Model Train

SESSION 7

OUR SCHEDULE

# DAY 1

Session 1 : Revision and Simple Command on Linux

- Quick Flashback on Phase 1
- Task Given, Creating package, workspace
- Revise back some useful command in Linux

Session 2: Revise back about Python

- Revise back basic of python
- Some exercises to test programming skills

Session 3: Revise back about OpenCV

- Detecting Edges and Applying Image Filters
- Blurring
- Edge detection
- Converting between different color spaces

OUR SCHEDULE

# DAY 2

Session 4: Continue OpenCV

- Motion blur
- Sharpening
- Embossing
- Object Tracking

Session 5: Installation of YOLO

- Introduction to YOLO Bbox Annotation Tool
- Installation the Tool on LinuxOS (Laptop)

Session 6: : Labelling and Training model

- Do the labelling
- Train the model by using Laptop

OUR SCHEDULE

# DAY 3

Session 7: Image Training

- Creating train.txt and test.txt
- Train your images

Session 8: ROS + OPENCV + YOLO

- Setup Turtlebot
- Deploying OpenCV and Yolo on Turtlebot for Follow-Person and Line-Following

cv2.filter2D()

Robotics Teaching Kit Modules

Module 1 Course Introduction	• Course Introduction and Overview • Introduction to Robotics • Introduction to Jetson TK1/TX1 • Jet Overview • Introduction to ROS
Module 2 Sensors	• Sonar, Accelerometer, Gyroscope • Camera, Motors, Encoders
Module 3 Computer Vision	• Introduction to Computer Vision • Image Filtering • Image Moments
Module 4 Machine Learning	• Introduction to Machine Learning • Neural Networks • Caffe

Robotics Teaching Kit Modules

Module 5 Dead Reckoning	• Introduction to Dead Reckoning • Calculating Positions • Sensor Fusion and Kalman Filters
Module 6 Path Planning	• Introduction to Path Planning • A* Planning
Module 7 Control	• Introduction to Control Systems • PID Control
Module 8 Robot Localization	• Introduction to Robot Localization • Localization with Particle Filters
Module 9 Mapping	• Introduction to Mapping • SLAM
Module 10 Final Project	• Motivation • Harvester • Robot Capture the Flag Game • Color Follower

Behind the scene of 2D convolutions

This method applied for any filter

cv2.filter2D()

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

Introduction to numpy

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

```
kernel_sharpen_3 = np.array([[-1,-1,-1,-1,-1],
                             [-1,2,2,2,-1],
                             [-1,2,8,2,-1],
                             [-1,2,2,2,-1],
                             [-1,-1,-1,-1,-1]]) / 8.0
```

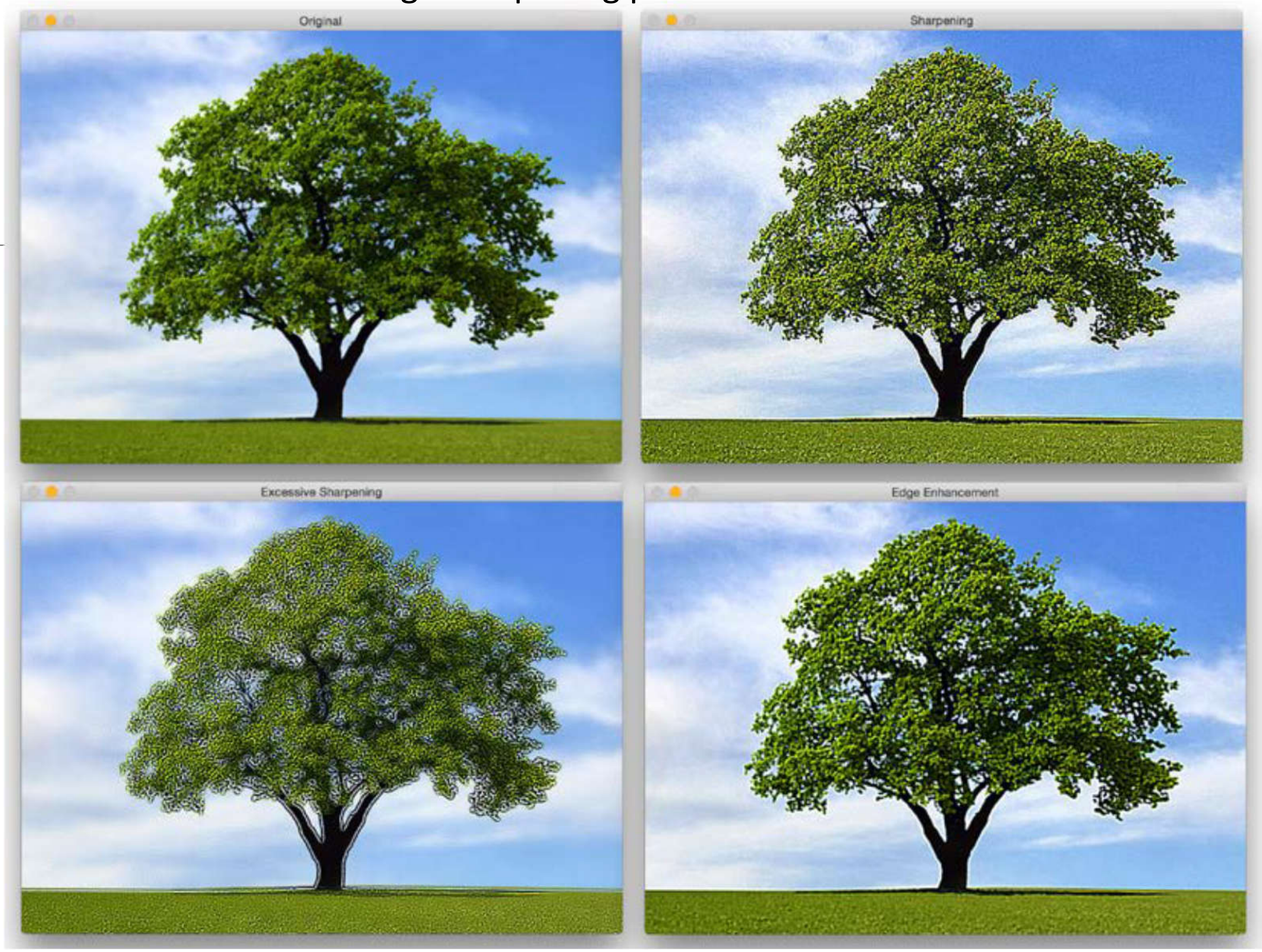
```
kernel = np.ones((5,5), np.uint8)
```

Sharpening

To sharpen the edges in the image.

To enhance the edges in an image that's not crisp.

Image sharpening process looks like:



As you can see in the preceding figure, the level of sharpening depends on the type of kernel we use.

We have a lot of freedom to customize the kernel here, and each kernel will give you a different kind of sharpening.

To just sharpen an image, like we are doing in the top right image in the preceding picture, we would use a kernel like this:

$$M = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

If we want to do excessive sharpening, like in the bottom left image, we would use the following kernel:

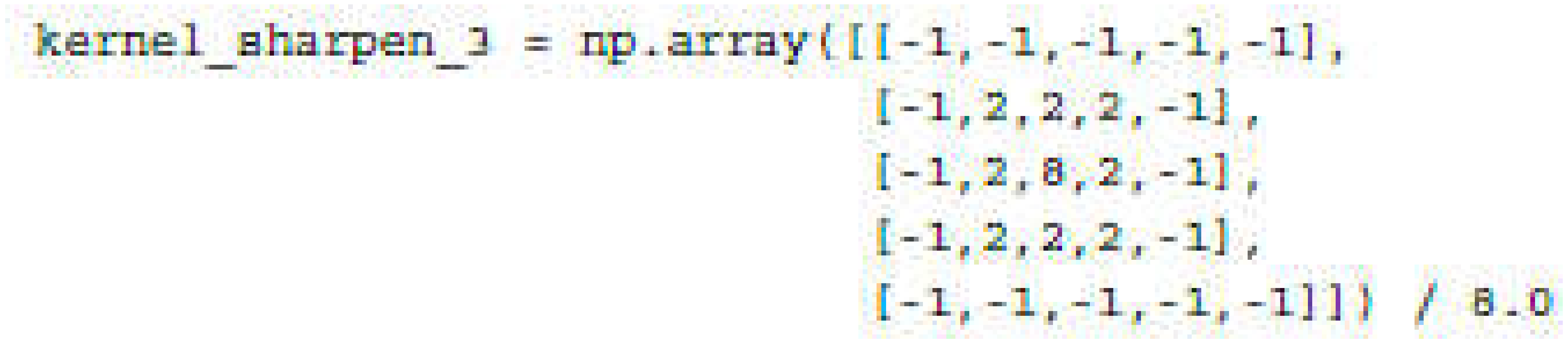
$$M = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -7 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

But the problem with these two kernels is that the output image looks artificially enhanced. If we want our images to look more natural, we would use an **Edge Enhancement** filter.

The underlying concept remains the same, but we use an approximate **Gaussian kernel** to build this filter.

It will help us smoothen the image when we enhance the edges, thus making the image look more natural.

Gaussian kernel

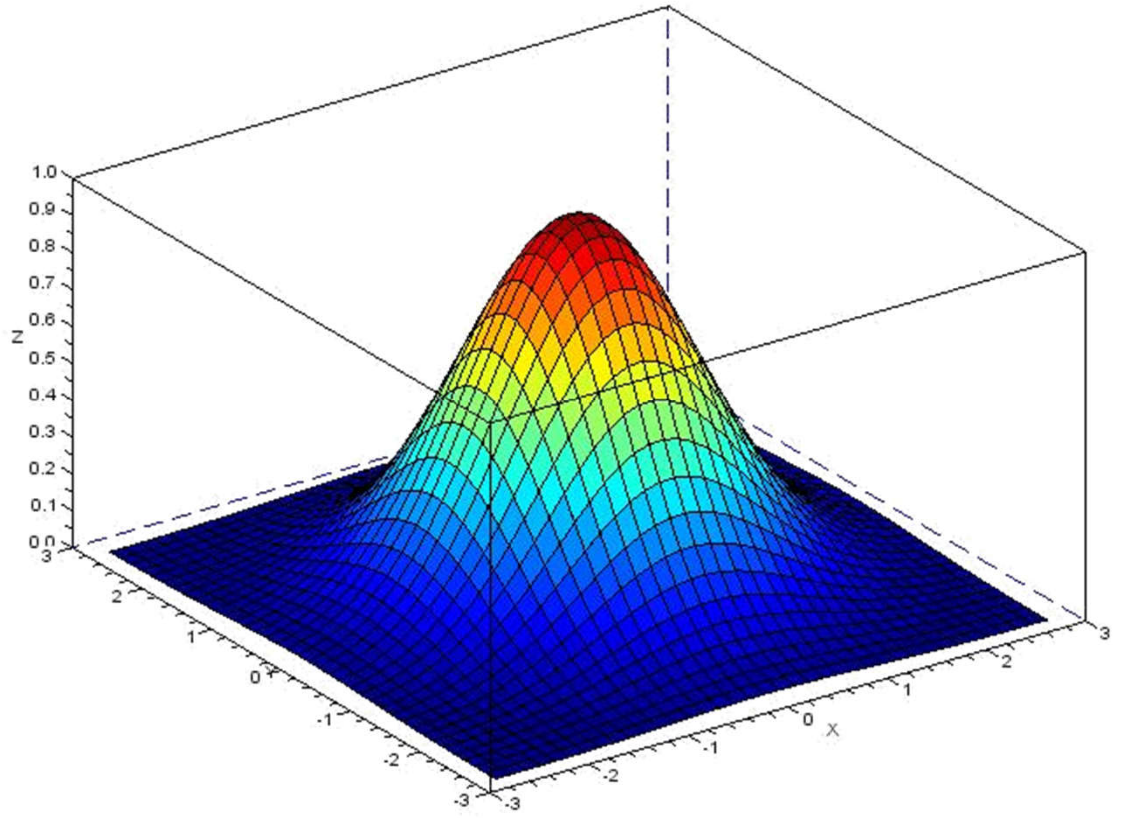


```
kernel_sharpen_3 = np.array([[-1,-1,-1,-1,-1],
                             [-1,2,2,2,-1],
                             [-1,2,8,2,-1],
                             [-1,2,2,2,-1],
                             [-1,-1,-1,-1,-1]]) / 8.0
```

Gaussian kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & 2 & 2 & 2 & -1 \\ -1 & 2 & 8 & 2 & -1 \\ -1 & 2 & 2 & 2 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$



Embossing

Convert it into an embossed image.

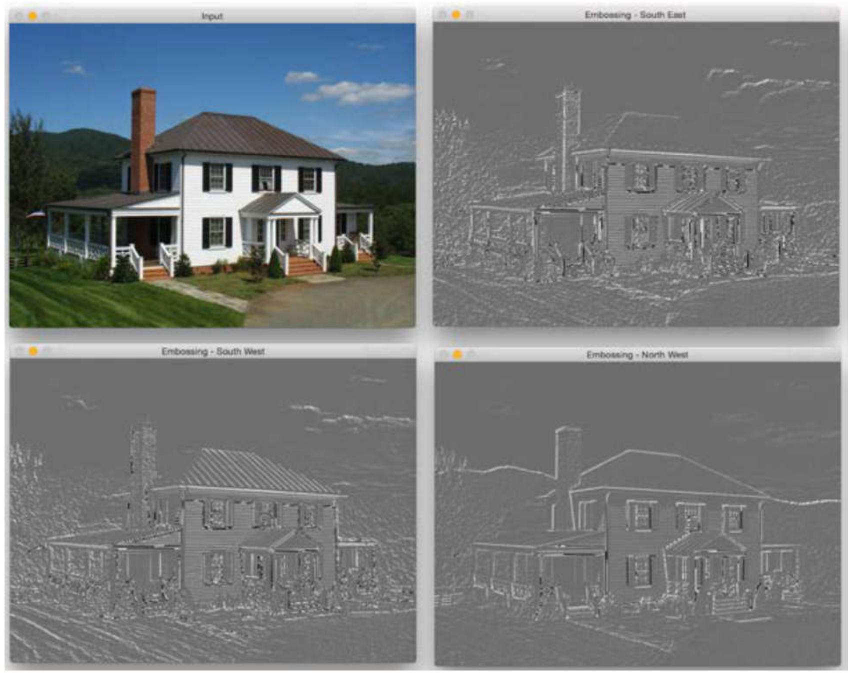
It take each pixel and replace it with a shadow or a highlight.

Let's say we are dealing with a relatively plain region in the image.

Need to replace it with plain gray color because there's not much information there.

If there is a lot of contrast in a particular region, we will replace it with a white pixel (highlight), or a dark pixel (shadow), depending on the direction in which we are embossing.

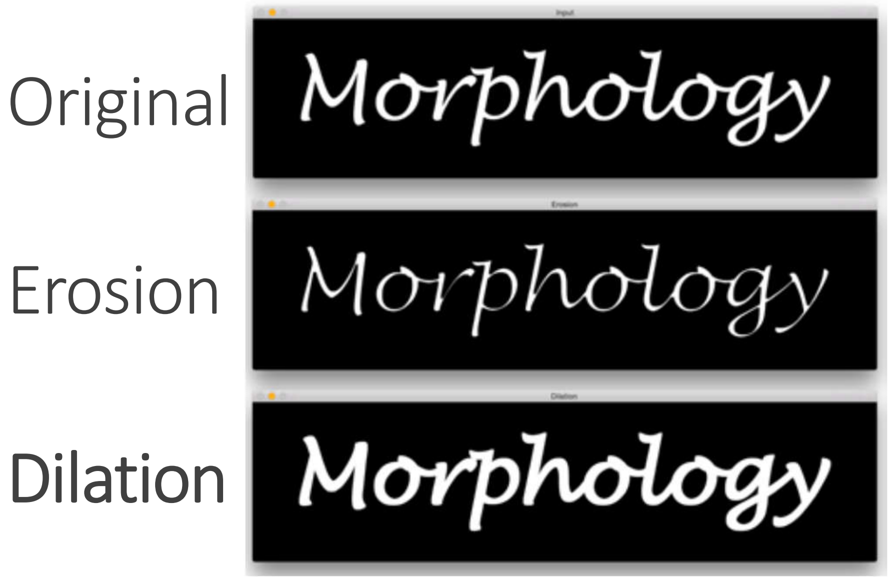
This is what it will look like:



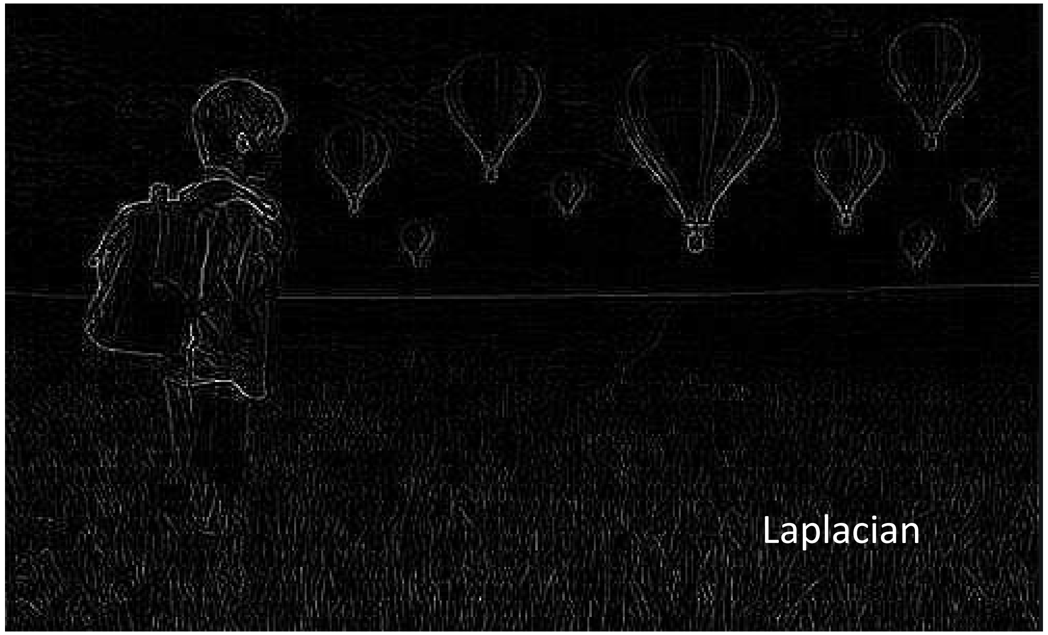
Erosion and dilation

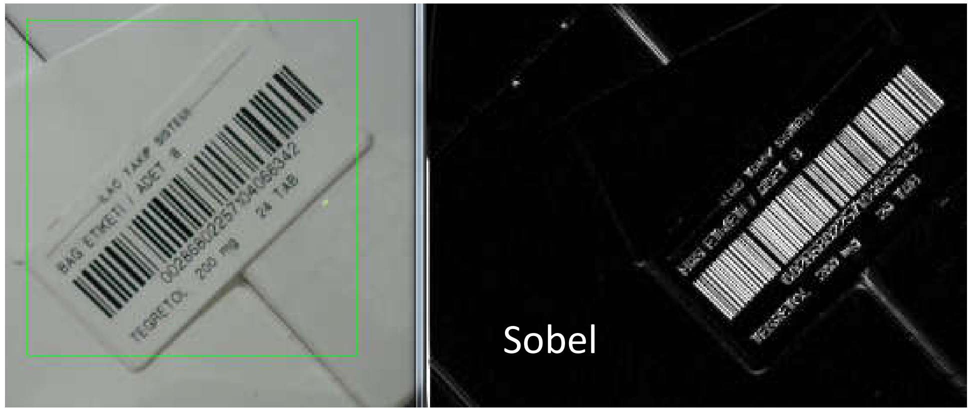
Erosion and **dilation** are morphological image processing operations

Let's see what these operations look like:



Erosion and dilation





Enhancing the contrast in an image

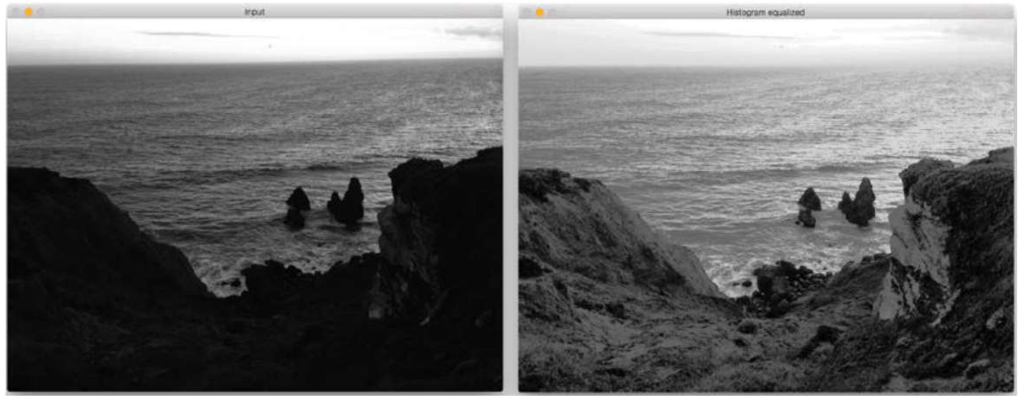
Whenever we capture images in low-light conditions, the images turn out to be dark.

This typically happens when you capture images in the evening or in a dimly lit room.

When this happens, a lot of details in the image are not clearly visible to the human eye.

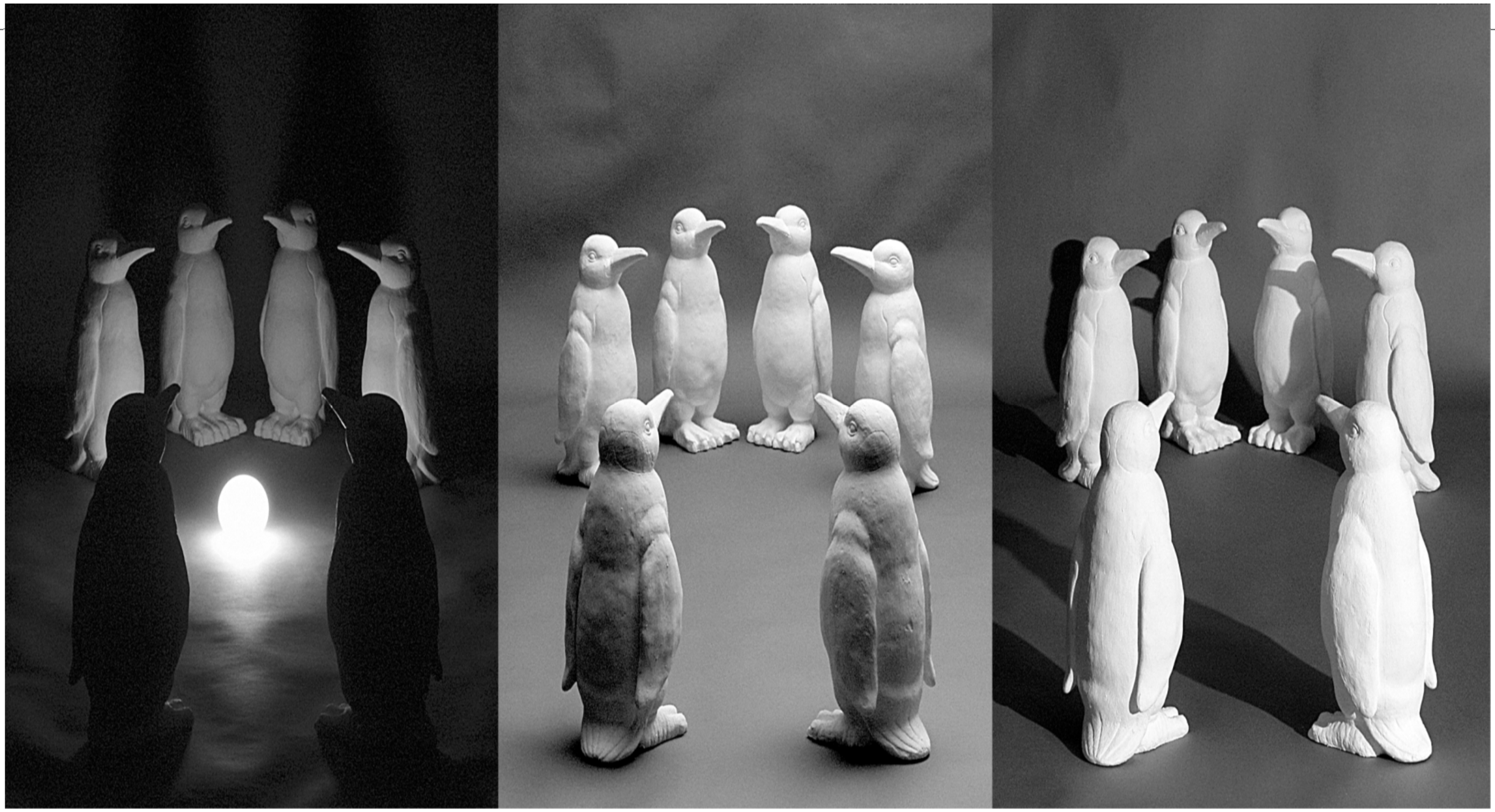
We use a process called **Histogram Equalization** to achieve this.

To give an example, this is what it looks like before and after contrast enhancement:



As we can see here, the input image on the left is really dark. To rectify this, we need to adjust the pixel values so that they are spread across the entire spectrum of values, that is, between 0 and 255.

Challenge: Lighting Conditions



Exercise

Use your webcam :
Perform enhancement of the image contrast

Drawing on an image

rectangle

cv2.rectangle(output, (320, 60), (420, 160), (0, 0, 255), 2)

Syntax: *cv2.rectangle(image, start_point, end_point, color, thickness)*

circle

cv2.circle(output, (300, 150), 20, (255, 0, 0), -1)

Syntax: *cv2.circle(image, center_coordinates, radius, color, thickness)*

line

cv2.line(output, (60, 20), (400, 200), (0, 0, 255), 5)

Syntax: *cv2.line(image, start_point, end_point, color, thickness)*

Text in your image: PutText

cv2.putText(output, "OpenCV + Jurassic Park!!!", (10, 25), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)

Syntax: *cv2.putText(image, text, org, font, fontScale, color[, thickness[, lineType[, bottomLeftOrigin]]])*

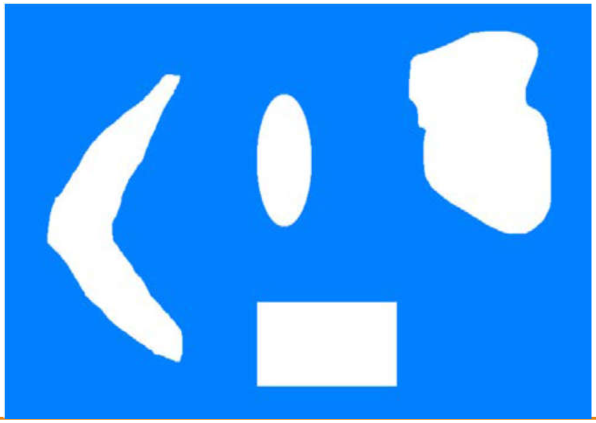
Detecting Shapes and Segmenting an Image

IN THIS CHAPTER, WE ARE GOING TO LEARN ABOUT SHAPE ANALYSIS

Contour analysis and shape matching

Contour analysis is a very useful tool in the field of computer vision. We deal with a lot of shapes in the real world and contour analysis helps in analyzing those shapes using various algorithms.

Let's say we want to identify the boomerang shape in the following image:

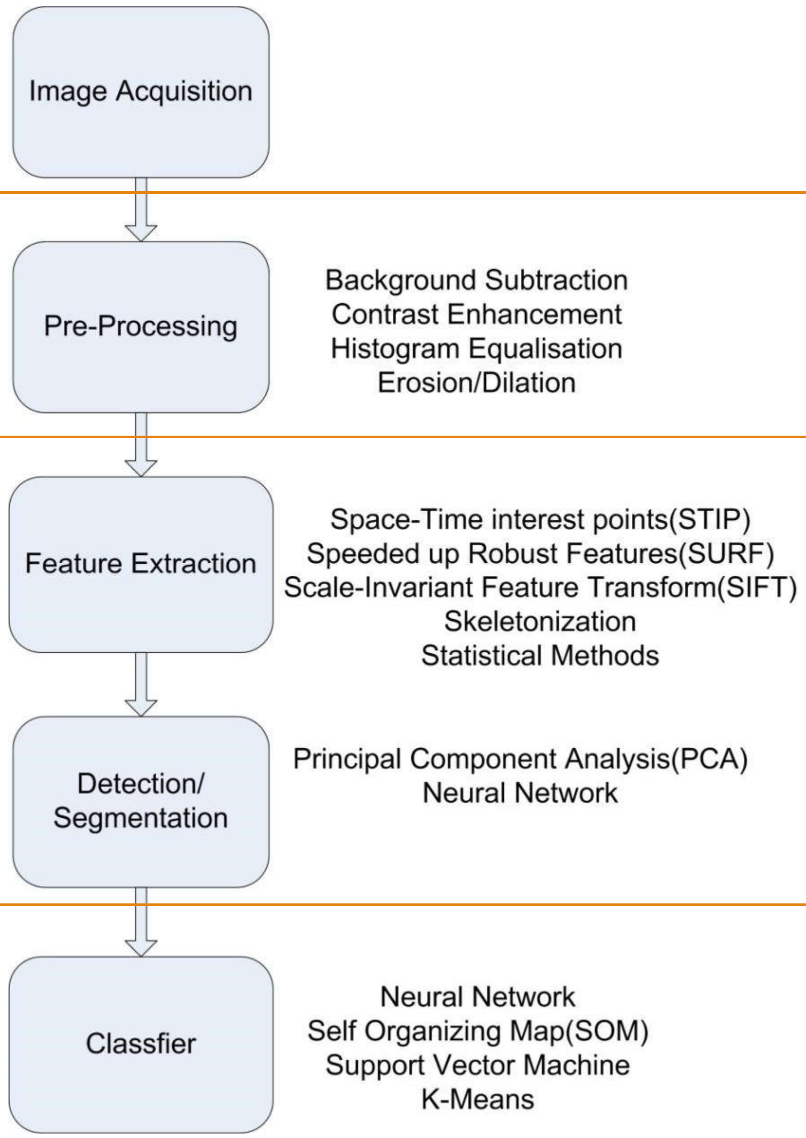


In order to do that, we first need to know what a regular boomerang looks like:

cv2.imread

cv2.bilateralFilter
cv2.cvtColor
cv2.Canny

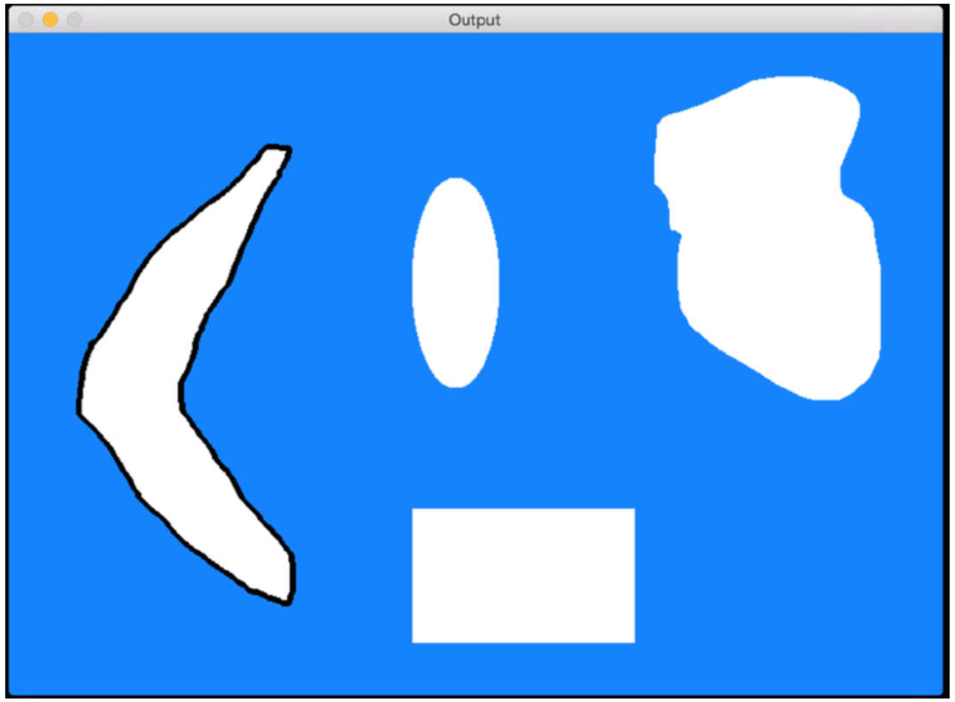
cv2.findContours



The concept of "image moments" basically refers to the weighted and power-raised summation of the pixels within a shape.

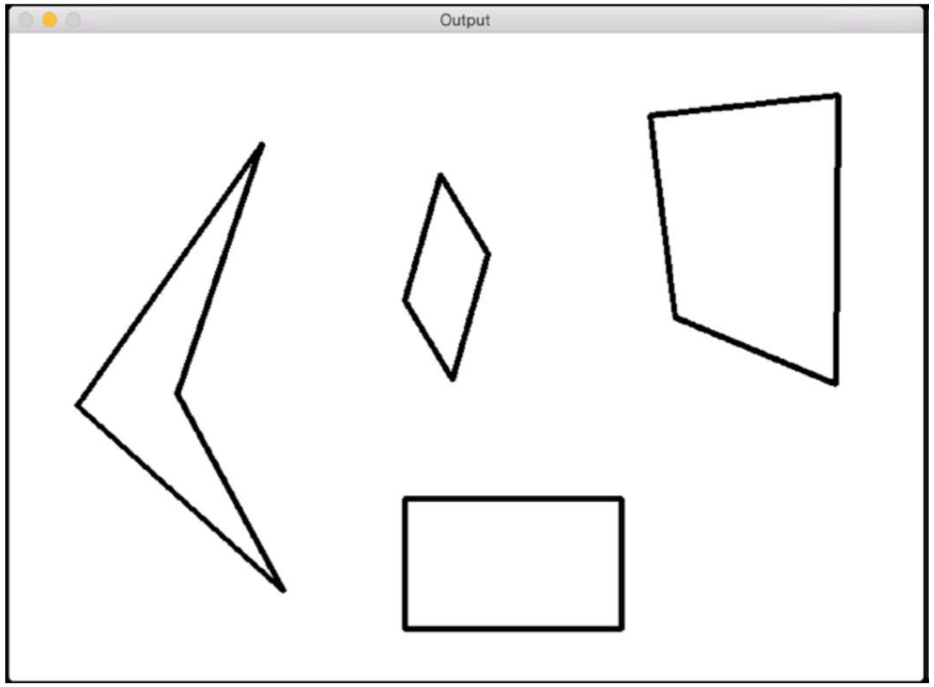
$$I = \sum_{i=0}^{N} w_i p_i^k$$

If we match the shapes, you will see something like this:

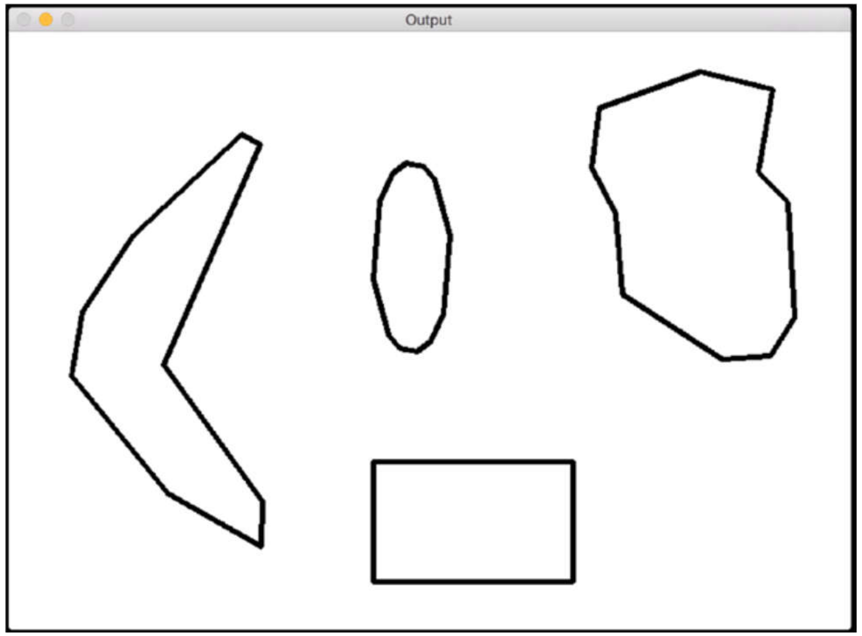


Approximating a contour

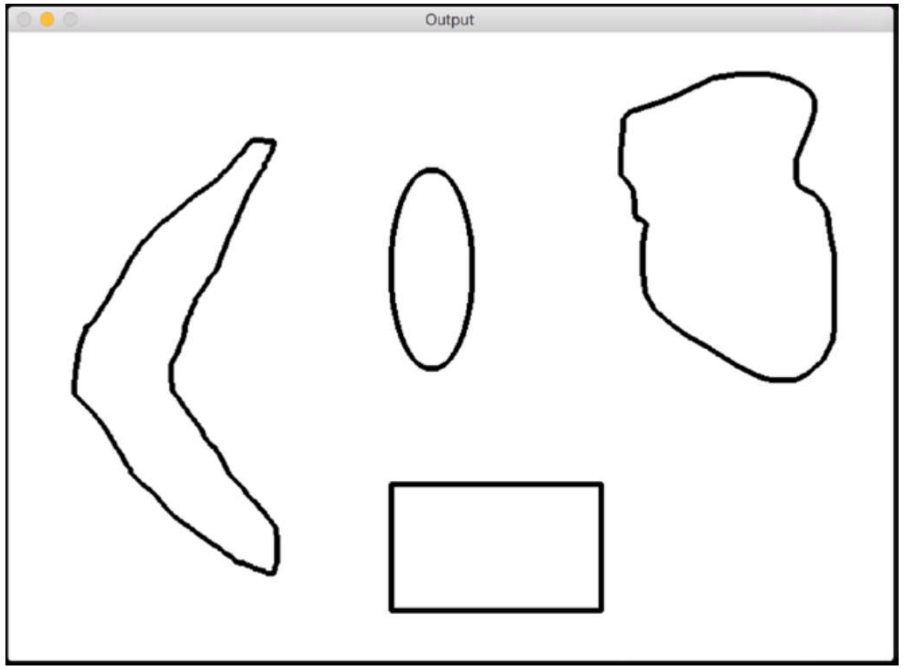
Let's start with a factor of 0.05:



If you reduce this factor, the contours will get smoother. Let's make it 0.01:

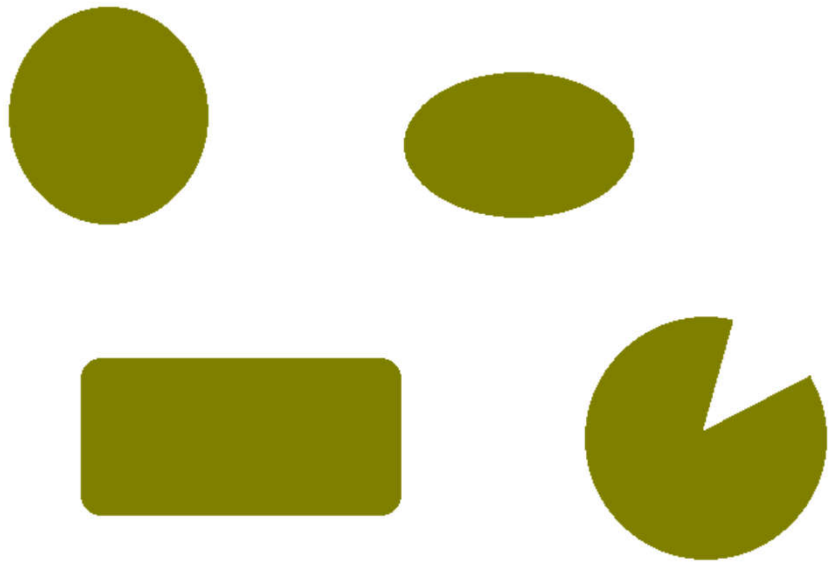


If you make it really small, say 0.00001, then it will look like the original image:

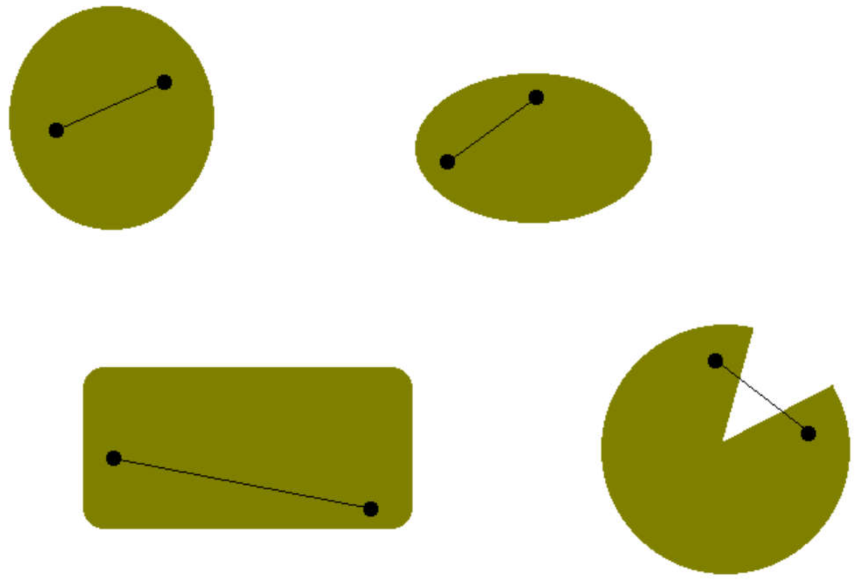


Identifying the pizza with the slice taken out

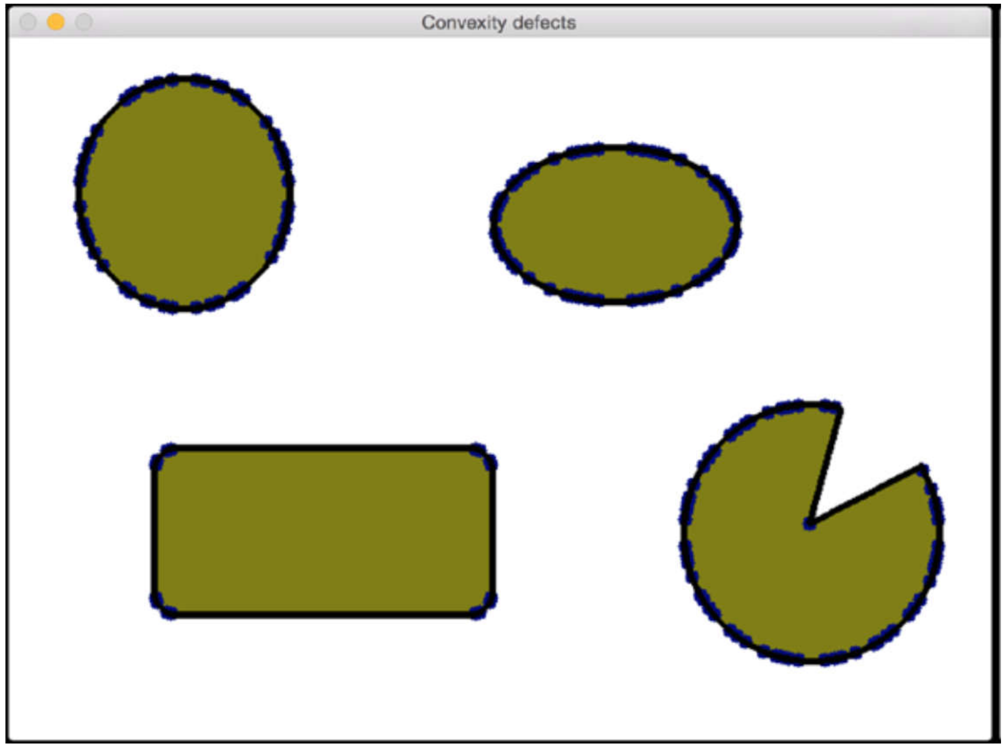
Let's consider the following image:



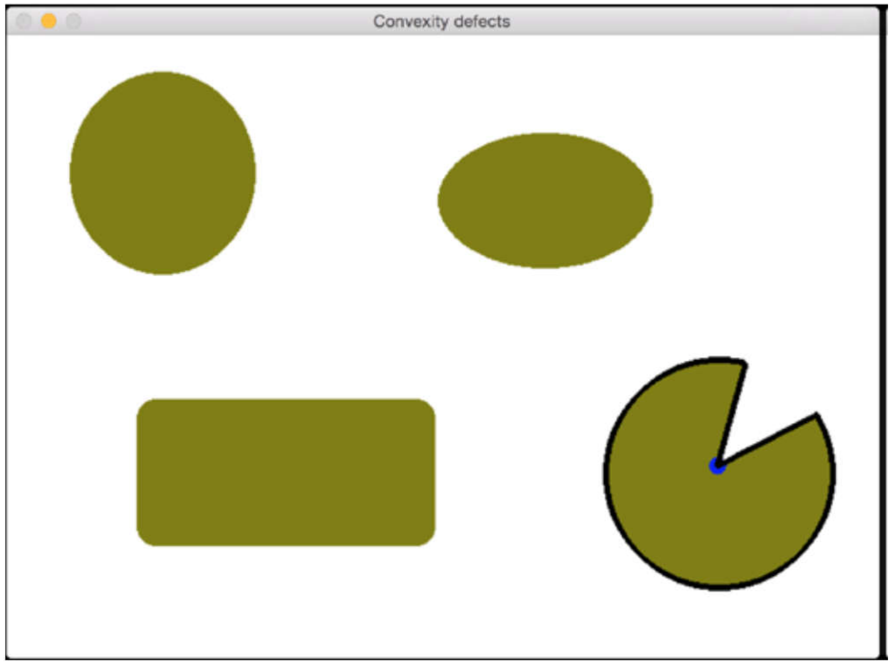
If you look at the sliced pizza shape, we can choose two points such that the line between them goes outside the shape as shown in the figure that follows:



If you run the above code, you will see something like this:

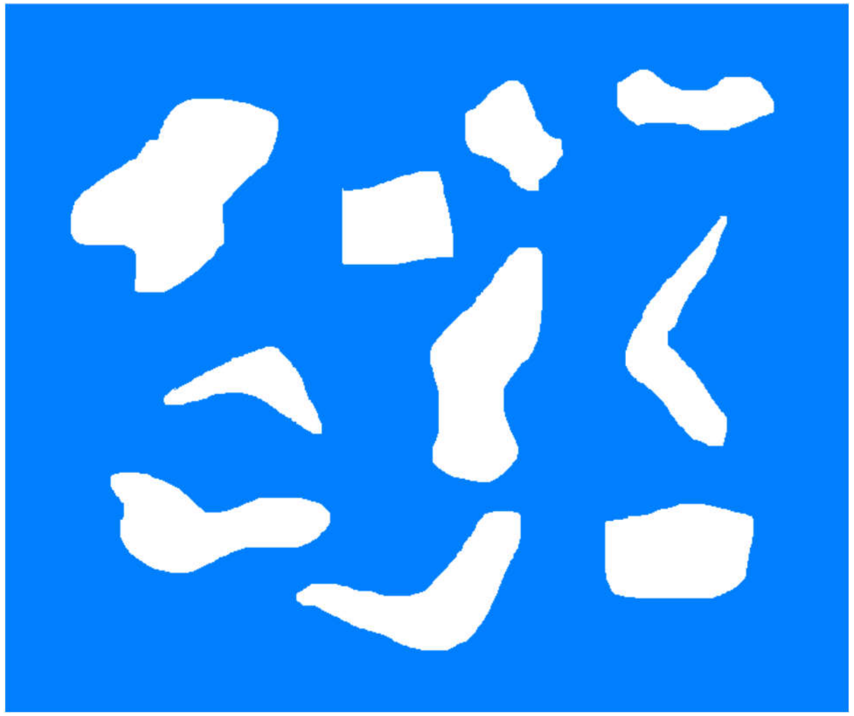


If you run the preceding code, the output will look like the following:

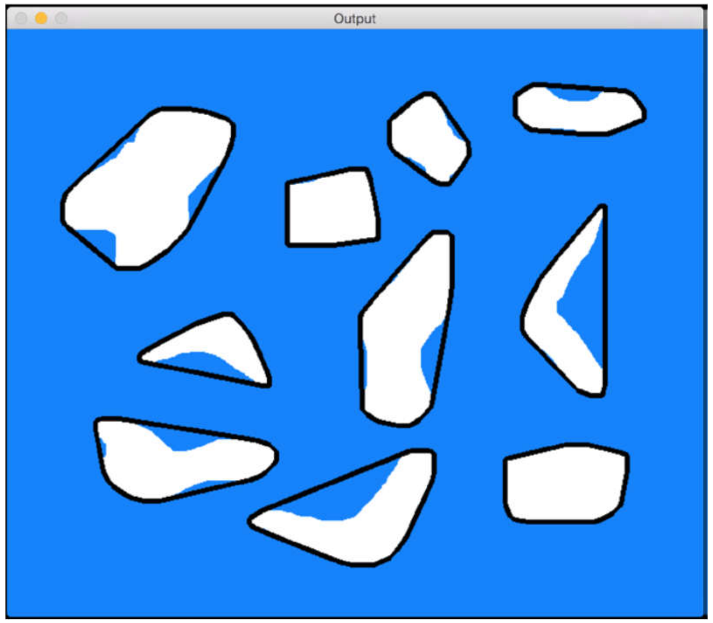


How to censor a shape?

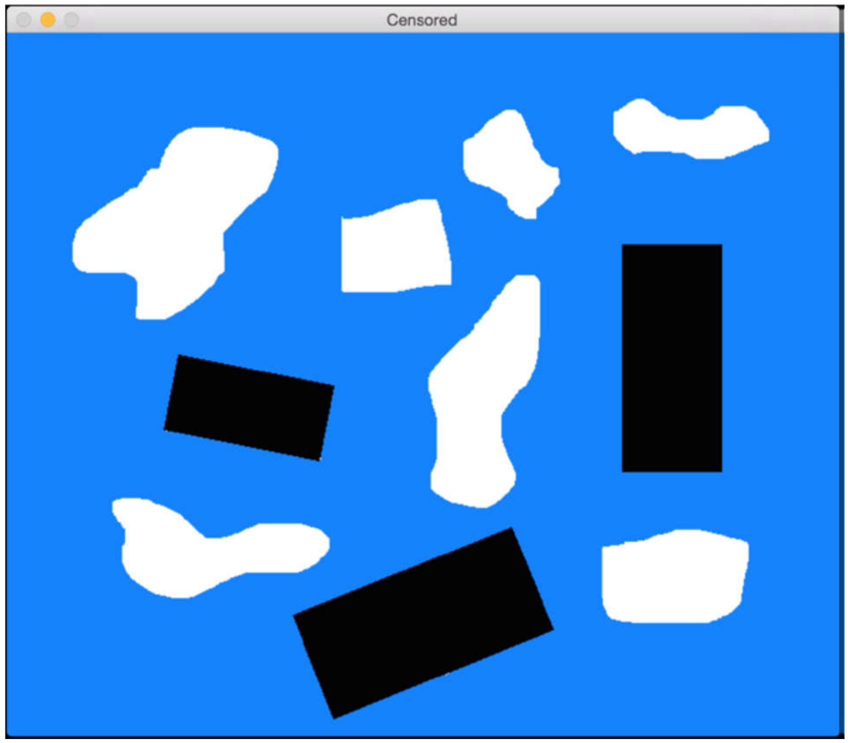
Let's consider the following figure:



This metric will have a lower value for the boomerang shapes because of the empty area that will be left out, as shown in the following figure:



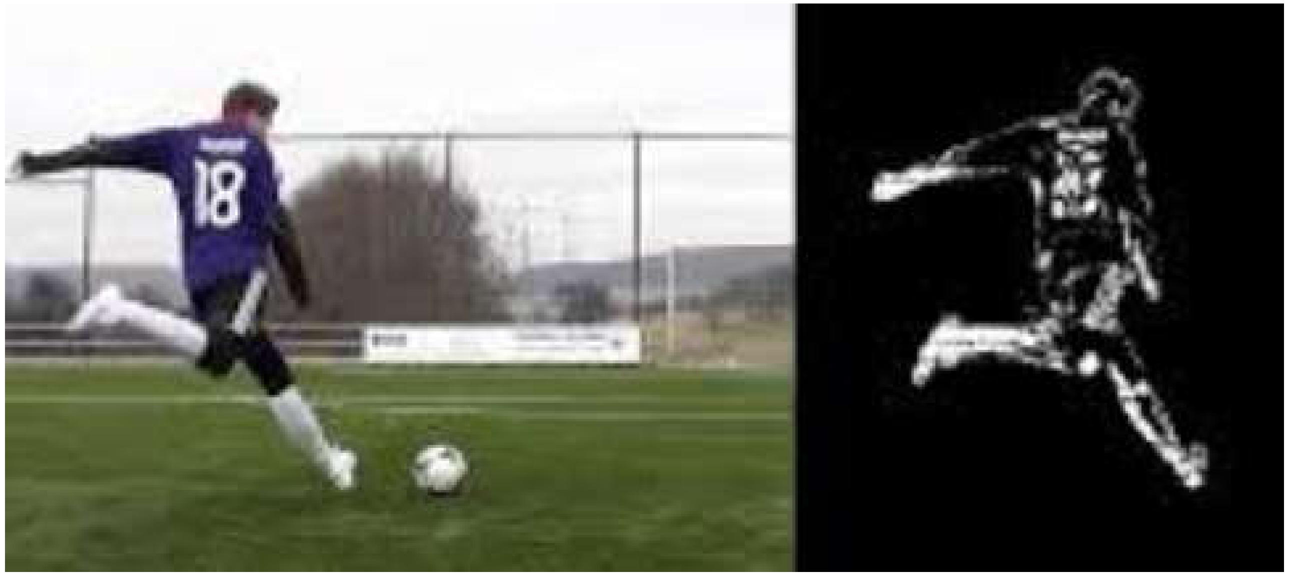
If you detect the shapes and block them out, it will look like this:



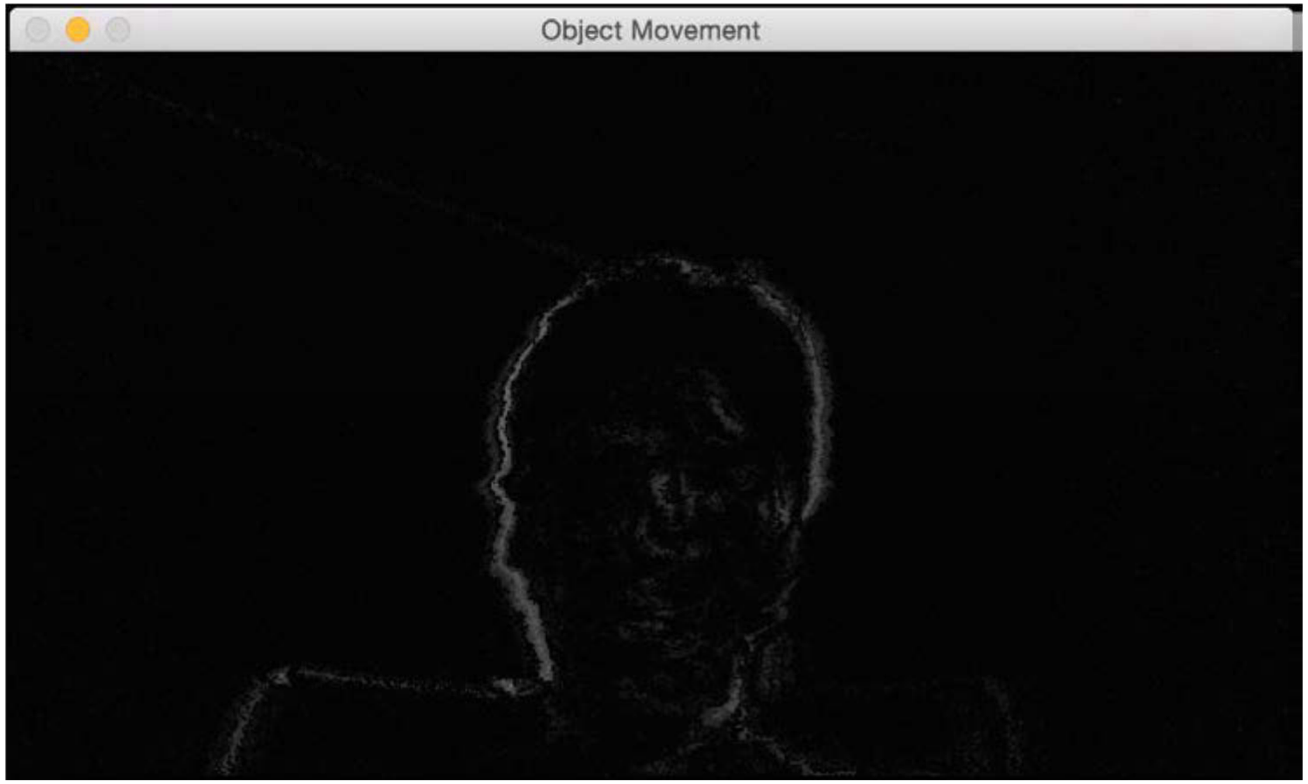
Object Tracking

Frame differencing

This is, possibly, the simplest technique we can use to see what parts of the video are moving. When we consider a live video stream, the difference between successive frames gives us a lot of information. The concept is fairly straightforward! We just take the difference between successive frames and display the differences.



If I move my laptop rapidly from left to right, we will see something like this:

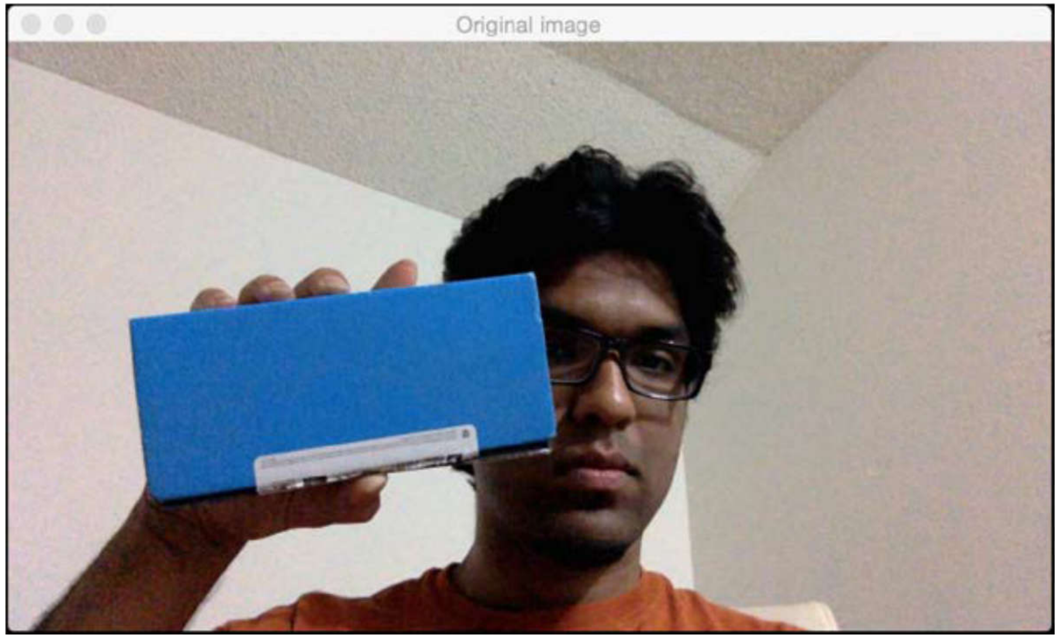


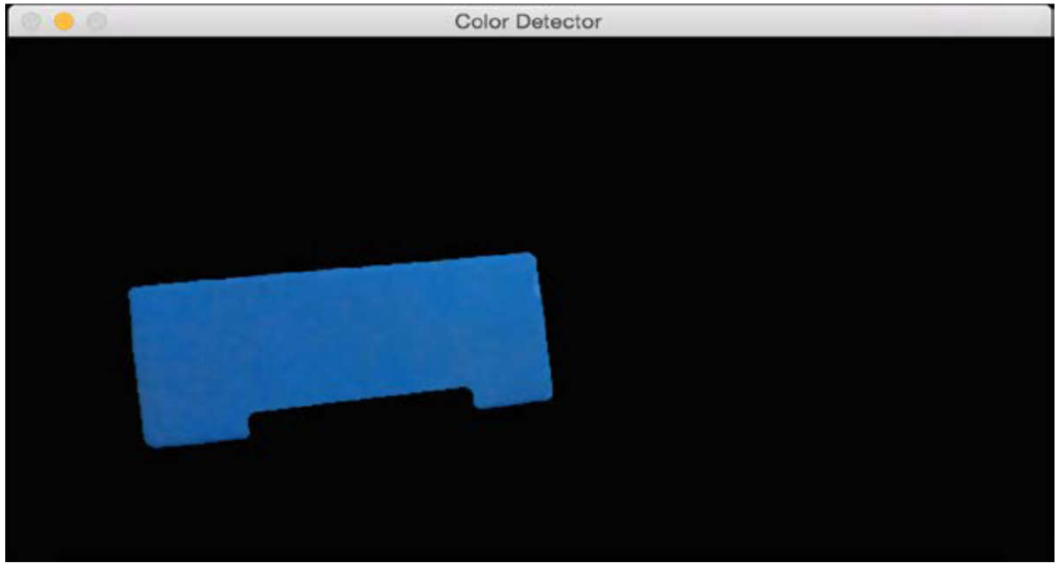
Colorspace based tracking

CAN CONVERT AN IMAGE TO THE HSV SPACE, AND THEN USE COLORSPACETHRESHOLDING TO TRACK A GIVEN OBJECT.

Consider the following frame in the video:

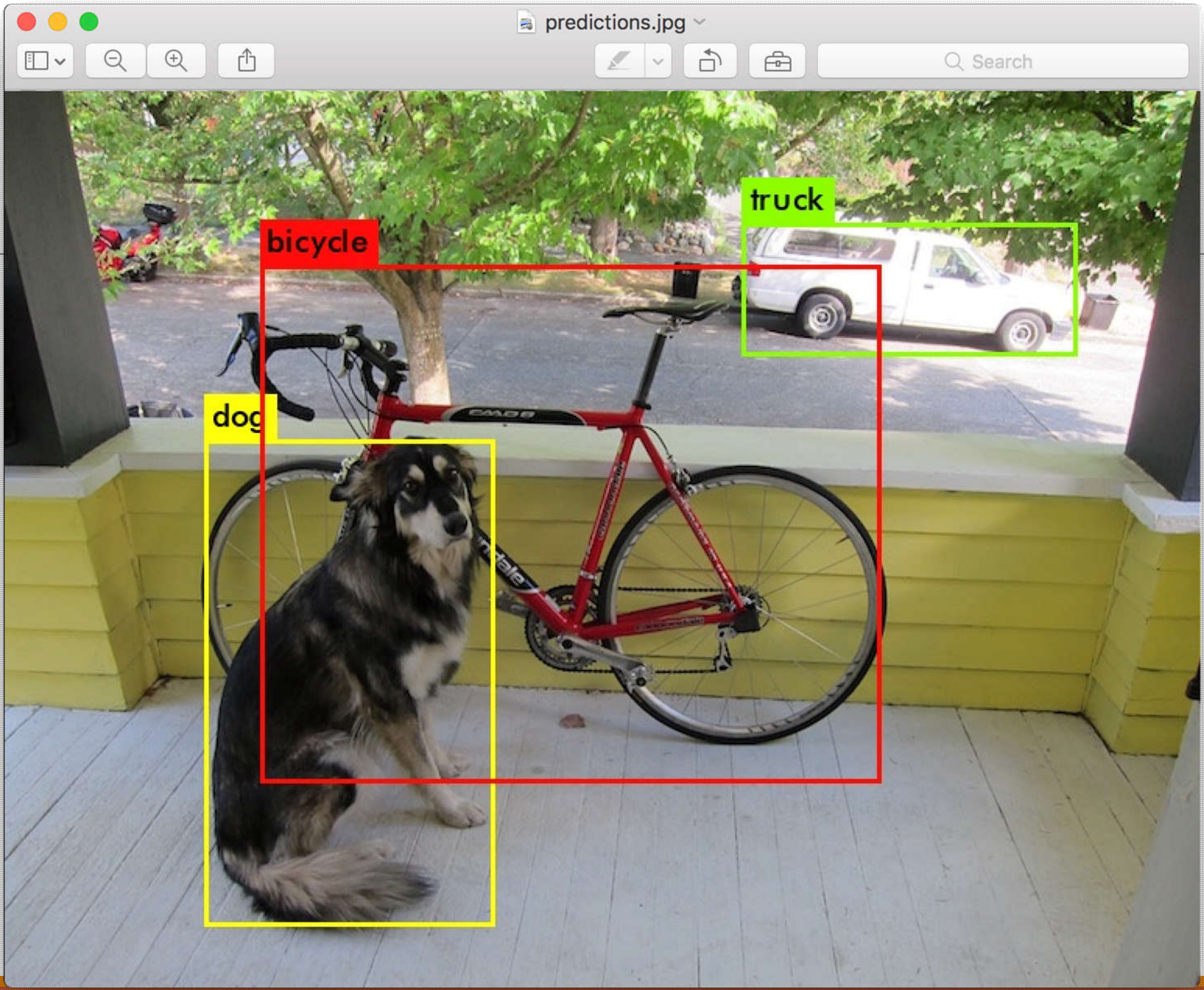
If you run it through the colorspace filter and track the object, you will see something like this:





Introduction to YOLO

YOU ONLY LOOK ONCE - REAL-TIME OBJECT DETECTION


predictions.jpg
Search
truck
bicycle
dog

Installation on YOLO

Open Terminal in your working directory, clone the Darknet repository, and build it:

```
git clone https://github.com/pjreddie/darknet
cd darknet
make
```

Then, run

```
./darknet
```

and you should have the output:

```
usage: ./darknet <function>
```

Testing Model

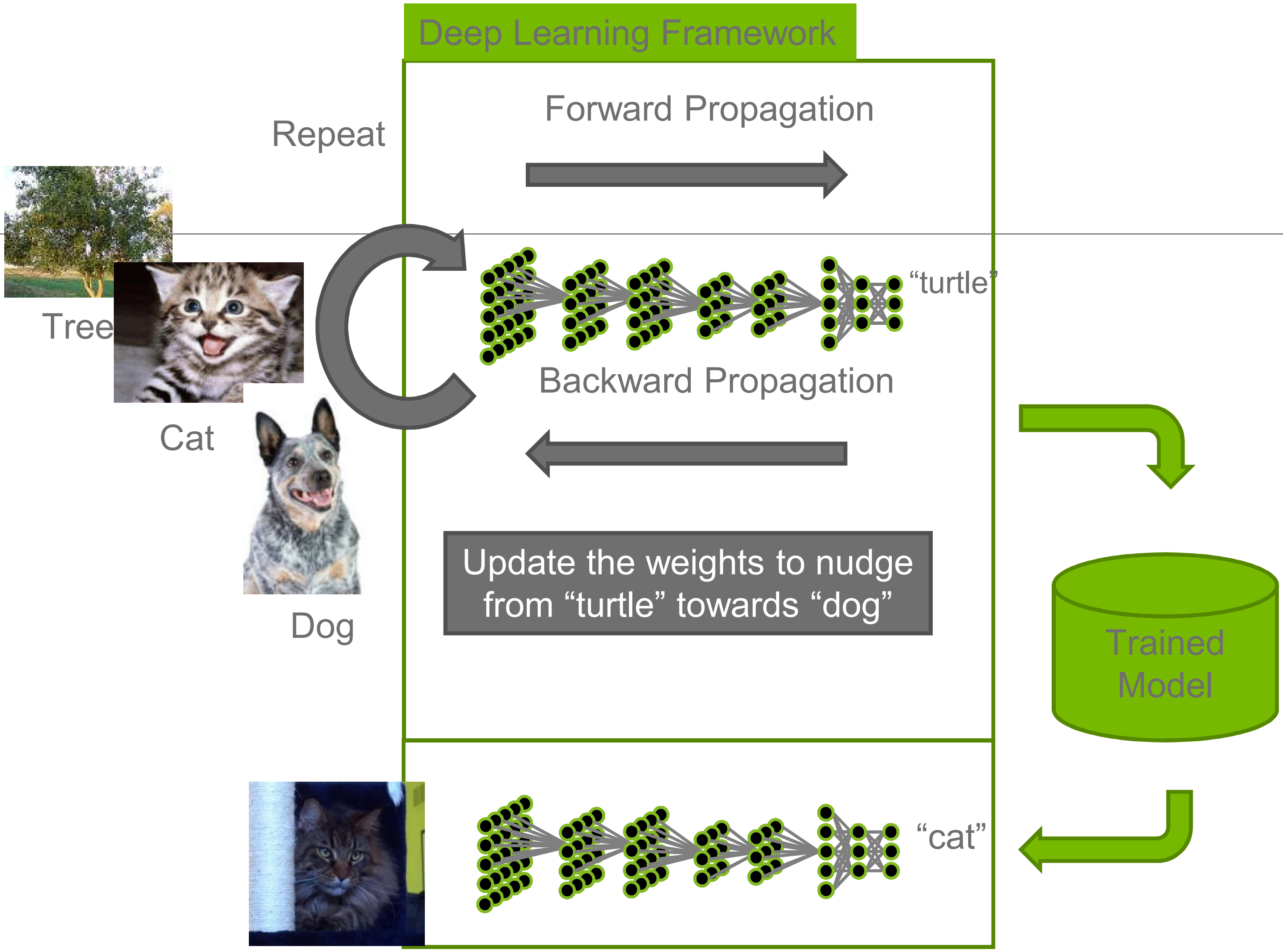
Tiny YOLOv3

To use this model, first download the weights:

```
wget https://pjreddie.com/media/files/yolov3-tiny.weights
```

Then run the detector with the tiny config file and weights:

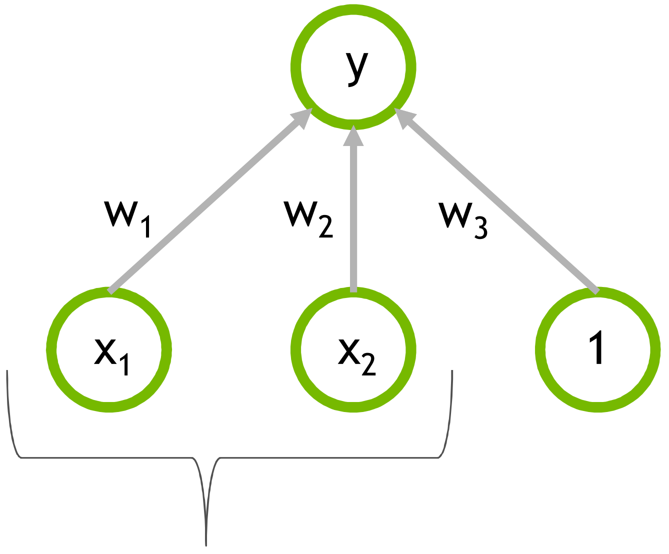
```
./darknet detect cfg/yolov3-tiny.cfg yolov3-tiny.weights data/dog.jpg
```


Deep Learning Framework
Forward Propagation
Repeat
Tree
Cat
Dog
“turtle”
Backward Propagation
Update the weights to nudge from “turtle” towards “dog”
Trained Model
“cat”

[redacted]

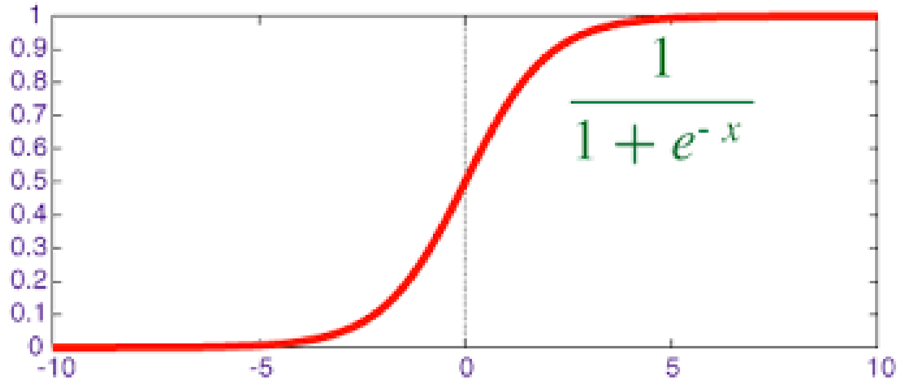
Artificial neurons

AND Gate



Input: 0 or 1

F(x) = Activation Function

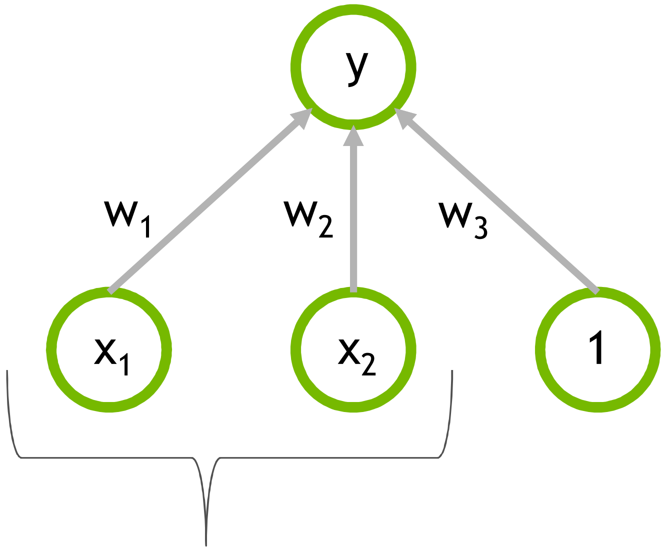


$y=F(w_1x_1+w_2x_2+w_3)$

x1	x2	y
0	0	0
1	0	0
0	1	0
1	1	1

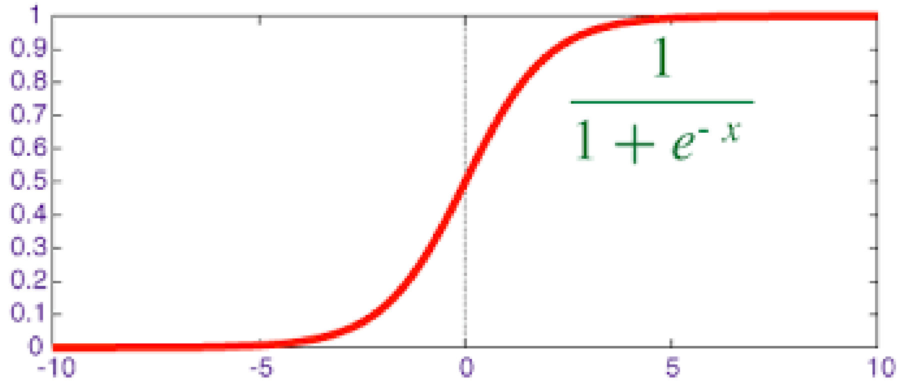
Artificial neurons

AND Gate



Input: 0 or 1

F(x) = Activation Function



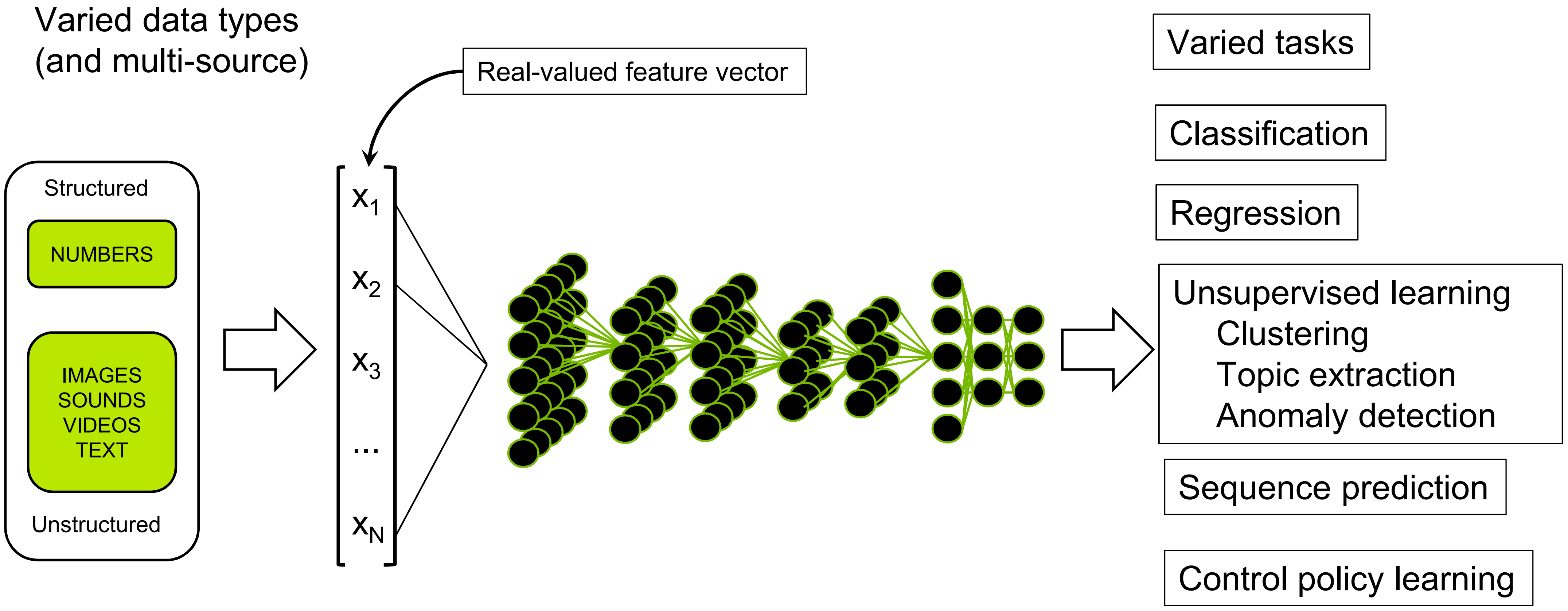
$y=F(w_1x_1+w_2x_2+w_3)$

Epoch 1: W1=5; W2=5; W3 = -10

Epoch 2: W1=6; W2=6; W3 = -10

Epoch 100: W1=20; W2=20; W3 = -30

Deep Learning generalizes solution across problems



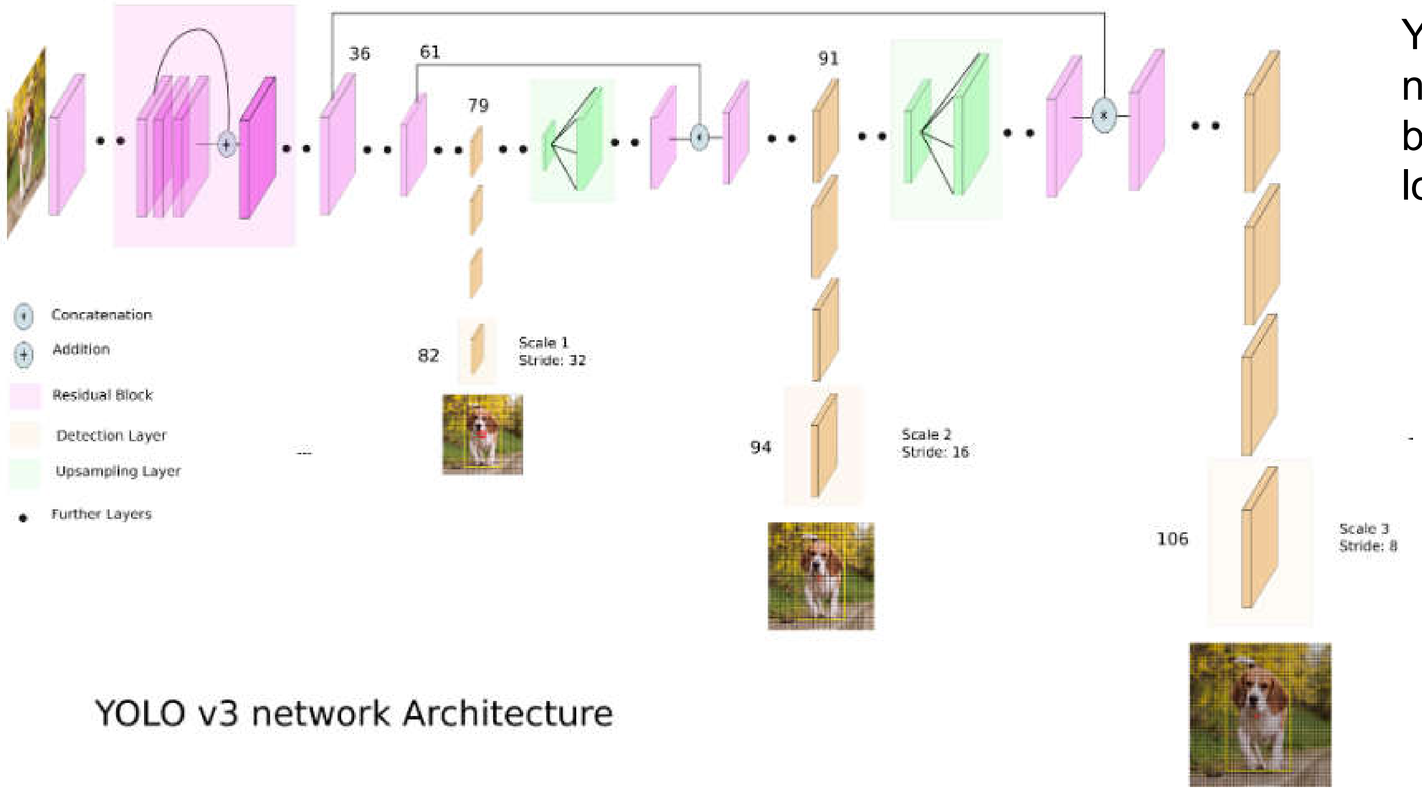
Constants: Big (high dimensional) Data + a complex function to learn

Install YOLO BBox Annotation Tool

Open the terminal and type

```
$ git clone https://github.com/drainingsun/boobs.git
```

What is YOLOv3?

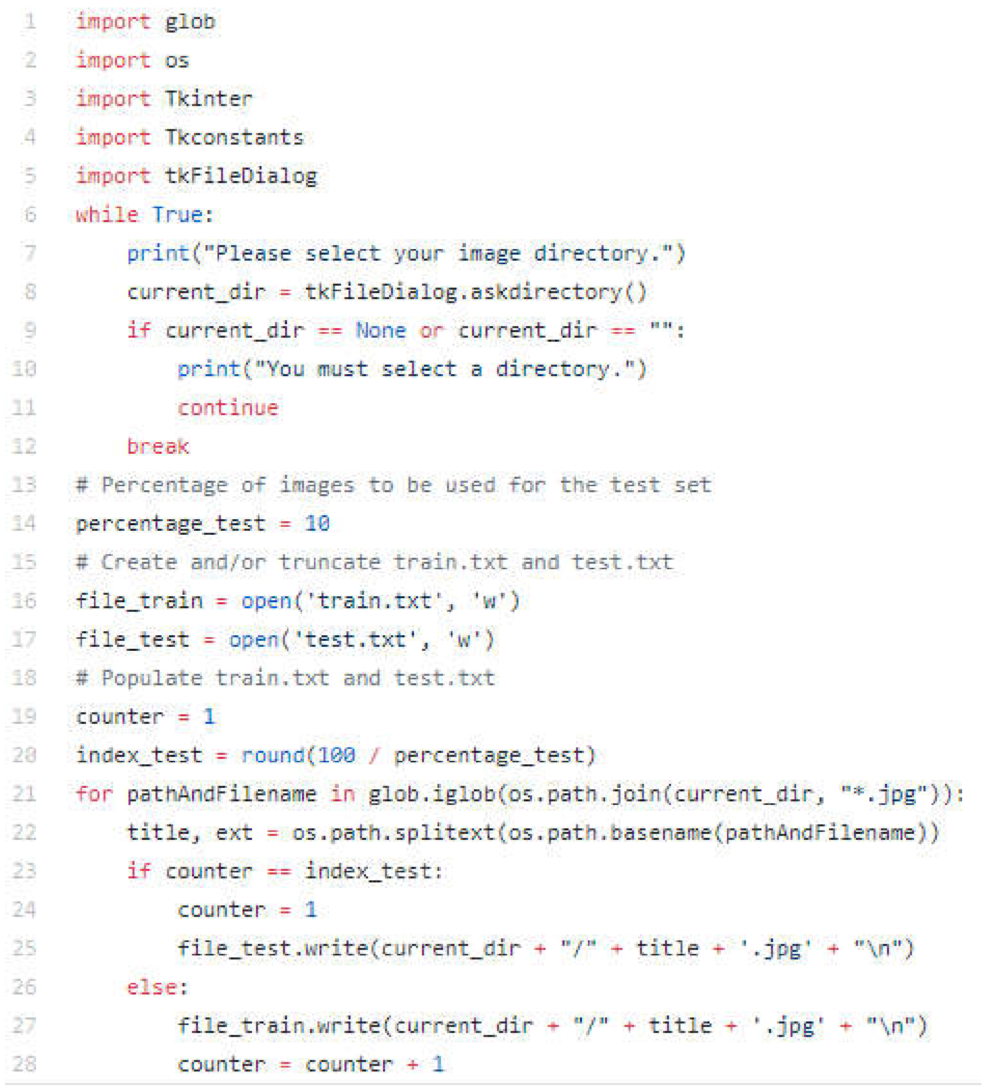


YOLO v3 network Architecture

YOLO applies a single neural network to the full image for both classification and localization.

Creating train.txt and test.txt

1.) save split.py in your working directory.



```python
import glob
import os
import Tkinter
import Tkconstants
import tkFileDialog
while True:
    print("Please select your image directory.")
    current_dir = tkFileDialog.askdirectory()
    if current_dir == None or current_dir == "":
        print("You must select a directory.")
        continue
    break
# Percentage of images to be used for the test set
percentage_test = 10
# Create and/or truncate train.txt and test.txt
file_train = open('train.txt', 'w')
file_test = open('test.txt', 'w')
# Populate train.txt and test.txt
counter = 1
index_test = round(100 / percentage_test)
for pathAndFilename in glob.iglob(os.path.join(current_dir, "*.jpg")):
    title, ext = os.path.splitext(os.path.basename(pathAndFilename))
    if counter == index_test:
        counter = 1
        file_test.write(current_dir + "/" + title + '.jpg' + "\n")
    else:
        file_train.write(current_dir + "/" + title + '.jpg' + "\n")
        counter = counter + 1
```

2.)open Terminal in the directory of split.py and run:

```
python split.py
```

3.) You should see that two new files have been created in your working directory after selecting the directory of where you've saved your images in the pop-up GUI.

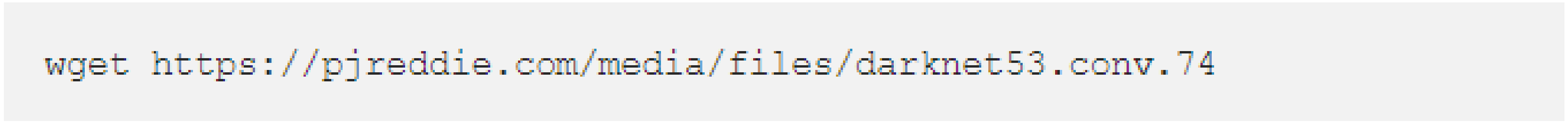
`train.txt`

`test.txt`;

4.) move `train.txt` and `test.txt`; to the Darknet directory.

Downloading pre-trained weights

5.)Save the weights for the convolutional layers to the Darknet directory:



```
wget https://pjreddie.com/media/files/darknet53.conv.74
```

Configuring .data

In your Darknet directory, create `obj.data` and fill in the contents with the following:

```
classes = 1
train = train.txt
valid = test.txt
names = obj.names
backup = backup/
```

Configuring .names

In your Darknet directory, create `obj.names` and fill in the contents with the name of your custom class. For example, the contents of your `obj.names` may simply be:

```
Object name
```

Configuring .cfg

We will now create a `.cfg` file to define our architecture. Let’s get us started.

1.) yolov3-tiny.cfg

Go to the cfg directory under the Darknet directory and make a copy of yolov3-tiny.cfg:

```
cd cfg
cp yolov3-tiny.cfg obj.cfg
```

Open obj.cfg with a text editor and edit as following:
In line 3, set batch=24 to use 24 images for every training step.
In line 4, set subdivisions=8 to subdivide the batch by 8 to speed up the training process and encourage generalization.
In line 127, set filters=(classes + 5)*3, e.g. filters=1.
In line 135, set classes=1, the number of custom classes.
In line 171, set filters=(classes + 5)*3, e.g. filters=1.
In line 177, set classes=1, the number of custom classes.
Then, save the file.

Change configurations to save weights at lower iteration intervals

To save your weights at lower iteration intervals,

1.) Open `detector.c` under `examples` directory in the Darknet directory with a text editor.

2.) edit line 138 to describe when you would like your weight to be saved.

```
if(i%10000==0 || (i < 1000 && i%100 == 0))
```

```
if(i%1000 == 0 || (i < 1000 && i%100 == 0))
```

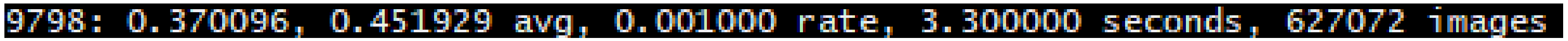
3.) Finally, save the file and run `make`.

Train models

To start training, open Terminal the directory of Darknet and run:

```
./darknet detector train obj.data cfg/obj.cfg darknet53.conv.74
```

Waiting



```
9798: 0.370096, 0.451929 avg, 0.001000 rate, 3.300000 seconds, 627072 images
```

Here, the loss is 0.370096

End of DAY 2

SEE YOU TOMORROW