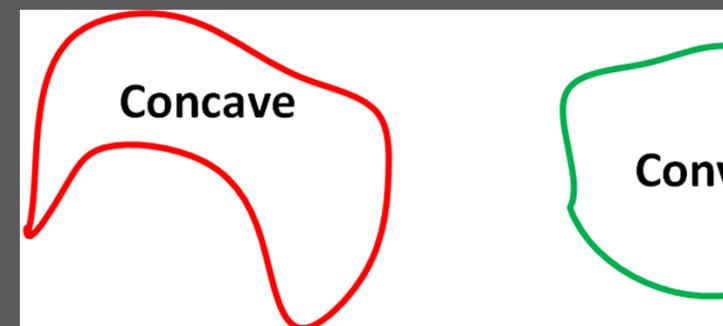
params.filterByCircularity = True/False
params.minCircularity = 1 being perfect circle, 0 the opposite

convexity – Area of blob / Area of Convex Hull

params.filterByConvexity = True/False
params.minConvexity = 0 to 1

inertia – Measure of ellipticalness (low being more elliptical, high being more circular)

params.filterByInertia = True/False
params.minInertiaRatio = 0.01



Section 5

Object Detection

Object Detection

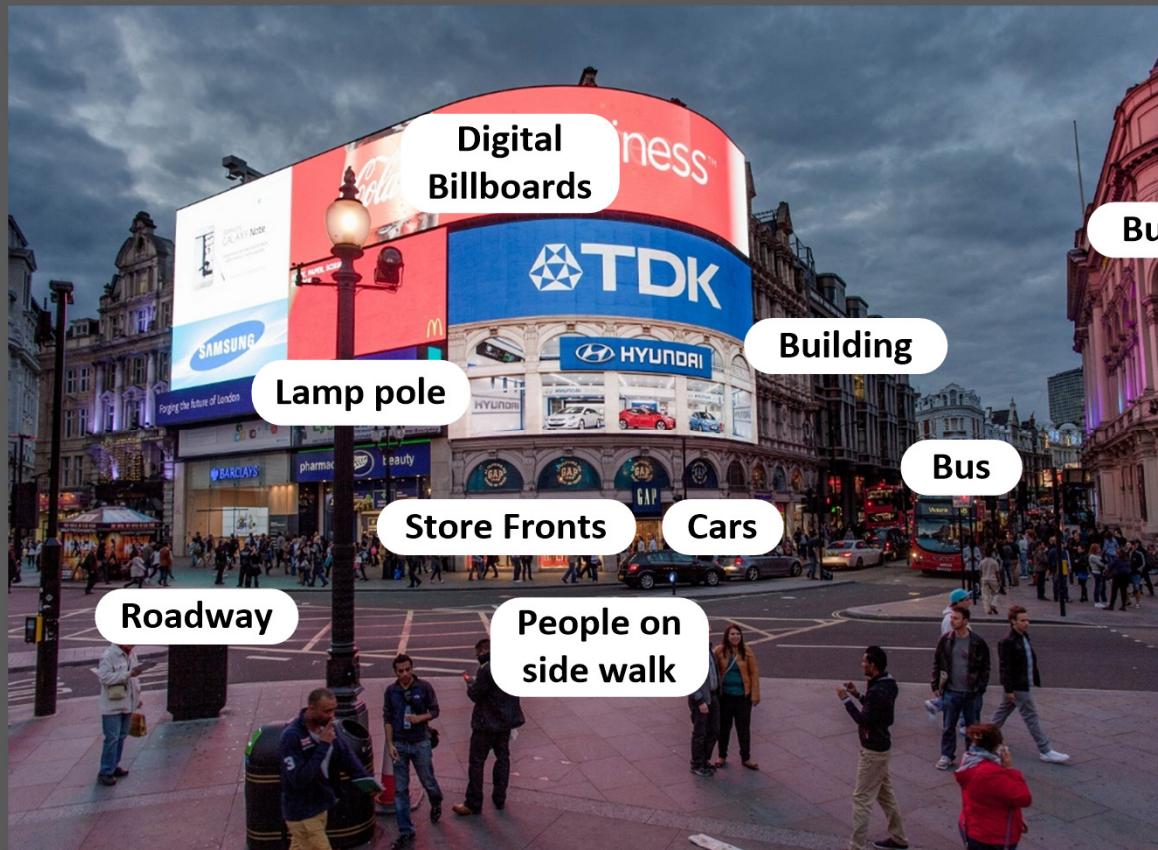
- . Object Detection using Template Matching
- . Mini Project – Finding Waldo
- . Feature Description Theory
- . Finding Corners
- . SIFT, SURF, FAST, BREIF & ORB
- . Mini Project – Object Detection using Features
- . Histogram of Gradients (HoG) as a Descriptor

Object Detection

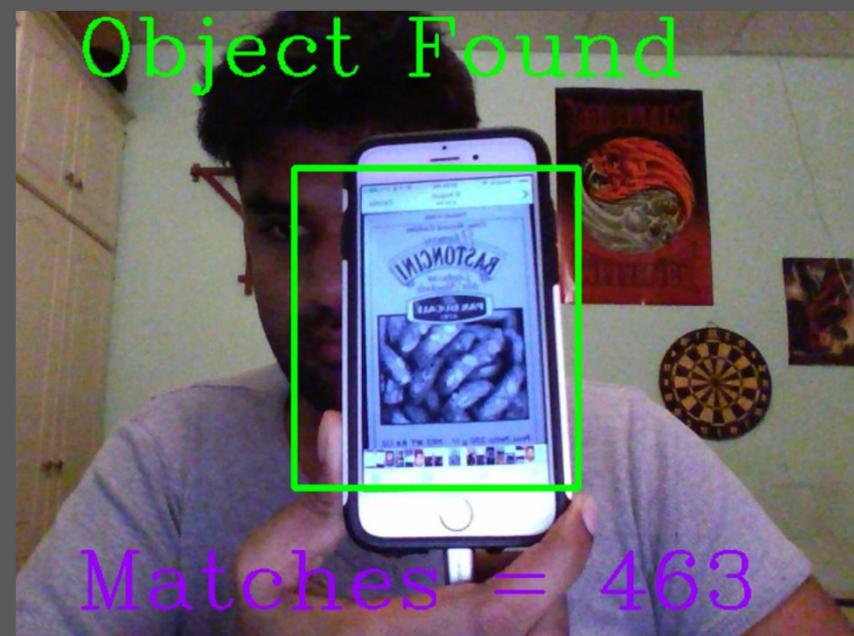
Why do we need **detect objects** in image?

- Labeling Scenes
- Robot Navigation
- Self Driving Cars
- Body Recognition (Microsoft Kinect)
- Disease & Cancer Detection
- Facial Recognition
- Handwriting Recognition
- Identifying objects in satellite images

C.

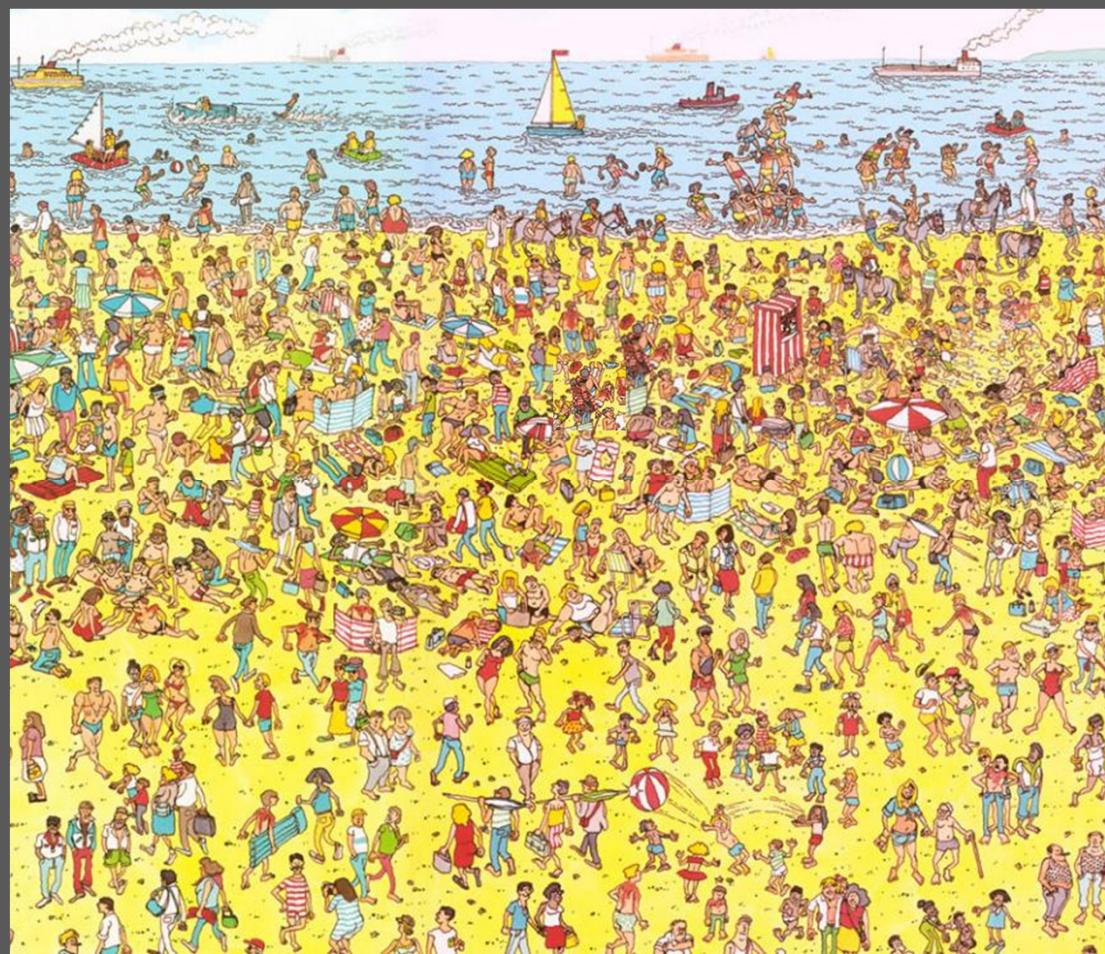


Object Detection vs Recognition

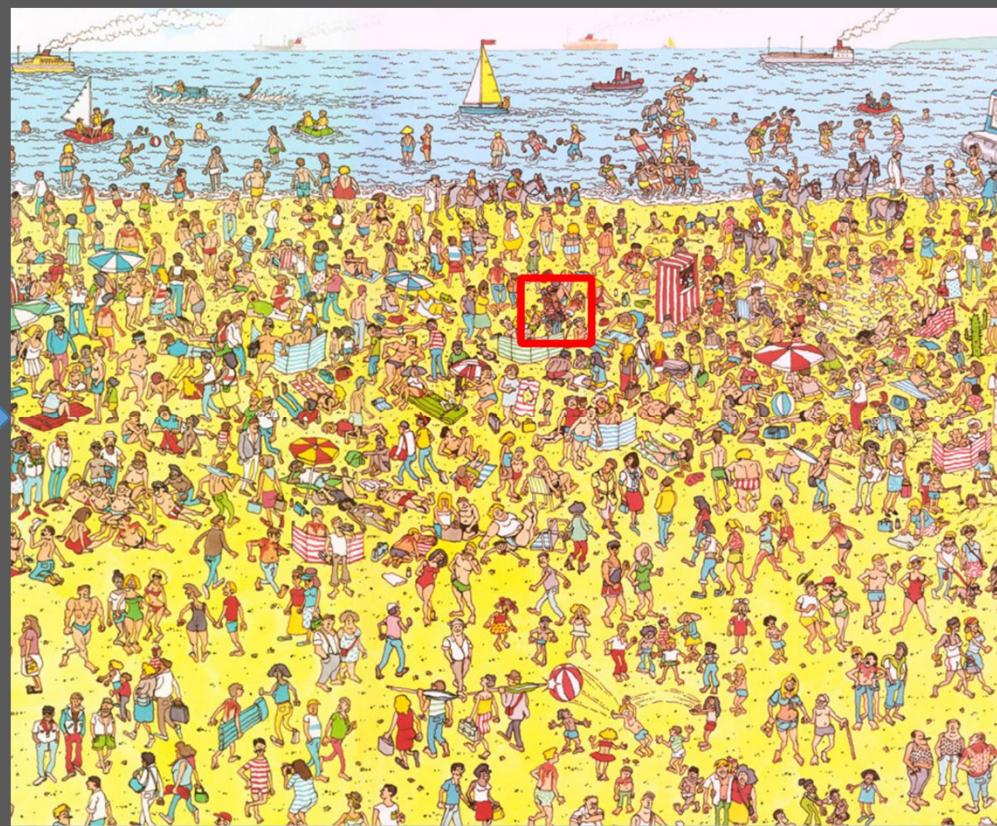
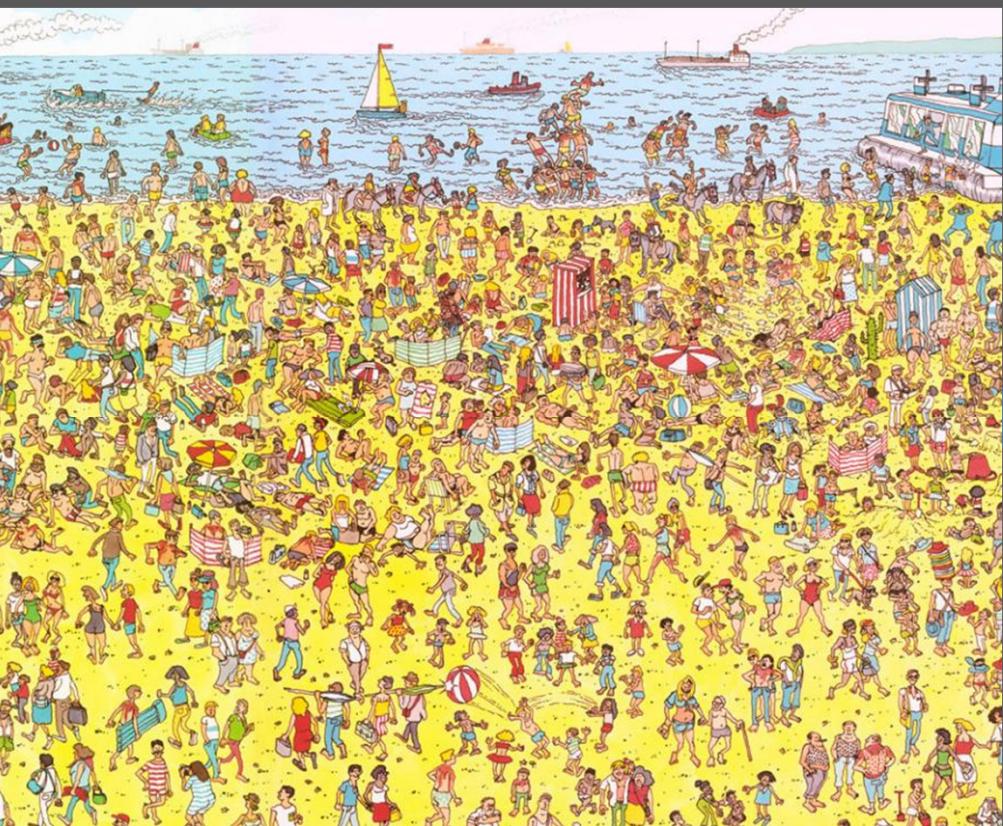


Template Matching

What if we slide a template image across a source image until the a match is found.



Mini Project # 4 – Finding Waldo



This method isn't very resilient

Rotation renders this method ineffective



Size (known as scaling) affects this as well



Photometric changes (e.g. brightness, contrast, hue etc.)

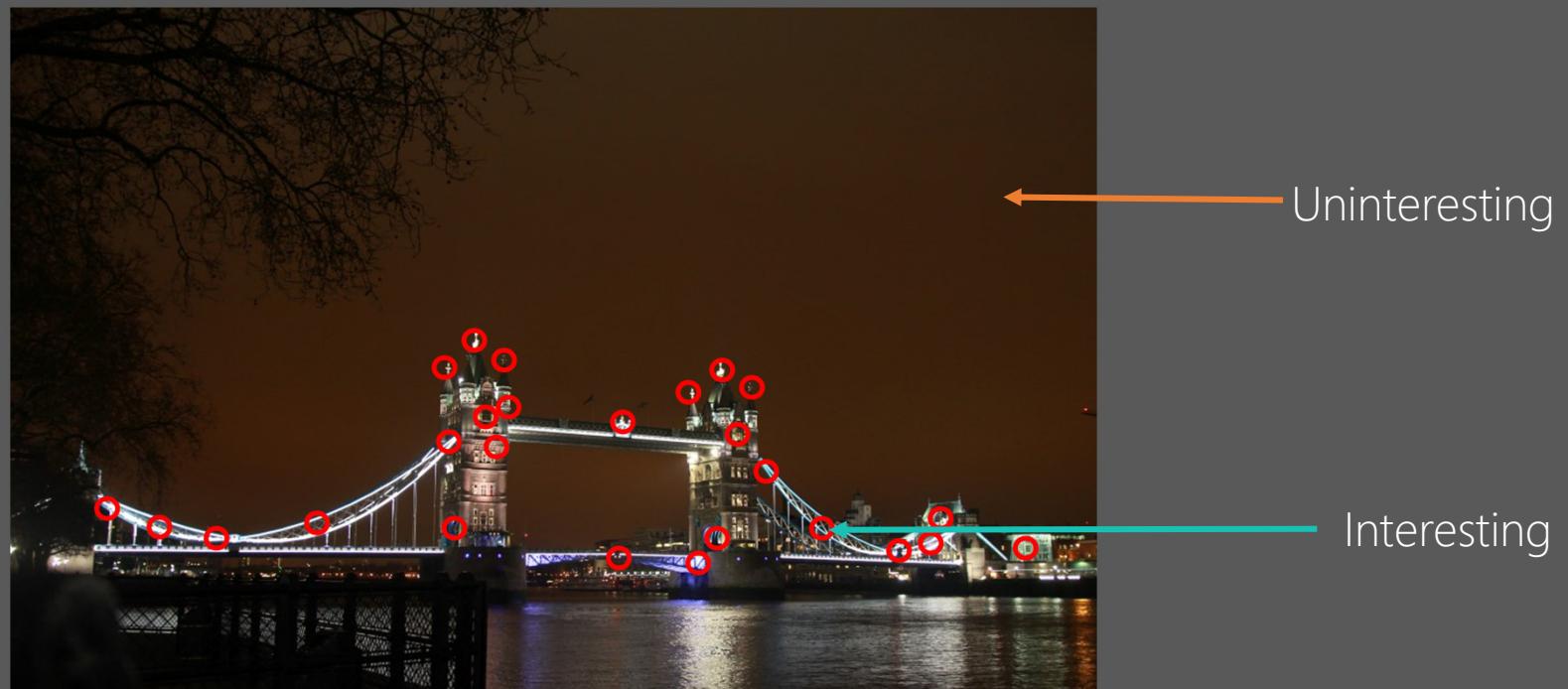


Distortions from view point changes (Affine)



Image Features

Image Features are **interesting** areas an image that are somewhat **unique** to that specific image. They are also popularly called **key point features** or **interest points**.

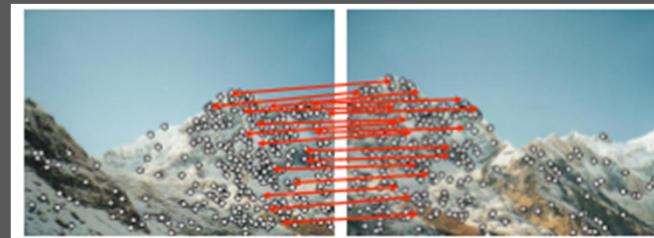
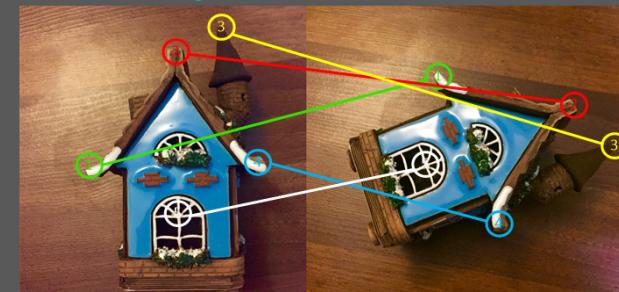


Why is this important?

Features are important as they can be used to analyze, describe and match images.

They are used in:

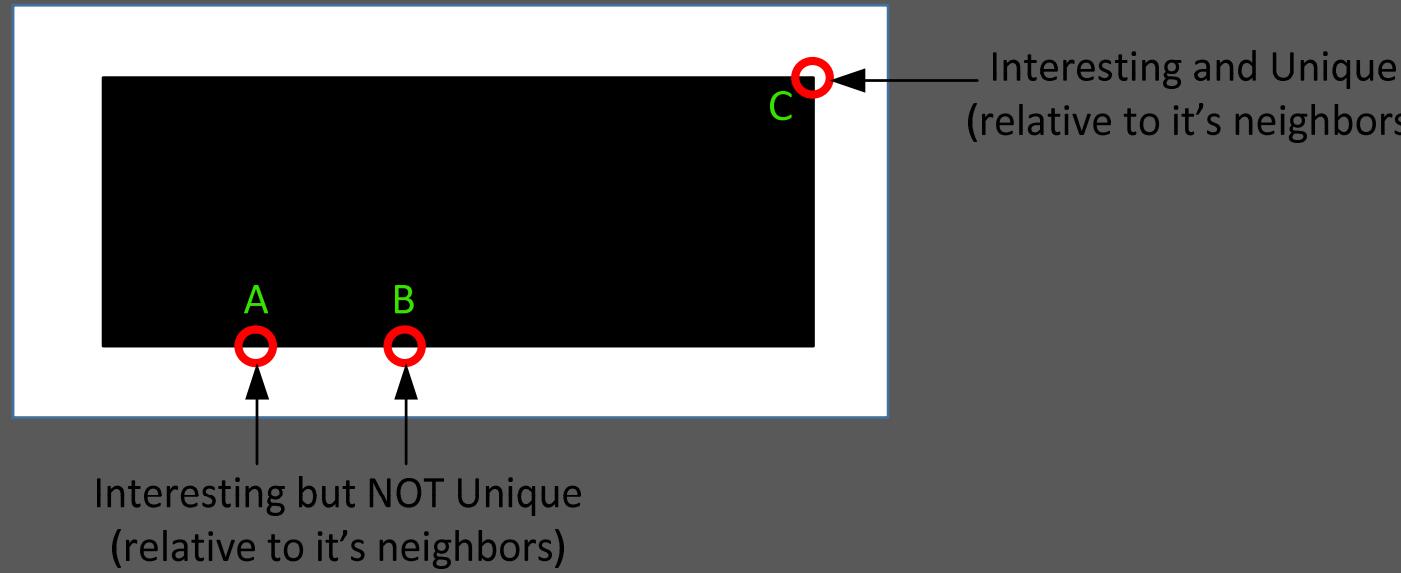
- **Image Alignment** – e.g. Panorama Stitching (finding corresponding matches so we can stitch images together)
- 3D Reconstruction
- Robot Navigation
- Object Recognition
- Motion Tracking
- And more!



What Defines Interesting?

Interesting areas carry a lot of **distinct** and **unique** information at that point.

- High change of intensity
- Corners or edges
- And more!



Be careful that noise can appear “informative” when it is not!

Characteristics of Good or Interesting Features

repeatable – they can be found in multiple
views of the same scene

distinctive – Each feature is somewhat unique
different to other features of the same scene

compactness/Efficiency – Significantly less
features than pixels in the image

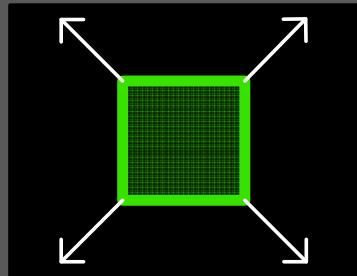
stability – Feature occupies a small area of the
image and is robust to clutter and occlusion



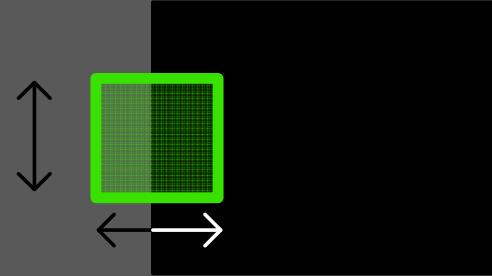
Corners as Features

Corners are identified when shifting a window in any direction over that point gives a large change in intensity.

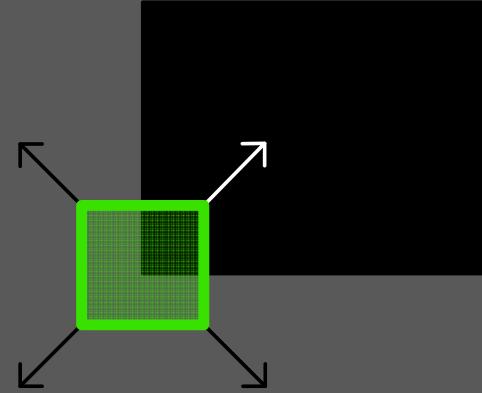
Flat – No change
in any direction



Edge – Change in
one direction



Corner – Change
in all directions



Corner Detection Methods – Harris Corners

Harris Corner Detection is an algorithm developed in 1998 for corner detection (<http://www.bmva.org/bmvc/1988/avc-88-023.pdf>) and works fairly well.

- `cv2.cornerHarris (input image, block size, ksize, k)`
 - Input image - should be grayscale and float32 type.
 - blockSize - the size of neighborhood considered for corner detection
 - ksize - aperture parameter of Sobel derivative used.
 - k - harris detector free parameter in the equation
 - Output – array of corner locations (x,y)

It was improved in 1994 by improving the scoring function when determining corners locations.

- `cv2.goodFeaturesToTrack`

Corner Detection Methods – ‘good Features To Track’

It was improved in 1994 by improving the scoring function when determining corners locations

- `cv2.goodFeaturesToTrack (input image, maxCorners, qualityLevel, minDistance)`

Input Image - 8-bit or floating-point 32-bit, single-channel image.

maxCorners – Maximum number of corners to return. If there are more corners than are found the strongest of them is returned.

qualityLevel – Parameter characterizing the minimal accepted quality of image corners. The parameter value is multiplied by the best corner quality measure (smallest eigenvalue). The corners with the quality measure less than the product are rejected. For example, if the best corner has the quality measure = 1500, and the qualityLevel=0.01 , then all the corners with the quality measure less than 15 are rejected.

minDistance – Minimum possible Euclidean distance between the returned corners.

Problems with corners as features

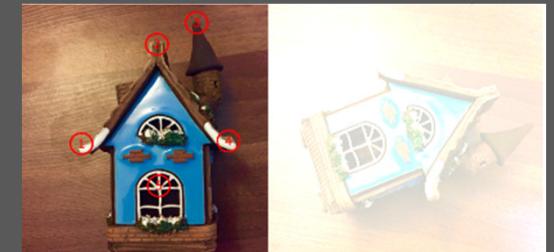
Corner matching in images is **tolerant** of:

- Rotations
- Translations (i.e. shifts in image)
- Slight photometric changes e.g. brightness or affine intensity

However, it is **intolerant** of:

- Large changes in intensity or photometric changes)
- Scaling (i.e. enlarging or shrinking)

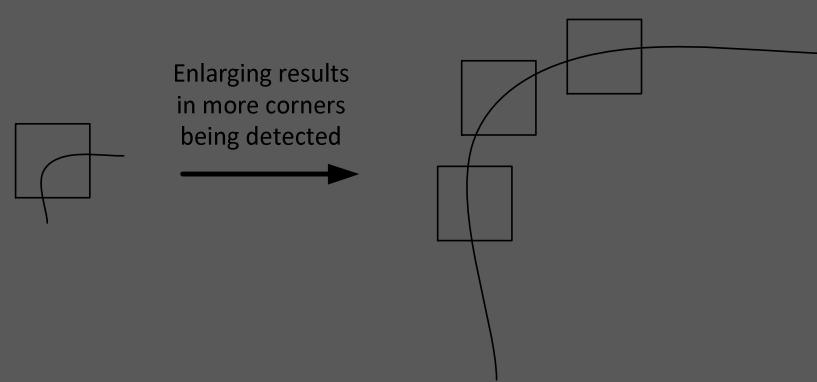
Intensity Issues



Scaling Issues



Enlarging results
in more corners
being detected



Introducing SIFT (Scale Invariant Feature Transform)

Needed tolerance to *scaling* (*known as scale invariance*)

SIFT is widely used (although patented) in computer vision as it very successfully dealt with the scale invariance issue

Patented and no longer freely available with OpenCV 3.0+



SIFT in a nutshell

We firstly detect **interesting key points** in an image using the Difference of Gaussian method. These are areas of the image where variation exceeds a certain threshold and are better than edge descriptors.

We then **create vector descriptor** for these interesting areas. Scale invariance is achieved via the following process:

- i. Interest points are scanned at **several different scales**
- ii. The scale at which we meet a **specific stability criteria**, is then selected and is encoded by the vector descriptor. Therefore, regardless of the initial size, the more stable scale is found which allows us to be scale invariant.

Rotation invariance is achieved by obtaining the **Orientation Assignment** of the key point using image gradient magnitudes. Once we know the 2D direction, we can normalize this direction.

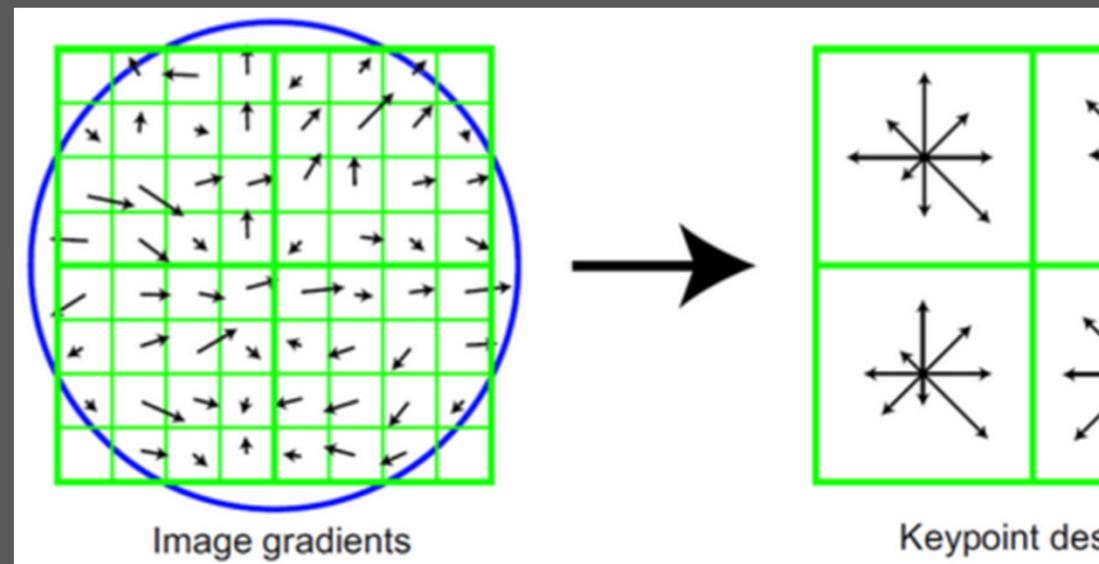
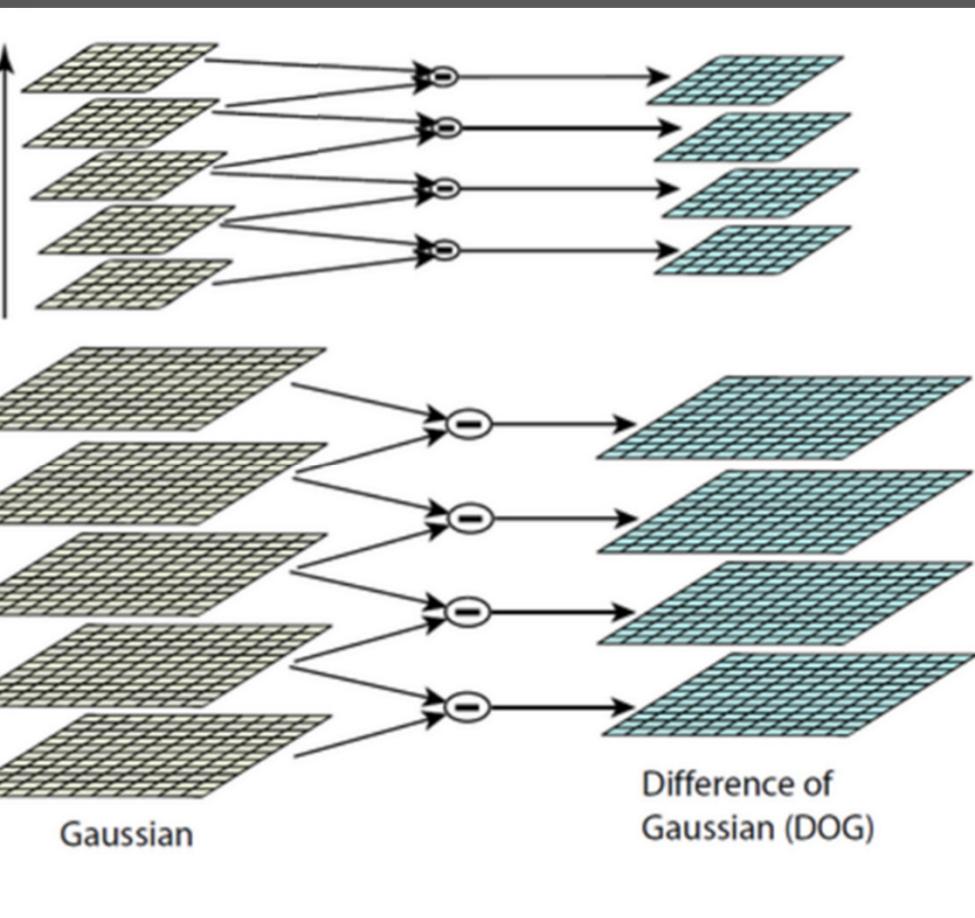
The full paper on SIFT can be read here:

- <http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf>

An excellent tutorial on SIFT also available here

- http://opencv-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html

SIFT (Scale Invariant Feature Transform)



Speeded Up Robust Features (SURF)

SIFT is quite effective but computationally expensive

SURF was developed to improve the speed of a scale invariant feature detector.

Instead of using the Difference of Gaussian approach, SURF uses Hessian matrix approximation to detect interesting points and use the sum of Haar wavelet responses for orientation assignment.

Alternatives to SIFT and SURF

Features from Accelerated Segment Test (FAST)

- Key point detection only (no descriptor, we can use SIFT or SURF to computer that)
- Used in real time applications

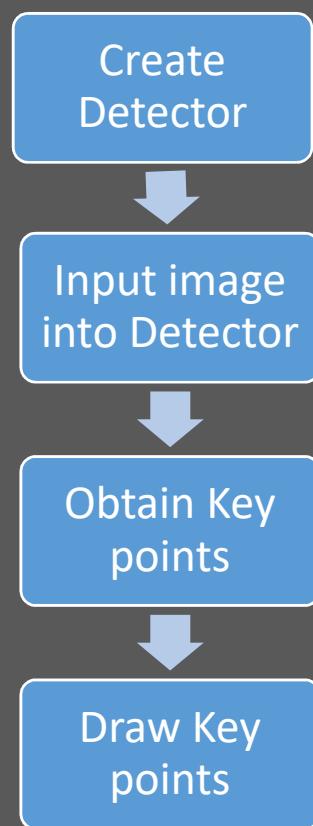
Binary Robust Independent Elementary Features (BRIEF)

- Computes descriptors quickly (instead of using SIFT or SURF)
- Fast

Oriented FAST and Rotated BRIEF (ORB) – Developed out of OpenCV Labs (no patented so free to use!)

- Combines both Fast and Brief
- http://www.willowgarage.com/sites/default/files/orb_final.pdf

Using SIFT, SURF, FAST, BRIEF & ORB in OpenCV



Mini Project # 5 – Object Detection



Histogram of Oriented Gradients (HOGs)

HOGs are a **feature descriptor** that has been widely and successfully used for object detection.

It represents objects as a **single feature vector** as opposed to a set of feature vectors where each represents a segment of the image.

It's computed **by sliding window detector** over an image, where a HOG descriptor is a computed for each position. Like SIFT the scale of the image is adjusted (pyramiding).

HOGs are often used with **SVM** (support vector machine) classifiers. Each HOG descriptor that is computed is fed to a SVM classifier to determine if the object was found or not).

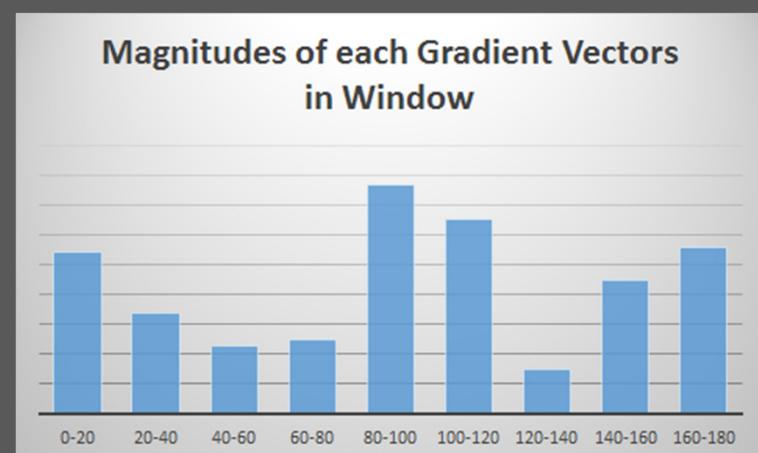
reat Paper by Dalal & Triggs on using **HOGs for Human Detection**:

- <https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf>

Histogram of Gradients (HOGs) Step by Step

Using an 8×8 pixel detection window or cell (in green), we compute the gradient vector or edge orientations at each pixel.

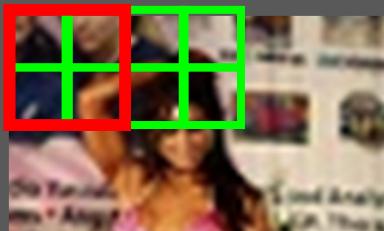
- . This generates 64 (8×8) gradient vectors which are then represented as a histogram.
- . Each cell is then split into angular bins, where each bin corresponds to a gradient direction (e.g. x, y). In the Dalal and Triggs paper, they used 9 bins $0-180^\circ$ (20° each bin).
- . This effectively reduces 64 vectors to just 9 values.
- . As it stores gradients magnitudes, it's relatively immune to deformations.



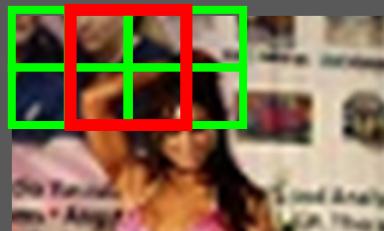
64×128

Histogram of Gradients (HoG) Step by Step

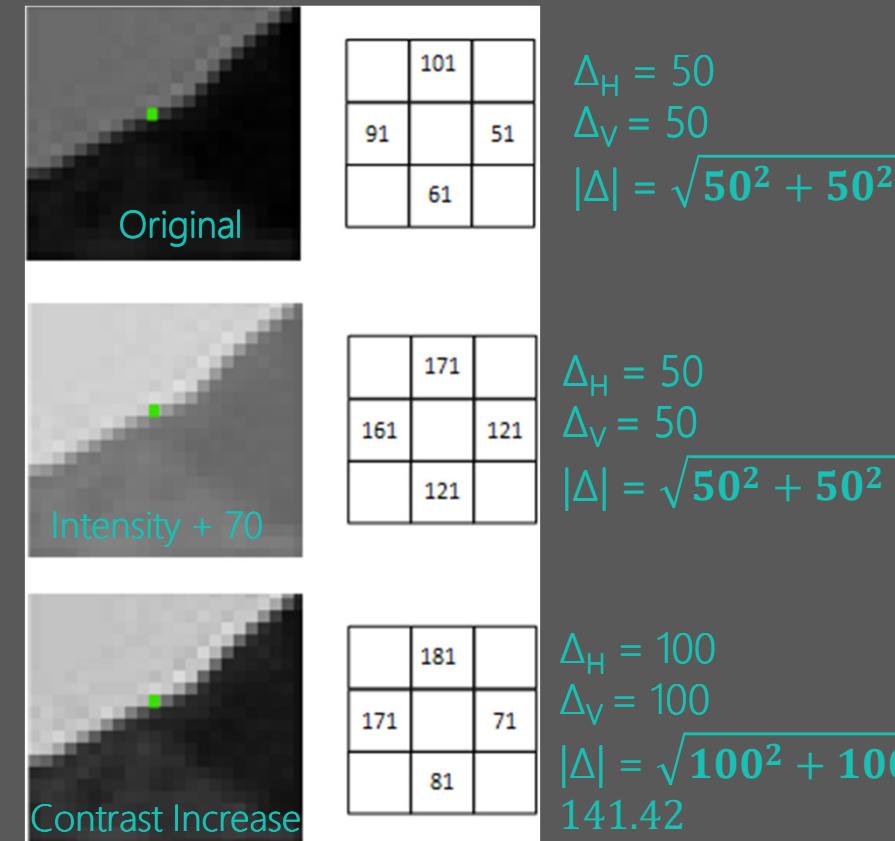
- We then **Normalize** the gradients to ensure invariance to illumination changes i.e. Brightness and Contrast. E.g. in the images on the right, if we **divide the vectors by the gradient magnitudes** we get 0.707 for all, this is normalization.
- Instead of individual window cell normalization, a method called Block Normalization is used. This takes into account neighboring blocks so we normalize taking into consideration larger segments of the image.



Block 1



Block 2



Section 6

Face, People & Car Detection

Face, People & Car Detection

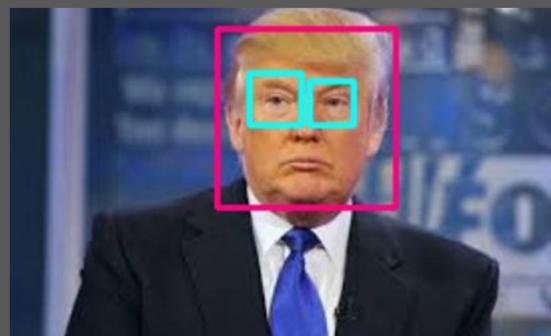
- . Understanding HAAR Cascade Classifiers
- . Face & Eye Detection
- . Mini Project # 6 - Car Detection & Pedestrian (Body) Detection

HAAR Cascade Classifiers

As we saw in the previous section, we can **extract features** from an image and use those features to **classify** objects.

What are HAAR Cascade Classifiers?

An object detection method that inputs **Haar features** into a series of classifiers (cascade) to identify objects in an image. They are trained to identify **one type of object**, however, we can use several of them in parallel e.g. detecting eyes and faces together.

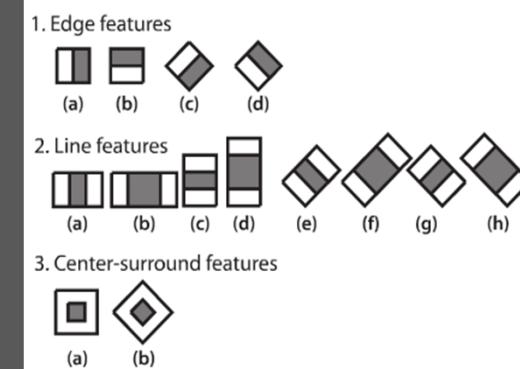
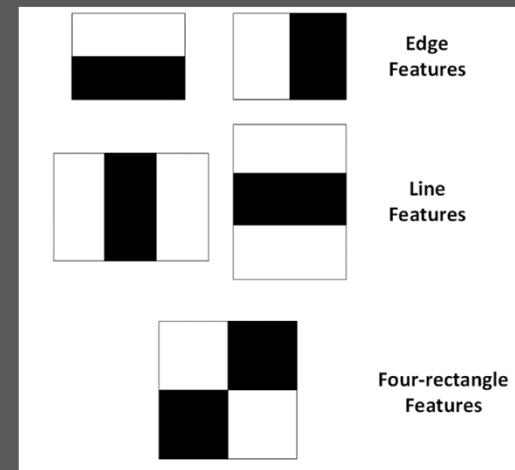


HAAR Classifiers Explained

HAAR Classifiers are trained using lots of **positive images** (i.e. images with the object present) and **negative images** (i.e. images without the object present).

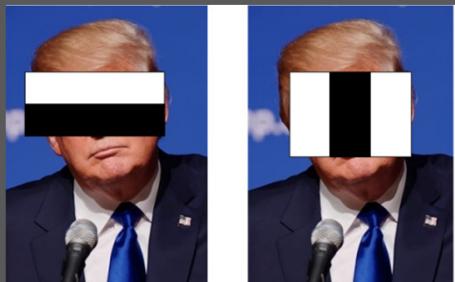


- We then extract features using **sliding windows of rectangular blocks**. These features are single valued and are calculated by subtracting the sum of pixel intensities under the white rectangles from the black rectangles. However, this is a ridiculous number of calculations, even for a base window of 24 x 24 pixels (180,000 features generated). So the researchers devised a method called **Integral Images** that computed this with four array references.

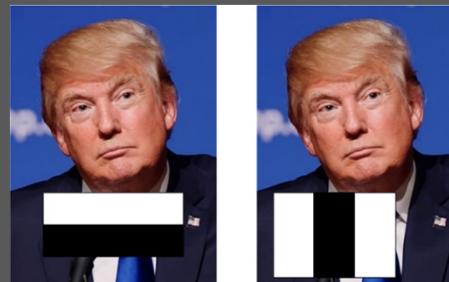


HAAR Classifiers Explained

- .. However, they still had 180,000 features and the majority of them added no real value.



Relevant

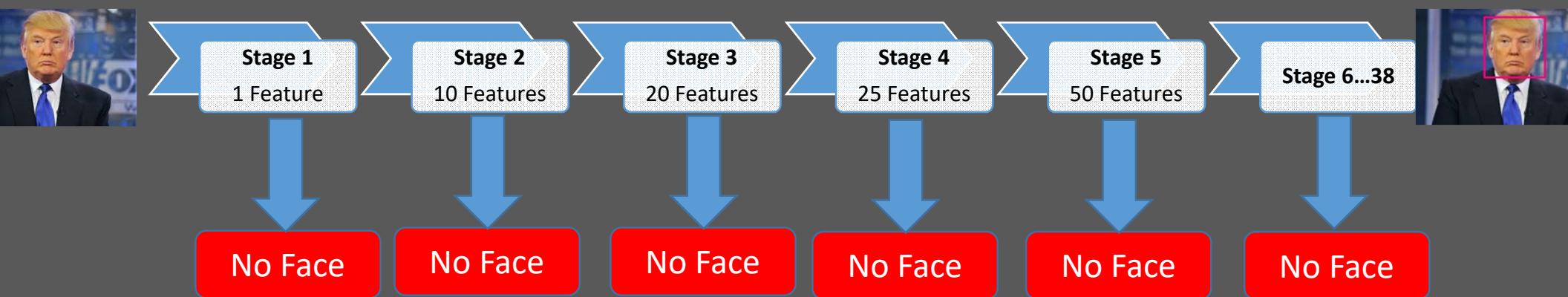


Irrelevant

- .. Boosting was then used to determine the most informative features, with Freund & Schapire's AdaBoost the algorithm of choice due to its ease of implementation. Boosting is the process by which we use weak classifiers to build strong classifiers, simply by assigning heavier weighted penalties on incorrect classifications. Reducing the 180,000 features to 6000, which is still quite a bit features.

HAAR Classifiers Explained

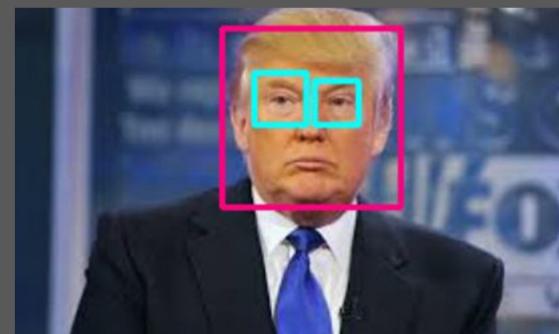
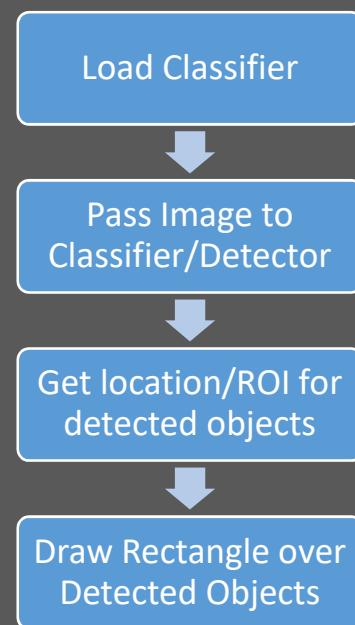
- .. Think about this intuitively, if of those 6000 features, some will be more informative than others. What if we used the most informative features to first check whether the region can potentially have a face (false positives will be no big deal). Doing so eliminates the need for calculating all 6000 features at once.
- . This concept is called the Cascade of Classifiers - for face detection, the Viola Jones method used 38 stages.



Face & Eye Detection

1. We use some pre-trained classifiers that have been provided by OpenCV
 - Stored as .XML files
2. These pre-trained classifiers can be found here:
 - <https://github.com/opencv/opencv/tree/master/data/haarcascades>

Cascade Classifiers Flow



Parameters for detectMultiScale

`ourClassifier.detectMultiScale(input image, Scale Factor , Min Neighbors)`

scale Factor

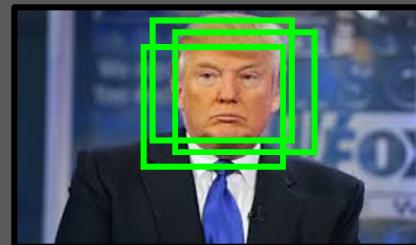
Specifies how much we reduce the image size each time we scale. E.g. in face detection we typically use 1.3. This means we reduce the image by 30% each time it's scaled. Smaller values, like 1.05 will take longer to compute, but will increase the rate of detection.



Min Neighbors

Specifies the number of neighbors each potential window should have in order to consider it a positive detection. Typically set between 3-6.

- It acts as sensitivity setting, low values will sometimes detect multiples faces over a single face. High values will ensure less false positives, but you may miss some faces.



List of OpenCV Pre-Trained Cascade Classifiers

found Here - <https://github.com/opencv/opencv/tree/master/data/haarcascades>

haarcascade_eye.xml	some attempts to tune the performance
haarcascade_eye_tree_eyeglasses.xml	some attempts to tune the performance
haarcascade_frontalcatface.xml	Removing whitespace to appease doc builder
haarcascade_frontalcatface_extende...	Removing whitespace to appease doc builder
haarcascade_frontalface_alt.xml	some attempts to tune the performance
haarcascade_frontalface_alt2.xml	some attempts to tune the performance
haarcascade_frontalface_alt_tree.xml	some attempts to tune the performance
haarcascade_frontalface_default.xml	some attempts to tune the performance
haarcascade_fullbody.xml	fixing wrong model sizes
haarcascade_lefteye_2splits.xml	some attempts to tune the performance
haarcascade_licence_plate_rus_16sta...	Added Haar cascade for russian cars licence plate detection, 16 stage...
haarcascade_lowerbody.xml	fixing wrong model sizes
haarcascade_profileface.xml	some attempts to tune the performance
haarcascade_righteye_2splits.xml	some attempts to tune the performance
haarcascade_russian_plate_number....	Create haarcascade_russian_plate_number.xml
haarcascade_smile.xml	fixing wrong model sizes
haarcascade_upperbody.xml	fixing wrong model sizes

Mini Project # 6- Car and Pedestrian Detection

