

YOLO: You Only Look Once is an Object Detection Algorithm that helps us to identify the objects in a given image.

Let's understand YOLO better by diving into the history of Computer Vision starting from CNN to Object Detection and its various algorithms that led to the invention of the YOLO and its successors.

Computer Vision

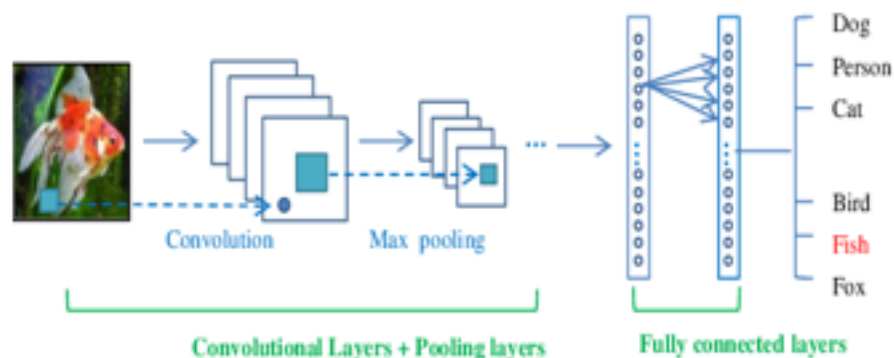
Computer Vision represents a relative understanding of visual environments and their contexts. In other words, they are used to make useful decisions about physical objects and scenes based on sensed images.

Some applications of Computer Vision is as below:

- o Face Recognition
- o Surveillance
- o Biometrics
- o Smart Cars

Convolution Neural Network (CNN)

CNN's are an integral part of applications that support image recognition. They take an input image and output a class (a cat, dog, etc) or a probability of classes that best describes the image.



Design of Convolution Neural Network

For more information on CNN's, please refer to my article at <https://towardsdatascience.com/convolution-neural-network-e9b864ac1e6c>

Object Detection

Object detection focusses on the problem of identifying different objects in a given image. It accomplishes this task by creating a bounding box across each object identified in an image.



CNN's alone are not suitable for object detection as locations of the object in an image are not constant.

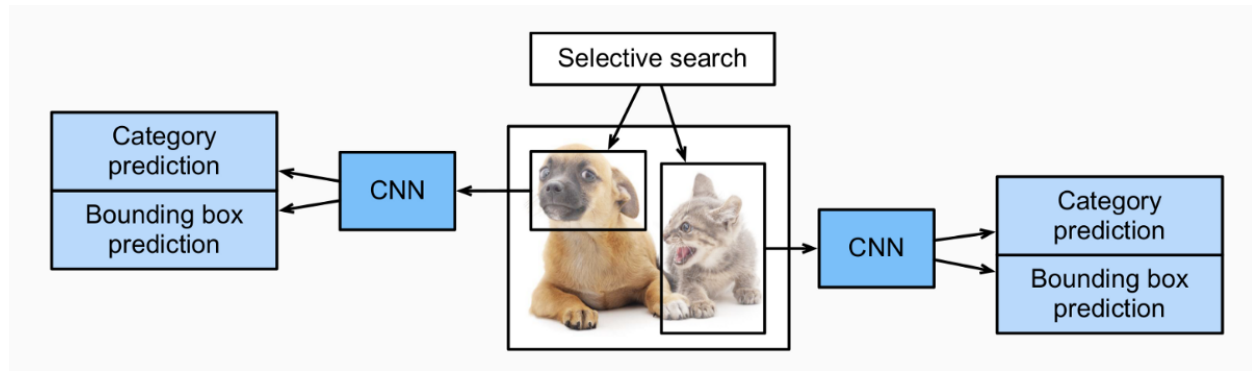
Solving such a problem would result in CNN being applied to a lot of sub-images of a given image (region of interest), that would blow up the computation cost.

R-CNN

R-CNN model selects 2000 regions of interest from a given image using the *Selective Search* algorithm.

Pre-trained CNNs are used to perform forward computations to extract features from each of the proposed regions.

These features are later used to predict the categories and bounding boxes of proposed regions.



R-CNN Model

The main downside of R-CNN is speed. It takes a huge amount of time to train the network as it requires a classification of 2000 region proposals per image.

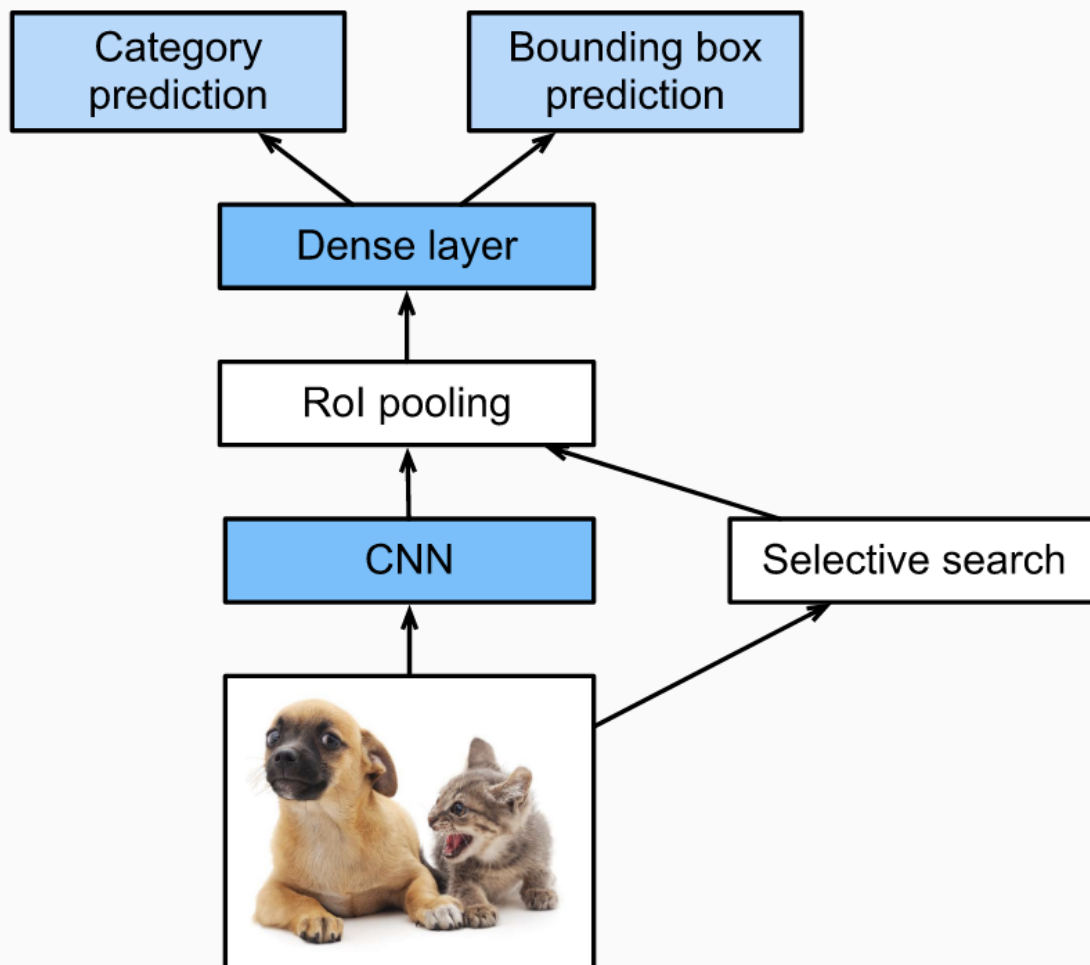
Fast R-CNN

Fast R-CNN improves on R-CNN by performing CNN forward computation on the image as a whole.

A *selective search* algorithm is used to identify 2000 regions of interest of different sizes from the preceding convolutional feature map.

An RoI pooling layer is then used to reshape these extracted features to a common shape that is later fed into a fully connected dense layer.

From the RoI feature vector, we use a softmax layer to predict the class of the proposed region and also the offset values for the bounding box

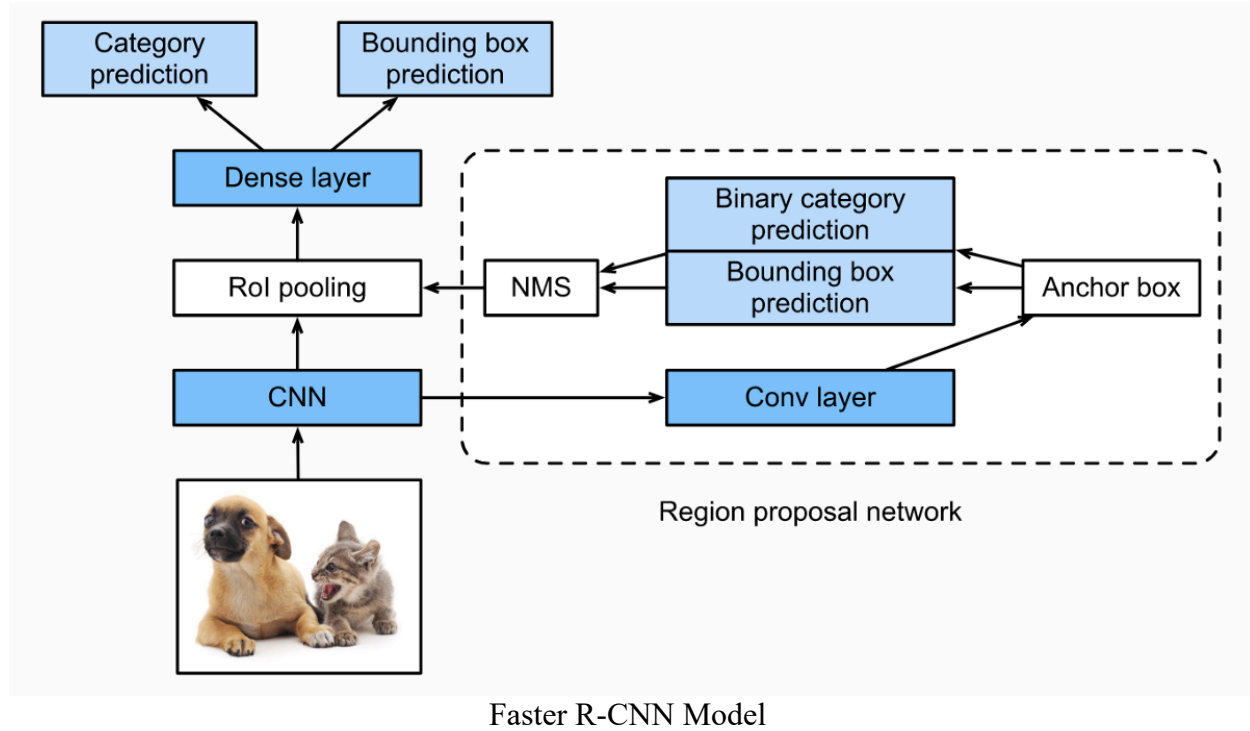


Fast R-CNN Model

The region proposals count becomes a bottleneck in the Fast R-CNN algorithm affecting its performance (speed).

Faster R-CNN

Faster R-CNN replaces the selective search used above with a network (RPN) that learns to identify the region proposals. This reduces the number of proposed regions generated while ensuring precise object detection.

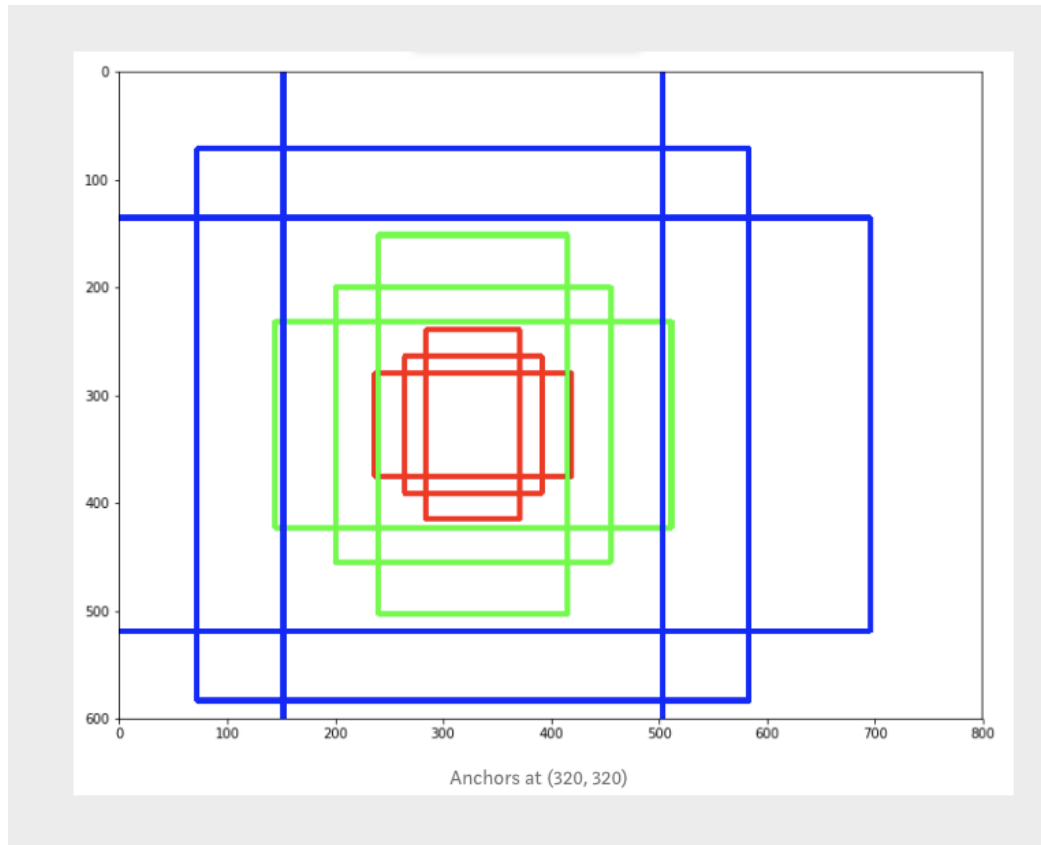


Anchor

An anchor is a box representing a combination of sliding window center, scale, and ratio. For example, 3 scales + 3 ratios => $k=9$ anchors at each sliding position (default configuration).

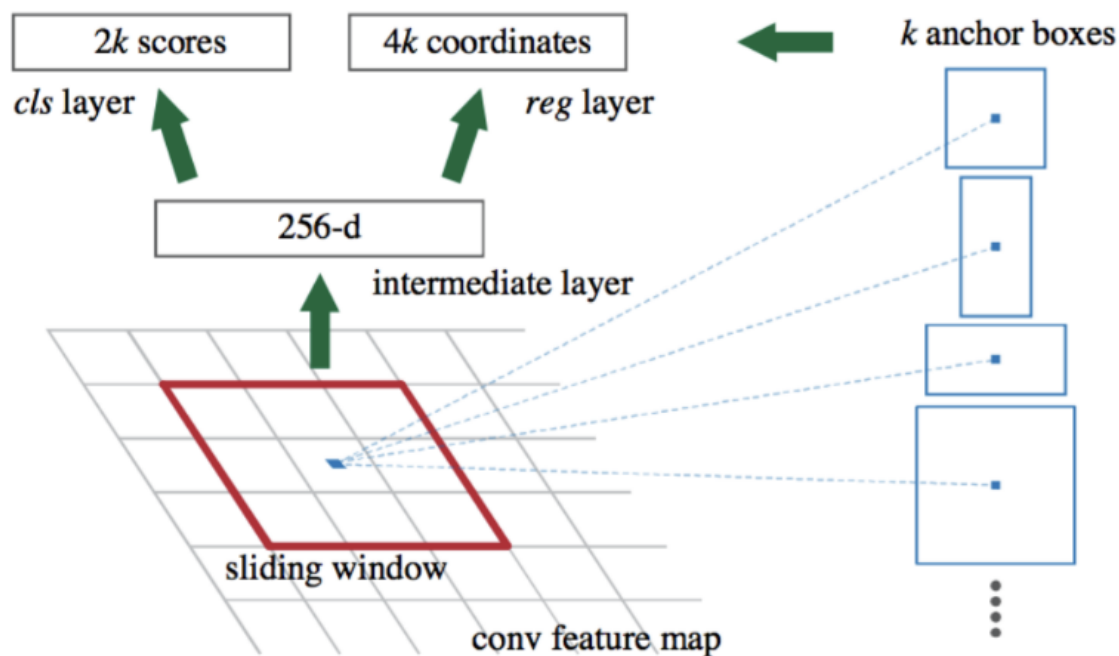
In below image:

- o Three colors represent three scales or sizes: 128x128, 256x256, 512x512.
- o These boxes have height-width ratios 1:1, 1:2 and 2:1 respectively



Region Proposal Network (RPN)

It ranks anchors and proposes the ones most likely containing objects.



RPN Network

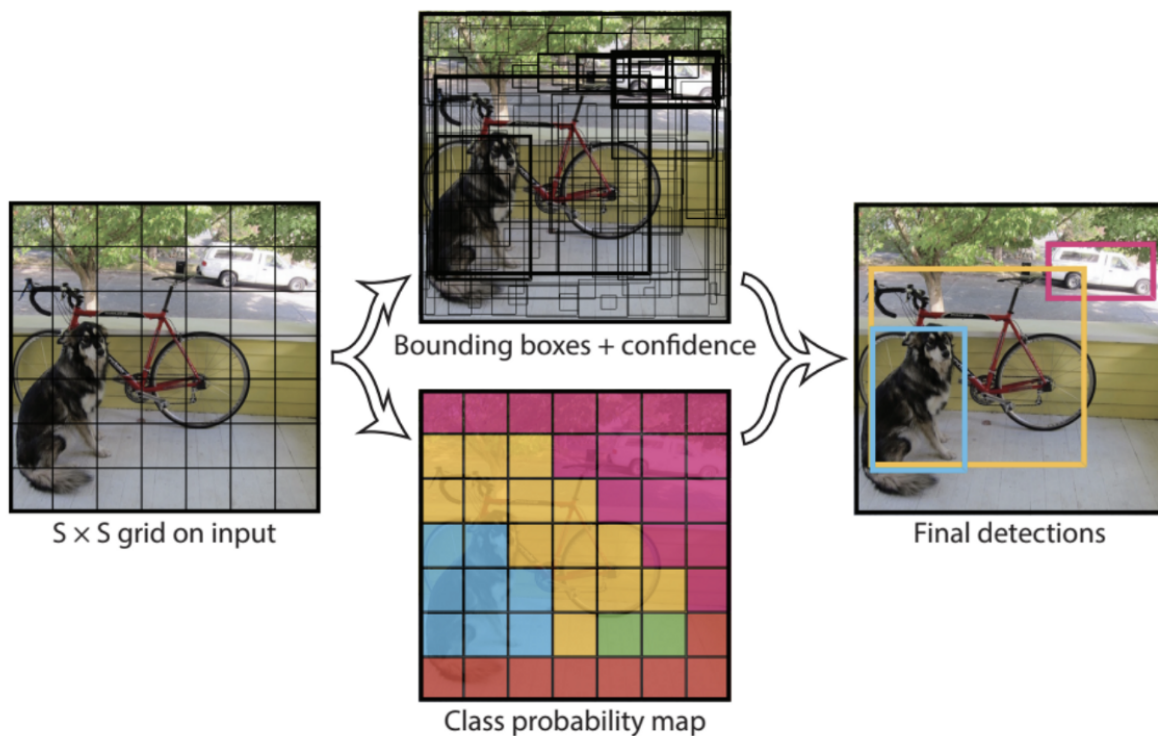
- o Slide a small $n \times n$ spatial window over the conv feature map of the entire image.
- o At the center of each sliding window, we predict multiple regions of various scales and ratios simultaneously using anchors.
- o Each anchor is then classified as background or foreground to predict the possibility of an object within it.
- o The bounding box coordinates of these anchors are then found.

The above process is repeated until the model loss is within an acceptable range.

YOLO

The previous object detection algorithms fail to look at the complete image. It uses regions / partial images to localize the object within the image.

In YOLO, 1 convolutional network predicts the bounding boxes and the class probabilities for these sub-images simultaneously. As a result, it is 45 frames per second faster than other object detection algorithms.



Architecture

- o We take an image and split it into an $S \times S$ grid
- o Each grid has M bounding boxes
- o YOLO outputs a class probability and offset values for each bounding box if the center of an object falls in the grid cell.

o The bounding boxes having class probability above a threshold value is selected and used to locate the object within the image

The limitation of the YOLO algorithm is that it struggles to identify small objects within an image.

For example, it might have difficulties in detecting a flock of birds due to the spatial constraints of the algorithm

YOLO: v1| v2 | v3

YOLO v1	YOLO v2	YOLO v3
1 object / grid	Multi-object / grid	Multi label prediction for each object
2 bounding box / cell: Box shapes are selected manually	5 bounding box / cell: Box shapes are learned during training	3 bounding box / cell: Box shapes are learned during training
	Batch Normalization	Better small object detection
	Multi-Scale Training	
	High Resolution Classifier (448 * 448)	

Custom Data Detection

1. Initial Setup

Follow the below steps to set up the initial environment on your local machine:

```
$ pip install Cython  
$ git clone https://github.com/thtrieu/darkflow.git
```

```
$ cd darkflow
$ python3 setup.py build_ext— inplace
$ pip install .
```

2. Data Crawling

Download 100–200 images each of your desired object detection choice in. /train/images

```
from icrawler.builtin import GoogleImageCrawler

for keyword in ['car', 'motorcycle']:

    google_crawler = GoogleImageCrawler(
        parser_threads=2,
        downloader_threads=4,
        storage={'root_dir': 'images/{}'.format(keyword)}
    )
    google_crawler.crawl(
        keyword=keyword, max_num=200, min_size=(200, 200))|
```

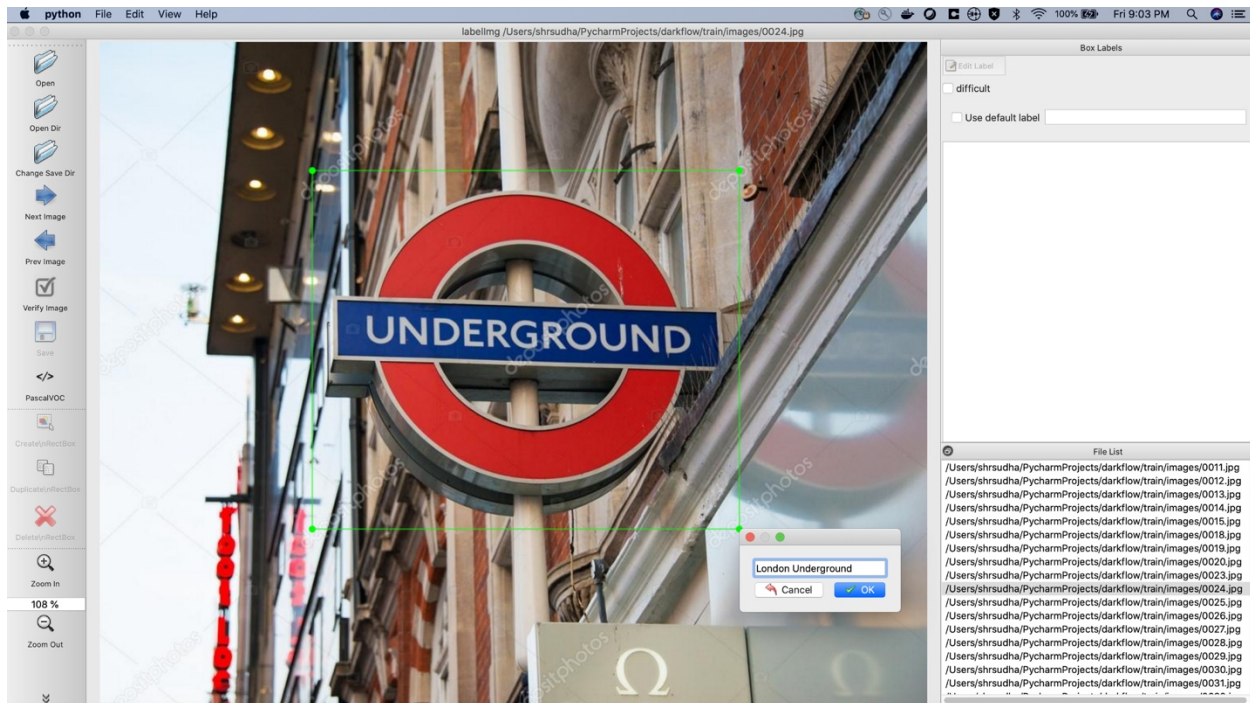
Rename downloaded files in number format to facilitate image identification during training

3. Annotate Data

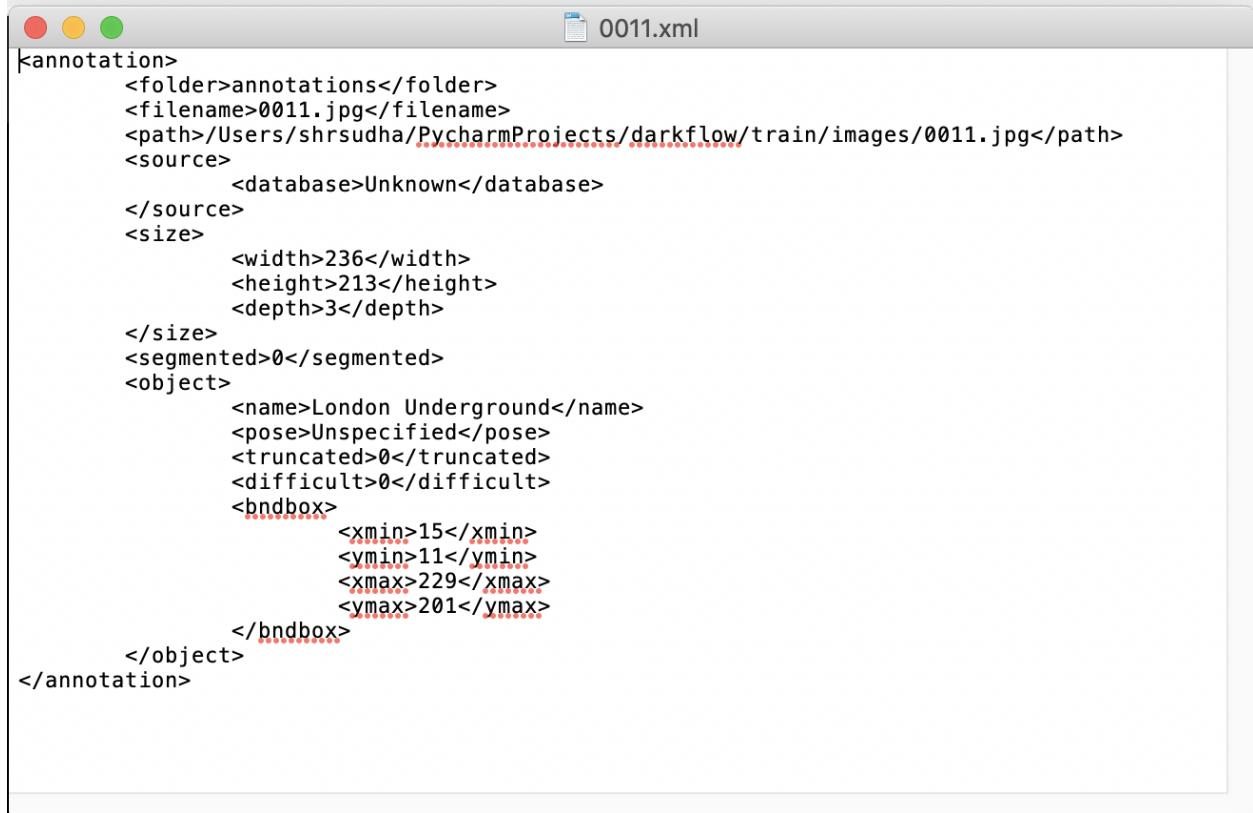
Annotate and save each of the downloaded images in .xml format using LabelImg
in **./train/annotations**

Steps

- o pip install labelImg
- o cd darkflow/train/images
- o labeling



- o File -> Open Dir -> Choose “darkflow/train/images”
- o File -> Change Save Dir -> Choose “darkflow/train/annotation”
- o Click on Create \nRectBox to be able to draw boxes around your desired objects in the selected Image
- o Save the object with desired name (eg., London Underground)
- o Repeat the process for all images present in the File List
- o Open any of the annotated .xml file saved and cross verify that file path points to /darkflow/train/images
- o If above is not true, redo the above LabelImg annotation process to get file path fixed. Process is a bit annoying, but if it's not taken care, training YOLO model won't be successful.



```
<annotation>
  <folder>annotations</folder>
  <filename>0011.jpg</filename>
  <path>/Users/shrsudha/PycharmProjects/darkflow/train/images/0011.jpg</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>236</width>
    <height>213</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>London Underground</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>15</xmin>
      <ymin>11</ymin>
      <xmax>229</xmax>
      <ymax>201</ymax>
    </bndbox>
  </object>
</annotation>
```

Eg.,

Correct:

Image is present in ./train/images

Annotated .xml file is in ./train/annaotation

Path mentioned in .xml file matches that of image present in ./train/images

Error:

Image is present in ./train/images

Annotated .xml file is in ./train/annaotation

But path mentioned in .xml file doesn't match that of image in ./train/images

YouTube Link for LabelImg Tutorial:

https://www.youtube.com/watch?v=_FC6mr7k694

4. Config & Weights

Download YOLO configuration (.cfg) and weights from <https://pjreddie.com/darknet/yolov2/>

Once downloaded, duplicate the config file and rename it.

Make below changes in the renamed config file:

At Top of file:

Width = 288
Height = 288
Batch Size = 64
Sub divisions = 64

At Bottom of file:

Last Conv layer
of filters = 1 ó (# of classes + 5) * 5
Region
Number of Classes = 1

Reducing the height and width of image results in faster training

Here, .cfg file has been renamed based on number of classes (yolo_1c)

5. Training

Train YOLO model from command line using steps below:

```
$ pwd
/Users/*****/PycharmProjects/darkflow/train/images
$ python flow—-model cfg/yolo-1c.cfg—-load bin/yolo.weights—-train—— annotation
train/annotations—-dataset train/images—— epoch 500
.....
.....
.....
step 3498—loss 0.8887977600097656—moving ave loss 0.5651443694831171
step 3499—loss 0.7236082553863525—moving ave loss 0.5809907580734406
step 3500—loss 0.1740107238292694—moving ave loss 0.5402927546490235
```

Checkpoint at step 3500

Finish 500 epoch(es)

Training finished, exit.

Train YOLO model from Jupyter Notebook line using steps mentioned below:

```
from darkflow.net.build import TFNet
import cv2
import tensorflow as tf
```

```
# Config TF, set True if using GPU
config = tf.ConfigProto(log_device_placement = False)
config.gpu_options.allow_growth = False
```

```
# Train YOLO Model

with tf.Session(config=config) as sess:
    options = {
        'model': 'cfg/yolov_1c.cfg',
        'load': '/bin/yyolo.weights',
        'threshold': 0.1,
        'batch': 5,
        'epoch': 500,
        'train': True,
        'annotations': 'train/annotations',
        'dataset': 'train/images'
        #'gpu': 1.0 # uncomment these if using GPU
    }
    tfnet = TFNet(options)
```

Once training is complete, you will see updated weights stored in the folder **ckpt** in form of **.profile** files (eg., yolo_1c-3500.profile)

Common Training Issues Solution

· *Path Not Found*

Define the full path of your training weights (.weights) and configuration file (.cfg)

Eg.,

```
/Users/****/PycharmProjects/darkflow/bin/yolo-1c.cfg  
/Users/****/PycharmProjects/darkflow/bin/yolo.weights
```

· *Offset Error*

Go to ./darkflow/utils/loader.py

Change offset value to from 16 to 20

i.e, self.offset = 16 à self.offset = 20

· *Weight Mismatch*

If you still face issue, download YOLO weights from below Google drive link:

https://drive.google.com/drive/folders/0B1tW_VtY7onidEwyQ2FtQVplWEU

7. Testing

Load trained YOLO model using the below steps in Jupyter notebook.

‘**load**’ refers to the last saved checkpoint during training. Herein, it corresponds to yolo_1c-3500.profile

‘**threshold**’ refers to the confidence level (0–1) above which the current object detected need to be classified as true/false

```

with tf.Session(config=config) as sess:
    options = {
        'model': 'cfg/yolo_1c.cfg',
        'load': 3500,
        'threshold': 0.575,
        #'gpu': 1.0 # uncomment these if using GPU
    }
    tfnet = TFNet(options)

```

```

Parsing /Users/shrsudha/PycharmProjects/darkflow/cfg/yolo_1c.cfg
Loading None ...
Finished in 0.00010609626770019531s

```

Building net ...

WARNING:tensorflow:From /Users/shrsudha/PycharmProjects/darkflow/darkflow/net/bu
s deprecated. Please use tf.compat.v1.placeholder instead.

Source	Train?	Layer description	Output size
--------	--------	-------------------	-------------

-----+-----+-----+-----

WARNING:tensorflow:From /Users/shrsudha/PycharmProjects/darkflow/darkflow/net/op
scope is deprecated. Please use tf.compat.v1.variable_scope instead.

WARNING:tensorflow:From /Users/shrsudha/PycharmProjects/darkflow/darkflow/net/op
ble is deprecated. Please use tf.compat.v1.get_variable instead.

WARNING:tensorflow:From /Users/shrsudha/PycharmProjects/darkflow/darkflow/net/op
er_with_default is deprecated. Please use tf.compat.v1.placeholder_with_default

		input	(?, 288, 288, 3)
Init	Yep!	conv 3x3p1_1 +bnorm leaky	(?, 288, 288, 32)

WARNING:tensorflow:From /Users/shrsudha/PycharmProjects/darkflow/darkflow/net/op
ool is deprecated. Please use tf.nn.max_pool2d instead.

Load	Yep!	maxp 2x2p0_2	(?, 144, 144, 32)
Init	Yep!	conv 3x3p1_1 +bnorm leaky	(?, 144, 144, 64)
Load	Yep!	maxp 2x2p0_2	(?, 72, 72, 64)
Init	Yep!	conv 3x3p1_1 +bnorm leaky	(?, 72, 72, 128)
Init	Yep!	conv 1x1p0 1 +bnorm leaky	(?, 72, 72, 64)

Test YOLO model on a sample image

```

img = cv2.imread('./sample_img/0000.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
results = tfnet.return_predict(img)
print(results)

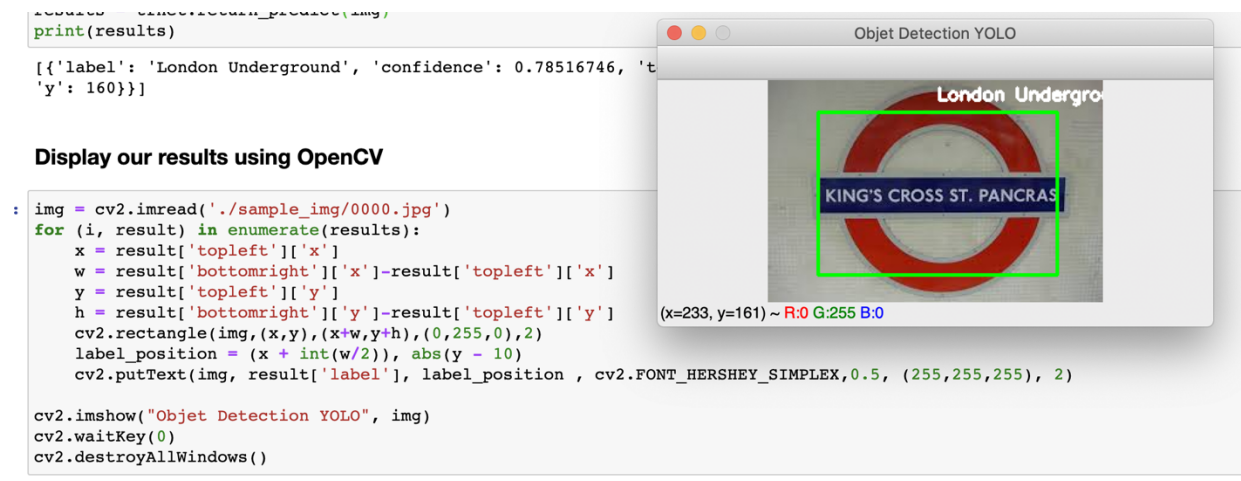
```

```

[{'label': 'London Underground', 'confidence': 0.78516746, 'topleft': {'x': 41, 'y': 26}, 'bottomright': {'x': 238, 'y': 160}}]

```


Display results using OpenCV



8. Code

<https://github.com/shree6791/Springboard>