COMPUTER VISION
INDUSTRIAL IOT

Software and Services Group
IoT Developer Relations, Intel

LEGAL NOTICES AND DISCLAIMERS

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at www.intel.com.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

Any forecasts of goods and services needed for Intel's operations are provided for discussion purposes only. Intel will have no liability to make any purchase in connection with forecasts published in this document.

ARDUINO 101 and the ARDUINO infinity logo are trademarks or registered trademarks of Arduino, LLC.

Intel, the Intel logo, Intel Inside, the Intel Inside logo, OpenVINO, Intel Atom, Celeron, Intel Core, and Intel Movidius Myriad 2 are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

MULTIPLE USAGES

CAPTURE

- Acquire Data from Sensors
- Imaging pipeline
- Initial processing
- Encoding

"MEDIA"

- Decoding
- Aggregation
- Muxing

COMPUTER-VISION

- "Traditional' Visual Understanding
- Pre-Processing for Deep Learning

DEEP LEARNING

- Inference using Deep Learning models
- Different models, tasks, for various models/topologies

"OUTPUT"

- Generate Insights
- Render to Screen
- Alert
- Store

CLEAR TRENDS

CAPTURE

- Acquire Data from Sensors
- Imaging pipeline
- Initial processing
- Encoding

"MEDIA"

- Decoding
- Aggregation
- Muxing

COMPUTER-VISION

- "Traditional' Visual Understanding
- Pre-Processing for Deep Learning

DEEP LEARNING

- Inference using Deep Learning models
- Different models, tasks, for various models/topologies

"OUTPUT"

- Generate Insights
- Render to Screen
- Alert
- Store

LESS STORAGE REQUIRED →

← FASTER RESPOND TIME, MORE CONTROLLABILITY ON THE EDGE

LESS BAND-WIDTH →

← MORE ANALYTICS TO THE EDGE

WHAT IS THE INTEL® CV SDK?

The Intel® Computer Vision SDK is a new software development package for development and optimization of computer vision and image processing pipelines for Intel System-on-Chips (SoCs).

- Intel-optimized implementation of the Khronos OpenVX 1.1 API
- Pre-built and fully-validated community OpenCV 3.3 binaries
- Vision Algorithm Designer (VAD)
- Deep Learning Model Optimizer tool
- Deep Learning Inference Engine

KHRONOS GROUP

Active Standards

COLLADA DataFormat EGL glTF NNEF

OpenCL OpenGL ES OpenGL OpenGL SC OpenVG

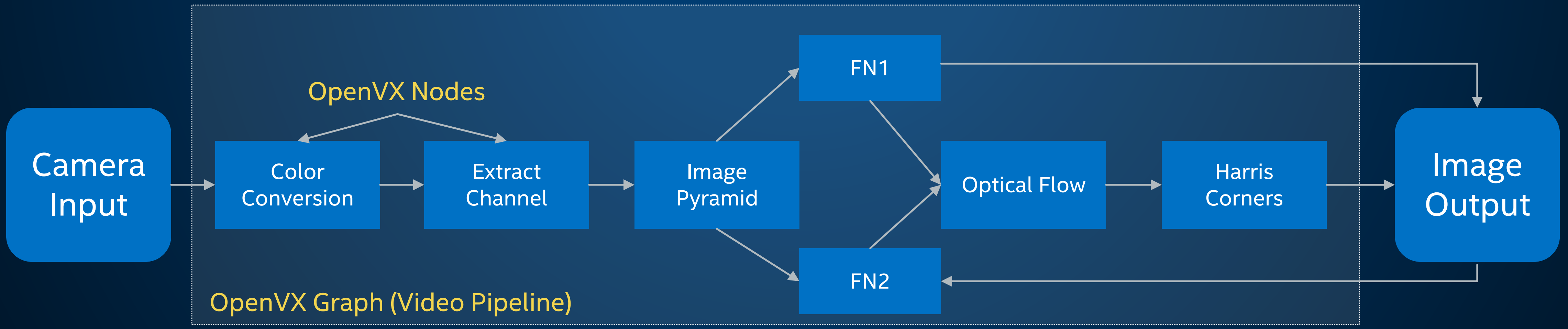
OpenXR OpenVX SPIR SYCL Vulkan

WebGL

WHAT IS OPENVX*?

OpenVX* is about standardized, portable, power-efficient vision processing.

- Faster development of computer vision applications
- Better optimization via directed graph scheduling of image kernels/algorithms. Each graph node can run on a separate piece of hardware.
- OpenVX chooses which hardware to run each step of your computer vision graph (e.g. CPU, GPU, VPU).
- Smaller number of algorithms than OpenCV. However, Khronos maintains a conformance test suite that can validate a vendors OpenVX implementation.



ALGORITHMS IN OPENVX* 1.1

- Absolute Difference
- Accumulate
- Accumulate Squared
- Accumulate Weighted
- Arithmetic Addition
- Arithmetic Subtraction
- Bitwise And
- Bitwise OR
- Box Filter
- Canny Edge Detector
- Channel Combine
- Channel Extract
- Color Covert
- Convert bit depth
- Custom Convolution
- Dilate Image
- Equalize Histogram
- Erode Image
- Fast Corners
- Gaussian Filter
- Harris Corners
- Histogram
- Image Pyramid
- Integral Image
- Magnitude
- Mean and Standard Deviation
- Media Filter
- Min Location
- Max Location
- Optical Flow Pyramid
- Phase
- Pixel-wise Mulitplication
- Remap
- Scale Image
- Sobel 3x3
- TableLookup
- Thresholding
- Warp Affine
- Warp Perspective

OPENVX* VS. OPENCV*

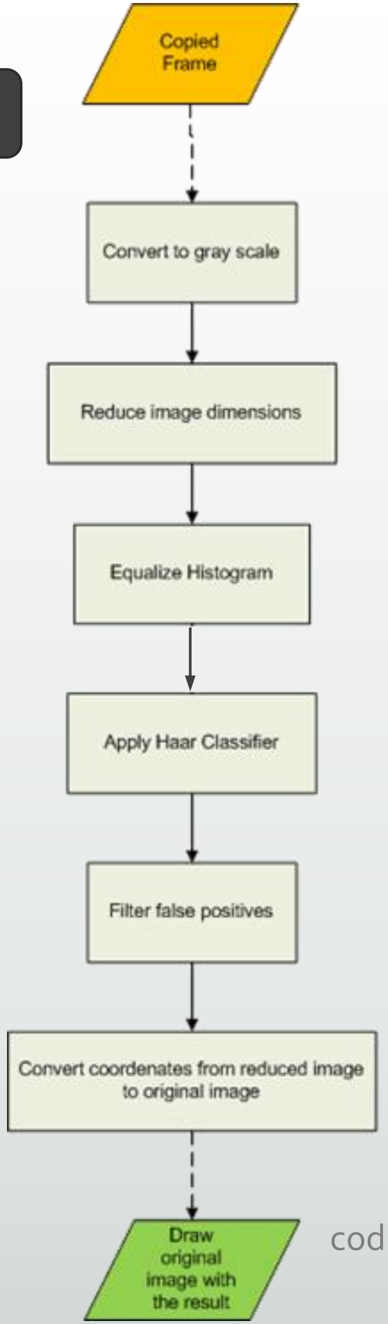
	OpenCV*	OpenVX*
Implementation	Community driven open source library	Open standard API designed to be implemented by hardware vendors
Conformance	Extensive OpenCV Test Suite but no formal Adopters program	Implementations must pass defined conformance test suite to use trademark
Consistency	Available functions can vary depending on implementation / platform	All core functions must be available in all conformant implementations
Scope	Very wide 1000s of imaging and vision functions Multiple camera APIs/interfaces	Tight focus on core hardware accelerated functions for mobile vision – but extensible Uses external/native camera API
Efficiency	Memory-based architecture Each operation reads and writes to memory	Graph-based execution Optimizable computation and data transfer
Typical Use Case	Rapid experimentation and prototyping - especially on desktop	Production development & deployment on mobile and embedded devices
Embedded Deployment	Re-usable code	Callable library

OpenCV*

- Open Source library for computer vision
- Written in C++
  - Bindings for most popular languages
- Block-like programming structure
  - Image data is sequentially passed through functions
  - Fundamental concepts allow for powerful image processing tools

```
# python code snippets go here
```

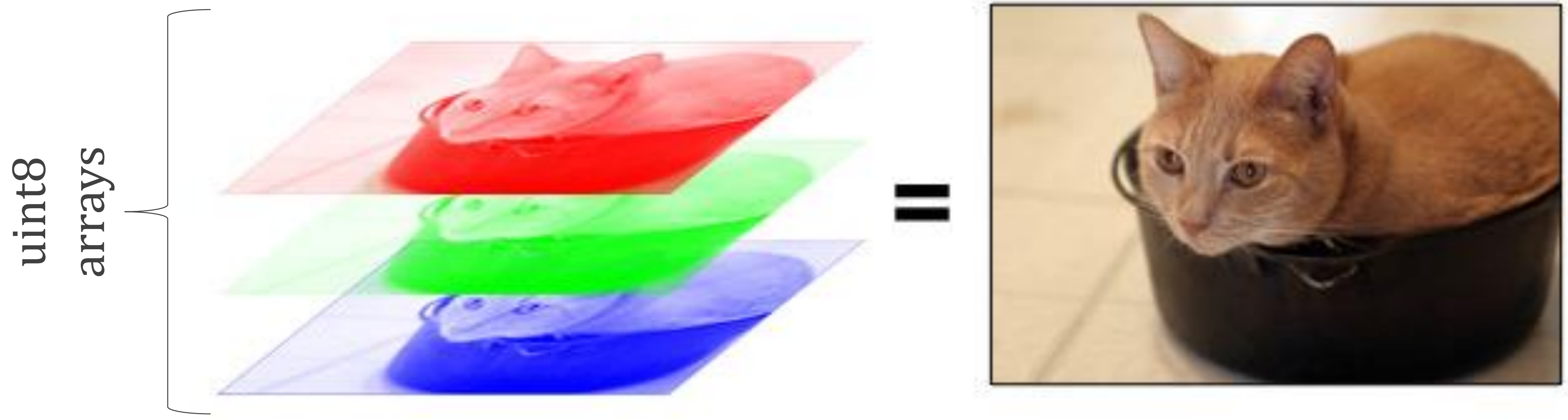
```
import cv2
import numpy as np
```



codingbox.com

COMPUTER VISION ALGORITHMS

Image Binary Representation



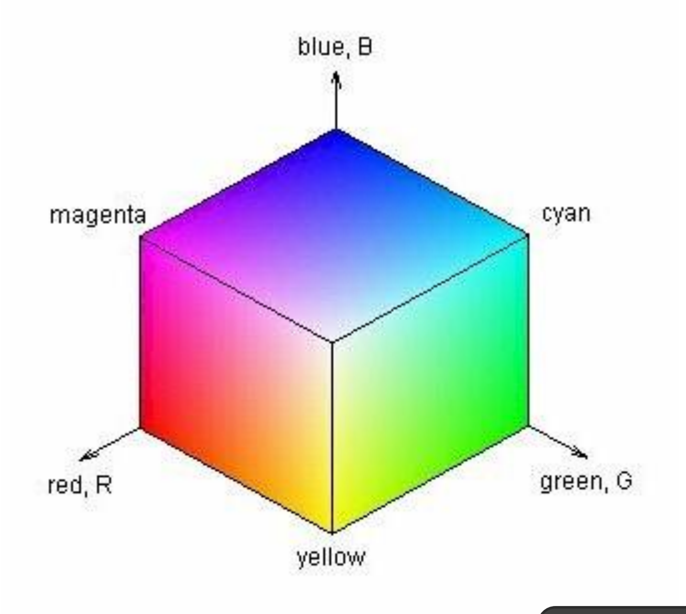
Cadin Batrack

```
img = cv2.imread( 'image.jpg') # open image
cv2.imshow( 'Title', img ) # show image
cv2.imwrite( 'image2.jpg', img ) # write image file
```

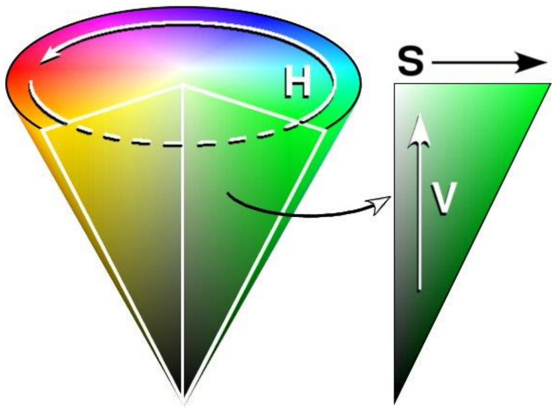
Colorspaces

$$(Blue, Red, Green) \leftrightarrow (Hue, Saturation, Value) \rightarrow (Intensity)$$

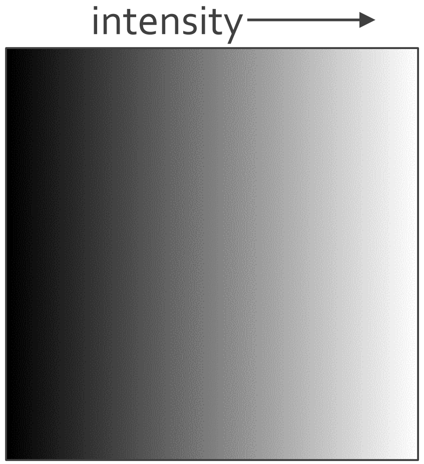
BGR



HSV



Grayscale

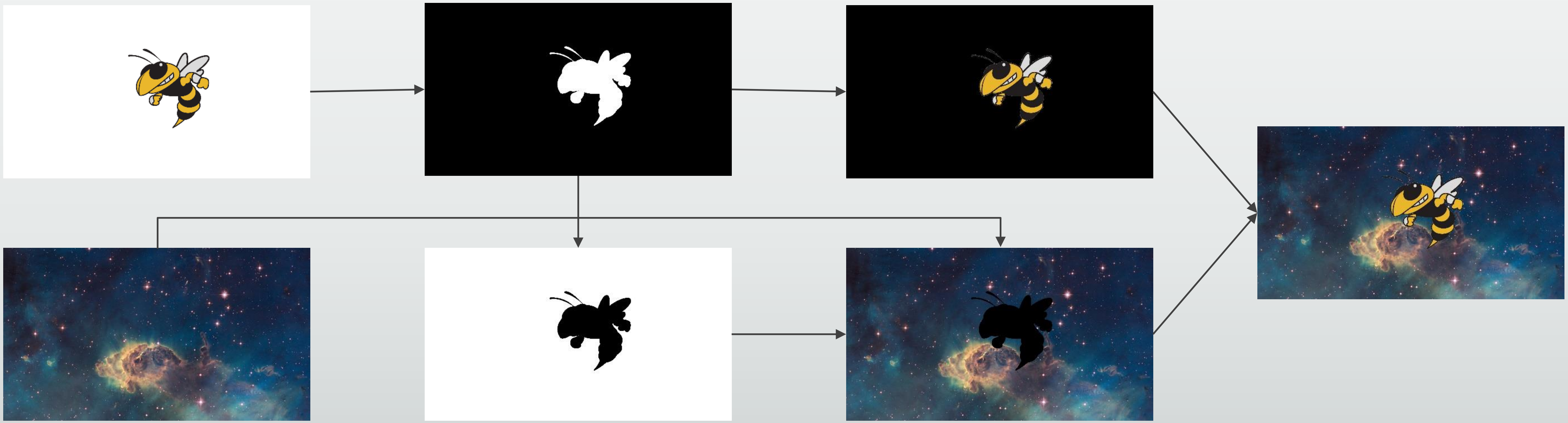



```
img2 = cv2.cvtColor( img, cv2.COLOR_BGR2HSV )
```

Masking

```
ret, thresh = cv2.threshold( single_channel, min_value , set_value_to, cv2.THRESH_BINARY )
mask = cv2.inRange( hsv_img, lower_color , higher_color )
cv2.bitwise_and( img, img, mask = mask )
```

inRange
threshold → Image Mask → Binary operations → Combine Image + Mask

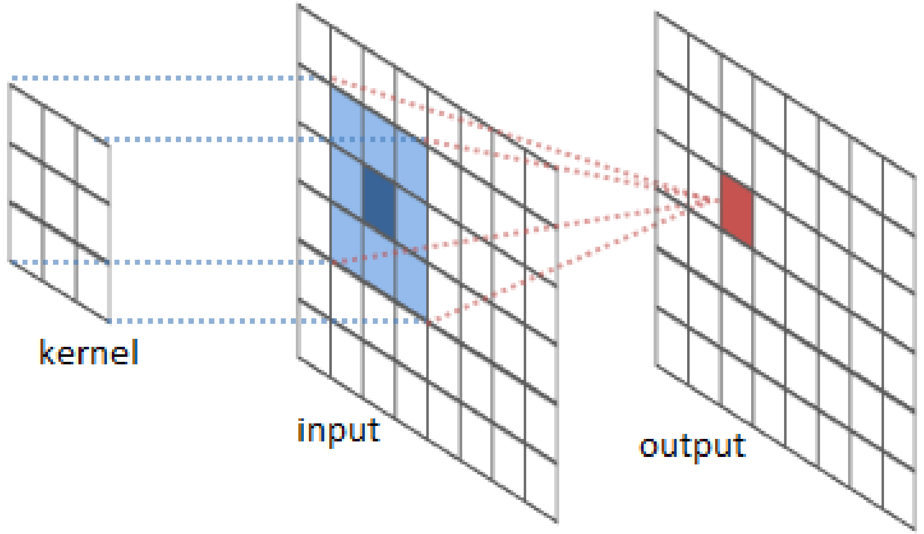


Filtering

- Noise Removal (LPF)
- Sharpness increase (HPF)
- Operate on Kernel

Filtering: Smoothing

- Uses 2D Convolution
- Blurring removes noise
- Kernel size determines blur amount
- Types:
  - Mean - *average*
  - Median – *less prone to outliers*
  - Gaussian – *applies Gaussian curve*
  - Bilateral – *blurs & preserves edges*

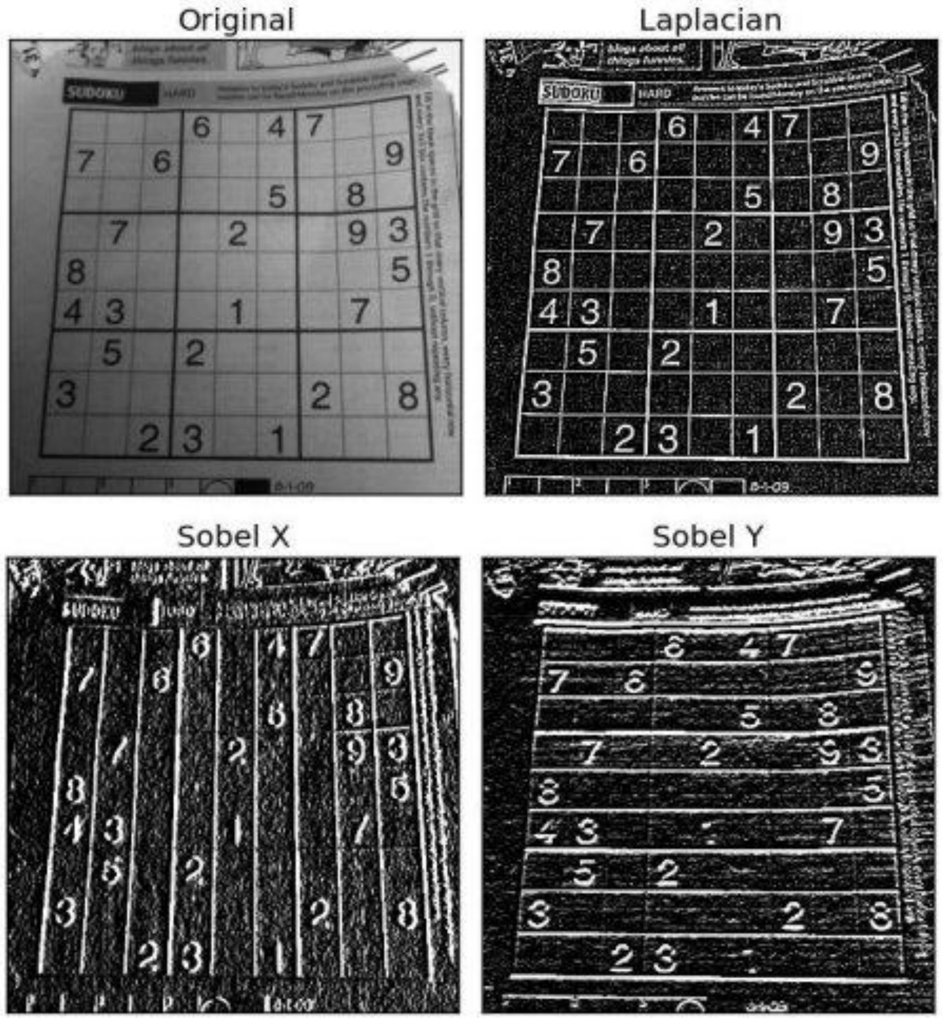



```
# 5x5 mean filter
kernel = np.ones( ( 5, 5 ) , np.float32 ) / 25
res = cv2.filter2D( img, -1, kernel )
```

Filtering: Gradients

- High Pass Filters
- Used to find edges in an image
- Specify Kernel size
- Types:
  - Sobel – *directional derivatives*
  - Scharr – *better for smaller kernel*
  - Laplacian – *relation on Sobel derivatives*

```
laplace = cv2.Laplacian( img,  cv2.CV_64F )
sobelX = cv2.Sobel( img, cv2.CV_64F, 0, 1, ksize = 5 )
```



OpenCV Documentation

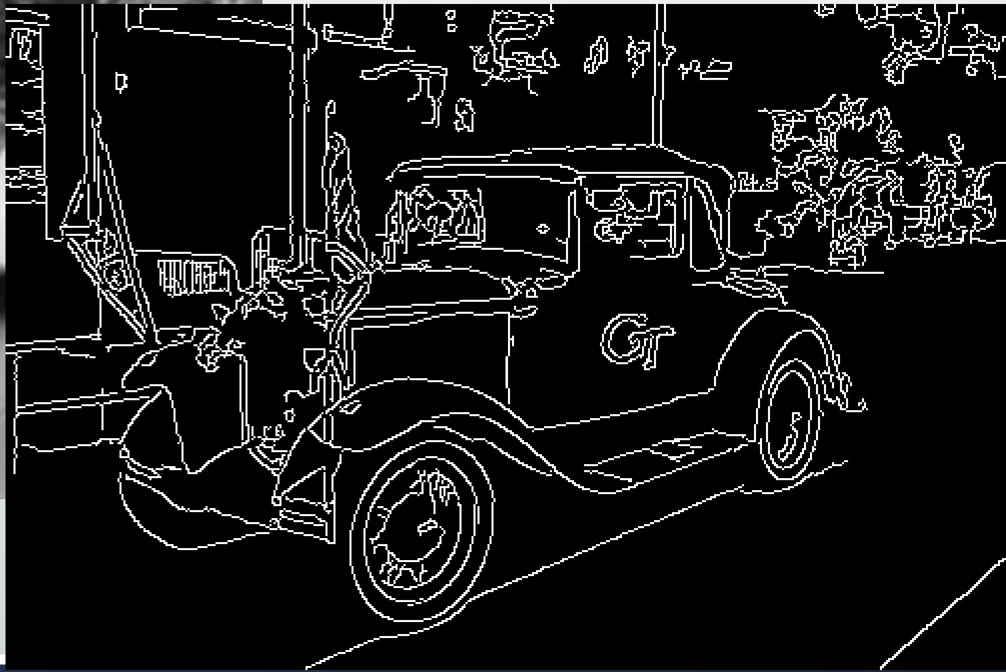
Gradients for Canny Edge Detection

```
# 4x4 mean filter
kernel = np.ones( ( 4, 4 ) , np.float32 ) / 16
wreck= cv2.filter2D( img, -1, kernel )

# apply canny edge detection
edges = cv2.Canny( wreck, 0, 200 )
```

Original

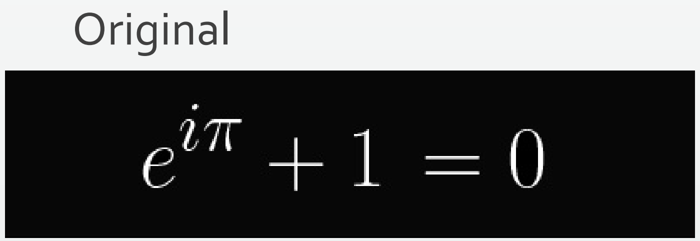
Smoothed

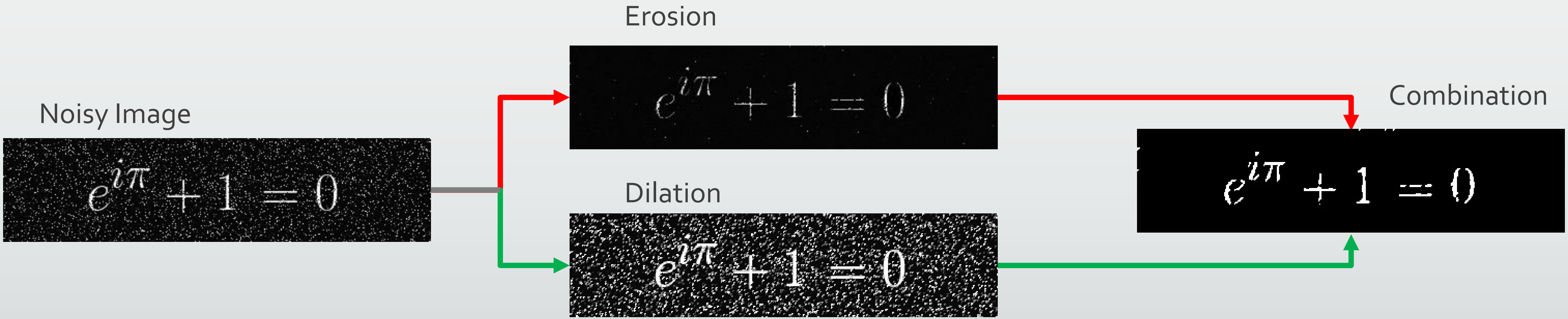
Canny



Filtering: Morphological Transformations

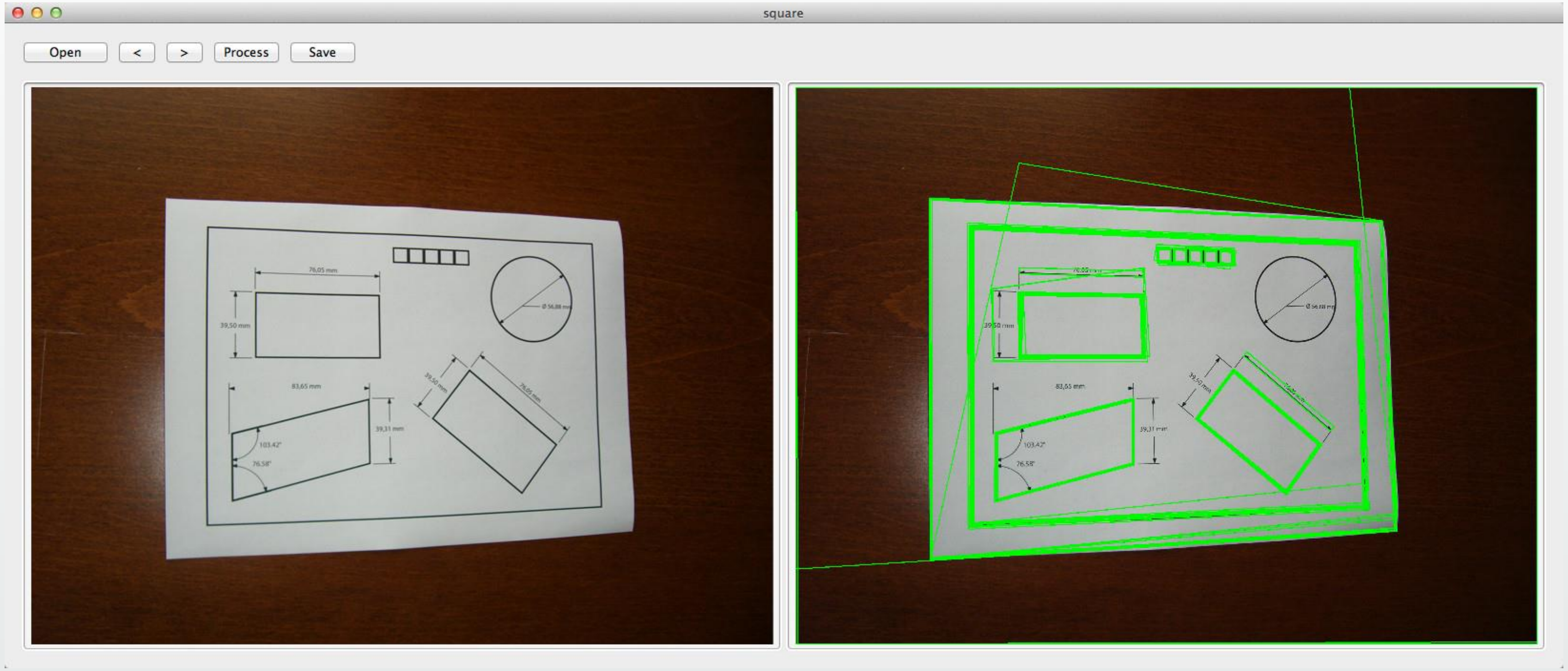
```
kernel = np.ones( ( 5, 5 ) , np.float32 ) / 25
erosion = cv2.erode( img, kernel,  iterations = 1 )
dilation = cv2.dilate( img, kernel,  iterations = 1 )
opening = cv2.morphologyEx( img, cv2.MORPH_OPEN, kernel )
closing = cv2.morphologyEx( img, cv2.MORPH_CLOSE, kernel )
```





Contours

- Join all continuous points along boundary
- Detect from logical mask
- Contour hierarchies allow selection of a contour based on its relation to others



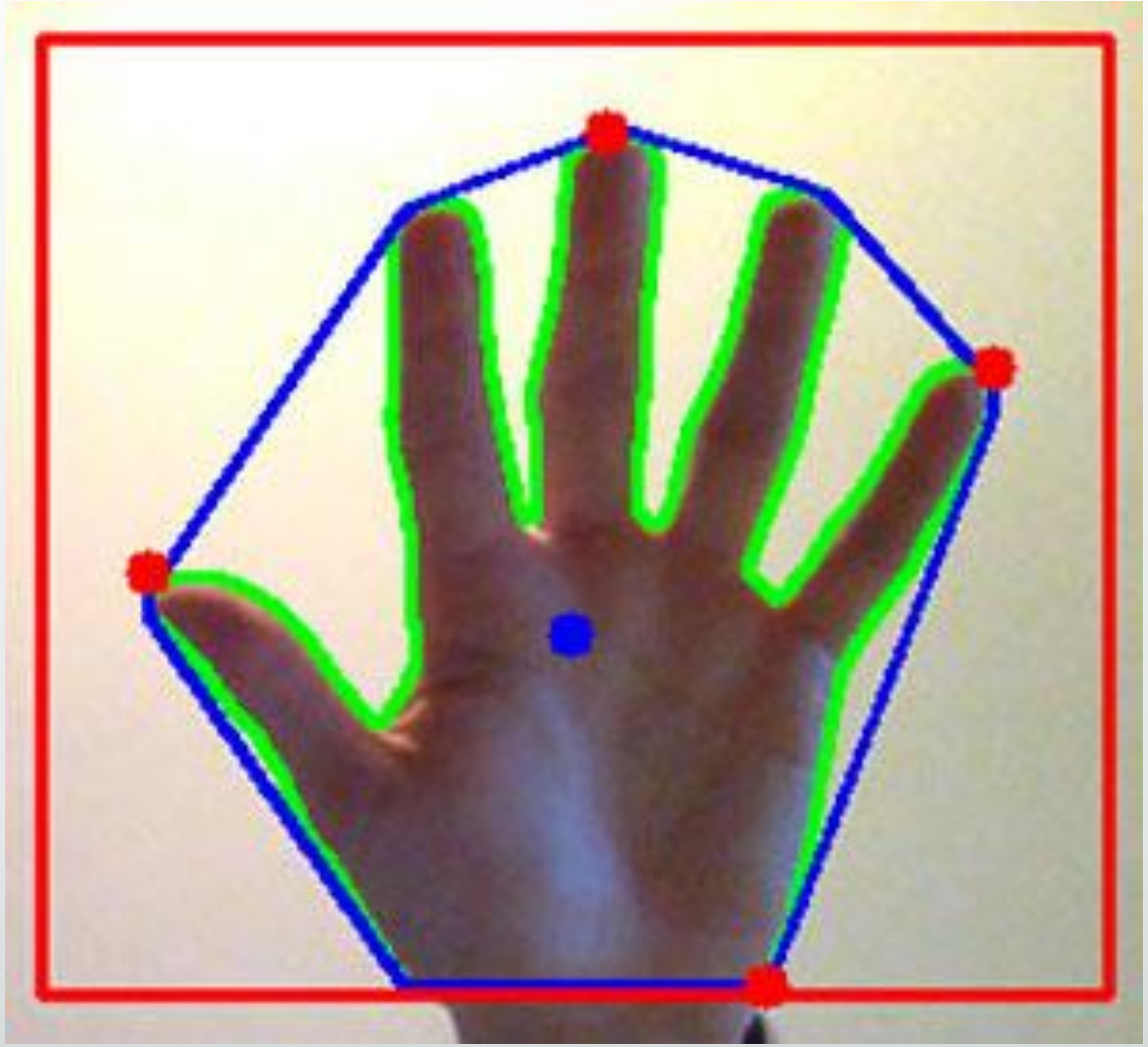

```
img2, contours, hierarchy = cv2.findContours( mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE )
cv2.drawContours( img, contours, contour_number, ( 0, 255, 0 ), thickness )
```

Contour Functions

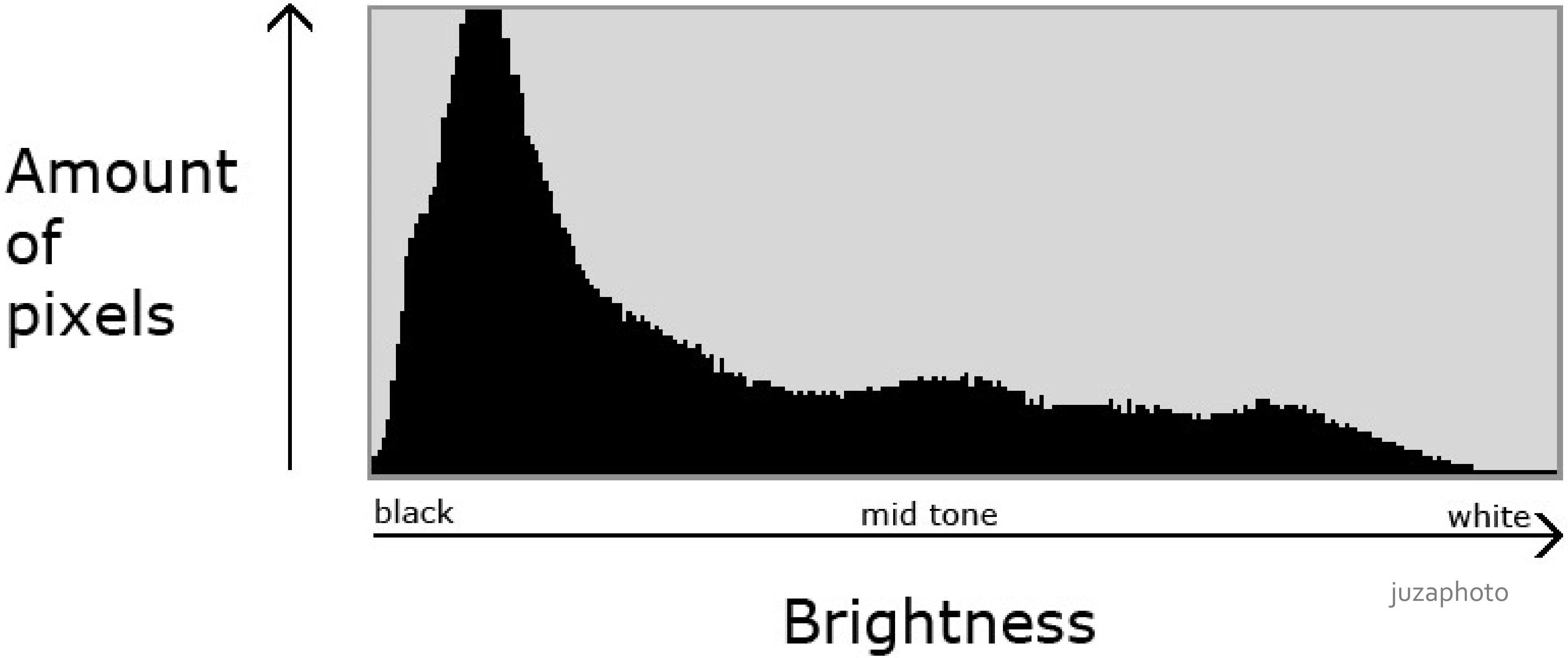
Contours can be mathematically analyzed

OpenCV functions give:

- Moments & Centroids
- Area & Perimeter
- Convex hulls
- Bounding Rectangles
- Fit curves & shapes
- Outermost points
- Shape Matching

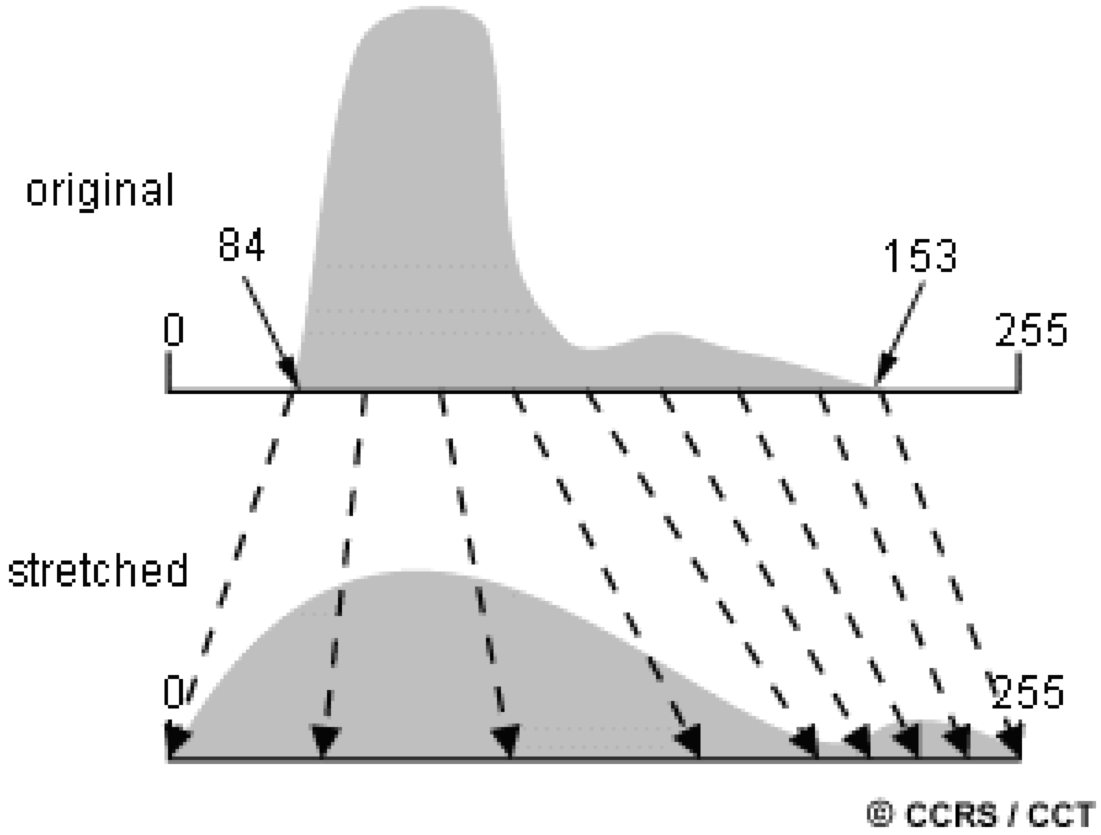


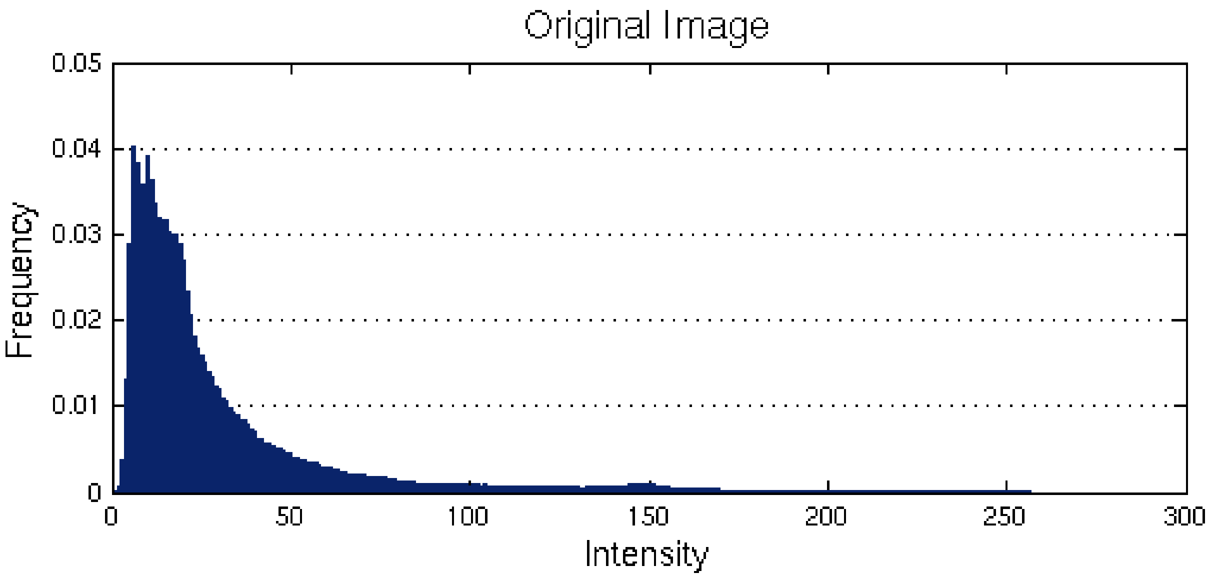
Histogram: $intensity \rightarrow frequency$

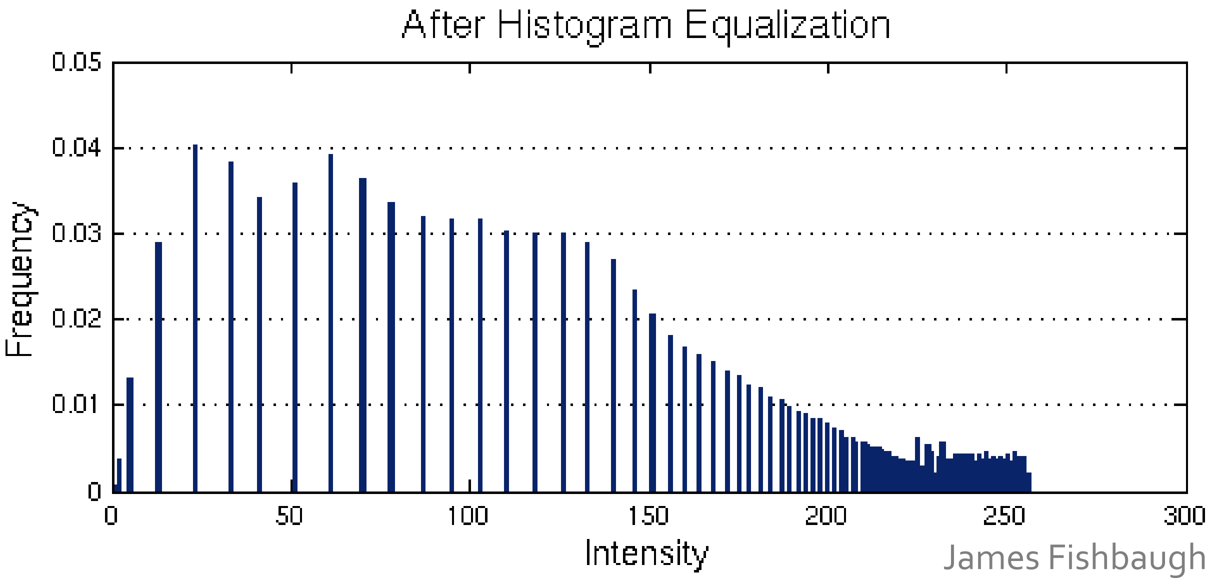



```
hist = cv2.calcHist( [img], [channel], mask, [number_bins], [min_val, max_val] )
```

Histogram Equalization



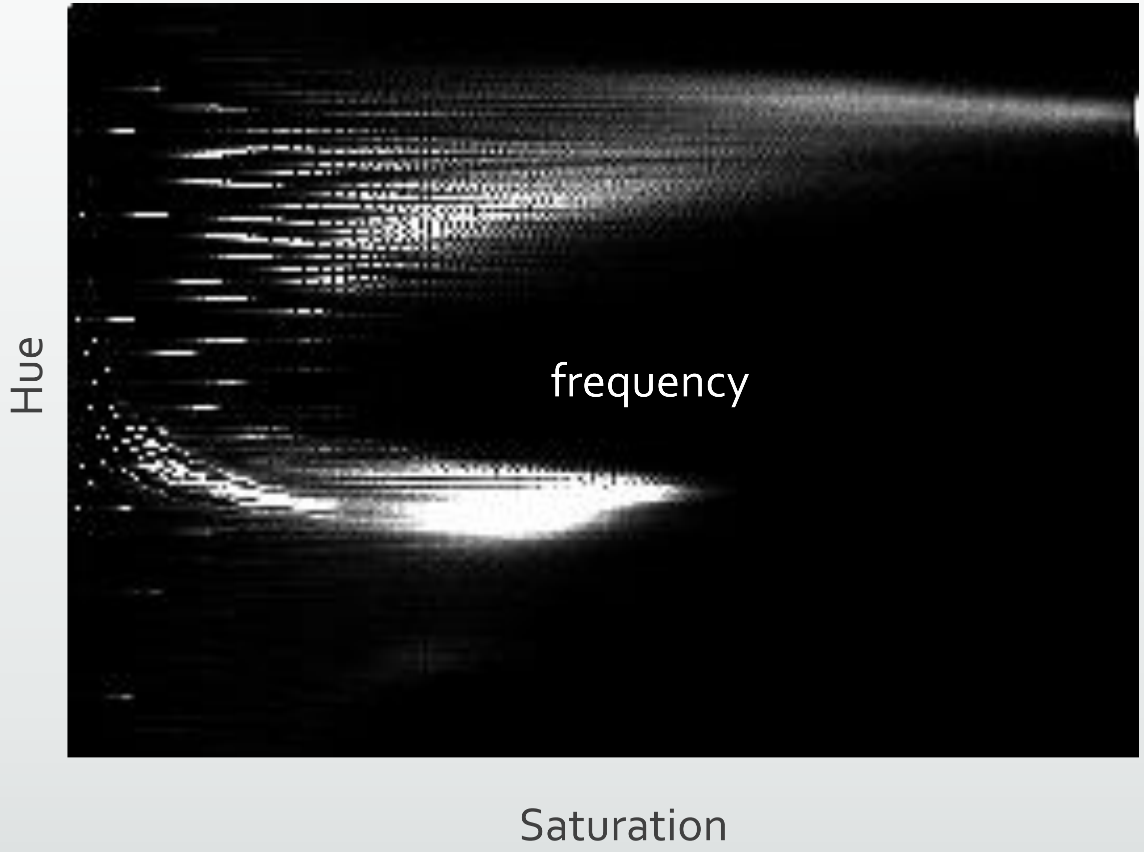




James Fishbaugh

```
eq = cv2.equalizeHist( img )
```

2D Histogram: $(hue, saturation) \rightarrow frequency$



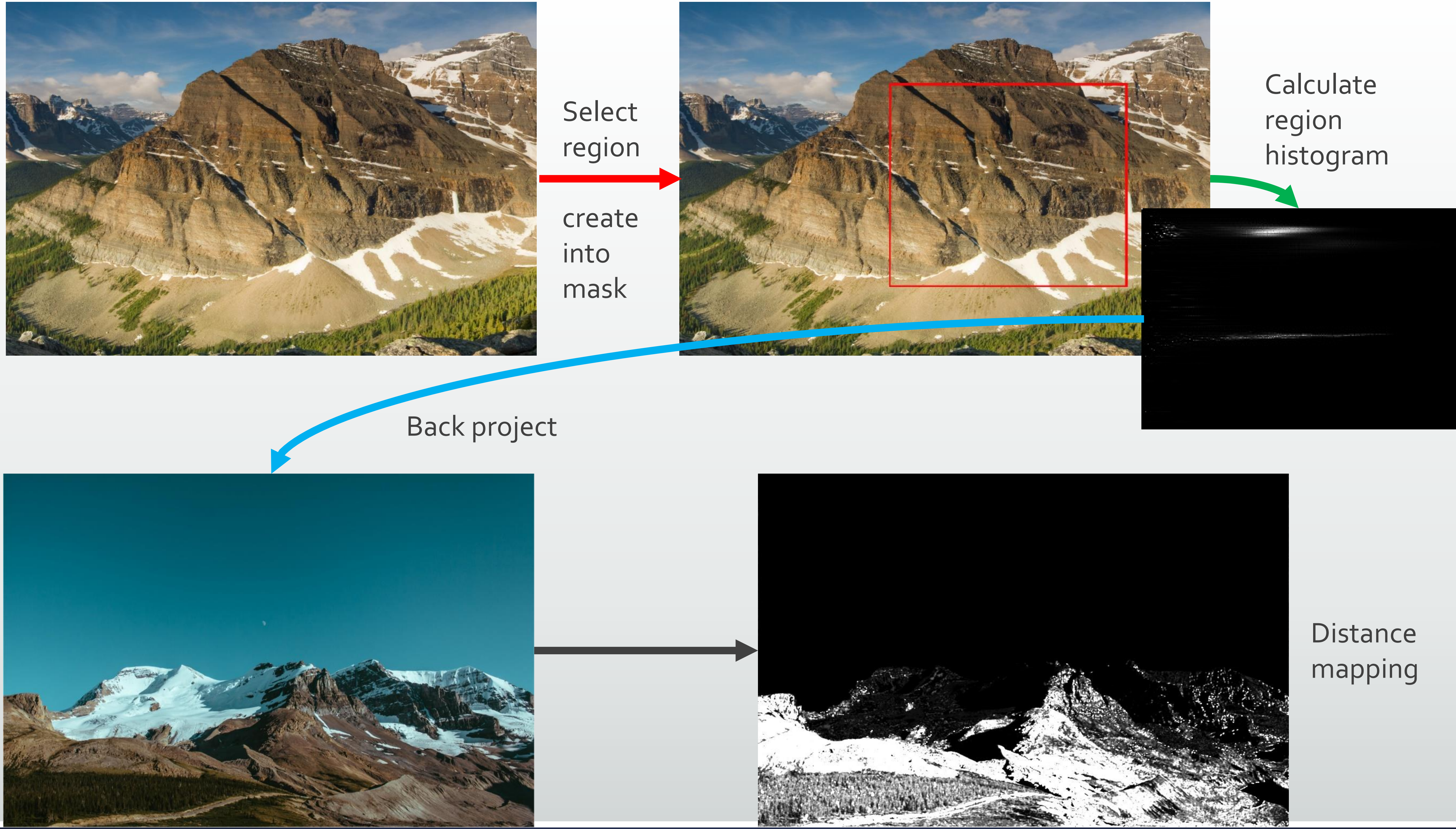

```
hist = cv2.calcHist( [img], [0, 1], mask, [180, 256], [0, 180, 0, 256] )
```

Histogram Backprojection

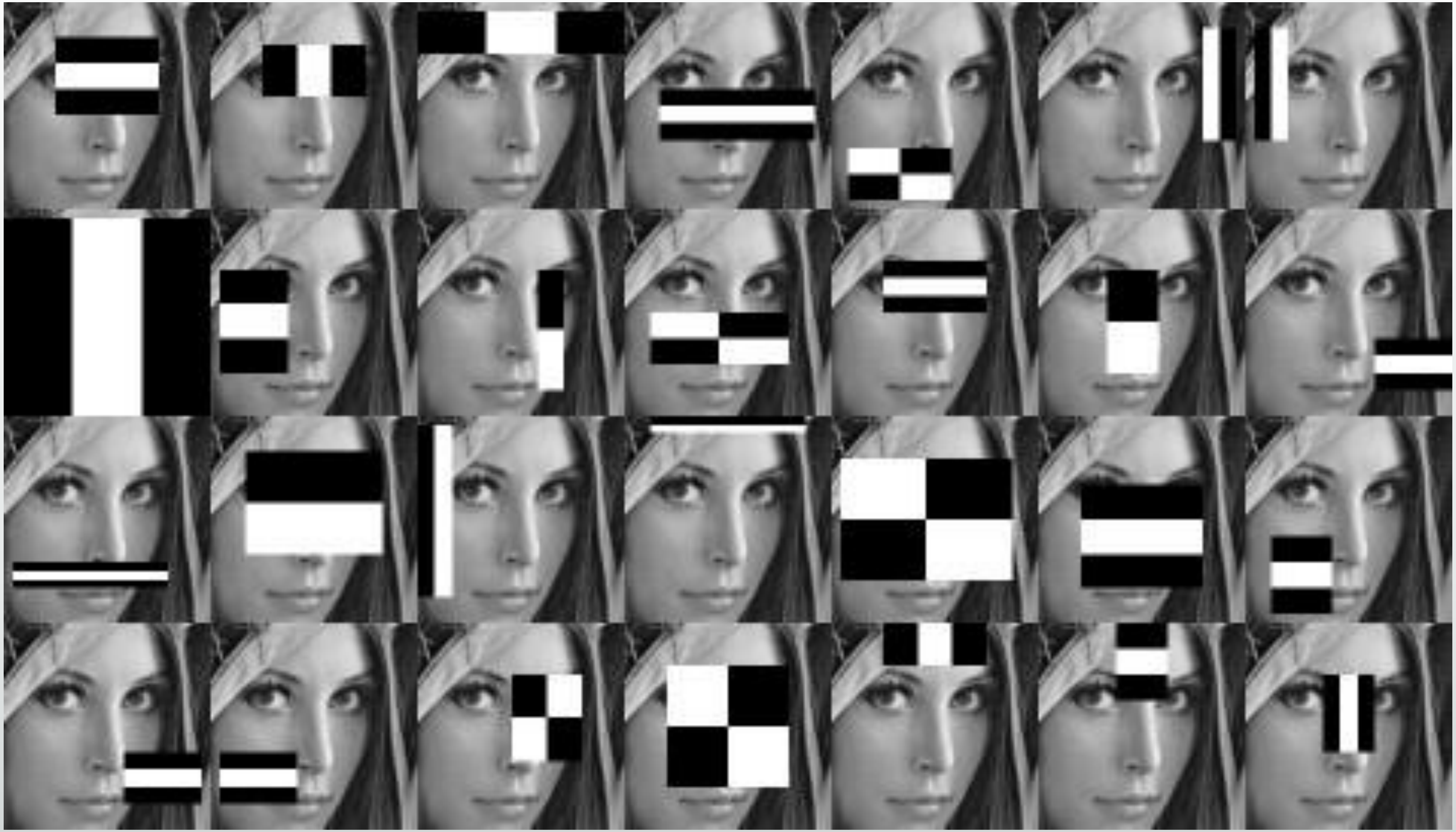
- Returns similarity of pixels in image region to the histogram
- Useful for object detection

Example of a mountain

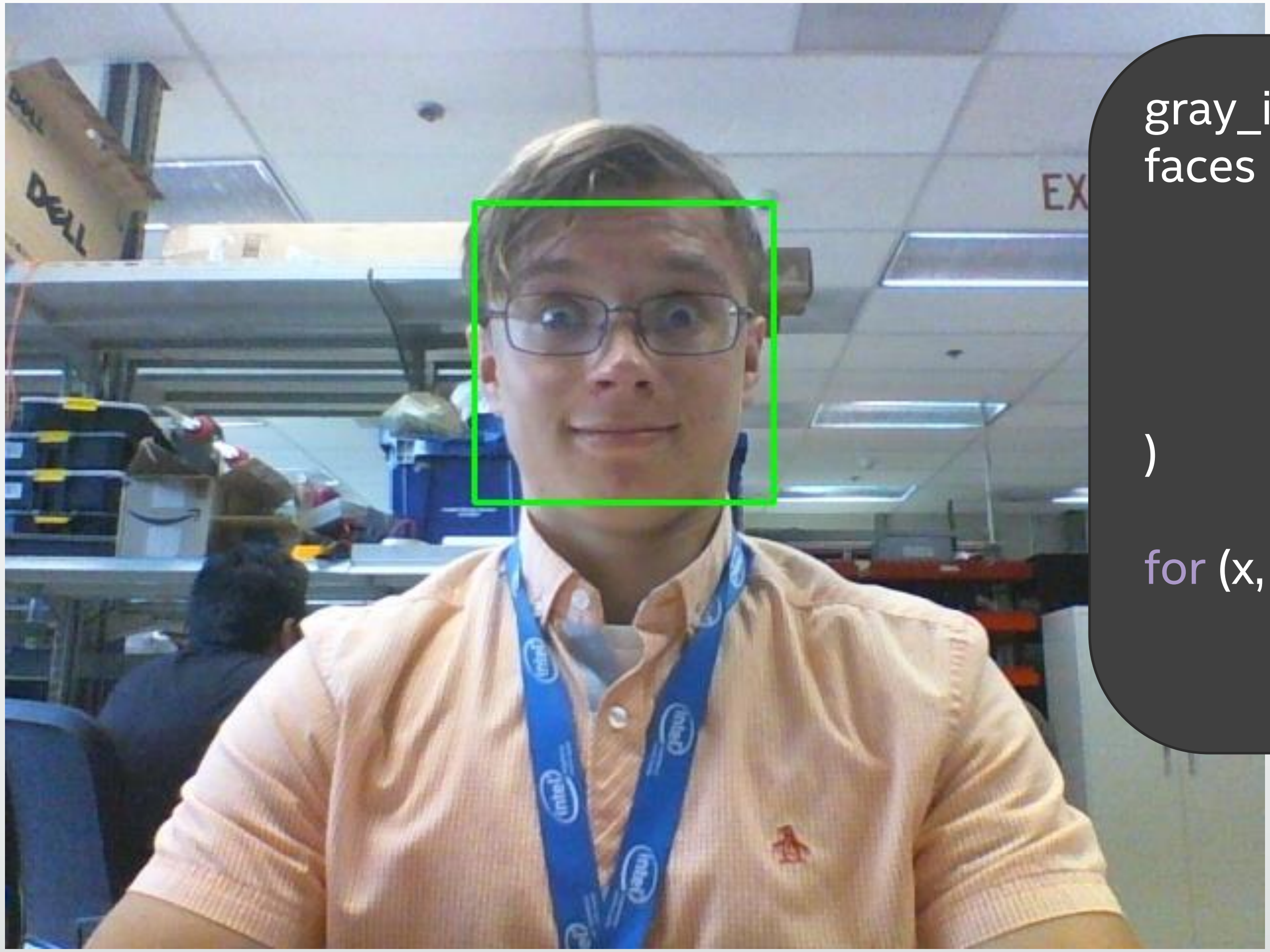
Find similar mountains


Select region
create into mask
Calculate region histogram
Back project
Distance mapping

Haar Cascades & Classifiers



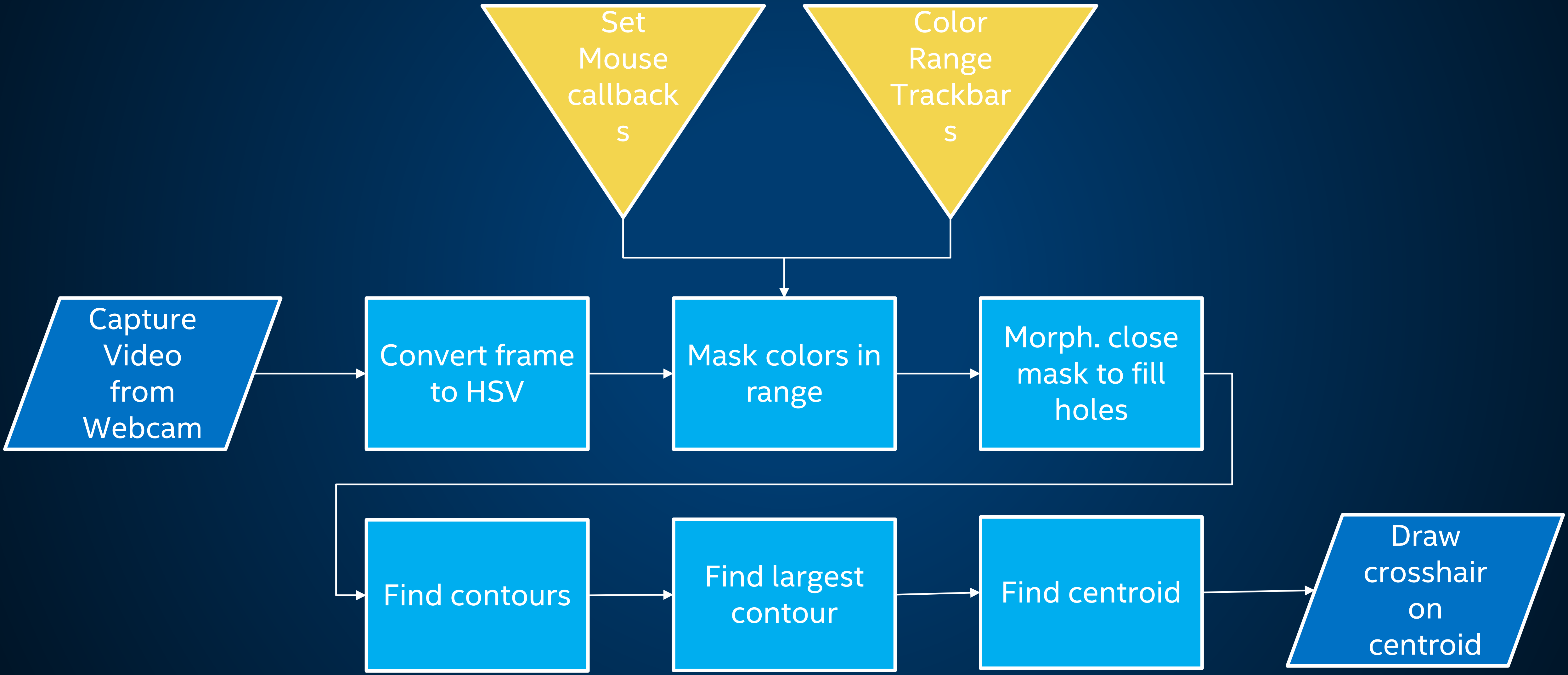
```python
cascade = cv2.CascadeClassifier( 'haarcascade_frontalface_default.xml' )
```



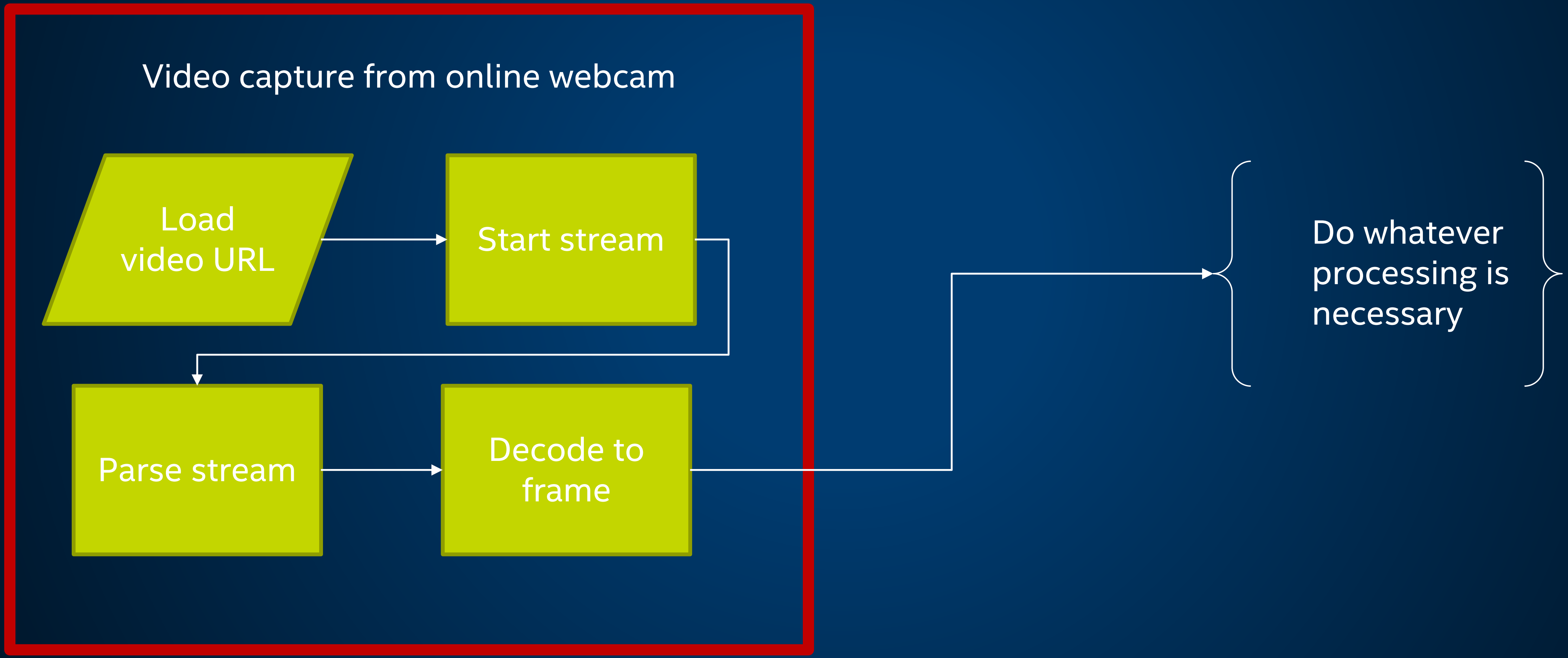

```python
gray_img = cv2.cvtColor( img, cv2.COLOR_BGR2GRAY )
faces = cascade.detectMultiScale(
            gray_img,
            scaleFactor = 1.25,
            minNeighbors = 5,
            minSize = ( 30, 30 ),
)

for (x, y, w, h) in faces:
            cv2.rectangle( img, x, y, (x+w, y+h), (0, 255, 0), 2)
```

COMPUTER VISION GRAPH EXAMPLES

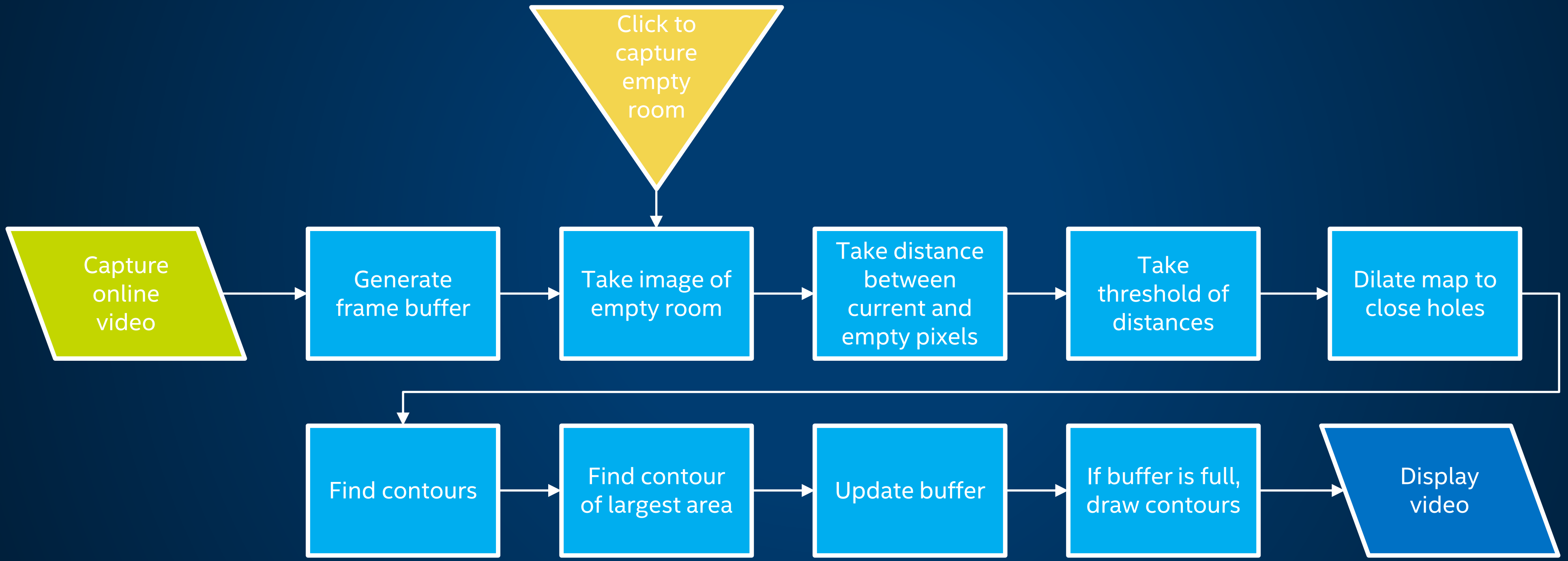
CROSSHAIR FOLLOWS OBJECT OF DISTINCT COLOR



MODULARITY: USING AN ONLINE VIDEO STREAM



PRESENCE DETECTION FROM AN ONLINE CAMERA



STORING VIDEO UPON FACE DETECTION