Click here to download the source code to this post

pyimagesearch
be awesome at building image search engines

YOLO object detection with OpenCV

by **Adrian Rosebrock** on November 12, 2018 in **Deep Learning**, **Object Detection**, **Tutorials**

In this tutorial, you'll learn how to use the YOLO object detector to detect objects in both images and video streams using Deep Learning, OpenCV, and Python.

By applying object detection, you'll not only be able to determine *what* is in an image, but also *where* a given object resides!

We'll start with a brief discussion of the YOLO object detector, including how the object detector works.

From there we'll use OpenCV, Python, and deep learning to:

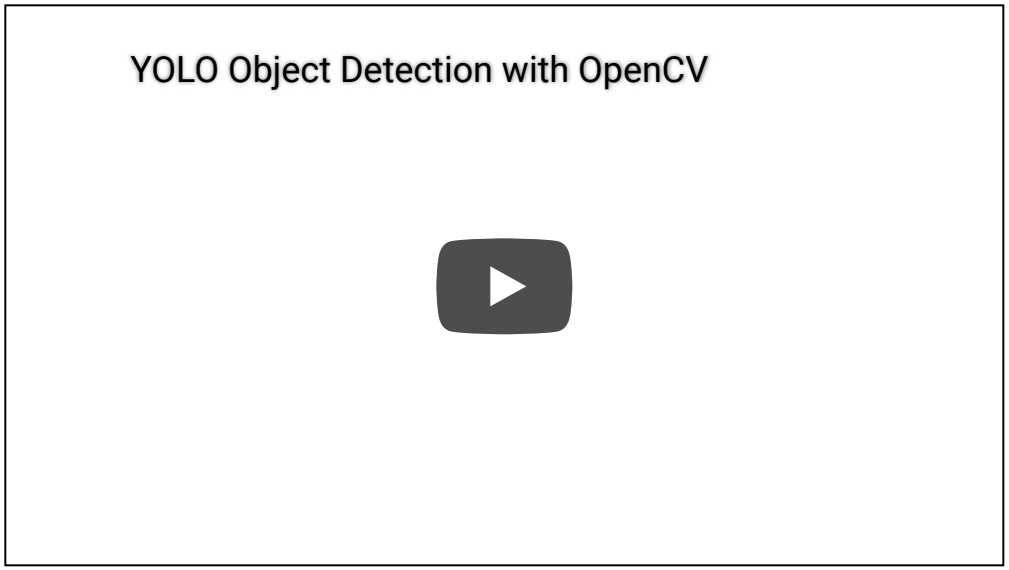
1. Apply the YOLO object detector to images
2. Apply YOLO to video streams

We'll wrap up the tutorial by discussing some of the limitations and drawbacks of the YOLO object detector, including some of my personal tips and suggestions.

To learn how use YOLO for object detection with OpenCV, *just keep reading!*

> **Looking for the source code to this post?**
> **Jump right to the downloads section.**

YOLO Object detection with OpenCV



In the rest of this tutorial we'll:

- Discuss the YOLO object detector model and architecture
- Utilize YOLO to detect objects in images
- Apply YOLO to detect objects in video streams
- Discuss some of the limitations and drawbacks of the YOLO object detector

Let's dive in!

What is the YOLO object detector?

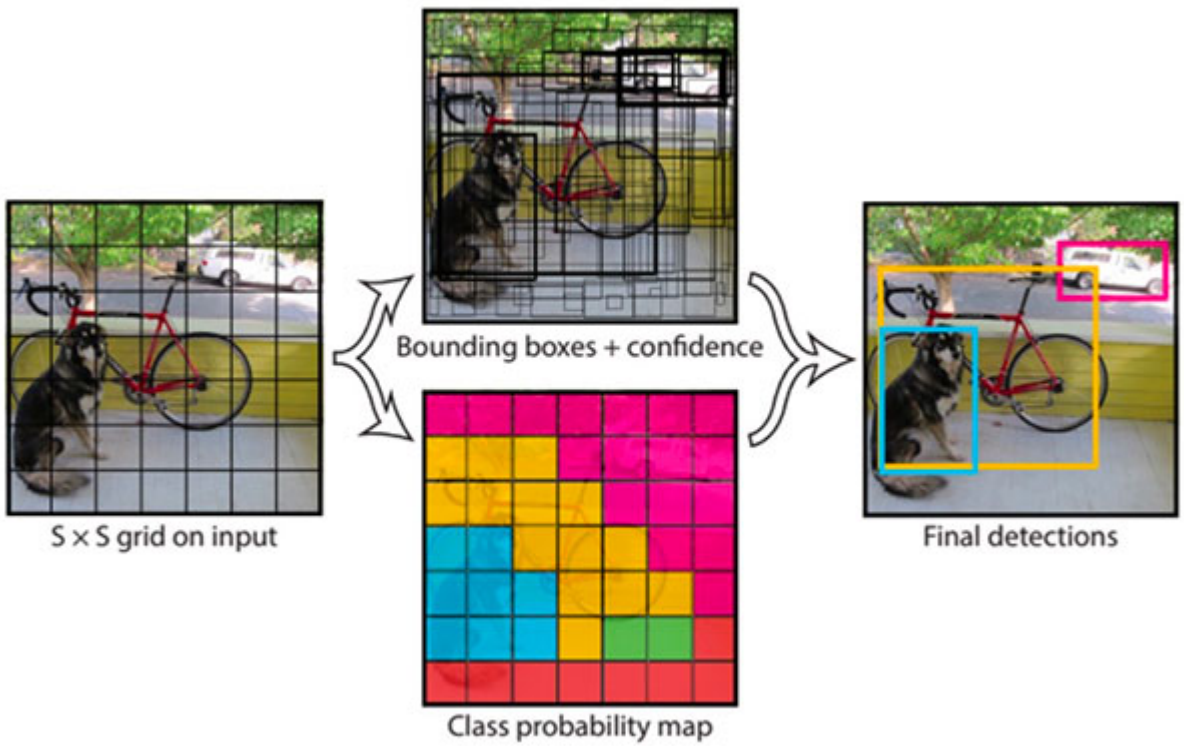


Figure 1: A simplified illustration of the YOLO object detector pipeline (source). We'll use YOLO with OpenCV in this blog post.

When it comes to deep learning-based object detection, there are three primary object detectors you'll encounter:

- R-CNN and their variants, including the original R-CNN, Fast R- CNN, and Faster R-CNN
- Single Shot Detector (SSDs)
- YOLO

R-CNNs are one of the first deep learning-based object detectors and are an example of a ***two-stage detector.***

1. In the first R-CNN publication, *Rich feature hierarchies for accurate object detection and semantic segmentation*, (2013) Girshick et al. proposed an object detector that required an algorithm such as Selective Search (or equivalent) to propose candidate bounding boxes that could contain objects.
2. These regions were then passed into a CNN for classification, ultimately leading to one of the first deep learning-based object detectors.

The problem with the standard R-CNN method was that it was *painfully slow* and not a complete end-to-end object detector.

Girshick et al. published a second paper in 2015, entitled *Fast R- CNN*. The Fast R-CNN algorithm made considerable improvements to the original R-CNN, namely increasing accuracy and reducing the time it took to perform a forward pass; however, the model still relied on an external region proposal algorithm.

It wasn't until Girshick et al.'s follow-up 2015 paper, *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*, that R-CNNs became a true end-to-end deep learning object detector by removing the Selective Search requirement and instead relying on a Region Proposal Network (RPN) that is (1) fully convolutional and (2) can predict the object bounding boxes and "objectness" scores (i.e., a score quantifying how likely it is a region of an image may contain an image). The outputs of the RPNs are then passed into the R-CNN component for final classification and labeling.

While R-CNNs tend to very accurate, the biggest problem with the R-CNN family of networks is their speed — they were incredibly slow, obtaining only 5 FPS on a GPU.

To help increase the speed of deep learning-based object detectors, both Single Shot Detectors (SSDs) and YOLO use a ***one-stage detector strategy***.

These algorithms treat object detection as a regression problem, taking a given input image and simultaneously learning bounding box coordinates and corresponding class label probabilities.

In general, single-stage detectors tend to be less accurate than two-stage detectors *but are significantly faster.*

YOLO is a great example of a single stage detector.

First introduced in 2015 by Redmon et al., their paper, *You Only Look Once: Unified, Real-Time Object Detection*, details an object detector capable of super real-time object detection, obtaining **45 FPS** on a GPU.

Note: *A smaller variant of their model called "Fast YOLO" claims to achieve 155 FPS on a GPU.*

YOLO has gone through a number of different iterations, including *YOLO9000: Better, Faster, Stronger* (i.e., YOLOv2), capable of detecting over 9,000 object detectors.

Redmon and Farhadi are able to achieve such a large number of object detections by performing joint training for both object detection and classification. Using joint training the authors trained YOLO9000 simultaneously on both the ImageNet classification dataset and COCO detection dataset. The result is a YOLO model, called YOLO9000, that can predict detections for object classes that don't have labeled detection data.

While interesting and novel, YOLOv2's performance was a bit underwhelming given the title and abstract of the paper.

On the 156 class version of COCO, YOLO9000 achieved 16% mean Average Precision (mAP), and yes, while YOLO can detect 9,000 separate classes, the accuracy is not quite what we would desire.

Redmon and Farhadi recently published a new YOLO paper, *YOLOv3: An Incremental Improvement* (2018). YOLOv3 is significantly larger than previous models but is, in my opinion, the best one yet out of the YOLO family of object detectors.

We'll be using YOLOv3 in this blog post, in particular, YOLO trained on the COCO dataset.

The COCO dataset consists of 80 labels, including, but not limited to:

- People
- Bicycles
- Cars and trucks
- Airplanes
- Stop signs and fire hydrants
- Animals, including cats, dogs, birds, horses, cows, and sheep, to name a few
- Kitchen and dining objects, such as wine glasses, cups, forks, knives, spoons, etc.
- *…and much more!*

You can find a full list of what YOLO trained on the COCO dataset can detect using this link.

I'll wrap up this section by saying that any academic needs to read Redmon's YOLO papers and tech reports — not only are they novel and insightful they are incredibly entertaining as well.

But seriously, if you do nothing else today **read the YOLOv3 tech report**.

It's only 6 pages and one of those pages is just references/citations.

Furthermore, the tech report is honest in a way that academic papers rarely, if ever, are.

Project structure

Let's take a look at today's project layout. You can use your OS's GUI (Finder for OSX, Nautilus for Ubuntu), but you may find it easier and faster to use the `tree` command in your terminal:

YOLO Object Detection with OpenCV — Shell

```
$ tree
.
├── images
│   ├── baggage_claim.jpg
│   ├── dining_table.jpg
│   ├── living_room.jpg
│   └── soccer.jpg
├── output
│   ├── airport_output.avi
│   ├── car_chase_01_output.avi
│   ├── car_chase_02_output.avi
│   └── overpass_output.avi
├── videos
│   ├── airport.mp4
│   ├── car_chase_01.mp4
│   ├── car_chase_02.mp4
│   └── overpass.mp4
├── yolo-coco
│   ├── coco.names
│   ├── yolov3.cfg
│   └── yolov3.weights
├── yolo.py
└── yolo_video.py

4 directories, 19 files
```

Our project today consists of 4 directories and two Python scripts.

The directories (in order of importance) are:

- `yolo-coco/` : The YOLOv3 object detector pre-trained (on the COCO dataset) model files. These were trained by the Darknet team.
- `images/` : This folder contains four static images which we'll perform object detection on for testing and evaluation purposes.
- `videos/` : After performing object detection with YOLO on images, we'll process videos in real time. This directory contains four sample videos for you to test with.
- `output/` : Output videos that have been processed by YOLO and annotated with bounding boxes and class names can go in this folder.

We're reviewing two Python scripts — `yolo.py` and `yolo_video.py` . The first script is for images and then we'll take what we learn and apply it to video in the second script.

Are you ready?

YOLO object detection in images

Let's get started applying the YOLO object detector to images!

Open up the `yolo.py` file in your project and insert the following code:

YOLO Object Detection with OpenCV — Python

```python
# import the necessary packages
import numpy as np
import argparse
import time
import cv2
import os

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
    help="path to input image")
ap.add_argument("-y", "--yolo", required=True,
    help="base path to YOLO directory")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
    help="minimum probability to filter weak detections")
ap.add_argument("-t", "--threshold", type=float, default=0.3,
    help="threshold when applying non-maxima suppression")
args = vars(ap.parse_args())
```

All you need installed for this script OpenCV 3.4.2+ with Python bindings. You can find my OpenCV installation tutorials here, just keep in mind that OpenCV 4 is in beta right now — you may run into issues installing or running certain scripts since it's not an official release. For the time being I recommend going for OpenCV 3.4.2+. You can actually be up and running in less than 5 minutes with pip as well.

First, we import our required packages — as long as OpenCV and NumPy are installed, your interpreter will breeze past these lines.

Now let's parse four command line arguments. Command line arguments are processed at runtime and allow us to change the inputs to our script from the terminal. If you aren't familiar with them, I encourage you to read more in my previous tutorial. Our command line arguments include:

- `--image` : The path to the input image. We'll detect objects in this image using YOLO.
- `--yolo` : The base path to the YOLO directory. Our script will then load the required YOLO files in order to perform object detection on the image.

- `--confidence` : Minimum probability to filter weak detections. I've given this a default value of 50% (`0.5`), but you should feel free to experiment with this value.
- `--threshold` : This is our non-maxima suppression threshold with a default value of `0.3` . You can read more about non-maxima suppression here.

After parsing, the `args` variable is now a dictionary containing the key-value pairs for the command line arguments. You'll see `args` a number of times in the rest of this script.

Let's load our class labels and set random colors for each:

YOLO Object Detection with OpenCV — Python

```
20 # load the COCO class labels our YOLO model was trained on
21 labelsPath = os.path.sep.join([args["yolo"], "coco.names"])
22 LABELS = open(labelsPath).read().strip().split("\n")
23
24 # initialize a list of colors to represent each possible class label
25 np.random.seed(42)
26 COLORS = np.random.randint(0, 255, size=(len(LABELS), 3),
27     dtype="uint8")
```

Here we load all of our class `LABELS` (notice the first command line argument, `args["yolo"]` being used) on **Lines 21 and 22**. Random `COLORS` are then assigned to each label on **Lines 25-27**.

Let's derive the paths to the YOLO weights and configuration files followed by loading YOLO from disk:

YOLO Object Detection with OpenCV — Python

```
29 # derive the paths to the YOLO weights and model configuration
30 weightsPath = os.path.sep.join([args["yolo"], "yolov3.weights"])
31 configPath = os.path.sep.join([args["yolo"], "yolov3.cfg"])
32
33 # load our YOLO object detector trained on COCO dataset (80 classes)
34 print("[INFO] loading YOLO from disk...")
35 net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)
```

To load YOLO from disk on **Line 35**, we'll take advantage of OpenCV's DNN function called `cv2.dnn.readNetFromDarknet` . This function requires both a `configPath` and `weightsPath` which are established via command line arguments on **Lines 30 and 31**.

I cannot stress this enough: you'll need at least OpenCV 3.4.2 to run this code as it has the updated `dnn` module required to load YOLO.

Let's load the image and send it through the network:

YOLO Object Detection with OpenCV — Python

```
37 # load our input image and grab its spatial dimensions
38 image = cv2.imread(args["image"])
39 (H, W) = image.shape[:2]
40
41 # determine only the *output* layer names that we need from YOLO
42 ln = net.getLayerNames()
43 ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]
44
45 # construct a blob from the input image and then perform a forward
46 # pass of the YOLO object detector, giving us our bounding boxes and
47 # associated probabilities
48 blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (416, 416),
49     swapRB=True, crop=False)
50 net.setInput(blob)
51 start = time.time()
52 layerOutputs = net.forward(ln)
53 end = time.time()
54
55 # show timing information on YOLO
56 print("[INFO] YOLO took {:.6f} seconds".format(end - start))
```

In this block we:

- Load the input `image` and extract its dimensions (**Lines 38 and 39**).
- Determine the output layer names from the YOLO model (**Lines 42 and 43**).
- Construct a `blob` from the image (**Lines 48 and 49**). Are you confused about what a blob is or what the `cv2.dnn.blobFromImage` does? Give this blog post a read.

Now that our blob is prepared, we'll

- Perform a forward pass through our YOLO network (**Lines 50 and 52**)
- Show the inference time for YOLO (**Line 56**)

What good is object detection unless we visualize our results? Let's take steps now to filter and visualize our results.

But first, let's initialize some lists we'll need in the process of doing so:

YOLO Object Detection with OpenCV — Python

```
58 # initialize our lists of detected bounding boxes, confidences, and
59 # class IDs, respectively
60 boxes = []
61 confidences = []
```

```
62 classIDs = []
```

These lists include:

- `boxes` : Our bounding boxes around the object.
- `confidences` : The confidence value that YOLO assigns to an object. Lower confidence values indicate that the object might not be what the network thinks it is. Remember from our command line arguments above that we'll filter out objects that don't meet the `0.5` threshold.
- `classIDs` : The detected object's class label.

Let's begin populating these lists with data from our YOLO `layerOutputs` :

YOLO Object Detection with OpenCV — Python

```
64 # loop over each of the layer outputs
65 for output in layerOutputs:
66     # loop over each of the detections
67     for detection in output:
68         # extract the class ID and confidence (i.e., probability) of
69         # the current object detection
70         scores = detection[5:]
71         classID = np.argmax(scores)
72         confidence = scores[classID]
73 
74         # filter out weak predictions by ensuring the detected
75         # probability is greater than the minimum probability
76         if confidence > args["confidence"]:
77             # scale the bounding box coordinates back relative to the
78             # size of the image, keeping in mind that YOLO actually
79             # returns the center (x, y)-coordinates of the bounding
80             # box followed by the boxes' width and height
81             box = detection[0:4] * np.array([W, H, W, H])
82             (centerX, centerY, width, height) = box.astype("int")
83 
84             # use the center (x, y)-coordinates to derive the top and
85             # and left corner of the bounding box
86             x = int(centerX - (width / 2))
87             y = int(centerY - (height / 2))
88 
89             # update our list of bounding box coordinates, confidences,
90             # and class IDs
91             boxes.append([x, y, int(width), int(height)])
92             confidences.append(float(confidence))
93             classIDs.append(classID)
```

There's a lot here in this code block — let's break it down.

In this block, we:

- Loop over each of the `layerOutputs` (beginning on **Line 65**).
- Loop over each `detection` in `output` (a nested loop beginning on **Line 67**).
- Extract the `classID` and `confidence` (**Lines 70-72**).
- Use the `confidence` to filter out weak detections (**Line 76**).

Now that we've filtered out unwanted detections, we're going to:

- Scale bounding box coordinates so we can display them properly on our original image (**Line 81**).
- Extract coordinates and dimensions of the bounding box (**Line 82**). YOLO returns bounding box coordinates in the form: `(centerX, centerY, width, and height)` .
- Use this information to derive the top-left *(x, y)*-coordinates of the bounding box (**Lines 86 and 87**).
- Update the `boxes` , `confidences` , and `classIDs` lists (**Lines 91-93**).

With this data, we're now going to apply what is called "non-maxima suppression":

YOLO Object Detection with OpenCV — Python

```
95 # apply non-maxima suppression to suppress weak, overlapping bounding
96 # boxes
97 idxs = cv2.dnn.NMSBoxes(boxes, confidences, args["confidence"],
98     args["threshold"])
```

YOLO does not apply non-maxima suppression for us, so we need to explicitly apply it.

Applying non-maxima suppression suppresses significantly overlapping bounding boxes, keeping only the most confident ones.

NMS also ensures that we do not have any redundant or extraneous bounding boxes.

Taking advantage of OpenCV's built-in DNN module implementation of NMS, we perform non-maxima suppression on **Lines 97 and 98**. All that is required is that we submit our bounding `boxes` , `confidences` , as well as both our confidence threshold and NMS threshold.

If you've been reading this blog, you might be wondering why we didn't use my imutils implementation of NMS. The primary reason is that the `NMSBoxes` function is now working in OpenCV. Previously it failed for some inputs and resulted in an error message. Now that the `NMSBoxes` function is working, we can use it in our own scripts.

Let's draw the boxes and class text on the image!

YOLO Object Detection with OpenCV — Python

```python
# ensure at least one detection exists
if len(idxs) > 0:
    # loop over the indexes we are keeping
    for i in idxs.flatten():
        # extract the bounding box coordinates
        (x, y) = (boxes[i][0], boxes[i][1])
        (w, h) = (boxes[i][2], boxes[i][3])

        # draw a bounding box rectangle and label on the image
        color = [int(c) for c in COLORS[classIDs[i]]]
        cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)
        text = "{}: {:.4f}".format(LABELS[classIDs[i]], confidences[i])
        cv2.putText(image, text, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX,
            0.5, color, 2)

# show the output image
cv2.imshow("Image", image)
cv2.waitKey(0)
```

Assuming at least one detection exists (**Line 101**), we proceed to loop over `idxs` determined by non-maxima suppression.

Then, we simply draw the bounding box and text on `image` using our random class colors (**Lines 105-113**).

Finally, we display our resulting image until the user presses any key on their keyboard (ensuring the window opened by OpenCV is selected and focused).

To follow along with this guide, make sure you use the ***"Downloads"*** section of this tutorial to download the source code, YOLO model, and example images.

From there, open up a terminal and execute the following command:

YOLO Object Detection with OpenCV — Shell

```
$ python yolo.py --image images/baggage_claim.jpg --yolo yolo-coco
[INFO] loading YOLO from disk...
[INFO] YOLO took 0.347815 seconds
```

Figure 2: YOLO with OpenCV is used to detect people and baggage in an airport.

Here you can see that YOLO has not only detected each person in the input image, but also the suitcases as well!

Furthermore, if you take a look at the right corner of the image you'll see that YOLO has also detected the handbag on the lady's shoulder.

Let's try another example:

YOLO Object Detection with OpenCV — Shell

```
$ python yolo.py --image images/living_room.jpg --yolo yolo-coco
[INFO] loading YOLO from disk...
```

```
[INFO] YOLO took 0.340221 seconds
```

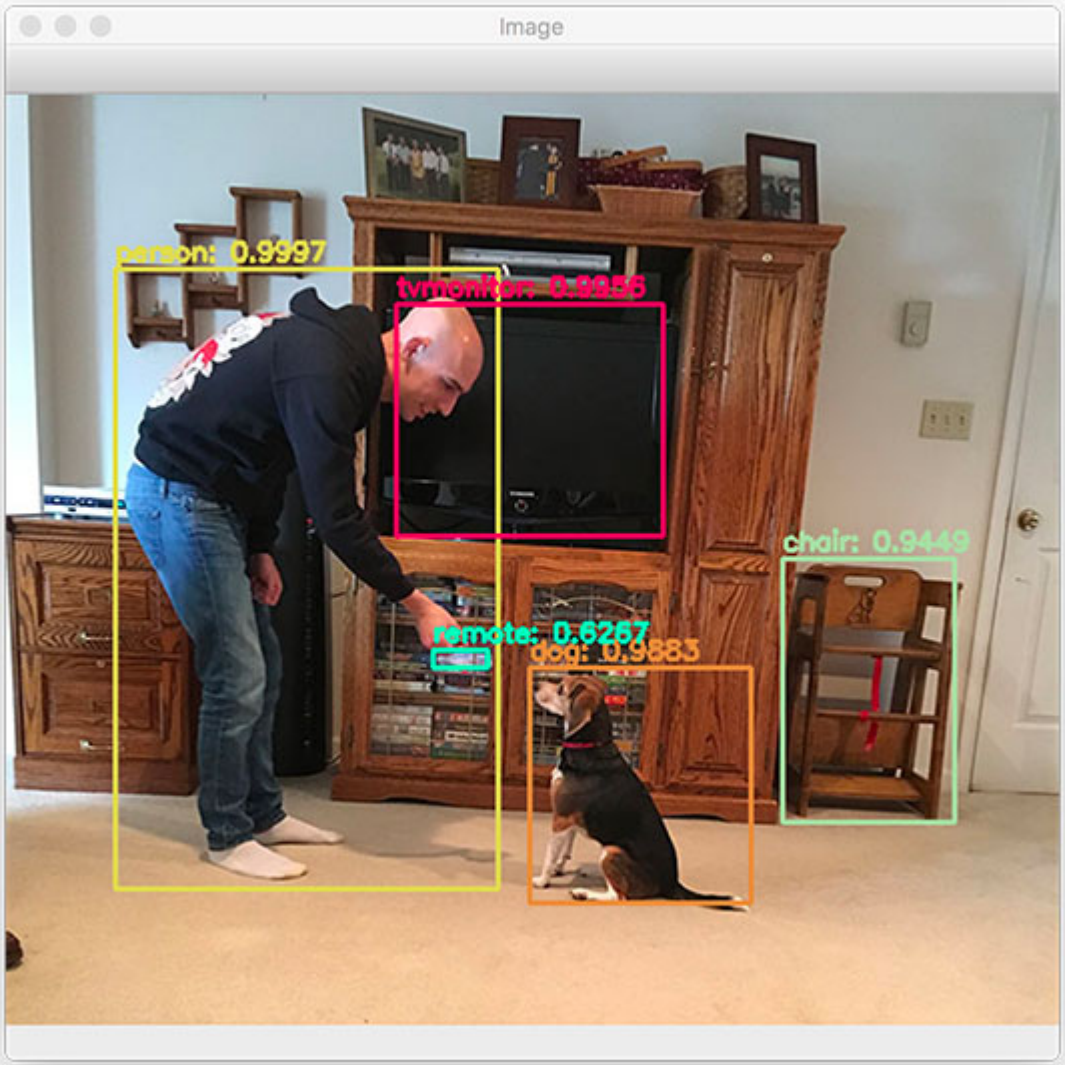


Figure 3: YOLO object detection with OpenCV is used to detect a person, dog, TV, and chair. The remote is a false-positive detection but looking at the ROI you could imagine that the area does share resemblances to a remote.

The image above contains a person (myself) and a dog (Jemma, the family beagle).

YOLO also detects the TV monitor and a chair as well. I'm particularly impressed that YOLO was able to detect the chair given that it's handmade, old fashioned "baby high chair".

Interestingly, YOLO thinks there is a "remote" in my hand. It's actually not a remote — it's the reflection of glass on a VHS tape; however, if you stare at the region it actually does look like it could be a remote.

The following example image demonstrates a limitation and weakness of the YOLO object detector:

YOLO Object Detection with OpenCV — Shell

```
$ python yolo.py --image images/dining_table.jpg --yolo yolo-coco
[INFO] loading YOLO from disk...
[INFO] YOLO took 0.362369 seconds
```

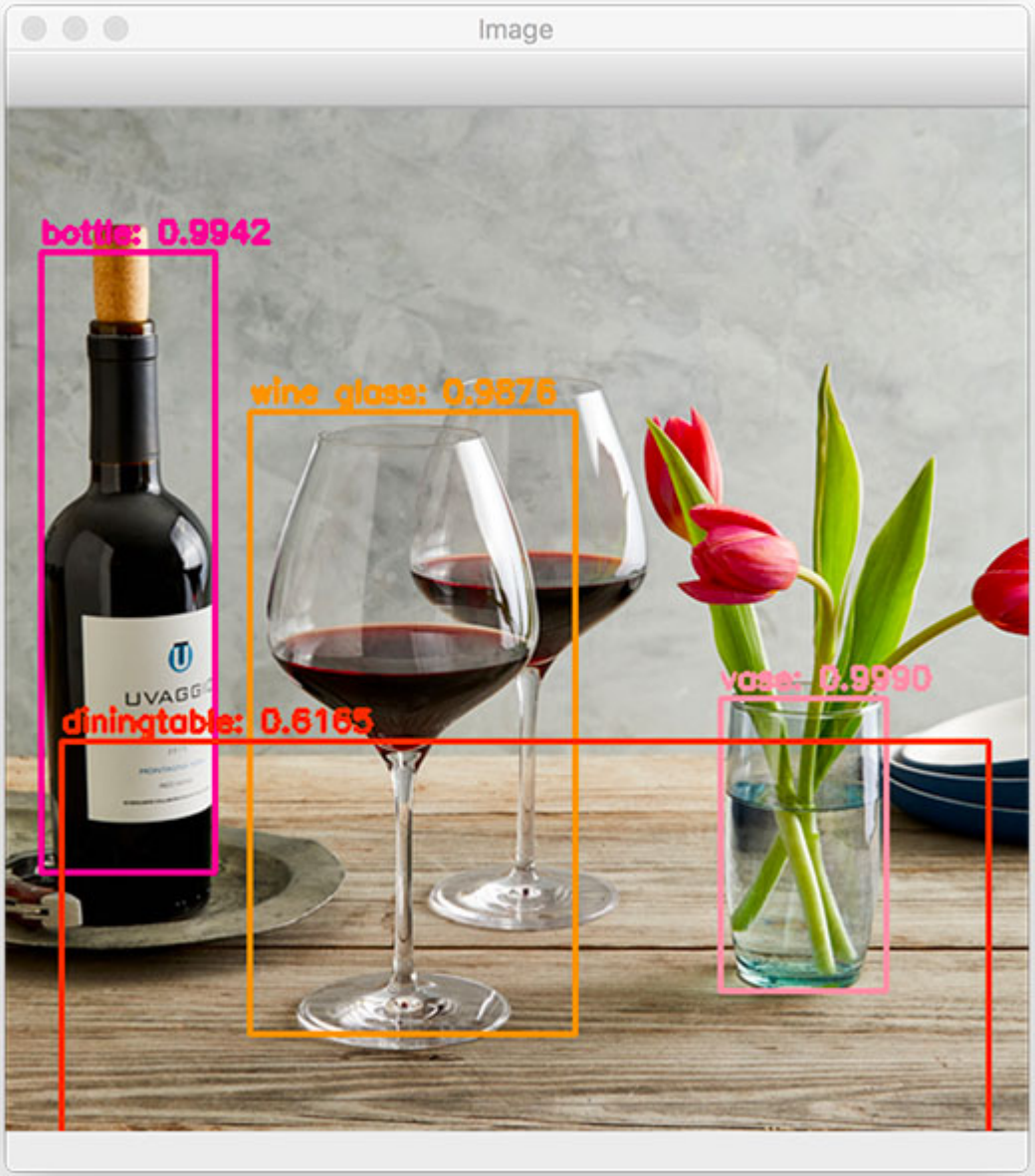


Figure 4: YOLO and OpenCV are used for object detection of a dining room table.

While both the wine bottle, dining table, and vase are correctly detected by YOLO, only one of the two wine glasses is properly detected.

We discuss why YOLO struggles with objects close together in the *"Limitations and drawbacks of the YOLO object detector"* section below.

Let's try one final image:

YOLO Object Detection with OpenCV — Shell

```
$ python yolo.py --image images/soccer.jpg --yolo yolo-coco
[INFO] loading YOLO from disk...
[INFO] YOLO took 0.345656 seconds
```

Figure 5: Soccer players and a soccer ball are detected with OpenCV using the YOLO object detector.

YOLO is able to correctly detect each of the players on the pitch, including the soccer ball itself. Notice the person in the background who is detected despite the area being highly blurred and partially obscured.

YOLO object detection in video streams

Now that we've learned how to apply the YOLO object detector to single images, let's also utilize YOLO to perform object detection in input video files as well.

Open up the `yolo_video.py` file and insert the following code:

YOLO Object Detection with OpenCV — Python

```python
# import the necessary packages
import numpy as np
import argparse
import imutils
import time
import cv2
import os

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--input", required=True,
    help="path to input video")
ap.add_argument("-o", "--output", required=True,
    help="path to output video")
ap.add_argument("-y", "--yolo", required=True,
    help="base path to YOLO directory")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
    help="minimum probability to filter weak detections")
ap.add_argument("-t", "--threshold", type=float, default=0.3,
    help="threshold when applyong non-maxima suppression")
args = vars(ap.parse_args())
```

We begin with our imports and command line arguments.

Notice that this script doesn't have the `--image` argument as before. To take its place, we now have two video-related arguments:

- `--input` : The path to the *input* video file.
- `--output` : Our path to the *output* video file.

Given these arguments, you can now use videos that you record of scenes with your smartphone or videos you find online. You can then process the video file producing an annotated output video. Of course if you want to use your webcam to process a live video stream, that is

possible too. Just find examples on PyImageSearch where the `VideoStream` class from `imutils.video` is utilized and make some minor changes.

Moving on, the next block is *identical* to the block from the YOLO image processing script:

YOLO Object Detection with OpenCV — Python

```
# load the COCO class labels our YOLO model was trained on
labelsPath = os.path.sep.join([args["yolo"], "coco.names"])
LABELS = open(labelsPath).read().strip().split("\n")

# initialize a list of colors to represent each possible class label
np.random.seed(42)
COLORS = np.random.randint(0, 255, size=(len(LABELS), 3),
    dtype="uint8")

# derive the paths to the YOLO weights and model configuration
weightsPath = os.path.sep.join([args["yolo"], "yolov3.weights"])
configPath = os.path.sep.join([args["yolo"], "yolov3.cfg"])

# load our YOLO object detector trained on COCO dataset (80 classes)
# and determine only the *output* layer names that we need from YOLO
print("[INFO] loading YOLO from disk...")
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)
ln = net.getLayerNames()
ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]
```

Here we load labels and generate colors followed by loading our YOLO model and determining output layer names.

Next, we'll take care of some video-specific tasks:

YOLO Object Detection with OpenCV — Python

```
# initialize the video stream, pointer to output video file, and
# frame dimensions
vs = cv2.VideoCapture(args["input"])
writer = None
(W, H) = (None, None)

# try to determine the total number of frames in the video file
try:
    prop = cv2.cv.CV_CAP_PROP_FRAME_COUNT if imutils.is_cv2() \
        else cv2.CAP_PROP_FRAME_COUNT
    total = int(vs.get(prop))
    print("[INFO] {} total frames in video".format(total))

# an error occurred while trying to determine the total
```

```
# number of frames in the video file
except:
    print("[INFO] could not determine # of frames in video")
    print("[INFO] no approx. completion time can be provided")
    total = -1
```

In this block, we:

- Open a file pointer to the video file for reading frames in the upcoming loop (**Line 45**).
- Initialize our video `writer` and frame dimensions (**Lines 46 and 47**).
- Try to determine the `total` number of frames in the video file so we can estimate how long processing the entire video will take (**Lines 50-61**).

Now we're ready to start processing frames one by one:

YOLO Object Detection with OpenCV — Python

```
# loop over frames from the video file stream
while True:
    # read the next frame from the file
    (grabbed, frame) = vs.read()

    # if the frame was not grabbed, then we have reached the end
    # of the stream
    if not grabbed:
        break

    # if the frame dimensions are empty, grab them
    if W is None or H is None:
        (H, W) = frame.shape[:2]
```

We define a `while` loop (**Line 64**) and then we grab our first frame (**Line 66**).

We make a check to see if it is the last frame of the video. If so we need to `break` from the `while` loop (**Lines 70 and 71**).

Next, we grab the frame dimensions if they haven't been grabbed yet (**Lines 74 and 75**).

Next, let's perform a forward pass of YOLO, using our current `frame` as the input:

YOLO Object Detection with OpenCV — Python

```
    # construct a blob from the input frame and then perform a forward
    # pass of the YOLO object detector, giving us our bounding boxes
    # and associated probabilities
    blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416),
```

```
81          swapRB=True, crop=False)
82      net.setInput(blob)
83      start = time.time()
84      layerOutputs = net.forward(ln)
85      end = time.time()
86
87      # initialize our lists of detected bounding boxes, confidences,
88      # and class IDs, respectively
89      boxes = []
90      confidences = []
91      classIDs = []
```

Here we construct a `blob` and pass it through the network, obtaining predictions. I've surrounded the forward pass operation with time stamps so we can calculate the elapsed time to make predictions on one frame — this will help us estimate the time needed to process the entire video.

We'll then go ahead and initialize the same three lists we used in our previous script: `boxes` , `confidences` , and `classIDs` .

This next block is, again, *identical* to our previous script:

YOLO Object Detection with OpenCV — Python

```
93      # loop over each of the layer outputs
94      for output in layerOutputs:
95          # loop over each of the detections
96          for detection in output:
97              # extract the class ID and confidence (i.e., probability)
98              # of the current object detection
99              scores = detection[5:]
100             classID = np.argmax(scores)
101             confidence = scores[classID]
102
103             # filter out weak predictions by ensuring the detected
104             # probability is greater than the minimum probability
105             if confidence > args["confidence"]:
106                 # scale the bounding box coordinates back relative to
107                 # the size of the image, keeping in mind that YOLO
108                 # actually returns the center (x, y)-coordinates of
109                 # the bounding box followed by the boxes' width and
110                 # height
111                 box = detection[0:4] * np.array([W, H, W, H])
112                 (centerX, centerY, width, height) = box.astype("int")
113
114                 # use the center (x, y)-coordinates to derive the top
115                 # and and left corner of the bounding box
116                 x = int(centerX - (width / 2))
117                 y = int(centerY - (height / 2))
```

```
118
119                 # update our list of bounding box coordinates,
120                 # confidences, and class IDs
121                 boxes.append([x, y, int(width), int(height)])
122                 confidences.append(float(confidence))
123                 classIDs.append(classID)
```

In this code block, we:

- Loop over output layers and detections (**Lines 94-96**).
- Extract the `classID` and filter out weak predictions (**Lines 99-105**).
- Compute bounding box coordinates (**Lines 111-117**).
- Update our respective lists (**Lines 121-123**).

Next, we'll apply non-maxima suppression and begin to proceed to annotate the frame:

YOLO Object Detection with OpenCV — Python

```
125     # apply non-maxima suppression to suppress weak, overlapping
126     # bounding boxes
127     idxs = cv2.dnn.NMSBoxes(boxes, confidences, args["confidence"],
128         args["threshold"])
129
130     # ensure at least one detection exists
131     if len(idxs) > 0:
132         # loop over the indexes we are keeping
133         for i in idxs.flatten():
134             # extract the bounding box coordinates
135             (x, y) = (boxes[i][0], boxes[i][1])
136             (w, h) = (boxes[i][2], boxes[i][3])
137
138             # draw a bounding box rectangle and label on the frame
139             color = [int(c) for c in COLORS[classIDs[i]]]
140             cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
141             text = "{}: {:.4f}".format(LABELS[classIDs[i]],
142                 confidences[i])
143             cv2.putText(frame, text, (x, y - 5),
144                 cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
```

You should recognize these lines as well. Here we:

- Apply NMS using the `cv2.dnn.NMSBoxes` function (**Lines 127 and 128**) to suppress weak, overlapping bounding boxes. You can read more about non-maxima suppression here.
- Loop over the `idxs` calculated by NMS and draw the corresponding bounding boxes + labels (**Lines 131-144**).

Let's finish out the script:

YOLO Object Detection with OpenCV — Python

```
146     # check if the video writer is None
147     if writer is None:
148         # initialize our video writer
149         fourcc = cv2.VideoWriter_fourcc(*"MJPG")
150         writer = cv2.VideoWriter(args["output"], fourcc, 30,
151             (frame.shape[1], frame.shape[0]), True)
152 
153         # some information on processing single frame
154         if total > 0:
155             elap = (end - start)
156             print("[INFO] single frame took {:.4f} seconds".format(elap))
157             print("[INFO] estimated total time to finish: {:.4f}".format(
158                 elap * total))
159 
160     # write the output frame to disk
161     writer.write(frame)
162 
163 # release the file pointers
164 print("[INFO] cleaning up...")
165 writer.release()
166 vs.release()
```

To wrap up, we simply:

- Initialize our video `writer` if necessary (**Lines 147-151**). The `writer` will be initialized on the first iteration of the loop.
- Print out our estimates of how long it will take to process the video (**Lines 154-158**).
- Write the `frame` to the output video file (**Line 161**).
- Cleanup and release pointers (**Lines 165 and 166**).

To apply YOLO object detection to video streams, make sure you use the ***"Downloads"*** section of this blog post to download the source, YOLO object detector, and example videos.

From there, open up a terminal and execute the following command:

YOLO Object Detection with OpenCV — Shell

```
$ python yolo_video.py --input videos/car_chase_01.mp4 \
    --output output/car_chase_01.avi --yolo yolo-coco
[INFO] loading YOLO from disk...
[INFO] 583 total frames in video
```

```
[INFO] single frame took 0.3500 seconds
[INFO] estimated total time to finish: 204.0238
[INFO] cleaning up...
```

Figure 6: YOLO deep learning object detection applied to a car crash video.

Above you can see a GIF excerpt from a car chase video I found on YouTube.

In the video/GIF, you can see not only the vehicles being detected, but people, as well as the traffic lights, are detected too!

The YOLO object detector is performing quite well here. Let's try a different video clip from the same car chase video:

YOLO Object Detection with OpenCV — Shell

```
$ python yolo_video.py --input videos/car_chase_02.mp4 \
    --output output/car_chase_02.avi --yolo yolo-coco
[INFO] loading YOLO from disk...
[INFO] 3132 total frames in video
[INFO] single frame took 0.3455 seconds
[INFO] estimated total time to finish: 1082.0806
[INFO] cleaning up...
```

Figure 7: In this video of a suspect on the run, we have used OpenCV and YOLO object detection to find the person.

The suspect has now fled the car and is running across a parking lot.

YOLO is once again able to detect people.

At one point the suspect is actually able to make it back to their car and continue the chase — let's see how YOLO performs there as well:

YOLO Object Detection with OpenCV — Shell

```
$ python yolo_video.py --input videos/car_chase_03.mp4 \
    --output output/car_chase_03.avi --yolo yolo-coco
[INFO] loading YOLO from disk...
[INFO] 749 total frames in video
[INFO] single frame took 0.3442 seconds
[INFO] estimated total time to finish: 257.8418
[INFO] cleaning up...
```

Figure 8: YOLO is a fast deep learning object detector capable of being used in real time video provided a GPU is utilized.

Note: *This video was simply too large for me to include in the* ***"Downloads"****. You may download the video from YouTube here.*

As a final example, let's see how we may use YOLO as a starting point to building a traffic counter:

YOLO Object Detection with OpenCV — Shell

```
$ python yolo_video.py --input videos/overpass.mp4 \
    --output output/overpass.avi --yolo yolo-coco
[INFO] loading YOLO from disk...
[INFO] 812 total frames in video
[INFO] single frame took 0.3534 seconds
[INFO] estimated total time to finish: 286.9583
[INFO] cleaning up...
```

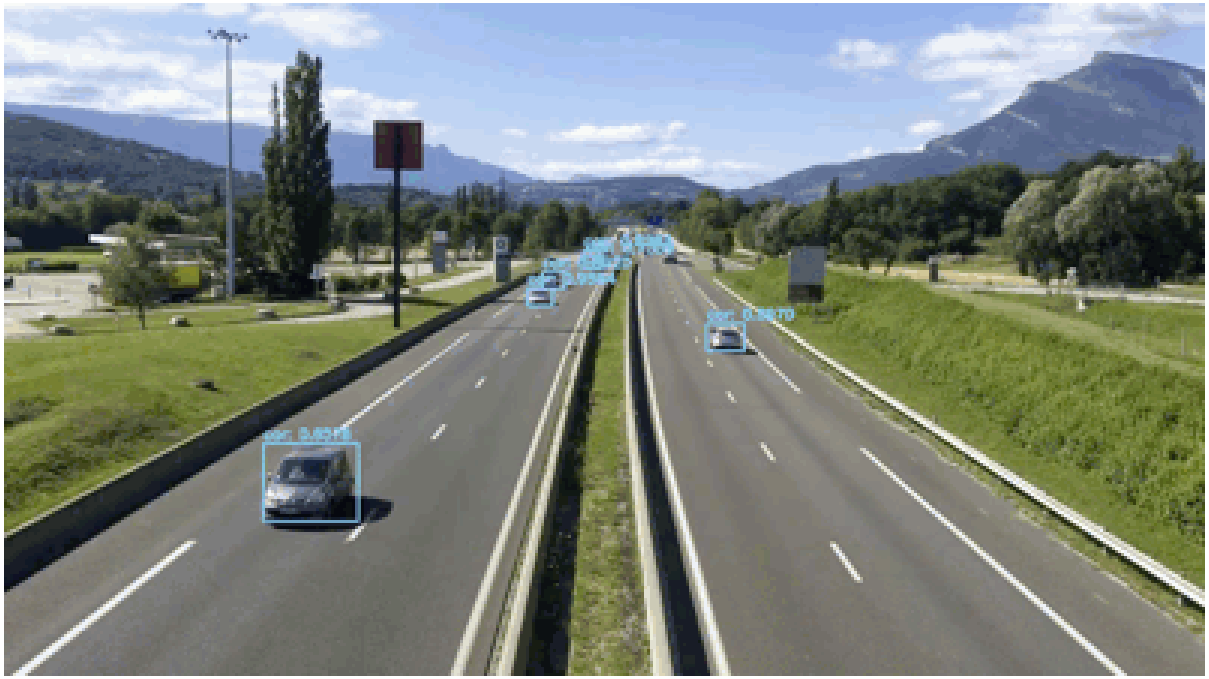
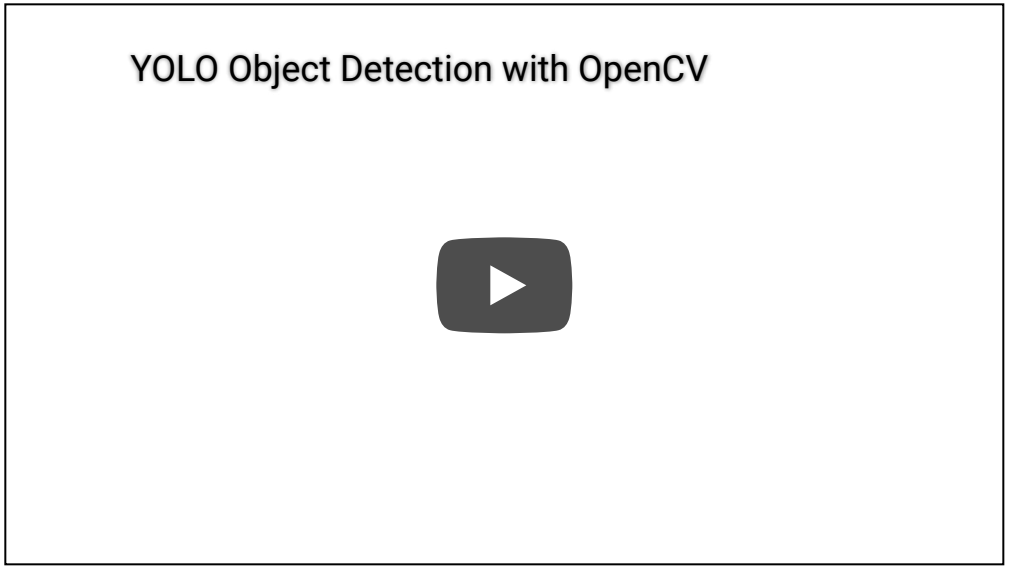


Figure 9: A video of traffic going under an overpass demonstrates that YOLO and OpenCV can be used to detect cars accurately and quickly.

I've put together a full video of YOLO object detection examples below:



Credits for video and audio:

- Car chase video posted on YouTube by Quaker Oats.
- Overpass video on YouTube by Vlad Kiraly.
- "White Crow" on the FreeMusicArchive by XTaKeRuX.

Limitations and drawbacks of the YOLO object detector

Arguably the largest limitation and drawback of the YOLO object detector is that:

1. It does not always handle small objects well
2. It *especially* does not handle objects grouped close together

The reason for this limitation is due to the YOLO algorithm itself:

- The YOLO object detector divides an input image into an *SxS* grid where each cell in the grid predicts only a single object.
- If there exist multiple, small objects in a single cell then YOLO will be unable to detect them, ultimately leading to missed object detections.

Therefore, if you know your dataset consists of many small objects grouped close together then you should not use the YOLO object detector.

In terms of small objects, Faster R-CNN tends to work the best; however, it's also the slowest.

SSDs can also be used here; however, SSDs can also struggle with smaller objects (but not as much as YOLO).

SSDs often give a nice tradeoff in terms of speed and accuracy as well.

It's also worth noting that YOLO ran slower than SSDs in this tutorial. In my previous tutorial on OpenCV object detection we utilized an SSD — a single forward pass of the SSD took ~0.03 seconds.

However, from this tutorial, we know that a forward pass of the YOLO object detector took ~0.3 seconds, *approximately an order of magnitude slower!*

If you're using the pre-trained deep learning object detectors OpenCV supplies you may want to consider using SSDs over YOLO. From my personal experience, I've rarely encountered situations where I needed to use YOLO over SSDs:

- I have found SSDs much easier to train and their performance in terms of accuracy almost always outperforms YOLO (at least for the datasets I've worked with).
- YOLO may have excellent results on the COCO dataset; however, I have not found that same level of accuracy for my own tasks.

I, therefore, tend to use the following guidelines when picking an object detector for a given problem:

1. If I know I need to detect small objects and speed is not a concern, I tend to use Faster R-CNN.
2. If speed is absolutely paramount, I use YOLO.
3. If I need a middle ground, I tend to go with SSDs.

In most of my situations I end up using SSDs or RetinaNet — both are a great balance between the YOLO/Faster R-CNN.

Want to train your own deep learning object detectors?

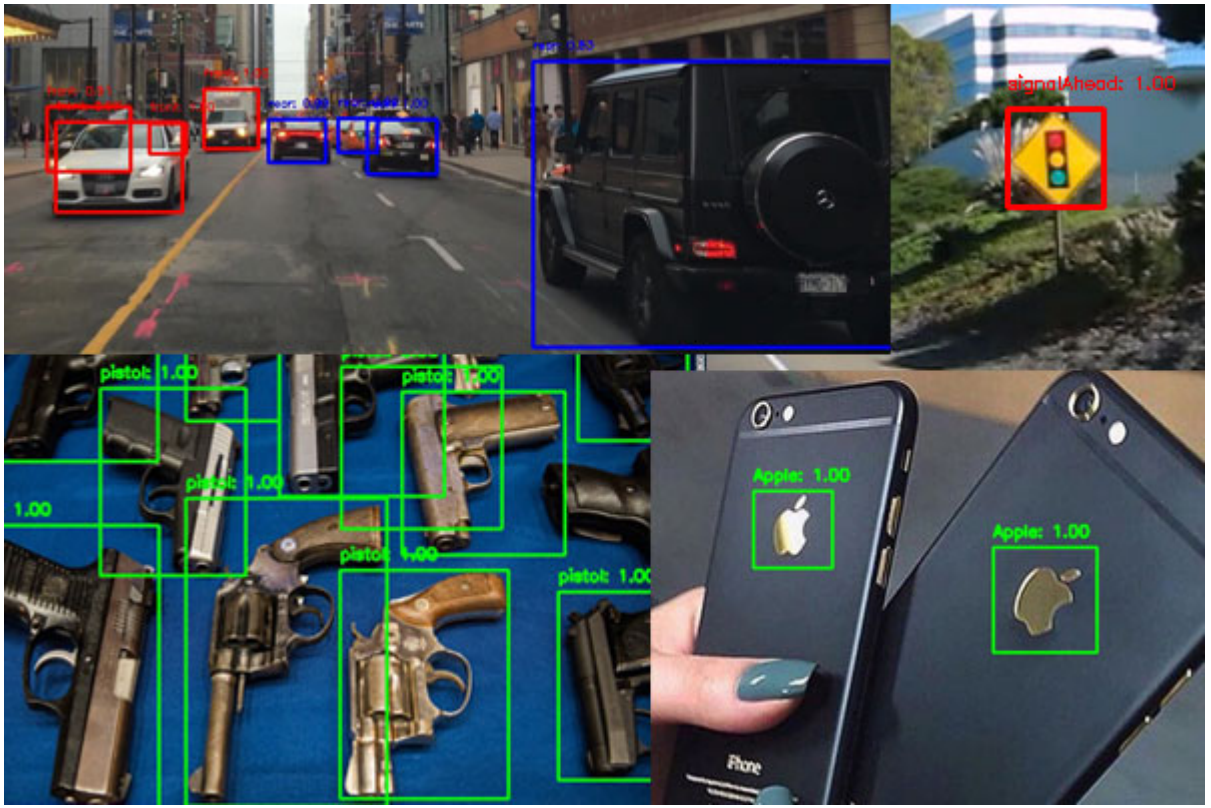


Figure 10: In my book, *Deep Learning for Computer Vision with Python*, I cover multiple object detection algorithms including Faster R-CNN, SSDs, and RetinaNet. Inside I will teach you how to create your object

detection image dataset, train the object detector, and make predictions. Not to mention I also cover deep learning fundamentals, best practices, and my personal set of rules of thumb. Grab your copy now so you can start learning new skills.

The YOLO model we used in this tutorial was *pre-trained* on the COCO dataset…

…but what if you wanted to train a deep learning object detector *on your own dataset?*

Inside my book, *Deep Learning for Computer Vision with Python*, I'll teach you how to train **Faster R-CNNs, Single Shot Detectors (SSDs), and RetinaNet to:**

- Detect **logos** in images
- Detect **traffic signs** (ex. stop sign, yield sign, etc.)
- Detect the front and rear views of **vehicles** (useful for building a self-driving car application)
- Detect **weapons** in images and video streams

All object detection chapters in the book include a detailed explanation of *both* the algorithm and code, **ensuring you will be able to successfully train your own object detectors.**

To learn more about my book (and grab your *free* set of sample chapters and table of contents), *just click here.*

Summary

In this tutorial we learned how to perform YOLO object detection using Deep Learning, OpenCV, and Python.

We then briefly discussed the YOLO architecture followed by implementing Python code to:

1. Apply YOLO object detection to single images
2. Apply the YOLO object detector to video streams

On my machine with a 3GHz Intel Xeon W processor, a single forward pass of YOLO took ~0.3 seconds; however, using a Single Shot Detector (SSD) from a previous tutorial, resulted in only 0.03 second detection, *an order of magnitude faster!*

For real-time deep learning-based object detection on your CPU with OpenCV and Python, you may want to consider using the SSD.

If you are interested in training your own deep learning object detectors on your own custom datasets, be sure to refer to my book, *Deep Learning for Computer Vision with Python*, where I provide detailed guides on how to successfully train your own detectors.

I hope you enjoyed today's YOLO object detection tutorial!

To download the source code to today's post, and be notified when future PyImageSearch blog posts are published, *just enter your email address in the form below*.

Downloads:

If you would like to download the code and images used in this post, please enter your email address in the form below. Not only will you get a .zip of the code, I'll also send you a **FREE 17-page Resource Guide on Computer Vision, OpenCV, and Deep Learning.** Inside you'll find my hand-picked tutorials, books, courses, and libraries to help you master CV and DL! Sound good? If so, enter your email address and I'll send you the code immediately!

Email address:

Your email address

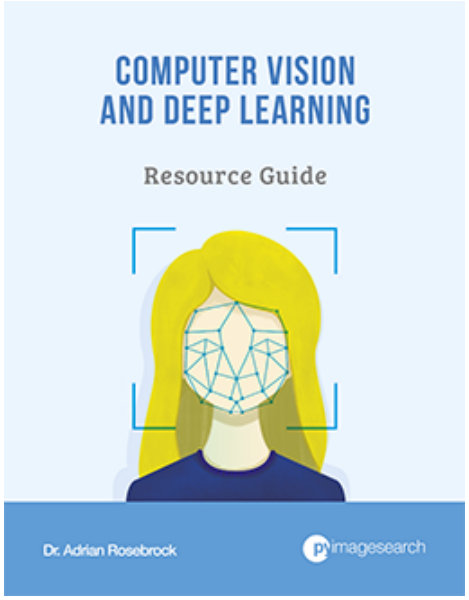
DOWNLOAD THE CODE!

Resource Guide (it's totally free).

Enter your email address below to get my **free 17-page Computer Vision, OpenCV, and Deep Learning Resource Guide PDF**. Inside you'll find my hand-picked tutorials, books, courses, and Python libraries to help you master computer vision and deep learning!

Your email address

DOWNLOAD THE GUIDE!



cnn, **convolutional neural network**, **machine learning**, **object detection**, **yolo**

‹ Creating GIFs with OpenCV

Mask R-CNN with OpenCV ›

270 Responses to *YOLO object detection with OpenCV*

Naser November 12, 2018 at 11:25 am #

REPLY

Great tutarial
Thanks adrian.
Can you make a tutarial and explain in that how can train yolo on our custom dataset??
Thank you.

Adrian Rosebrock November 12, 2018 at 12:10 pm #

REPLY

I actually cover how to train your own custom object detectors inside Deep Learning for Computer Vision with Python. I would suggest starting there.

Gary November 12, 2018 at 6:40 pm #

REPLY

Hi Adrian,

did you show in your book training custom objects with different frameworks like Yolo,YoloV3,Tensorflow,Mxnet and Caffe with faster-RNN vs. SSD?

If not, that would be great to see which framework has the best object multi detector for small and close objects. Hope you will think about this.

Thanks a lot for all your great tutorials.

Adrian Rosebrock November 13, 2018 at 4:26 pm #

REPLY

Inside the book I focus on Faster R-CNN, SSDs, and RetinaNet. Per my suggestions in this blog post I don't tend to use YOLO that often.

yan November 14, 2018 at 12:14 pm #

When I was using Raspberry 3B +, I encountered the error of Attribute Error:' NoneType' Object Has No Attribute' Shape', but I don't know how to fix it. I hope I can get your guidance

Adrian Rosebrock November 15, 2018 at 12:00 pm #

Your path to the input image is invalid and `cv2.imread` is returning "None". Double-check the path to your input image. Also read this tutorial on NoneType errors.

Yonten November 29, 2018 at 3:31 am #

REPLY

I have trained my dataset on darknet and I am using your code to detect my trained images but I cannot see the bounding box. When I run in darknet, I can cleary see the output with the bounding box. Can you tell me which code I should edit?

Adrian Rosebrock November 30, 2018 at 9:00 am #

REPLY

I would raise that question with the OpenCV developers. Your architecture may be different or some additional model conversion may need to take place.

issaiass March 21, 2019 at 3:33 pm #

REPLY

I had a similar error in my PC. There were two major issues:
1 – Something with Python 2.7 (not sure)
2 – Something with OpenCV below 3.4.2

Solutions:
1 – Anaconda3, Python version 3.6.x
2 – OpenCV over 3.4.2

aiwen November 12, 2018 at 11:42 am #

REPLY

so good!

Adrian Rosebrock November 12, 2018 at 12:09 pm #

REPLY

Thanks Aiwen, I'm glad you liked it!

ShivaGuntuku November 12, 2018 at 11:54 am #

REPLY

Hi Adrian, thank you for the tutorial,

Although I am getting this error in ubuntu18 , python3.6 and cv2 version '3.4.0'

…
error: (-212) Unknown layer type: shortcut in function ReadDarknetFromCfgFile,

Please help me out.

Adrian Rosebrock November 12, 2018 at 12:12 pm #

REPLY

You need at least OpenCV 3.4.2 for this tutorial. OpenCV 4 would work as well.

ShivaGuntuku November 12, 2018 at 1:43 pm #

REPLY

Thanks Adrian, it worked. Nice

Adrian Rosebrock November 13, 2018 at 4:35 pm #

REPLY

Awesome, I'm glad that worked.

Richard Wiseman November 14, 2018 at 5:04 am #

Might be worth updating the article to say 3.4.2 rather than 3.4 as it currently does. This caught me out too.

Adrian Rosebrock November 15, 2018 at 12:06 pm #

Thanks for catching that Richard. I've updated the post 😉

John Kang November 13, 2018 at 6:15 pm #

REPLY

I got same error on Windows. I have OpenCV-Python 3.4.0 installed. How to install opencv-python 3.4.2 on windows?

thanks in advance

Adrian Rosebrock November 15, 2018 at 12:16 pm #

REPLY

I'm sorry to hear about the error message. You would indeed need to install OpenCV 3.4.2 or higher. That said, I do not officially support Windows here on the PyImageSearch blog (I haven't even used a Windows machine in 11+ years now). When it comes to computer vision and deep learning I *highly recommend* you use a Unix-based machine such as Ubuntu or macOS. I have a number of OpenCV install tutorials for those operating systems. If you need help with Windows I would need to refer you to the official OpenCV website.

Ulrich January 1, 2019 at 11:23 am #

REPLY

Hello John,

I use python 2.7 and opencv 2.4.11 on a windows10 System. I updated my opencv by using "pip install opencv-contrib-python" and opencv 3.4.5 was installed in a view minutes.

The code is running well!

Sourav November 12, 2018 at 12:02 pm #

REPLY

can it be implemented on a pi 3B?

Adrian Rosebrock November 12, 2018 at 12:10 pm #

REPLY

Yes, but it would be extremely slow, under 1 FPS (at least for the OpenCV + YOLO version). The Movidius NCS does have a YOLO model that supposedly works but I have never tried it — that would likely get you to a FPS.

wally kulecz November 12, 2018 at 5:34 pm #

REPLY

If you are talking about this TinyYolo model for the Movidius from the appzoo:

https://github.com/movidius/ncappzoo/tree/master/caffe/TinyYolo

Its not the same model used in this tutorial.

I've played with it and it was really poor at detecting people and really good at finding people in shadows (false positives) so it was useless for my purposes.

The YOLOv3 model used here has performed admirably on the test images where the TinyYolo model from the NCS appzoo (linked above) failed miserably.

If there is a Movidius version of this YOLOv3 model, point me to it and I'll give it a try and report back.

Adrian Rosebrock November 13, 2018 at 4:26 pm #

REPLY

That was the one I was thinking of, thanks Wally. I'm not aware of a YOLOv3 model for the Movidius though.

wally kulecz November 14, 2018 at 4:51 pm #

Looks like a Movidius NCS2 using the Myriad X is available, the splash pages suggests “up to 8X faster” than the Movidius:
https://software.intel.com/en-us/neural-compute-stick

They are available, I just ordered one from Mouser for $99 + tax and shipping.

No mention of Raspberry Pi support, for now. It looks like the the OpenVINO toolkit will be required to use it. They are supporting Windows 10 for this one. Its a free download:
https://software.intel.com/en-us/openvino-toolkit/choose-download/free-download-linux

The Pi is where this improved device could really help, but it looks like it needs USB3 and a specific driver which may explain the lack of Pi support.

I’m expecting a challenge to get the tool kit install.

Adrian Rosebrock November 15, 2018 at 11:56 am #

Intel sent me a NCS2 but I must admit that I never unboxed it 😕 I’ve been too busy releasing the 2nd edition of DL4CV. I’ll have to carve out some time and play with it as well 🙂 Thanks for the motivation, Wally.

wally kulecz November 19, 2018 at 8:30 pm #

Perhaps a bit more motivation.

I installed the openVINO SDK on that old i3 system that I mentioned in another reply (failed with library version errors on Ubuntu 18.04 , so I installed 16.04 to the free space on the drive and dual boot).

Running their C++ interactive_facedetection_demo sample code with a USB WebCam I get these results:

NCS facedetection: ~16 fps face analysis: ~3 fps
NCS2 ~42 fps ~10 fps
CPU ~17 fps ~5.4 fps

Note that the CPU needed an FP32 model where the NCS used fp16. As with my MobleNet-SSD Python code, the CPU on the i3 is about the same as the Movidius NCS, but the NCS2 shows very worthwhile improvement.

The SDK auto-detects NCS vs NCS2 so it was just a matter of unplugging the NCS and plugging int the NCS2 to get these numbers from the live openCV overlay.

The SDK compiles openCV v4.0.0-pre. It appears to support Python Virtual Environment, although I didn't use one.

The GPU support seems not to work on this old i3-i915 motherboard.

There is a C++ example for YOLOv3 object detection in the installed sample code.

But my first task will be to see if I can re-write my Python code to use the openVINO Python support as from my limited test it looks like one NCS2 might be able to exceed the fps I get with three NCS sticks.

Devin November 19, 2018 at 8:52 pm #

REPLY

Hi, Doctor Adrian, very glad to read ur blog. i have a project that should recognize and detect object in the video based on Raspberry Pi 3B+ , my boss wanna i use deep learning method, such as resnet, ssd, yolov3, etc… but, in your blog, i know it's difficult to achieve real time…what should i do? could u please give me some advice?
thanks!

Adrian Rosebrock November 20, 2018 at 9:14 am #

REPLY

Hey Devin — I cover how to train your own custom object detectors (Faster R-CNN, SSDs, RetinaNet, etc.) inside my book, Deep Learning for Computer Vision with Python. I also discuss and demonstrate how to obtain real-time performance and which model is suitable for various tasks. I would suggest you start there.

Irwin May 20, 2019 at 10:52 pm #

What an elegant post, and very understandable. When I try this on a video file that has a pick-up truck, sometimes it detects the truck in one frame and detects the same truck as a car in another frame. Do you have any quick suggestions on how

to correct this? I would still like to detect both “trucks” and “cars” from the same video.

Adrian Rosebrock May 23, 2019 at 9:43 am #

You could do a rolling average over time. Keep track of the top 2-3 predictions over consecutive frames, average them, and pick the label with the highest average probability.

wally kulecz November 12, 2018 at 5:49 pm #

REPLY

The downloaded tutorial code runs fine on my Pi3B+ with python3 and openCV 3.4.2, but it takes 14 seconds to process an image. Can’t imagine how this could be of any use beyond a demo.

sset November 12, 2018 at 12:44 pm #

REPLY

Thanks for great article.

How do we custom train for customized dataset?

Adrian Rosebrock November 13, 2018 at 4:37 pm #

REPLY

I provide code and discuss how to train your own custom object detectors on your own datasets inside my book, Deep Learning for Computer Vision with Python.

Alex November 12, 2018 at 1:05 pm #

REPLY

Hello Adrian, which GPU did you use to achieve this performance?

Adrian Rosebrock November 13, 2018 at 4:37 pm #

REPLY

I did not use a GPU, it was CPU only. OpenCV's "dnn" module does not yet support many GPUs.

Cenk Camkoy November 12, 2018 at 1:41 pm #

REPLY

This is really very cool. Thanks for sharing all these together with your valuable benchmarks. By the way, out of my curiosity, do you know what type of object detector is used in Google's autonomous cars? SSD or other?

Adrian Rosebrock November 13, 2018 at 4:36 pm #

REPLY

Hm, no, I don't know what Google is using in their autonomous cars. SSDs are rooted in Google research though so that would likely be my guess.

JBeale November 12, 2018 at 1:54 pm #

REPLY

YOLO may not win on real-world metrics, but it is clearly #1 in readability of the associated papers.

Adrian Rosebrock November 13, 2018 at 4:34 pm #

REPLY

Agreed 🙂

Max November 12, 2018 at 2:31 pm #

REPLY

Hi,
It is possible to make it up and running on a GPU?

Adrian Rosebrock November 13, 2018 at 4:34 pm #

REPLY

That depends. OpenCV's "dnn" module currently does not support NVIDIA GPUs. It does work with some Intel GPUs though.

Bob Estes November 12, 2018 at 2:55 pm #

REPLY

To be clear, your performance numbers for YOLO and SSD are for a CPU version, not a GPU version, right? Thanks.

Adrian Rosebrock November 13, 2018 at 4:32 pm #

REPLY

That is correct. YOLO can run 40+ FPS on a GPU. Tiny-YOLO can reportedly get past 100+ FPS.

julio November 12, 2018 at 3:21 pm #

REPLY

If you work OpenCV with CUDA support, can you achieve 30FPS in real time? … I mean …
1. YoloV3 + module dnn + CPU is very slow
2. YoloV3 + module dnn + GPU that FPS speed could reach for real-time applications?

How could I use Yolo in real time on a laptop GPU like Asus' GeForce 930MX?

Adrian Rosebrock November 13, 2018 at 4:32 pm #

REPLY

See my replies to the other comments in this post — OpenCV does not yet support NVIDIA GPUs for their "dnn" module (hopefully soon though). That said, YOLO by itself can achieve 40+ FPS when ran on a GPU.

kelemu November 12, 2018 at 3:36 pm #

REPLY

Hi Adrian, I am waits like this tutorials but now I am lucky to get from you really tanks a lot. How to train YOLO with our datasets?

Adrian Rosebrock November 13, 2018 at 4:31 pm #

REPLY

I don't have any tutorials for training YOLO from scratch. Typically I recommend using SSDs or RetinaNet, both of which (and Faster R-CNNs), are covered inside Deep Learning for Computer Vision with Python.

Sam November 12, 2018 at 3:37 pm #

REPLY

Thanks Adrian.. great post.

Can I use it with Movidius NCS with custom dataset?

Adrian Rosebrock November 13, 2018 at 4:30 pm #

REPLY

Take a look at Wally's comment.

Robert November 12, 2018 at 3:56 pm #

REPLY

Thanks for suggesting to read the Yolo v3 research paper, that's easily the most entertaining and honest research paper I've ever read, all the way to the last line!

Adrian Rosebrock November 13, 2018 at 4:30 pm #

REPLY

Awesome, I'm glad you enjoyed it Robert!

Hemant November 12, 2018 at 4:26 pm #

REPLY

Hey Adrian, nice article and very useful. I tried it on Pi 3 and as you stated, it is very slow. I am getting object detection rate of 1 frame per 16 seconds. Processing of the airport.mp4 took little less than 4 hours. Looking forward to your second edition of the book.

Adrian Rosebrock November 13, 2018 at 4:29 pm #

REPLY

Thank you for checking YOLO performance on the Pi, Hemant!

wally kulecz November 12, 2018 at 4:59 pm #

REPLY

Nice timing on this, I just finished installing Ubuntu-Mate 18.04 on an i3 system. The installation of the Movidius v.1 SDK pulled in openCV 3.4.3 (presumably from PyPi) so I grabbed this sample code and gave it a try.

The yolo is taking ~1.47 seconds.

This is not a powerful machine (1.8 GHz if I remember right), but I'm getting about 10 fps with MobilenetSSD (from a previous tutorial) and one NCS stick handling 4 cameras (round-robin sampling) and near linear speed up with multiple sticks — 19.5 fps with 2 sticks 29 fps with 3 sticks. This is heavily threaded Python code with one main thread and one thread for each NCS stick and one thread for each Onvif network camera. A 4th NCS (9 threads) may be too much of a good thing as it drops to 24.6 fps. Although I had to have two sticks on a powered hub when I added the 4th stick for lack of ports, this may be a bit of a bottleneck as re-running the 3 stick test with two of them on hub dropped about 2 fps.

I hope one of the AI gurus can compile this yolo model for the NCS, although I realize this may not be possible.

Does your Xeon system use GPU (CUDA) acceleration? If so how many cuda cores?

My i7 Desktop has a GTX-950 with 2GB ram and 768 cuda cores, so I'm wondering if its worth the trouble to try and enable it. I need to update its openCV from 3.3.0 to 3.4.3 before I can run this tutorial, so this could be a good time for me to try and activate cuda.

Adrian Rosebrock November 13, 2018 at 4:28 pm #

REPLY

I love your multi-Movidius NCS setup, Wally! I would love to learn more about it and how you are using it.

As for my Xeon system, no, there is no CUDA acceleration. Although my iMac does have a Vega GPU so I suppose I could look into trying out the Intel + OpenCV + dnn drivers.

In your case don't bother with it. OpenCV doesn't yet support NVIDIA GPUs with their "dnn" module (hopefully soon though!)

wally kulecz November 13, 2018 at 11:37 pm #

REPLY

Thanks for the most useful info about openCV and CUDA, maybe for openCV 4.x.x it'll be worth revisiting. I really appreciate shared experience that saves me from a dead end!

My multi-Movidius Python code uses NCSDK API v.1 and has been tested with Python 3.6 and 2.7 on Ubuntu-Mate 18.04, Raspbian Stretch on a Pi3B+ with Python 2.7 and 3.5, and Ubuntu-Mate 16.04 with Python 3.5 virtual environment (I never setup the virtual environment for python 2.7). If no Movidius are found, it drops down to using your Caffe version of Mobilenet-SSD on the CPU with one thread per camera.

On my i7 with four cameras and three NCS I'm getting ~30 fps (8 threads) and with no NCS I'm getting about the same ~30 fps (9 threads). In each case there is evidence that the AI spends significant time waiting for images

On an i3 (same four cameras) its getting ~29 fps with three NCS, but it falls apart with no NCS only getting ~8 fps and its clear the camera threads that are waiting for the AI threads. Just not enough cores for the CPU AI.

On a Pi3B+ with three cameras its getting ~6.7 fps with one NCS (5 threads), ~11 fps with two NCS (6 threads), and ~13 fps with three NCS (7 threads). Two NCS seems to spend significant time waiting on the AI, while three NCS appears to spend significant time waiting on images, based on summary counts in the threads that the camera thread would block on queue.put() and the NCS thread would block on queue.get().

Right now its only supported input is Onvif netcameras via their "snapshot" URL. The single stick version used your imutils to optionally use USB cameras or the PiCamera module, but I ripped this support out of the multi-stick version as few USB cameras work with IR illumination and only one PiCamera module can be used on a Pi as far as I know.

I need four cameras minimum, my use is for a video security system where a commercial “security DVR” provides 24/7 video recording while the AI provides near zero false positive rate high priority “push” notifications when it is armed in “not home mode”, audio alerts (via espeak-ng) if armed in “at home mode”, and nothing when in “idle mode”.

The Python code does the AI, node-red does the controlling and notifications, and MQTT glues it all together. The basic system has been running since early July and it works extremely well. It continues to evolve, mostly to improve the frame rate and reduce the detection latency — think “bad guys” marshaling on your property for a “home invasion”. But we love it for when the mailman comes or a package is delivered 🙂

I’d be happy to send you the Python code if you are interested, in fact I’d like to see if it works on a Mac. The CPU only part runs on Windows 10 and 7 (no NCS support without way more effort than I’m willing to apply) in limited testing with the single stick (AI thread) version (I’ve removed the Windows support from the multi-stick code). I’ve totally given up on Windows since I retired, but a couple of Windows only friends were interested early on (hence the Win7 and Win10 tests), and I must say that this was by far the best cross-platform development experience I’ve ever had! Python has really impressed me!

I plan to put it up on GitHub eventually, the Ubuntu 18.04 and PyPi openCV install was so easy I finally think I could write a README.md (in a reasonable amount of time) that someone could actually use from a fresh install of Raspbian or Ubuntu.

Adrian Rosebrock November 15, 2018 at 12:11 pm #

REPLY

Thanks for the detailed writeup, Wally! Let me know when you publish it on GitHub and I’ll take a look 🙂

faurog November 19, 2018 at 11:20 pm #

REPLY

Hi, Dr. Adrian. It would be nice if you tried using an Intel iGPU + OpenCV + dnn module. My laptop has a Nvidia GPU (not well supported yet) and an integrated Intel GPU, but I couldn’t make it work (net.setPreferableTarget(cv2.dnn.DNN_TARGET_OPENCL)). Anyway, if you try something, let us know. I would like to know if it indeed improves the performance. Thank you for another incredible post. Cheers.

Adrian Rosebrock November 20, 2018 at 9:12 am #

REPLY

I unfortunately do not have an Intel GPU right now. I hope to try it in the future though. Perhaps another reader can share their experience.

blank November 12, 2018 at 8:55 pm #

REPLY

always cool tutorial, keep it up, have a great day! 🙂

Adrian Rosebrock November 13, 2018 at 4:25 pm #

REPLY

Thanks, you too 🙂

Shivam Sahil November 12, 2018 at 9:30 pm #

REPLY

I always had this question in mind, even though it should be the fastest detector, whenever I use it in real time video detection, it gets slowest even than normal cnn which works pretty fast in my laptop. Is it because I have amd graphics card instead of NVidea or something else? Was just confused… let me know if you have suggestions regarding the same. I saw it takes about 1.3 sec to detect all the individual objects in one frame. But how is it able to detect objects quickly in your predefined videos, I just changed those to make it real time and it again went super slow, please let me know what is the actual issue.

Adrian Rosebrock November 13, 2018 at 4:25 pm #

REPLY

Keep in mind that the YOLO model is not accessing your GPU here. The YOLO + OpenCV implementation is running on your CPU which is why it's taking a long time for inference.

Jason November 15, 2018 at 11:06 am #

REPLY

Adrian, as always, you have a nice tutorial. Thanks a lot.

You can speed up the YOLO model on CPU by using OpenMP. Open makefile, and set AVX=1 and OPENMP=1.

Adrian Rosebrock November 15, 2018 at 11:49 am #

REPLY

Thanks Jason. How much of a speed increase are you seeing with that change?

Jason November 15, 2018 at 10:04 pm #

I have not had the chance to download your codes yet. I am currently using my own data to train YOLOv3. It takes a lot time to prepare the images for training because you have to draw a bounding box for each objects in each images. Once I finish the training, I will let you know the speed difference between turning OpenMP on and off in prediction.

By the way, you can also set OpenCV on and off in YOLO.

git-scientist November 28, 2018 at 10:10 pm #

REPLY

Hi Jason, could you give some detailed info about OpenMP? How one should make use of it? And, where does that makefile reside?

Balaji November 12, 2018 at 10:23 pm #

REPLY

Hi,

Nice tutorial for Yolo and valid comparsion with other object detection models.
I want to detect small objects, so more interested in Faster-Rcnn resnet models, In this blog I can see you have mentioned they will outperform with ~5fps. I am using Faster-Rcnn resnet101 model in GPU 1080, but I am getting only 1.5 fps.
Can you please suggest how to improve the speed.
And as a user want to ask, When can we except a blog on Faster Rcnn Models and their advantages with custom training.

Thank You

Adrian Rosebrock November 13, 2018 at 4:23 pm #

REPLY

Hey Balaji — I actually show you how to train your own custom Faster R-CNN models on your own datasets inside my book, Deep Learning for Computer Vision with Python. I also provide you with my tips, best practices, and suggestions on how to improve your model performance and speed. Be sure to take a look, I think it will really help you out.

Jacob November 12, 2018 at 10:28 pm #

REPLY

What performance do you expect when run with a Tesla V100 GPU with 608×608 images? With darknet, I can process images with yolo between 80-90 fps. Yolo is typically much slower when implemented in python–does this opencv implementation also have a significant reduction in performance compared to darknet?

Adrian Rosebrock November 13, 2018 at 4:22 pm #

REPLY

OpenCV doesn't yet support NVIDIA GPUs with their "dnn" module so we cannot yet obtain that benchmark. NVIDIA GPU support is coming soon but it's not quite there yet.

adam_Viz November 13, 2018 at 1:26 am #

REPLY

Oh Adrain!!! Awesome,am implemented successfully without any hasle..thankx for your contribution .

Adrian Rosebrock November 13, 2018 at 4:21 pm #

REPLY

Thanks Adam — and thank you for being a PyImageSearch reader.

Alexander November 13, 2018 at 2:34 am #

REPLY

Hello, Adrian!

What could you think about problem with real-time video from web-cameras? In our project (on-line detecting cars and peoples) when we used OpenCV3 with real-time video, we got big delay between frames… We solved this problem, but now we don`t using real-time video-streams from OpenCV.

Could you have sample with real-time stream, not mp4 or avi-files?

Best wishes, Alexander,
Russia, Novosibirsk.

Adrian Rosebrock November 13, 2018 at 4:21 pm #

REPLY

Keep in mind that deep learning models will run significantly faster on a GPU. You might want to refactor your code to use pure Keras, TensorFlow, Caffe, or whatever your model was trained with, enabling you to access your GPU. More GPU support with OpenCV is coming soon but it's not quite there yet.

TAYFUN ARABACI November 13, 2018 at 2:50 am #

REPLY

very very nice Adrian :=)

Adrian Rosebrock November 13, 2018 at 4:20 pm #

REPLY

Thanks Tayfun!

Riad November 13, 2018 at 6:16 am #

REPLY

Great tutorial ! But I notice that the code doesn't work with grayscale images. Is there some parameters I can tweak to make it work?

Adrian Rosebrock November 13, 2018 at 4:16 pm #

REPLY

YOLO expects three channel RGB input images. If you have an input grayscale image just stack it to create a "faux" RGB/grayscale image:

```
image = np.dstack([gray] * 3)
```

Anusha November 13, 2018 at 8:55 am #

REPLY

Hey Adrian, this is a great post and I really liked the way you put everything in sequential order. I have a question though. I was wondering how can I replace the YOLO model for this object detection with Faster RCNN to suit my purposes as I have fairly small objects in my videos which I need to detect. I mean is there a deploy model and prototxt available for Faster RCNN?

Adrian Rosebrock November 13, 2018 at 4:14 pm #

REPLY

Yes, you would:

1. Train your Faster R-CNN on whatever dataset you are using
2. Then take the prototxt and Caffe model weights and swap them in

Keep in mind that loading Faster R-CNN models is not yet 100% supported by OpenCV yet. It's partially supported but it can be a bit of a pain.

Sophia November 13, 2018 at 10:55 am #

REPLY

yet another amazingly informative tutorial! how does the speed-accuracy tradeoff of SSD compare with that of RetinaNet? thanks,

Adrian Rosebrock November 13, 2018 at 4:14 pm #

REPLY

In my experience RetinaNet tends to be slightly slower but also (1) slightly more accurate and (2) a bit easier to train.

sophia November 13, 2018 at 5:05 pm #

REPLY

thank you for replying, Adrian. that's helpful information.

joeSIX November 13, 2018 at 2:58 pm #

REPLY

This is a great tutorial, can't thank you enough.
unfortunately, I was unable to test it on my own (macbook pro, anaconda environment, opencv 3.4.2):
error: (-215:Assertion failed) ifile.is_open() in function 'ReadDarknetFromWeightsFile'

Adrian Rosebrock November 13, 2018 at 4:08 pm #

REPLY

Double-check your path to the input weights and configuration file. It sounds like your paths may be incorrect.

Yurii November 13, 2018 at 9:23 pm #

REPLY

Hi Adrian,
Is there a way to specify particular object to detect? For instance only cars and stop signs. It should speed up process I suppose as resources are not wasted on recognition of other objects.

Adrian Rosebrock November 15, 2018 at 12:14 pm #

REPLY

You can fine-tune the model to remove classes you're not wanted in but keep in mind the number of classes isn't going to dramatically slow down or speedup the network — all the computation is happening earlier in the network.

Ramkumar November 14, 2018 at 4:53 am #

REPLY

Hi Adrian,
The article you explained very interestingly for a beginner. Can we implement a smoke detection from image using yolo? Or only for hard objects?

Adrian Rosebrock November 15, 2018 at 12:07 pm #

REPLY

Object detectors work best for objects that have some sort of "form". Smoke, like water, doesn't have a true rigid form hence YOLO and other object detectors would not work well for smoke detection.

Marcelo Mota November 14, 2018 at 5:45 am #

REPLY

Thanks for another great tutorial, Adrian!

Could you please explain in more details lines 41 to 43? Why do you get layer names, and unconnected layers? And why that " – 1″? code below:

```
# determine only the *output* layer names that we need from YOLO
ln = net.getLayerNames()
ln = [ln[i[0] – 1] for i in net.getUnconnectedOutLayers()]
```

And also line 52, why do you need to do a forward pass in just the "ln" layers? code below:

```
layerOutputs = net.forward(ln)
```

thank you!

Marcelo

Adrian Rosebrock November 15, 2018 at 12:05 pm #

REPLY

The YOLO model is trained via the Darknet framework. We need to explicitly supply the output layer names into the call to ".forward()". It's a requirement when using Darknet models with OpenCV.

Taha November 14, 2018 at 9:24 am #

REPLY

I'm getting 0.5 fps on a 1.7Ghz processor which intel core i3 4th gen. Is that okay speed for this model and system.

Adrian Rosebrock November 15, 2018 at 12:02 pm #

REPLY

Given that the model is running on the CPU, yes, those results seem accurate.

Andrew November 14, 2018 at 12:41 pm #

REPLY

Hello Adrian,

Nice tutorial…have you tried running YOLOv3 in C, given that it was originally written in C?
I think there are some python wrappers out there for the datatypes

Adrian Rosebrock November 15, 2018 at 11:58 am #

REPLY

I haven't tried in C but I know there is the darknetpy wrapper which can be used to run YOLO on a GPU.

Thanks a lot! November 14, 2018 at 2:39 pm #

REPLY

Hi, Can you please tell me if I can run this code on Windows or not? I am stuck in Windows and cannot find a comprehensive tutorial of Yolo in Windows. Please help.

Adrian Rosebrock November 15, 2018 at 11:58 am #

REPLY

This tutorial will work on Windows provided you:

1. Use the “Downloads” section of the tutorial to download the code + trained YOLO model
2. Have OpenCV 3.4.2 or higher installed

Sunny November 17, 2018 at 11:33 pm #

REPLY

Hi Adrian,

If I want to only detect the red car inside the car chasing video by using YOLO, any suggestions on fulfilling the goal? Thank you

Adrian Rosebrock November 19, 2018 at 12:38 pm #

REPLY

You would:

1. Filter on the “car” class, ignoring all other non-car detections
2. Determine the object color

moxran November 19, 2018 at 3:16 am #

REPLY

Hi,

Since this is a yolo detector with OpenCV, it is not using the gpu, right? I'm getting 1 fps on a Intel core i7 2.2 Ghz processor, which is really slow. any reason you can see? Thanks!

Adrian Rosebrock November 19, 2018 at 12:25 pm #

REPLY

Correct, the YOLO detector is running on the CPU, not the GPU. Please see the other comments on this page where I've addressed OpenCV's GPU capabilities.

Dheeraj November 20, 2018 at 4:34 am #

REPLY

Can we count people using YOLO based Approach? If so, then what changes should be made in the code or need to use my own data set to train the model?

Adrian Rosebrock November 20, 2018 at 9:03 am #

REPLY

I actually describe how to build an people counter in this tutorial.

Jelo November 20, 2018 at 11:51 am #

REPLY

Hi Adrian,

First at all let me thank you for your all posts, really it is very useful for all.
I would like to ask you can we use the deep learning to estimate the detect object position ? can you share some links ?
Thank you again

Adrian Rosebrock November 21, 2018 at 9:37 am #

REPLY

Could you elaborate on what you mean by “object position”? What specifically are you trying to measure?

Oscar Mejia November 20, 2018 at 11:01 pm #

REPLY

Hi Adrian, first of all, Thanks for your help and time to do this kind of tutorials. I really appreciate your help.

Adrian, I would like to know if you recommend these algorithms to apply in a project to identifying, tracking and counting people in real time, if not what technique would you recommend me?

Thanks in advance.

Oscar Mejia November 20, 2018 at 11:03 pm #

REPLY

I forgot mentioning that the project is for doing it on a raspberry pi 3 b+.

Thanks.

Adrian Rosebrock November 21, 2018 at 9:26 am #

REPLY

Take a look at my tutorial on building an OpenCV people counter. I include suggestions on how to adapt it to the Raspberry Pi.

Steve November 20, 2018 at 11:48 pm #

REPLY

Hi,
After running the yolo_video.py, it does’t display the video window, Why?

Adrian Rosebrock November 21, 2018 at 9:25 am #

REPLY

The YOLO video file does not display the frame on your screen, it just writes it to disk. To display the frame to your screen you can use `cv2.imshow`

Aiwenj November 22, 2018 at 2:18 am #

REPLY

Hello ,Adrian.Thank you for your post!and i have a question that i want to know the number of people in an image.how to do it using YOLO?

Adrian Rosebrock November 25, 2018 at 9:33 am #

REPLY

You would loop over the number of detected objects, use an “if” statement to check if it’s a person, and then increment a counter.

Abhijeet November 22, 2018 at 7:03 am #

REPLY

hi am getting this error while running your code No such file or directory: ‘yolo-coco\\coco.names please reply me

Adrian Rosebrock November 25, 2018 at 9:27 am #

REPLY

Make sure you are using the “Downloads” section of this blog post to download the source code and example models. It sounds like you don’t have them downloaded on your system yet.

Abhijeet November 30, 2018 at 7:47 am #

REPLY

Thanks man code is really understandable.. please tell my purpose of unconnectedoutput layer (ln) in code

Abhishek November 22, 2018 at 4:12 pm #

REPLY

Does this work with GIFs? Also I'm getting "error: (-215:Assertion failed) !ssize.empty() in function 'cv::resize'. Any fixes? maybe the input image seems to be empty but I'm not sure about it

Adrian Rosebrock November 25, 2018 at 9:19 am #

REPLY

No, OpenCV does not support loading GIFs. You'll want to convert the GIF to a series of JPEG or PNG frames, then feed them through the YOLO object detector.

frank November 23, 2018 at 9:14 am #

REPLY

Hi Adrian, thanks for your great tutorials. I have a question about the line 70 of source code in yolo.py. the length of detection is 85. detection[0:4] represent coordinates ,width and height. detection[5:] represent the probability of 80 objects. I find the detection[4] is not used. so I want to know what detection[4] stands for.

Daniele Bagni November 25, 2018 at 9:59 am #

REPLY

EXCELLENT TUTORIAL, Adrian as usual from you. Thank you very much for sharing your knowledge!

Adrian Rosebrock November 26, 2018 at 2:33 pm #

REPLY

Thanks so much Daniele!

Guille Lopez November 28, 2018 at 7:45 am #

REPLY

Hi Adrian. Great tutorial! Extending your code I've been able to add the SORT algorithm to create a first approach to a traffic counter. I was thinking on open sourcing the code on Github (link removed by spam filter). Is it ok for you if I do that? I will cite your tutorial, pointing back to this page.

Adrian Rosebrock November 30, 2018 at 9:12 am #

REPLY

Hi Guille — congratulations on building a traffic counter, awesome job! Yes, feel free to open source the project, please just link back to the PyImageSearch blog from the GitHub readme page. Thank you!

Dnyaneshwar December 3, 2018 at 10:50 pm #

REPLY

hi Adrian ,

i am new to deep learning computer vision , i have downloaded the source code from your site.

please let me know what set up is required and steps to run the program on windows

Adrian Rosebrock December 4, 2018 at 9:43 am #

REPLY

Please note that I only support Linux and macOS on this blog. I do not officially support Windows nor do I provide Windows install tutorials. I would suggest you follow my OpenCV install guides on either Linux or macOS to get up and running.

Dnyaneshwar December 4, 2018 at 2:27 am #

REPLY

HI Adrian ,

i am new to deep learning computer vision. can you help me how to get started building application for object detection using intel openvino toolkit.please provide the steps to create application and run it

Adrian Rosebrock December 4, 2018 at 9:39 am #

REPLY

At this time I do not have any tutorials on Intel's OpenVINO toolkit. I will consider it for the future but I cannot guarantee if/when I may write about it.

Nahael December 4, 2018 at 11:47 am #

REPLY

Hi Adrian, thank you for the tutorial,I;ve been following your post for a while now,I would like to know if its possible to traiin my own dataset to detect violent scenes in videos,it will be very kind if you could help us with that,thank you again.

Adrian Rosebrock December 6, 2018 at 9:55 am #

REPLY

What you are referring to is called "activity recognition". I don't have any tutorials regarding activity recognition (yet) but I do have a chapter inside Deep Learning for Computer Vision with Python which does show you how to detect and recognize weapons in images and video. That may be a good starting point for your project.

Chris December 5, 2018 at 7:20 am #

REPLY

Cheers for this Adrian, it's been exactly what I needed.

I've now adapted the code to work with my home CCTV!

Before my CCTV would FTP a short video when motion was detected to my server, and then I would use python to split the video into frames and email a picture of frames from 1sec, 3sec and 5sec to my email address, so where ever I am in the world I get an image of what triggered the motion sensor.

Problem is, that I kept getting images of cats, birds, heavy rain etc.

What I've done now is edit your code, so now when I get images from 1, 3 and 5secs. I run them through the code, check if the label is "person", "car", "truck" etc. and if so then attach the images to the email and send it.

No more false alerts!!

Thanks again and I love the Guru course too, having some real fun with that

Adrian Rosebrock December 6, 2018 at 9:40 am #

REPLY

Awesome, congratulations on adapting the code to your own project Chris!

Jammula December 8, 2018 at 5:00 am #

REPLY

hello,
Adrian Rosebrock,

Thank you for great tutorial ,I am facing problem while executing the yolo.py for object detection in images through terminal in Jupiter notebook.

Issue:
I am getting “cannot connect to X SERVER ” error

My server details:
i am using Nvidia GeForce GTX 1080 Ti with 11173 MiB memory

Thank You in advance.

Adrian Rosebrock December 11, 2018 at 12:58 pm #

REPLY

What line of code is throwing that error?

Kunal Gupta December 8, 2018 at 2:02 pm #

REPLY

Thanks for the post Adrian!
I’d like to know that it’s said that YOLO is faster than SSD’s so technically, on real time video feed, it should outperform them?
But, when I ran it, I got 15fps in MobileNet SSD’s and around, 3fps in YOLO.

What can be the issue?
Thanks!

Adrian Rosebrock December 11, 2018 at 12:57 pm #

REPLY

You are correct that YOLO should be faster than SSD but as you found out and as I noted in the “Limitations and drawbacks of the YOLO object detector” section of the guide YOLO appears to be slower. I’m not sure why that is.

Ramar December 15, 2018 at 11:42 am #

REPLY

adrian
how to downloads the yolomodule file.and how is it open for the windows os system

Adrian Rosebrock December 18, 2018 at 9:14 am #

REPLY

You can use the “Downloads” section of the blog post to download the YOLO model. The code will work on Windows.

Sanjay Swami December 12, 2018 at 1:25 pm #

REPLY

Hello Mr. Adrian Rosebrock

I am very glad that I found your blog and I have started tutorials given by you.
I am using this tutorial in my project. My project is to find the ball and track it. So will this program work on “Raspberry Pi” ?

Thank you in advance.

Adrian Rosebrock December 13, 2018 at 9:01 am #

REPLY

No, the object detector will run far, far too slow on the Raspberry Pi. Is your ball a colored one? If so, follow this tutorial on simple color thresholding and you'll be able to complete your project.

Sanjay Swami` December 14, 2018 at 5:05 am #

REPLY

Thank you for your information.

Mike December 17, 2018 at 2:23 pm #

REPLY

Hi Adriian,

You have mentioned that the Raspberry Pi is too slow for object detection…does the newer Pi hardware perform any better? I've thought about using your tutorial to build a weapon detector using a Pi B+ with your pre trained dataset.

Would this be too under powered?

Thank you,

Mike

Adrian Rosebrock December 18, 2018 at 8:57 am #

REPLY

No, the new hardware is still too underpowered. You should look into using the Movidius NCS.

Mansoor alam December 19, 2018 at 11:39 am #

REPLY

Hello sir, hope you will be fine.

sir, when I run this code I always get this error as below.

yolo.py: error: the following arguments are required: -i/–image, -y/–yolo

Sir, could you please tell me that what I am doing wrong and what is the solution to this error?

Adrian Rosebrock December 19, 2018 at 1:44 pm #

REPLY

If you are new to command line arguments, no worries, just make sure you read this tutorial first.

Mansoor alam December 21, 2018 at 12:58 pm #

REPLY

thank you sir. it worked 🙂

Mansoor alam December 21, 2018 at 1:11 pm #

REPLY

hello, sir hope you will be fine.
sir, I used ur code which is ok and working for images but when I run the code for video it works ok and all results displayed like total time and frame time etc but at the end, the video does not display in which detection occur.
so, what should I do???

Adrian Rosebrock December 27, 2018 at 11:00 am #

REPLY

The output of the video is not displayed to your screen, it's instead written to disk as an output video file. Check your output video file.

smalldroid December 25, 2018 at 2:32 am #

REPLY

Hi Mr Adrian, thanks for your great tutorial about YOLO? Are you going to make addtional tutorial about how to train YOLOv3 model with Cocodataset (using Keras, Pytorch)?

Adrian Rosebrock December 27, 2018 at 10:30 am #

REPLY

I actually show you how to train your own custom object detectors inside Deep Learning for Computer Vision with Python.

Aqsa December 25, 2018 at 4:28 am #

REPLY

Hi Adrian,

Firstly, amazing tutorial. Great help (y)

I want to run yolo on imagenet dataset. I downloaded weights and configuration files for imagenet YOLO from darknet project website. Then I plugged them in the code (weightspath, cofigpath, lines 30, 31 respectively). I also edited line 21 as required.
But it did not work for me. Is there anything additional that I should be doing?

Thank you in advance.

Daniel Spencer December 25, 2018 at 2:13 pm #

REPLY

Hey Adrian! ,
Great work , I just want to ask something real quick. It appears that you've trained yolo to detect a wide variety of objects which is amazing, but it takes somewhat of a long time on my computer to run!. The problem is , I'm only interested in detecting cats and dogs in my project (using the coco dataset as well) , so what would the procedure be in order to train a YOLO model similar to the one you did for these two specific classes.
Thanks!

Adrian Rosebrock December 27, 2018 at 10:23 am #

REPLY

Instead of training your network from scratch I would instead recommend performing fine-tuning. I cover how to train your own custom object detectors, including fine-tuning them, inside my book, Deep Learning for Computer Vision with Python.

Shivam December 27, 2018 at 2:48 pm #

REPLY

Hello Sir,
This is a great tutorial. Can you please elaborate more on Real time Object Detection using my own laptop's webcam?

Adrian Rosebrock January 2, 2019 at 9:49 am #

REPLY

Sure, see this tutorial.

Sannan January 10, 2019 at 4:46 am #

REPLY

Hy Adrian it's good tutorial. I want to detect objects from my webcam(Live Video). So how can i implement this code?

Adrian Rosebrock January 11, 2019 at 9:38 am #

REPLY

This method will be too slow to detect methods in real-time on a GPU. For real-time object detection on a CPU I would recommend you follow this tutorial.

RJ January 11, 2019 at 2:42 am #

REPLY

Why does this code not display the output video..but only prints the result for video????…Can you help me with how can I get the output video on screen with bounding boxes..

Adrian Rosebrock January 11, 2019 at 9:27 am #

REPLY

This code takes the output detections and writes them to a video file. You can use `cv2.imshow` to display the results to your screen.

OzgurG January 11, 2019 at 10:44 pm #

REPLY

Hi Adrian,
Do you think is there a way to calculate the speed of the vehicles from a fixed camera using YOLO or other modern CNN algorithms while doing vehicle tracking?
It is like to replace the traditional traffic speed cameras with CNN…

Adrian Rosebrock January 16, 2019 at 10:18 am #

REPLY

Yes, I will be covering speed calculation in my computing Computer Vision with Raspberry Pi book. Stay tuned!

Bruce Dai January 20, 2019 at 2:31 am #

REPLY

Great post, I never know CV2 has a dnn.py module that is so useful. I tried this script on some test images, but it seems that YOLO is doing poorly recognizing cars. It mislabeled some of the cars as cell-phones in my images. 🙁

Adrian Rosebrock January 22, 2019 at 9:32 am #

REPLY

Hey Bruce — you may want to try a more accurate object detector such Single Shot Detectors or Faster R-CNNs. In practice I've found that both SSDs and Faster R-CNN perform better "in the wild".

Maning January 24, 2019 at 9:28 am #

REPLY

Hello! Thanks for the tutorial! I followed it but used a model that I trained on yolov3-tiny-obj.cfg instead. However, the results are different compared to the results I get when I run the detector using the command line. Any idea what could be the cause of it?

adel January 27, 2019 at 6:00 pm #

REPLY

i use pycharm , anaconda , visual studio and google colab . i need all of them .
in vs and pycharm : where should i address the parametes ?
–image baggage_claim.jpg –yolo yolo-coco

i just can tun the cod in pycharm and i have error in othe enviroment
tnx for your great contents

Adrian Rosebrock January 29, 2019 at 6:45 am #

REPLY

You'll need to set the command line arguments via PyCharm. Make sure you read this tutorial which includes an example.

Eduardo January 30, 2019 at 11:07 am #

REPLY

Hi! this was a great tutorial! I was wondering if there could be some way to use the same code but with the YOLO9000 .cfg and .names file. I have already tryed on my own but the program crashes.

Shaon January 30, 2019 at 2:12 pm #

REPLY

Is YOLO object detection available with OpenCV.js ?

Mohammd January 31, 2019 at 4:33 pm #

REPLY

Hi Adrian,
I run that your python script and ./darknet detect …. from original website YOLO, but the result of them not same , why ? the result of original command website YOLO is very correct.
why ?????

Adrian Rosebrock February 1, 2019 at 6:42 am #

REPLY

Are you sure you're using the same YOLO model versions? If so, it could be a difference in the NMS parameters. I'm not sure what the default DarkNet NMS parameters are, you may need to refer to the documentation/source code.

Jean-Michel February 1, 2019 at 11:45 am #

REPLY

Hello,

I'm comparing the results obtained with (1) yolov3 built from https://github.com/jaskarannagi19/yolov3 with (2) the results obtained with your code.

I've noted that the results are not the same at all (and not only the probability). For example, in some cases, the 1st yolov3 detects a car while the « dnn » yolov3 detects nothing.

For the 1st case, the command is :
./darknet detect cfg/yolov3.cfg yolov3.weights data/test.jpg -i 0 -thresh 0.25

For the 2nd case, the command is :
python yolo.py –image images/test.jpg –yolo yolo-coco –confidence 0.25

It should be noted that the config files (yolov3.cfg) are strictly the same. The weights files (yolov3.weights) too.

Best Regards.

sidrah February 1, 2019 at 1:36 am #

REPLY

nonetype attribute error

Adrian Rosebrock February 1, 2019 at 6:37 am #

REPLY

Your path to the input image is incorrect and “cv2.imread” is returning “None”. Double-check your input image path. You can read more about NoneType errors, including how to resolve them, here.

karan February 5, 2019 at 7:45 am #

REPLY

Hi Adrian. Detection[0:4] return centerX, centerY, width, height. Detection[5:] returns the probability score for each classes. What does Detection[4] mean?

Adrian Rosebrock February 5, 2019 at 9:12 am #

REPLY

The “detection[4]” would be the height of the bounding box.

yoming February 12, 2019 at 2:16 am #

REPLY

how can i use centroidtracker in yolo-detection，the coordinates are always wrong,i use “simple object tracking” to modify.

Edouard February 13, 2019 at 8:33 am #

REPLY

Hi Adrian and all,
I am training a Densenet model on X-Ray pictures. I would like to localize some indications on these X-Ray pictures. Do you have hints to adapt Yolo or any advice ?
Thanks

Adrian Rosebrock February 14, 2019 at 12:56 pm #

REPLY

That's much more of a complicated problem but absolutely doable. I would recommend referring to Deep Learning for Computer Vision with Python where I suggest how to train your own custom deep learning object detectors and instance segmentation networks.

I would also be very curious to know which dataset you are using. I like medical image datasets 🙂

Manohar Sonwan February 18, 2019 at 6:10 am #

REPLY

Hi Adrian,
This tutorial was very helpful.
I have been working on object collision problem where I want to detect the collision between tennis racket and ball. I have used "YOLO.h5" model to detect those two object and I have achieved that detection part in my code, but I want to detect the collision and pause the video whenever a collision happens.

Adrian Rosebrock February 20, 2019 at 12:31 pm #

REPLY

I would recommend you instead perform instance segmentation with Mask R-CNN. You can then check and see if the two masks overlap (use bitwise operations for that). If you're new to image processing make sure you read through Practical Python and OpenCV so you can learn the basics first, including bitwise operations.

Elena February 25, 2019 at 12:00 am #

REPLY

Hello Adrian,
Thank you for the post. I have tested this method on my video, but as you mentioned this method does not use GPU and it is not real time.
My project is about detecting objects through live and stream camera. I should use YOLO V3 for that. I think OpenCV would be useless for my project since it does not support GPU.
what frameworks should I use to get this project done?
Could you please guide me? I am quite desperate.

Adrian Rosebrock February 27, 2019 at 6:00 am #

REPLY

Take a look at “darknet”, the author’s implementation of YOLO. It is capable of running on a GPU.

Elena February 27, 2019 at 8:35 am #

REPLY

Thanks for the reply. I think I did not clearly ask my question.
My question is:
Does “dnn” module of OpenCV support GPU now? How we can run OpenCV on GPU?

Adrian Rosebrock February 28, 2019 at 1:52 pm #

REPLY

It really depends on what type of GPU you are using. The most popular types of GPUs are NVIDIA GPUs which OpenCV’s “dnn” module does not yet support (hopefully soon though!)

Elena March 7, 2019 at 9:44 am #

REPLY

I am a bit confused. I run your method, and I am giving 30sec video as an input and the output (processed video) is also 30 sec. I expected to get a longer video in the output since the code is running in CPU. How this is possible that the motions in the video are not sluggish and the video is smooth.

Adrian Rosebrock March 7, 2019 at 4:15 pm #

REPLY

The output video doesn’t care how long it takes for a new frame to be added to it. We just specify the output video FPS and that is the FPS rate that the video plays back at.

Rishabh Kachhwaha February 25, 2019 at 2:41 pm #

REPLY

Hey Adrian,

My yolo_video.py file is running without any error but there is a bug the output file is of few a milliseconds, instead of being of full lenght as of the original video.

Help needed..

Akbar February 26, 2019 at 11:32 pm #

REPLY

how to implement the video stream using flask, so the video stream will play on web? not just using the command line only

thanks

Adrian Rosebrock February 27, 2019 at 5:31 am #

REPLY

I'll be showing how to stream the output of the YOLO object detector to the web using Flask in my upcoming Computer Vision + Raspberry Pi book, stay tuned!

mhadi February 27, 2019 at 4:19 pm #

REPLY

Hi

could you please explain or reference the YoLo architecture ?

Adrian Rosebrock February 28, 2019 at 1:44 pm #

REPLY

I provide links to the YOLO papers in the guide, you can refer to them for details on the YOLO architecture.

Vasilis March 1, 2019 at 3:52 am #

REPLY

Hello,

Thank you for this tutorial but unfortunately when I run it in terminal I get this error:

writer.release()
AttributeError: ‘NoneType’ object has no attribute ‘release’

I saw some other people had similar problem on this forum so I went to the ‘Nonetype’ tutorial you suggested, but the causes described there do not match my issue.

I downloaded the source code so the input file path should not have a problem.
The webcam is accessible via OpenCV because I tried it in other scripts.
‘Not having the proper video codecs installed’? Not sure what this is but I successfully ran your face detection tutorial so maybe that is not the case either. Any ideas please?

Kind regards

Amir March 9, 2019 at 9:12 am #

REPLY

Hi
this issue appear when the packages of openCV did not install correctly.
therefore, you have to install openCV and all its corresponding packages again and with more accurate.

Khushboo Katariya March 5, 2019 at 9:45 am #

REPLY

can we applied that code on Live videoes capture at that time by camera.

Adrian Rosebrock March 5, 2019 at 9:51 am #

REPLY

Technically yes, but the FPS is going to be pretty low (in the order of 1-5 FPS on a CPU). OpenCV's "dnn" module doesn't yet support NVIDIA GPUs for the "dnn" module.

Amir March 9, 2019 at 9:16 am #

REPLY

Hi
can you introduce some tutorials and codes on Live video capture?
also a tutorial for run that code on NVIDIA GPUs?
tnx

Adrian Rosebrock March 13, 2019 at 3:55 pm #

REPLY

I cover how to perform object detection in real-time, including using NVIDIA GPUs, inside my book, Deep Learning for Computer Vision with Python. I would suggest starting there.

Elena March 7, 2019 at 9:54 am #

REPLY

Hello Adrian,
I was wondering if you could tell me why the original YOLO3 (from the original website) is capable of detection more object than the implementation of YOLO3 in Keras and OpenCV with the same weights?

and could you please explain why there is no training in this method that you posted here?

Thanks in advance.

Adrian Rosebrock March 7, 2019 at 4:14 pm #

REPLY

1. Do you mean capable of detecting more object classes?

2. The model we are using here is pre-trained. Since it's pre-trained on the COCO dataset we do not have to train it ourselves.

pavan March 9, 2019 at 9:31 am #

REPLY

This project is very helpful for me
Thanks a lot
Can i have the documentation for this project?

Adrian Rosebrock March 13, 2019 at 3:54 pm #

REPLY

I'm not sure what you mean by "documentation". The entire post and code is documented.

student1 March 18, 2019 at 6:59 am #

REPLY

hi,

nice work , i successfully run it on some different videos.
it's detetcs very accurate.
one question : it's possible to yolo_video.py to also counting the vehicules and the pedestrians?

best regards.

Adrian Rosebrock March 19, 2019 at 10:00 am #

REPLY

Absolutely. See this tutorial.

James Adams March 22, 2019 at 5:22 pm #

REPLY

Brilliant post, Adrian. Thanks for all the effort that goes into making your tutorials so easy to understand and put into practice.

My understanding is that this model (or any other object detection model) can be trained against any (properly labeled and formatted) dataset in order to detect objects not detected by the available pre-trained models, such as those trained against ImageNet or COCO. If so then is the training process fundamentally different from the type of training that's described in the [tutorial for facial recognition](https://www.pyimagesearch.com/2018/09/24/opencv-face-recognition/)?

Also is there a way to incorporate [fastai](https://www.fast.ai/) into the process? I ask because fastai seems to be next-level over something like Keras, but I don't know enough about it yet to know if it's very relevant or necessary for a model training effort like this.

Kind regards, and thanks again.

Adrian Rosebrock March 27, 2019 at 9:27 am #

REPLY

I've only briefly played around with the fast.ai library. I felt that it abstracted Keras/TensorFlow a bit too much. I think Keras is the perfect level to sit on top of TensorFlow. Plus, if you need any additional TF functionality you can drop down into it from Keras.

As for your training question, yes, provided you have the bounding boxes + labels you can train an object detection network. However, exactly how that network is trained can be very different based on the (1) the network itself and (2) the library that has implemented it.

If you or anyone else is interested in learning more about how to train your own custom deep learning-based object detectors you should definitely read through Deep Learning for Computer Vision with Python where I cover them in detail (including code).

CVee March 23, 2019 at 7:14 pm #

REPLY

Hi Adrian,

this is an awesome tutorial! Both the image and video implementations work great.
One thing I'm not overly excited about though is the speed issue, especially when the video file you're testing is long.

Talking about the speed issue, I have a few questions to ask you:

1) What do I need to change in the code to make it work for real-time object detection on CPU without too much of a delay?
2) If I want to detect just a few objects instead of all the 80 classes the model is trained on, can I just replace the yolov3.weights with a pre-trained weight file from other sources?

I'm going to get your book because I want to learn more about computer vision.

REPLY

Adrian Rosebrock March 27, 2019 at 9:14 am #

You can't use this method of OpenCV + YOLO for true real-time performance on a CPU. Follow this tutorial which uses a SSD with OpenCV to achieve real-time performance.

REPLY

Nicolò March 27, 2019 at 12:43 pm #

Hi Adrian,
Thank for the tutorial, i successfully run on some pics and videos.

I don't understand how you are parsing the output of net.forward().
What is the object 'detection'?
Where can I found the documentation of this output?
The format is the same for every model or this case refers only to YOLO?
I searched within the opencv reference but I can't find it.

Thanks

REPLY

Adrian Rosebrock April 2, 2019 at 6:40 am #

The format of the output volume is dependent on what model you are using. If you take a look at the post + code I have documented what the output variables are. Please give it another read as that should clarify your questions.

REPLY

med March 28, 2019 at 4:40 pm #

hi ; thnks for the code but it takes long times for video ;i hava a raspberry pi 3 b+ ;

Adrian Rosebrock April 2, 2019 at 6:27 am #

REPLY

The Raspberry Pi 3B+ is not suitable for running YOLO as-is. I'll be covering how to run YOLO on the Pi in my upcoming book, stay tuned!

Dina March 29, 2019 at 6:58 am #

REPLY

Your tutorial is very great.
I try and succes.
But i have problem, when the INFO Cleanup will show up ?
Open CV will be show up or we must open the output in folder output ?
I try to write cv2.imshow('frame',frame) after write release. But i haven't get [INFO] Cleanup in my terminal, so i can't showup my opencv.
Could you please help me ???

Adrian Rosebrock April 2, 2019 at 6:16 am #

REPLY

The "cleaning up" is only executed after the script has finished processing all frames in the video file.

chow April 1, 2019 at 1:35 am #

REPLY

I am trying to learn. Can you please help in making a counter when a car passes and also show speed just above the frame.

Adrian Rosebrock April 2, 2019 at 5:55 am #

REPLY

I'll be covering that in my upcoming Computer Vision + Raspberry Pi book, stay tuned!

Emma April 1, 2019 at 6:05 am #

REPLY

Hi, Adrian,
I've read the article "Python, argparse, and command line arguments" but still had problem executing on Jupyter notebook.
Can you give me some hint, please?
Thanks and have a nice day.

Adrian Rosebrock April 2, 2019 at 5:53 am #

REPLY

There is a section in that post that shows you how to modify the code to work with Jupyter Notebooks. Have you given it a try?

sidra April 2, 2019 at 1:50 pm #

REPLY

thankyou sir for sharing code with proper guide. can you please tell me about how to run yolo with open cv in real time.? plz plz, this means alot .it will be a great support .

Adrian Rosebrock April 4, 2019 at 1:33 pm #

REPLY

Please refer to the post and the comments in the post. I have already discussed what would be required to run YOLO in real-time.

ArxivfromFrance April 5, 2019 at 9:13 am #

REPLY

Hi,
thank you, i have two questions,

– Where can i find the official documentation of Yolo-OpenCV
– Is your code Open source ? i mean can i use it for my projects in company ?

Thank you and have a nice day 🙂

Erdal.J April 5, 2019 at 12:32 pm #

REPLY

Hi Adrian,thanks for sharing this kind of things. I know that also we can count the objects detected in video, but I want to know: How we can know in which seconds of video which objects are detected? I need youre suggestioni,thanks 🙂

Samuel Leong April 6, 2019 at 3:08 am #

REPLY

Hi! This is a great article. I'd like to make some observations:

1) In terms of speed, YOLOv2 is almost 3 times faster than YOLOv3 (~0.15 seconds/frame vs ~0.6 seconds/frame). The difference in accuracy (at least in my testing) isn't that great compared to YOLOv2, so use that if you cannot take the slow speed. (Of course, the YOLO2/3-tiny versions are the fastest, but also super inaccurate)

2) From the comments above, it's uncertain if the latest versions of opencv or dlib have GPU support now, but I actually suspect that the answer is YES.

Running YOLO2 on darknet's repo (supposed to use CUDA/GPU) gave me images processed in around 0.15 seconds. When I ran it with no GPU support, it was almost 10s. This python code gave images processed in about 0.2 seconds, so I think that GPU is supported now. (Not sure tho)

Adrian Rosebrock April 12, 2019 at 12:56 pm #

REPLY

The dlib library does have GPU support but I'm not sure why you're mentioning that in this particular post?

OpenCV's "dnn" module supports some GPUs but not NVIDIA Ones. Hopefully that will be changing this summer.

kenneth April 12, 2019 at 4:21 am #

REPLY

"On my machine with a 3GHz Intel Xeon W processor, a single forward pass of YOLO took ~0.3 seconds; however, using a Single Shot Detector (SSD) from a previous tutorial, resulted in only 0.03 second detection, an order of magnitude faster!"

But YOLO claims it has a very high FPS (capable of processing 40-90 FPS on a Titan X GPU. The super fast variant of YOLO can even get up to 155 FPS.)

Why it takes 0.3s (FPS = 3) now?

Thanks….

Adrian Rosebrock April 12, 2019 at 11:18 am #

REPLY

Correct and I discuss that in the post — I believe it's an issue related to the OpenCV implementation.

Mik April 15, 2019 at 12:46 am #

REPLY

Hi! Adrian Thank you for this article the YOLO object detection in images works perfectly fine
But I am having a problem with the the YOLO object detection in video streams. I am encountering an Error in line 168
writer.release()
AttributeError: 'NoneType' object has no attribute 'release'

can you provide an explanation?

Adrian Rosebrock April 18, 2019 at 7:22 am #

REPLY

Did you use the "Downloads" section of this post to download the source code? Or did you copy and paste? It sounds like it may be a copy and paste or accidental error inserted into the code. If you're not writing to video then the "writer" object should be "None". In this case, somehow, the "writer" object thinks it was instantiated. Go back and check any edits you have made to the code.

Wahyu Istanto Bram W April 15, 2019 at 6:09 am #

REPLY

Mr. Rosebrock, do you know python module to draw in laptop / PC screen?
imagine if we open laptop, open browser, watch Youtube, then…the line show up and people think it come from youtube…?

or watching movie in full screen mode and the line show up to? people think it from the movie, but the truth it's come from the screen, so we don't need compute to capture and edit the video , only capture the screen, everything show up in laptop / pc screen will detect (like streaming webcam, then draw the line on the screen.
combine the YOLO…

and so many ways to implement it…CCTV and many others.

the most powerfull if we can make a super mini projector, that can projecting a mini line / retangle on the glass. it's more great than google glas i think.

Adrian Rosebrock April 18, 2019 at 7:15 am #

REPLY

I would suggest trying to use this method.

Elena April 15, 2019 at 12:02 pm #

REPLY

Hi, Adrian! Many thanks – great tutorial!
Just a couple of questions: I get YOLOv3 output images with blue color dominating (everything looks dark+ blue), I understand that opencv transforms images, could you please maybe give a hint on this?

Another question: is it possible to increase output window size?

Thanks a lot in advance!

Elena April 15, 2019 at 1:54 pm #

REPLY

Actually I found answers to my questions :), writing it below, maybe it could be useful for someone:

I am working in Jupyter notebooks, using matplotlib to display output images:

1. To convert BGR to RGB, one more line of code is needed:
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

2. To resize image display window in jupyter notebook:
plt.rcParams['figure.figsize'] = [10, 5] #10,5 – as example only

Thank you, Adrian, for the great tutorial!

Adrian Rosebrock April 18, 2019 at 7:08 am #

REPLY

Congrats on resolving the issue, Elena!

ibrahim April 22, 2019 at 4:21 am #

REPLY

Hi Adrian,

thanks for this post,and its thorough explanation along with your other blog posts! 🙂

i want to ask if cv2.dnn.NMSBoxes() apply Intersection Over Union when selecting the bounding boxes? or if it is applied anywhere in this implementation. if not do you think that applying it would give better results for the bounding boxes and what would be the easiest way to do it?

Palbha April 25, 2019 at 3:33 am #

REPLY

Hi Adrian ,

I am using only CPU and have integrated face detection along with object detection and I know its bound to be slow , is there any way i can limit the objects and might increase the FPS .Please suggest anything that might help increasing the output fps also I want to video to be normal sized like frame = imutils.resize(frame, width=800)

Thanks in advance ,
Palbha

Adrian Rosebrock April 25, 2019 at 8:33 am #

REPLY

If you're using a CPU the only thing you can really do is resize your image dimensions and make the smaller. Otherwise you should look into using a GPU.

AYOUNI aymen April 30, 2019 at 6:33 pm #

REPLY

hello
can you help me with this tasks
how can we count vehicles and class them using YOLO ?
how to measure vehicle speed using YOLO?

Adrian Rosebrock May 1, 2019 at 11:23 am #

REPLY

I'm covering how to detect and compute vehicle speed inside Raspberry Pi for Computer Vision. I would definitely suggest starting there.

AYOUNI aymen May 3, 2019 at 1:09 pm #

REPLY

what about counting traffic vehicles ?
I found OpenCV People Counter in your blog but there's not more information about vehicles ..
can you help me please I really need this

Adrian Rosebrock May 8, 2019 at 1:31 pm #

REPLY

All you need to do is change the people counter class from "person" to "vehicle". Then you can track vehicles.

Adam May 5, 2019 at 1:37 am #

REPLY

Hello
Great work. I am used your sample code and I am able to do detection on my video. However, do you have some sample code or what I need to add to it so it does tracking? I have intersection with cars and I just want to give every car that enters, move and exit its own id. I saw your pople counter example but it does not use Yolo and I was wondering if perhaps you have some code around for that or how I can do it

I am java person, so python is not my strength 🙂 THANK you

Adrian Rosebrock May 8, 2019 at 1:09 pm #

REPLY

The people counter example you're referring to would work here. Just change the class you want to detect from "person" to "vehicle".

Ayush Sahay May 5, 2019 at 3:43 pm #

REPLY

Hey!!
I compiled the video code, and it compiles successfully, but nothing appears on the screen. Basically I wait for 5-6min for code to compiler but after that it terminates and nothing appears on the screen, and I go back to command line(ubuntu). Not really sure what's going on here.

Adrian Rosebrock May 8, 2019 at 1:13 pm #

REPLY

Are you running it on a video file? If so, nothing is supposed to display to your screen. It will generate an output video file for you.

student May 9, 2019 at 6:05 pm #

REPLY

Hello Adrian

I want to use two cameras. I want to detect the object from the first camera and then when the object Is passed from the second camera. I want the YOLO to know this object is the same object. (matches the first object).
How can I do so?
I heard that I need to add temporal info. Do you know any reference, that I can read about?

Adrian Rosebrock May 15, 2019 at 3:16 pm #

REPLY

Sorry, I don't have any tutorials for object tracking across multiple cameras. I'll consider it for a future tutorial but cannot guarantee if/when I will cover it.

rakib May 12, 2019 at 12:49 pm #

REPLY

how can I use it for real-time with a raspberry camera?

Adrian Rosebrock May 15, 2019 at 2:58 pm #

REPLY

No, the RPi is too underpowered to run YOLO object detection in real-time.

Hanna May 12, 2019 at 6:17 pm #

REPLY

Hi adrian, thank you for providing this tutorial. It is really helpful for a newbie like me. I have a request. Could you show tutorial on how to generate caption from traffic images using python?.

Adrian Rosebrock May 15, 2019 at 2:56 pm #

REPLY

Thanks for the suggestion. I'll consider it but I cannot guarantee if/when I will cover it.

mayur May 13, 2019 at 3:50 pm #

REPLY

using yolo on video if i want to get the label of the detected object, is it possible ?
plz let me know if anyone know the way to solve it

mayur May 13, 2019 at 3:54 pm #

REPLY

in text format.

Adrian Rosebrock May 15, 2019 at 2:50 pm #

REPLY

The code already shows you how to do that so perhaps I'm misunderstanding your question?

Edi May 14, 2019 at 9:39 am #

REPLY

Dear Adrian
I've implemented your great work on my raspberry pi3 B+.
but it's extremely slow.it takes near 40s to process only one frame.

Adrian Rosebrock May 15, 2019 at 2:38 pm #

REPLY

Yes, the RPi is far too underpowered for YOLO. You would want to use a Movidius NCS or Google Coral USB Accelerator to speed up inference.

Yashu May 26, 2019 at 9:41 am #

REPLY

Hey how good is yolo for custom object detection? … like if i want to build a apparel detection model will it be able to perform well?

Adrian Rosebrock May 30, 2019 at 9:26 am #

REPLY

In my opinion, I would suggest you use SSD or RetinaNet in place of YOLO. YOLO, while fast on a GPU, can be harder to train and is more prone to false-positive detections. If speed is not an issue then Faster R-CNN would be a good choice as well.

I cover how to train each of those networks inside Deep Learning for Computer Vision with Python.

Sambhavi May 28, 2019 at 7:55 am #

REPLY

Hello Adrian,

As usual, another great blog post. I learn so much of Computer Vision from your blogs and books. Thank you so much.

I was able to successfully detect objects in images and videos using your code. To take it further, I custom trained few other objects which is not pre-trained by yolo. My need is, in any given image, I want to detect the objects that I trained as well as few categories already trained by yolo. Is that possible at all? They are 2 different weight files, in such a case how should I go about it? I did some search and couldn't find any straight forward answer. So I was wondering whether I should re-train those categories which were already trained by yolo along with my custom objects so that my requirement will be handled. You think it makes sense?

Just to make my question clearer, apples and oranges are pre-trained. I trained tomatoes and melons. Now in an image if I want my detector to detect all 4, what would be a better approach.

Thanks in advance.

Adrian Rosebrock May 30, 2019 at 9:15 am #

REPLY

What you want to do is fine-tune the YOLO model. I don't have any tutorials on that, but I cover how to fine-tune object detectors to recognize classes they were not originally trained on inside Deep Learning for Computer Vision with Python.

Jeff Xu June 3, 2019 at 8:14 am #

REPLY

Hi Adrian,
I had read the fine-turn in your book once. I will read it more times later. I also have the same question. If the pre-trained model can detect 100 classes , after fine-turn to train it on my own data(added two calsses), can it detect 102 classed or just the new 2 classes? Thank you.

Adrian Rosebrock June 6, 2019 at 8:02 am #

REPLY

No, it would only detect the 2 new classes.

Rheza Aditya May 29, 2019 at 2:21 pm #

REPLY

Hello Adrian.

This is a great tutorial. However, how can i integrate a live video from my webcam with this method? Thank you in advance!

Adrian Rosebrock May 30, 2019 at 9:04 am #

REPLY

Technically yes, but YOLO is not going to run in real-time using OpenCV's "dnn" module (it's just too computationally expensive).

Phat Dao June 2, 2019 at 9:30 am #

REPLY

Thank you very much Adrian <3

Adrian Rosebrock June 6, 2019 at 8:30 am #

REPLY

You are welcome!

Jeff Xu June 3, 2019 at 5:26 am #

REPLY

Hi Adrian,

Lots of thanks for your great posts as well as your patient replays! Here I have 2 questions that hope you can help answer.

Q1: From the comments, I found yolov2 was faster than yolov2, so I downloaded the yolov2 cfg and weights(https://pjreddie.com/darknet/yolov2/). However, when replaced the yolov3 cfg and weight, although the speed is faster, the labels are wrong(for instance, person will be detected as bird), why?

Q2: When I run the polo_video.py, it costed 95% memory of GPU while occupied 99% of CPUS, why?(I added cv2.imshow() and remove the writer() function)

Thanks in advance!

Adrian Rosebrock June 6, 2019 at 8:03 am #

REPLY

1. Sorry, I'm not sure about that.

2. Make sure you're reading the tutorial and the comments. I've mentioned that OpenCV's "dnn" module does not support NVIDIA GPUs (yet). Hopefully support for them is coming this summer.

student June 3, 2019 at 11:17 pm #

REPLY

Hi Adrian,

Thank you so much for this tutorial.
I am trying to apply this approach further by creating a warning system if objects are in a certain area of the screen (ex. creating a region of

interest for the left and right half of the screen and outputting that an object is detected on the respective side). Im unsure where/how in the code i can implement this.
Any help would be greatly appreciated!

Adrian Rosebrock June 6, 2019 at 6:57 am #

REPLY

Hey there — I actually cover how to build that exact project inside the PyImageSearch Gurus course. I would definitely suggest starting there.

student June 6, 2019 at 11:56 pm #

REPLY

Hello Adrian,

I want to do detection and tracking through video/Image. I will train the model for my own dataset. My project is about a surveillance system. I want to put a camera outside and do detection using camera data. I heard that I would need a raspberry pi or other tools. But I also heard that raspberry pi is not powerful enough to run YOLO in real time. Do you think that I need both an embeded device and pc for training?
I really need an expert to tell me what to buy that it meets my need. I would appreciate if you could tell me what is the best to buy?
What would be your suggestion for choosing devices for my project?

Thanks in advance

Adrian Rosebrock June 12, 2019 at 2:00 pm #

REPLY

Hi there — I cover your exact question inside Raspberry Pi for Computer Vision.

Leave a Reply

Name (required)

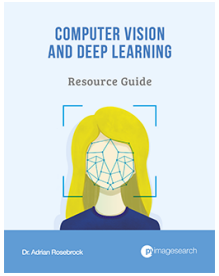
Email (will not be published) (required)

Website

SUBMIT COMMENT

Search...

Resource Guide (it's totally free).



Get your **FREE 17 page Computer Vision, OpenCV, and Deep Learning Resource Guide PDF.** Inside you'll find my hand-picked tutorials, books, courses, and libraries to help you master CV and DL.

Download for Free!

Raspberry Pi for Computer Vision

You can teach your **Raspberry Pi** to “see” using **Computer Vision**, **Deep Learning**, and **OpenCV**. **Let me show you how.**

CLICK HERE TO LEARN MORE

Deep Learning for Computer Vision with Python Book — OUT NOW!

You're interested in deep learning and computer vision, *but you don't know how to get started.* Let me help. **My new book will teach you all you need to know about deep learning.**

CLICK HERE TO MASTER DEEP LEARNING

You can detect faces in images & video.

Are you interested in **detecting faces in images & video?** But **tired of Googling for tutorials** that *never work?* Then let me help! I guarantee that my new book will turn you into a **face detection ninja** by the end of this weekend. Click here to give it a shot yourself.

CLICK HERE TO MASTER FACE DETECTION

PyImageSearch Gurus: NOW ENROLLING!

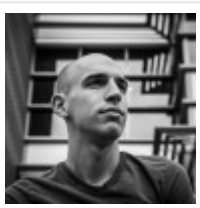
The PyImageSearch Gurus course is *now enrolling!* Inside the course you'll learn how to perform:

- Automatic License Plate Recognition (ANPR)
- Deep Learning
- Face Recognition
- *and much more!*

Click the button below to learn more about the course, take a tour, and get 10 (FREE) sample lessons.

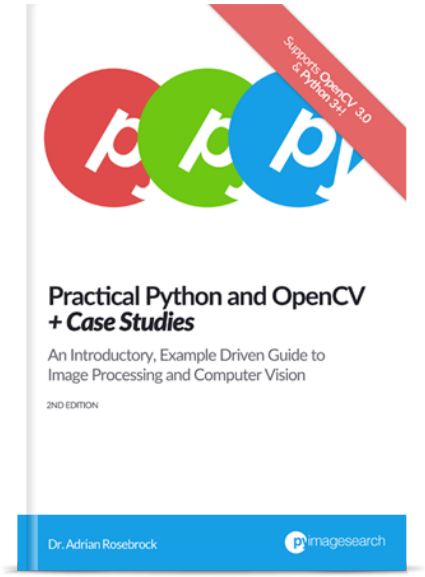
TAKE A TOUR & GET 10 (FREE) LESSONS

Hello! I'm Adrian Rosebrock.



I'm Ph.D and entrepreneur who has spent his entire adult life studying Computer Vision and Deep Learning. I'm here to help you master CV, DL, and OpenCV. Learn More

Learn computer vision in a single weekend.



Want to learn computer vision & OpenCV? I can teach you in a **single weekend**. I know. It sounds crazy, but it's no joke. My new book is your **guaranteed, quick-start guide** to becoming an OpenCV Ninja. So why not give it a try? Click here to become a computer vision ninja.

CLICK HERE TO BECOME AN OPENCV NINJA

Subscribe via RSS

Never miss a post! Subscribe to the PyImageSearch RSS Feed and keep up to date with my image search engine tutorials, tips, and tricks

POPULAR

Raspbian Stretch: Install OpenCV 3 + Python on your Raspberry Pi
SEPTEMBER 4, 2017

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3
APRIL 18, 2016

Home surveillance and motion detection with the Raspberry Pi, Python, OpenCV, and Dropbox
JUNE 1, 2015

Install OpenCV and Python on your Raspberry Pi 2 and B+
FEBRUARY 23, 2015

Face recognition with OpenCV, Python, and deep learning
JUNE 18, 2018

Ubuntu 16.04: How to install OpenCV
OCTOBER 24, 2016

Real-time object detection with deep learning and OpenCV
SEPTEMBER 18, 2017

Find me on **Twitter**, **Facebook**, and **LinkedIn**.
Privacy Policy
© 2019 PyImageSearch. All Rights Reserved.