How to implement a YOLO (v3) object detector from scratch in PyTorch

Part 1: Understanding How YOLO works (https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/)

Github repo (https://github.com/ayooshkathuria/YOLO_v3_tutorial_from_scratch)

Part 2 : Creating the layers of the network architecture (https://blog.paperspace.com/how-to-implement-a-yolo-v3-object-detector-from-scratch-in-pytorch-part-2/)

In [1]:
```
# mkdir cfg
# cd cfg
# wget https://raw.githubusercontent.com/pjreddie/darknet/master/cfg/yolov3.cfg
```

In [2]:
```
from __future__ import division

import time
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.autograd import Variable
import numpy as np
import cv2
import argparse
import os
import os.path as osp
import pickle as pkl
import pandas as pd
import random
```

In [3]:
```
def parse_cfg(cfgfile):
    """
    Takes a configuration file

    Returns a list of blocks. Each blocks describes a block in the neural
    network to be built. Block is represented as a dictionary in the list

    """
    file = open(cfgfile, 'r')
    lines = file.read().split('\n')                        # store the lines in a list
    lines = [x for x in lines if len(x) > 0]               # get read of the empty lines
    lines = [x for x in lines if x[0] != '#']              # get rid of comments
    lines = [x.rstrip().lstrip() for x in lines]           # get rid of fringe whitespaces

    block = {}
    blocks = []

    for line in lines:
        if line[0] == "[":               # This marks the start of a new block
            if len(block) != 0:          # If block is not empty, implies it is storing values of previous block.
                blocks.append(block)     # add it the blocks list
                block = {}               # re-init the block
            block["type"] = line[1:-1].rstrip()
        else:
            key,value = line.split("=")
            block[key.rstrip()] = value.lstrip()
    blocks.append(block)

    return blocks
```

In [4]:
```
class EmptyLayer(nn.Module):
    def __init__(self):
        super(EmptyLayer, self).__init__()
```

In [5]:
```
class DetectionLayer(nn.Module):
    def __init__(self, anchors):
        super(DetectionLayer, self).__init__()
        self.anchors = anchors
```

In [6]:
```
def create_modules(blocks):
    net_info = blocks[0]     #Captures the information about the input and pre-processing
    module_list = nn.ModuleList()
```

```
    prev_filters = 3
    output_filters = []

    for index, x in enumerate(blocks[1:]):
        module = nn.Sequential()

        #check the type of block
        #create a new module for the block
        #append to module_list

        if (x["type"] == "convolutional"):
            #Get the info about the layer
            activation = x["activation"]
            try:
                batch_normalize = int(x["batch_normalize"])
                bias = False
            except:
                batch_normalize = 0
                bias = True

            filters= int(x["filters"])
            padding = int(x["pad"])
            kernel_size = int(x["size"])
            stride = int(x["stride"])

            if padding:
                pad = (kernel_size - 1) // 2
            else:
                pad = 0

            #Add the convolutional layer
            conv = nn.Conv2d(prev_filters, filters, kernel_size, st
ride, pad, bias = bias)
            module.add_module("conv_{0}".format(index), conv)

            #Add the Batch Norm Layer
            if batch_normalize:
                bn = nn.BatchNorm2d(filters)
                module.add_module("batch_norm_{0}".format(index), b
n)

            #Check the activation.
            #It is either Linear or a Leaky ReLU for YOLO
            if activation == "leaky":
                activn = nn.LeakyReLU(0.1, inplace = True)
                module.add_module("leaky_{0}".format(index), activn
)

        #If it's an upsampling layer
        #We use Bilinear2dUpsampling
        elif (x["type"] == "upsample"):
            stride = int(x["stride"])
            upsample = nn.Upsample(scale_factor = 2, mode = "biline
```

```
ar")
            module.add_module("upsample_{}".format(index), upsample
)
        #If it is a route layer
        elif (x["type"] == "route"):
            x["layers"] = x["layers"].split(',')
            #Start  of a route
            start = int(x["layers"][0])
            #end, if there exists one.
            try:
                end = int(x["layers"][1])
            except:
                end = 0
            #Positive anotation
            if start > 0:
                start = start - index
            if end > 0:
                end = end - index
            route = EmptyLayer()
            module.add_module("route_{0}".format(index), route)
            if end < 0:
                filters = output_filters[index + start] + output_fi
lters[index + end]
            else:
                filters= output_filters[index + start]

        #shortcut corresponds to skip connection
        elif x["type"] == "shortcut":
            shortcut = EmptyLayer()
            module.add_module("shortcut_{}".format(index), shortcut
)
        elif x["type"] == "yolo":
            mask = x["mask"].split(",")
            mask = [int(x) for x in mask]

            anchors = x["anchors"].split(",")
            anchors = [int(a) for a in anchors]
            anchors = [(anchors[i], anchors[i+1]) for i in range(0,
len(anchors),2)]
            anchors = [anchors[i] for i in mask]

            detection = DetectionLayer(anchors)
            module.add_module("Detection_{}".format(index), detecti
on)
        module_list.append(module)
        prev_filters = filters
        output_filters.append(filters)
    return (net_info, module_list)
```

In [7]:
```
blocks = parse_cfg("cfg/yolov3.cfg")
print(create_modules(blocks))
```

```
({'type': 'net', 'batch': '64', 'subdivisions': '16', 'width': '41
```

```
6', 'height': '416', 'channels': '3', 'momentum': '0.9', 'decay':
'0.0005', 'angle': '0', 'saturation': '1.5', 'exposure': '1.5', 'h
ue': '.1', 'learning_rate': '0.001', 'burn_in': '1000', 'max_batch
es': '500200', 'policy': 'steps', 'steps': '400000,450000', 'scale
s': '.1,.1'}, ModuleList(
  (0): Sequential(
    (conv_0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), pad
ding=(1, 1), bias=False)
    (batch_norm_0): BatchNorm2d(32, eps=1e-05, momentum=0.1, affin
e=True, track_running_stats=True)
    (leaky_0): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (1): Sequential(
    (conv_1): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), pa
dding=(1, 1), bias=False)
    (batch_norm_1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affin
e=True, track_running_stats=True)
    (leaky_1): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (2): Sequential(
    (conv_2): Conv2d(64, 32, kernel_size=(1, 1), stride=(1, 1), bi
as=False)
    (batch_norm_2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affin
e=True, track_running_stats=True)
    (leaky_2): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (3): Sequential(
    (conv_3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), pa
dding=(1, 1), bias=False)
    (batch_norm_3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affin
e=True, track_running_stats=True)
    (leaky_3): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (4): Sequential(
    (shortcut_4): EmptyLayer()
  )
  (5): Sequential(
    (conv_5): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), p
adding=(1, 1), bias=False)
    (batch_norm_5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affi
ne=True, track_running_stats=True)
    (leaky_5): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (6): Sequential(
    (conv_6): Conv2d(128, 64, kernel_size=(1, 1), stride=(1, 1), b
ias=False)
    (batch_norm_6): BatchNorm2d(64, eps=1e-05, momentum=0.1, affin
e=True, track_running_stats=True)
    (leaky_6): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (7): Sequential(
    (conv_7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), p
adding=(1, 1), bias=False)
```

```
    (batch_norm_7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affi
ne=True, track_running_stats=True)
    (leaky_7): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (8): Sequential(
    (shortcut_8): EmptyLayer()
  )
  (9): Sequential(
    (conv_9): Conv2d(128, 64, kernel_size=(1, 1), stride=(1, 1), b
ias=False)
    (batch_norm_9): BatchNorm2d(64, eps=1e-05, momentum=0.1, affin
e=True, track_running_stats=True)
    (leaky_9): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (10): Sequential(
    (conv_10): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (batch_norm_10): BatchNorm2d(128, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_10): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (11): Sequential(
    (shortcut_11): EmptyLayer()
  )
  (12): Sequential(
    (conv_12): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
    (batch_norm_12): BatchNorm2d(256, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_12): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (13): Sequential(
    (conv_13): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (batch_norm_13): BatchNorm2d(128, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_13): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (14): Sequential(
    (conv_14): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (batch_norm_14): BatchNorm2d(256, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_14): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (15): Sequential(
    (shortcut_15): EmptyLayer()
  )
  (16): Sequential(
    (conv_16): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (batch_norm_16): BatchNorm2d(128, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
```

```
    (leaky_16): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (17): Sequential(
    (conv_17): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (batch_norm_17): BatchNorm2d(256, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_17): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (18): Sequential(
    (shortcut_18): EmptyLayer()
  )
  (19): Sequential(
    (conv_19): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (batch_norm_19): BatchNorm2d(128, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_19): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (20): Sequential(
    (conv_20): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (batch_norm_20): BatchNorm2d(256, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_20): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (21): Sequential(
    (shortcut_21): EmptyLayer()
  )
  (22): Sequential(
    (conv_22): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (batch_norm_22): BatchNorm2d(128, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_22): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (23): Sequential(
    (conv_23): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (batch_norm_23): BatchNorm2d(256, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_23): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (24): Sequential(
    (shortcut_24): EmptyLayer()
  )
  (25): Sequential(
    (conv_25): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (batch_norm_25): BatchNorm2d(128, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_25): LeakyReLU(negative_slope=0.1, inplace=True)
  )
```

```
  (26): Sequential(
    (conv_26): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (batch_norm_26): BatchNorm2d(256, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_26): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (27): Sequential(
    (shortcut_27): EmptyLayer()
  )
  (28): Sequential(
    (conv_28): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (batch_norm_28): BatchNorm2d(128, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_28): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (29): Sequential(
    (conv_29): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (batch_norm_29): BatchNorm2d(256, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_29): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (30): Sequential(
    (shortcut_30): EmptyLayer()
  )
  (31): Sequential(
    (conv_31): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (batch_norm_31): BatchNorm2d(128, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_31): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (32): Sequential(
    (conv_32): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (batch_norm_32): BatchNorm2d(256, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_32): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (33): Sequential(
    (shortcut_33): EmptyLayer()
  )
  (34): Sequential(
    (conv_34): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (batch_norm_34): BatchNorm2d(128, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_34): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (35): Sequential(
    (conv_35): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1),
```

```
padding=(1, 1), bias=False)
    (batch_norm_35): BatchNorm2d(256, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_35): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (36): Sequential(
    (shortcut_36): EmptyLayer()
  )
  (37): Sequential(
    (conv_37): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
    (batch_norm_37): BatchNorm2d(512, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_37): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (38): Sequential(
    (conv_38): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (batch_norm_38): BatchNorm2d(256, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_38): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (39): Sequential(
    (conv_39): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (batch_norm_39): BatchNorm2d(512, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_39): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (40): Sequential(
    (shortcut_40): EmptyLayer()
  )
  (41): Sequential(
    (conv_41): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (batch_norm_41): BatchNorm2d(256, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_41): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (42): Sequential(
    (conv_42): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (batch_norm_42): BatchNorm2d(512, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_42): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (43): Sequential(
    (shortcut_43): EmptyLayer()
  )
  (44): Sequential(
    (conv_44): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (batch_norm_44): BatchNorm2d(256, eps=1e-05, momentum=0.1, aff
```

```
ine=True, track_running_stats=True)
    (leaky_44): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (45): Sequential(
    (conv_45): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (batch_norm_45): BatchNorm2d(512, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_45): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (46): Sequential(
    (shortcut_46): EmptyLayer()
  )
  (47): Sequential(
    (conv_47): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (batch_norm_47): BatchNorm2d(256, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_47): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (48): Sequential(
    (conv_48): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (batch_norm_48): BatchNorm2d(512, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_48): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (49): Sequential(
    (shortcut_49): EmptyLayer()
  )
  (50): Sequential(
    (conv_50): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (batch_norm_50): BatchNorm2d(256, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_50): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (51): Sequential(
    (conv_51): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (batch_norm_51): BatchNorm2d(512, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_51): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (52): Sequential(
    (shortcut_52): EmptyLayer()
  )
  (53): Sequential(
    (conv_53): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (batch_norm_53): BatchNorm2d(256, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_53): LeakyReLU(negative_slope=0.1, inplace=True)
```

```
  )
  (54): Sequential(
    (conv_54): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (batch_norm_54): BatchNorm2d(512, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_54): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (55): Sequential(
    (shortcut_55): EmptyLayer()
  )
  (56): Sequential(
    (conv_56): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (batch_norm_56): BatchNorm2d(256, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_56): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (57): Sequential(
    (conv_57): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (batch_norm_57): BatchNorm2d(512, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_57): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (58): Sequential(
    (shortcut_58): EmptyLayer()
  )
  (59): Sequential(
    (conv_59): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (batch_norm_59): BatchNorm2d(256, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_59): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (60): Sequential(
    (conv_60): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (batch_norm_60): BatchNorm2d(512, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_60): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (61): Sequential(
    (shortcut_61): EmptyLayer()
  )
  (62): Sequential(
    (conv_62): Conv2d(512, 1024, kernel_size=(3, 3), stride=(2, 2)
, padding=(1, 1), bias=False)
    (batch_norm_62): BatchNorm2d(1024, eps=1e-05, momentum=0.1, af
fine=True, track_running_stats=True)
    (leaky_62): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (63): Sequential(
```

```
    (conv_63): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1)
, bias=False)
    (batch_norm_63): BatchNorm2d(512, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_63): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (64): Sequential(
    (conv_64): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1)
, padding=(1, 1), bias=False)
    (batch_norm_64): BatchNorm2d(1024, eps=1e-05, momentum=0.1, af
fine=True, track_running_stats=True)
    (leaky_64): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (65): Sequential(
    (shortcut_65): EmptyLayer()
  )
  (66): Sequential(
    (conv_66): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1)
, bias=False)
    (batch_norm_66): BatchNorm2d(512, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_66): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (67): Sequential(
    (conv_67): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1)
, padding=(1, 1), bias=False)
    (batch_norm_67): BatchNorm2d(1024, eps=1e-05, momentum=0.1, af
fine=True, track_running_stats=True)
    (leaky_67): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (68): Sequential(
    (shortcut_68): EmptyLayer()
  )
  (69): Sequential(
    (conv_69): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1)
, bias=False)
    (batch_norm_69): BatchNorm2d(512, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_69): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (70): Sequential(
    (conv_70): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1)
, padding=(1, 1), bias=False)
    (batch_norm_70): BatchNorm2d(1024, eps=1e-05, momentum=0.1, af
fine=True, track_running_stats=True)
    (leaky_70): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (71): Sequential(
    (shortcut_71): EmptyLayer()
  )
  (72): Sequential(
    (conv_72): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1)
, bias=False)
```

```
    (batch_norm_72): BatchNorm2d(512, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_72): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (73): Sequential(
    (conv_73): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1)
, padding=(1, 1), bias=False)
    (batch_norm_73): BatchNorm2d(1024, eps=1e-05, momentum=0.1, af
fine=True, track_running_stats=True)
    (leaky_73): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (74): Sequential(
    (shortcut_74): EmptyLayer()
  )
  (75): Sequential(
    (conv_75): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1)
, bias=False)
    (batch_norm_75): BatchNorm2d(512, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_75): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (76): Sequential(
    (conv_76): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1)
, padding=(1, 1), bias=False)
    (batch_norm_76): BatchNorm2d(1024, eps=1e-05, momentum=0.1, af
fine=True, track_running_stats=True)
    (leaky_76): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (77): Sequential(
    (conv_77): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1)
, bias=False)
    (batch_norm_77): BatchNorm2d(512, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_77): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (78): Sequential(
    (conv_78): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1)
, padding=(1, 1), bias=False)
    (batch_norm_78): BatchNorm2d(1024, eps=1e-05, momentum=0.1, af
fine=True, track_running_stats=True)
    (leaky_78): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (79): Sequential(
    (conv_79): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1)
, bias=False)
    (batch_norm_79): BatchNorm2d(512, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_79): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (80): Sequential(
    (conv_80): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1)
, padding=(1, 1), bias=False)
    (batch_norm_80): BatchNorm2d(1024, eps=1e-05, momentum=0.1, af
```

```
fine=True, track_running_stats=True)
    (leaky_80): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (81): Sequential(
    (conv_81): Conv2d(1024, 255, kernel_size=(1, 1), stride=(1, 1)
)
  )
  (82): Sequential(
    (Detection_82): DetectionLayer()
  )
  (83): Sequential(
    (route_83): EmptyLayer()
  )
  (84): Sequential(
    (conv_84): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (batch_norm_84): BatchNorm2d(256, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_84): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (85): Sequential(
    (upsample_85): Upsample(scale_factor=2.0, mode=bilinear)
  )
  (86): Sequential(
    (route_86): EmptyLayer()
  )
  (87): Sequential(
    (conv_87): Conv2d(768, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (batch_norm_87): BatchNorm2d(256, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_87): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (88): Sequential(
    (conv_88): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (batch_norm_88): BatchNorm2d(512, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_88): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (89): Sequential(
    (conv_89): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (batch_norm_89): BatchNorm2d(256, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_89): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (90): Sequential(
    (conv_90): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (batch_norm_90): BatchNorm2d(512, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_90): LeakyReLU(negative_slope=0.1, inplace=True)
```

```
  )
  (91): Sequential(
    (conv_91): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (batch_norm_91): BatchNorm2d(256, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_91): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (92): Sequential(
    (conv_92): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (batch_norm_92): BatchNorm2d(512, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_92): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (93): Sequential(
    (conv_93): Conv2d(512, 255, kernel_size=(1, 1), stride=(1, 1))
  )
  (94): Sequential(
    (Detection_94): DetectionLayer()
  )
  (95): Sequential(
    (route_95): EmptyLayer()
  )
  (96): Sequential(
    (conv_96): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (batch_norm_96): BatchNorm2d(128, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_96): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (97): Sequential(
    (upsample_97): Upsample(scale_factor=2.0, mode=bilinear)
  )
  (98): Sequential(
    (route_98): EmptyLayer()
  )
  (99): Sequential(
    (conv_99): Conv2d(384, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (batch_norm_99): BatchNorm2d(128, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
    (leaky_99): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (100): Sequential(
    (conv_100): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1)
, padding=(1, 1), bias=False)
    (batch_norm_100): BatchNorm2d(256, eps=1e-05, momentum=0.1, af
fine=True, track_running_stats=True)
    (leaky_100): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (101): Sequential(
    (conv_101): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1)
```

```
, bias=False)
    (batch_norm_101): BatchNorm2d(128, eps=1e-05, momentum=0.1, af
fine=True, track_running_stats=True)
    (leaky_101): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (102): Sequential(
    (conv_102): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1)
, padding=(1, 1), bias=False)
    (batch_norm_102): BatchNorm2d(256, eps=1e-05, momentum=0.1, af
fine=True, track_running_stats=True)
    (leaky_102): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (103): Sequential(
    (conv_103): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1)
, bias=False)
    (batch_norm_103): BatchNorm2d(128, eps=1e-05, momentum=0.1, af
fine=True, track_running_stats=True)
    (leaky_103): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (104): Sequential(
    (conv_104): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1)
, padding=(1, 1), bias=False)
    (batch_norm_104): BatchNorm2d(256, eps=1e-05, momentum=0.1, af
fine=True, track_running_stats=True)
    (leaky_104): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (105): Sequential(
    (conv_105): Conv2d(256, 255, kernel_size=(1, 1), stride=(1, 1)
)
  )
  (106): Sequential(
    (Detection_106): DetectionLayer()
  )
))
```

Part 3 : Implementing the the forward pass of the network (https://blog.paperspace.com/how-to-implement-a-yolo-v3-object-detector-from-scratch-in-pytorch-part-3/)

In [8]:
```
class Darknet(nn.Module):
    def __init__(self, cfgfile):
        super(Darknet, self).__init__()
        self.blocks = parse_cfg(cfgfile)
        self.net_info, self.module_list = create_modules(self.block
s)

    def forward(self, x, CUDA):
        modules = self.blocks[1:]
        outputs = {}   #We cache the outputs for the route layer
```

```
        write = 0
        for i, module in enumerate(modules):
            module_type = (module["type"])

            if module_type == "convolutional" or module_type == "up
sample":
                x = self.module_list[i](x)

            elif module_type == "route":
                layers = module["layers"]
                layers = [int(a) for a in layers]

                if (layers[0]) > 0:
                    layers[0] = layers[0] - i

                if len(layers) == 1:
                    x = outputs[i + (layers[0])]

                else:
                    if (layers[1]) > 0:
                        layers[1] = layers[1] - i

                    map1 = outputs[i + layers[0]]
                    map2 = outputs[i + layers[1]]
                    x = torch.cat((map1, map2), 1)


            elif  module_type == "shortcut":
                from_ = int(module["from"])
                x = outputs[i-1] + outputs[i+from_]

            elif module_type == 'yolo':
                anchors = self.module_list[i][0].anchors
                #Get the input dimensions
                inp_dim = int (self.net_info["height"])

                #Get the number of classes
                num_classes = int (module["classes"])

                #Transform
                x = x.data
                x = predict_transform(x, inp_dim, anchors, num_clas
ses, CUDA)
                if not write:              #if no collector has bee
n intialised.
                    detections = x
                    write = 1

                else:
                    detections = torch.cat((detections, x), 1)

            outputs[i] = x
```

```
        return detections

    def load_weights(self, weightfile):
        #Open the weights file
        fp = open(weightfile, "rb")

        #The first 5 values are header information
        # 1. Major version number
        # 2. Minor Version Number
        # 3. Subversion number
        # 4,5. Images seen by the network (during training)
        header = np.fromfile(fp, dtype = np.int32, count = 5)
        self.header = torch.from_numpy(header)
        self.seen = self.header[3]

        weights = np.fromfile(fp, dtype = np.float32)

        ptr = 0
        for i in range(len(self.module_list)):
            module_type = self.blocks[i + 1]["type"]

            #If module_type is convolutional load weights
            #Otherwise ignore.

            if module_type == "convolutional":
                model = self.module_list[i]
                try:
                    batch_normalize = int(self.blocks[i+1]["batch_n
ormalize"])
                except:
                    batch_normalize = 0

                conv = model[0]


                if (batch_normalize):
                    bn = model[1]

                    #Get the number of weights of Batch Norm Layer
                    num_bn_biases = bn.bias.numel()

                    #Load the weights
                    bn_biases = torch.from_numpy(weights[ptr:ptr +
num_bn_biases])
                    ptr += num_bn_biases

                    bn_weights = torch.from_numpy(weights[ptr: ptr
+ num_bn_biases])
                    ptr  += num_bn_biases

                    bn_running_mean = torch.from_numpy(weights[ptr:
ptr + num_bn_biases])
                    ptr  += num_bn_biases
```

```
                    bn_running_var = torch.from_numpy(weights[ptr:
ptr + num_bn_biases])
                    ptr  += num_bn_biases

                    #Cast the loaded weights into dims of model wei
ghts.
                    bn_biases = bn_biases.view_as(bn.bias.data)
                    bn_weights = bn_weights.view_as(bn.weight.data)
                    bn_running_mean = bn_running_mean.view_as(bn.ru
nning_mean)
                    bn_running_var = bn_running_var.view_as(bn.runn
ing_var)

                    #Copy the data to model
                    bn.bias.data.copy_(bn_biases)
                    bn.weight.data.copy_(bn_weights)
                    bn.running_mean.copy_(bn_running_mean)
                    bn.running_var.copy_(bn_running_var)

                else:
                    #Number of biases
                    num_biases = conv.bias.numel()

                    #Load the weights
                    conv_biases = torch.from_numpy(weights[ptr: ptr
+ num_biases])
                    ptr = ptr + num_biases

                    #reshape the loaded weights according to the di
ms of the model weights
                    conv_biases = conv_biases.view_as(conv.bias.dat
a)

                    #Finally copy the data
                    conv.bias.data.copy_(conv_biases)

                #Let us load the weights for the Convolutional laye
rs
                num_weights = conv.weight.numel()

                #Do the same as above for weights
                conv_weights = torch.from_numpy(weights[ptr:ptr+num
_weights])
                ptr = ptr + num_weights

                conv_weights = conv_weights.view_as(conv.weight.dat
a)
                conv.weight.data.copy_(conv_weights)
```

In [9]:
```
def predict_transform(prediction, inp_dim, anchors, num_classes, CU
DA = True):

    batch_size = prediction.size(0)
    stride =  inp_dim // prediction.size(2)
    grid_size = inp_dim // stride
    bbox_attrs = 5 + num_classes
    num_anchors = len(anchors)

    prediction = prediction.view(batch_size, bbox_attrs*num_anchors
, grid_size*grid_size)
    prediction = prediction.transpose(1,2).contiguous()
    prediction = prediction.view(batch_size, grid_size*grid_size*nu
m_anchors, bbox_attrs)
    anchors = [(a[0]/stride, a[1]/stride) for a in anchors]

    #Sigmoid the  centre_X, centre_Y. and object confidencce
    prediction[:,:,0] = torch.sigmoid(prediction[:,:,0])
    prediction[:,:,1] = torch.sigmoid(prediction[:,:,1])
    prediction[:,:,4] = torch.sigmoid(prediction[:,:,4])

    #Add the center offsets
    grid = np.arange(grid_size)
    a,b = np.meshgrid(grid, grid)

    x_offset = torch.FloatTensor(a).view(-1,1)
    y_offset = torch.FloatTensor(b).view(-1,1)

    if CUDA:
        x_offset = x_offset.cuda()
        y_offset = y_offset.cuda()

    x_y_offset = torch.cat((x_offset, y_offset), 1).repeat(1,num_an
chors).view(-1,2).unsqueeze(0)

    prediction[:,:,:2] += x_y_offset

    #log space transform height and the width
    anchors = torch.FloatTensor(anchors)

    if CUDA:
        anchors = anchors.cuda()

    anchors = anchors.repeat(grid_size*grid_size, 1).unsqueeze(0)
    prediction[:,:,2:4] = torch.exp(prediction[:,:,2:4])*anchors

    prediction[:,:,5: 5 + num_classes] = torch.sigmoid((prediction[
:,:, 5 : 5 + num_classes]))

    prediction[:,:,:4] *= stride

    return prediction
```

In [10]:
```
# wget https://github.com/ayooshkathuria/pytorch-yolo-v3/raw/master/dog-cycle-car.png
```

In [11]:
```
def get_test_input():
    img = cv2.imread("img/dog-cycle-car.png")
    img = cv2.resize(img, (416, 416))          #Resize to the input dimension
    img_ = img[:,:,::-1].transpose((2,0,1))    #BGR -> RGB | H X W C -> C X H X W
    img_ = img_[np.newaxis,:,:,:]/255.0        #Add a channel at 0 (for batch) | Normalise
    img_ = torch.from_numpy(img_).float()      #Convert to float
    img_ = Variable(img_)                      #Convert to Variable
    return img_
```

In [12]:
```
model = Darknet("cfg/yolov3.cfg")
inp = get_test_input()
print(inp.shape)
pred = model(inp, torch.cuda.is_available())
print (pred)
```

```
torch.Size([1, 3, 416, 416])
tensor([[[1.6675e+01, 1.4159e+01, 6.8440e+01,  ..., 4.1797e-01,
          5.1057e-01, 5.2733e-01],
         [1.3412e+01, 1.5306e+01, 1.4919e+02,  ..., 5.8069e-01,
          4.8104e-01, 5.1910e-01],
         [1.8618e+01, 1.9154e+01, 3.1965e+02,  ..., 4.8069e-01,
          5.4098e-01, 5.5036e-01],
         ...,
         [4.1208e+02, 4.1248e+02, 1.1816e+01,  ..., 5.2179e-01,
          5.2875e-01, 4.2780e-01],
         [4.1234e+02, 4.1198e+02, 1.7884e+01,  ..., 5.7071e-01,
          4.9250e-01, 5.0524e-01],
         [4.1318e+02, 4.1198e+02, 5.2446e+01,  ..., 3.9043e-01,
          4.6008e-01, 5.2686e-01]]])

/Users/chanho/miniconda3/envs/yolo/lib/python3.8/site-packages/torch/nn/functional.py:2503: UserWarning: Default upsampling behavior when mode=bilinear is changed to align_corners=False since 0.4.0. Please specify align_corners=True if the old behavior is desired. See the documentation of nn.Upsample for details.
  warnings.warn("Default upsampling behavior when mode={} is changed "
```

In [13]:
```
# wget https://pjreddie.com/media/files/yolov3.weights
```

In [14]:
```
model = Darknet("cfg/yolov3.cfg")
model.load_weights("yolov3.weights")
```

In [15]:
```
def bbox_iou(box1, box2):
    """
    Returns the IoU of two bounding boxes


    """
    #Get the coordinates of bounding boxes
    b1_x1, b1_y1, b1_x2, b1_y2 = box1[:,0], box1[:,1], box1[:,2], b
ox1[:,3]
    b2_x1, b2_y1, b2_x2, b2_y2 = box2[:,0], box2[:,1], box2[:,2], b
ox2[:,3]

    #get the corrdinates of the intersection rectangle
    inter_rect_x1 =  torch.max(b1_x1, b2_x1)
    inter_rect_y1 =  torch.max(b1_y1, b2_y1)
    inter_rect_x2 =  torch.min(b1_x2, b2_x2)
    inter_rect_y2 =  torch.min(b1_y2, b2_y2)

    #Intersection area
    inter_area = torch.clamp(inter_rect_x2 - inter_rect_x1 + 1, min
=0) * torch.clamp(inter_rect_y2 - inter_rect_y1 + 1, min=0)

    #Union Area
    b1_area = (b1_x2 - b1_x1 + 1)*(b1_y2 - b1_y1 + 1)
    b2_area = (b2_x2 - b2_x1 + 1)*(b2_y2 - b2_y1 + 1)

    iou = inter_area / (b1_area + b2_area - inter_area)

    return iou
```

Part 4 : Objectness score thresholding and Non-maximum suppression (https://blog.paperspace.com/how-to-implement-a-yolo-v3-object-detector-from-scratch-in-pytorch-part-4/)

In [16]:
```
def write_results(prediction, confidence, num_classes, nms_conf = 0
.4):
    conf_mask = (prediction[:,:,4] > confidence).float().unsqueeze(
2)
    prediction = prediction*conf_mask

    box_corner = prediction.new(prediction.shape)
    box_corner[:,:,0] = (prediction[:,:,0] - prediction[:,:,2]/2)
    box_corner[:,:,1] = (prediction[:,:,1] - prediction[:,:,3]/2)
    box_corner[:,:,2] = (prediction[:,:,0] + prediction[:,:,2]/2)
    box_corner[:,:,3] = (prediction[:,:,1] + prediction[:,:,3]/2)
    prediction[:,:,:4] = box_corner[:,:,:4]

    batch_size = prediction.size(0)
```

```
    write = False


    for ind in range(batch_size):
        image_pred = prediction[ind]          #image Tensor
       #confidence threshholding
       #NMS

        max_conf, max_conf_score = torch.max(image_pred[:,5:5+ num_
classes], 1)
        max_conf = max_conf.float().unsqueeze(1)
        max_conf_score = max_conf_score.float().unsqueeze(1)
        seq = (image_pred[:,:5], max_conf, max_conf_score)
        image_pred = torch.cat(seq, 1)

        non_zero_ind =  (torch.nonzero(image_pred[:,4]))
        try:
            image_pred_ = image_pred[non_zero_ind.squeeze(),:].view
(-1,7)
        except:
            continue

        if image_pred_.shape[0] == 0:
            continue
#

        #Get the various classes detected in the image
        img_classes = unique(image_pred_[:,-1])  # -1 index holds t
he class index


        for cls in img_classes:
            #perform NMS


            #get the detections with one particular class
            cls_mask = image_pred_*(image_pred_[:,-1] == cls).float
().unsqueeze(1)
            class_mask_ind = torch.nonzero(cls_mask[:,-2]).squeeze(
)
            image_pred_class = image_pred_[class_mask_ind].view(-1,
7)

            #sort the detections such that the entry with the maxim
um objectness
            #confidence is at the top
            conf_sort_index = torch.sort(image_pred_class[:,4], des
cending = True )[1]
            image_pred_class = image_pred_class[conf_sort_index]
            idx = image_pred_class.size(0)   #Number of detections
```

```
            for i in range(idx):
                #Get the IOUs of all boxes that come after the one
we are looking at
                #in the loop
                try:
                    ious = bbox_iou(image_pred_class[i].unsqueeze(0
), image_pred_class[i+1:])
                except ValueError:
                    break

                except IndexError:
                    break

                #Zero out all the detections that have IoU > treshh
old
                iou_mask = (ious < nms_conf).float().unsqueeze(1)
                image_pred_class[i+1:] *= iou_mask

                #Remove the non-zero entries
                non_zero_ind = torch.nonzero(image_pred_class[:,4])
.squeeze()
                image_pred_class = image_pred_class[non_zero_ind].v
iew(-1,7)

            batch_ind = image_pred_class.new(image_pred_class.size(
0), 1).fill_(ind)      #Repeat the batch_id for as many detections
of the class cls in the image
            seq = batch_ind, image_pred_class

            if not write:
                output = torch.cat(seq,1)
                write = True
            else:
                out = torch.cat(seq,1)
                output = torch.cat((output,out))

    try:
        return output
    except:
        return 0
```

Part 5 : Designing the input and the output pipelines (https://blog.paperspace.com/how-to-implement-a-yolo-v3-object-detector-from-scratch-in-pytorch-part-5/)

In [17]:
```
img_dir = 'img'
det_dir = 'det'
batch_size = 1
confidence = 0.5
nms_thesh = 0.4
cfgfile = 'cfg/yolov3.cfg'
weightsfile = 'yolov3.weights'
start = 0
reso = 416
CUDA = torch.cuda.is_available()
```

In [18]:
```
# mkdir data
# cd data
# wget https://raw.githubusercontent.com/ayooshkathuria/YOLO_v3_tut
orial_from_scratch/master/data/coco.names
```

In [19]:
```
def load_classes(namesfile):
    fp = open(namesfile, "r")
    names = fp.read().split("\n")[:-1]
    return names
```

In [20]:
```
num_classes = 80    #For COCO
classes = load_classes("data/coco.names")
```

In [21]:
```
#Set up the neural network
print("Loading network.....")
model = Darknet(cfgfile)
model.load_weights(weightsfile)
print("Network successfully loaded")

model.net_info["height"] = reso
inp_dim = int(model.net_info["height"])
assert inp_dim % 32 == 0
assert inp_dim > 32

#If there's a GPU availible, put the model on GPU
if CUDA:
    model.cuda()

#Set the model in evaluation mode
model.eval()
```

```
Loading network.....
Network successfully loaded
```

Out[21]:
```
Darknet(
  (module_list): ModuleList(
    (0): Sequential(
      (conv_0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), p
adding=(1, 1), bias=False)
      (batch_norm_0): BatchNorm2d(32, eps=1e-05, momentum=0.1, aff
```

```
ine=True, track_running_stats=True)
      (leaky_0): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (1): Sequential(
      (conv_1): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
      (batch_norm_1): BatchNorm2d(64, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
      (leaky_1): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (2): Sequential(
      (conv_2): Conv2d(64, 32, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (batch_norm_2): BatchNorm2d(32, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
      (leaky_2): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (3): Sequential(
      (conv_3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (batch_norm_3): BatchNorm2d(64, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
      (leaky_3): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (4): Sequential(
      (shortcut_4): EmptyLayer()
    )
    (5): Sequential(
      (conv_5): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
      (batch_norm_5): BatchNorm2d(128, eps=1e-05, momentum=0.1, af
fine=True, track_running_stats=True)
      (leaky_5): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (6): Sequential(
      (conv_6): Conv2d(128, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (batch_norm_6): BatchNorm2d(64, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
      (leaky_6): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (7): Sequential(
      (conv_7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (batch_norm_7): BatchNorm2d(128, eps=1e-05, momentum=0.1, af
fine=True, track_running_stats=True)
      (leaky_7): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (8): Sequential(
      (shortcut_8): EmptyLayer()
    )
    (9): Sequential(
      (conv_9): Conv2d(128, 64, kernel_size=(1, 1), stride=(1, 1),
```

```
bias=False)
      (batch_norm_9): BatchNorm2d(64, eps=1e-05, momentum=0.1, aff
ine=True, track_running_stats=True)
      (leaky_9): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (10): Sequential(
      (conv_10): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1)
, padding=(1, 1), bias=False)
      (batch_norm_10): BatchNorm2d(128, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_10): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (11): Sequential(
      (shortcut_11): EmptyLayer()
    )
    (12): Sequential(
      (conv_12): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2
), padding=(1, 1), bias=False)
      (batch_norm_12): BatchNorm2d(256, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_12): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (13): Sequential(
      (conv_13): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1
), bias=False)
      (batch_norm_13): BatchNorm2d(128, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_13): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (14): Sequential(
      (conv_14): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1
), padding=(1, 1), bias=False)
      (batch_norm_14): BatchNorm2d(256, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_14): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (15): Sequential(
      (shortcut_15): EmptyLayer()
    )
    (16): Sequential(
      (conv_16): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1
), bias=False)
      (batch_norm_16): BatchNorm2d(128, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_16): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (17): Sequential(
      (conv_17): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1
), padding=(1, 1), bias=False)
      (batch_norm_17): BatchNorm2d(256, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_17): LeakyReLU(negative_slope=0.1, inplace=True)
    )
```

```
    (18): Sequential(
      (shortcut_18): EmptyLayer()
    )
    (19): Sequential(
      (conv_19): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1
), bias=False)
      (batch_norm_19): BatchNorm2d(128, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_19): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (20): Sequential(
      (conv_20): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1
), padding=(1, 1), bias=False)
      (batch_norm_20): BatchNorm2d(256, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_20): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (21): Sequential(
      (shortcut_21): EmptyLayer()
    )
    (22): Sequential(
      (conv_22): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1
), bias=False)
      (batch_norm_22): BatchNorm2d(128, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_22): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (23): Sequential(
      (conv_23): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1
), padding=(1, 1), bias=False)
      (batch_norm_23): BatchNorm2d(256, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_23): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (24): Sequential(
      (shortcut_24): EmptyLayer()
    )
    (25): Sequential(
      (conv_25): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1
), bias=False)
      (batch_norm_25): BatchNorm2d(128, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_25): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (26): Sequential(
      (conv_26): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1
), padding=(1, 1), bias=False)
      (batch_norm_26): BatchNorm2d(256, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_26): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (27): Sequential(
      (shortcut_27): EmptyLayer()
```

```
    )
    (28): Sequential(
      (conv_28): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1
), bias=False)
      (batch_norm_28): BatchNorm2d(128, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_28): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (29): Sequential(
      (conv_29): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1
), padding=(1, 1), bias=False)
      (batch_norm_29): BatchNorm2d(256, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_29): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (30): Sequential(
      (shortcut_30): EmptyLayer()
    )
    (31): Sequential(
      (conv_31): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1
), bias=False)
      (batch_norm_31): BatchNorm2d(128, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_31): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (32): Sequential(
      (conv_32): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1
), padding=(1, 1), bias=False)
      (batch_norm_32): BatchNorm2d(256, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_32): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (33): Sequential(
      (shortcut_33): EmptyLayer()
    )
    (34): Sequential(
      (conv_34): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1
), bias=False)
      (batch_norm_34): BatchNorm2d(128, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_34): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (35): Sequential(
      (conv_35): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1
), padding=(1, 1), bias=False)
      (batch_norm_35): BatchNorm2d(256, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_35): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (36): Sequential(
      (shortcut_36): EmptyLayer()
    )
    (37): Sequential(
```

```
      (conv_37): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2
), padding=(1, 1), bias=False)
      (batch_norm_37): BatchNorm2d(512, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_37): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (38): Sequential(
      (conv_38): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1
), bias=False)
      (batch_norm_38): BatchNorm2d(256, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_38): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (39): Sequential(
      (conv_39): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1
), padding=(1, 1), bias=False)
      (batch_norm_39): BatchNorm2d(512, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_39): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (40): Sequential(
      (shortcut_40): EmptyLayer()
    )
    (41): Sequential(
      (conv_41): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1
), bias=False)
      (batch_norm_41): BatchNorm2d(256, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_41): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (42): Sequential(
      (conv_42): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1
), padding=(1, 1), bias=False)
      (batch_norm_42): BatchNorm2d(512, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_42): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (43): Sequential(
      (shortcut_43): EmptyLayer()
    )
    (44): Sequential(
      (conv_44): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1
), bias=False)
      (batch_norm_44): BatchNorm2d(256, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_44): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (45): Sequential(
      (conv_45): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1
), padding=(1, 1), bias=False)
      (batch_norm_45): BatchNorm2d(512, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_45): LeakyReLU(negative_slope=0.1, inplace=True)
```

```
    )
    (46): Sequential(
      (shortcut_46): EmptyLayer()
    )
    (47): Sequential(
      (conv_47): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1
), bias=False)
      (batch_norm_47): BatchNorm2d(256, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_47): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (48): Sequential(
      (conv_48): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1
), padding=(1, 1), bias=False)
      (batch_norm_48): BatchNorm2d(512, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_48): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (49): Sequential(
      (shortcut_49): EmptyLayer()
    )
    (50): Sequential(
      (conv_50): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1
), bias=False)
      (batch_norm_50): BatchNorm2d(256, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_50): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (51): Sequential(
      (conv_51): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1
), padding=(1, 1), bias=False)
      (batch_norm_51): BatchNorm2d(512, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_51): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (52): Sequential(
      (shortcut_52): EmptyLayer()
    )
    (53): Sequential(
      (conv_53): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1
), bias=False)
      (batch_norm_53): BatchNorm2d(256, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_53): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (54): Sequential(
      (conv_54): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1
), padding=(1, 1), bias=False)
      (batch_norm_54): BatchNorm2d(512, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_54): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (55): Sequential(
```

```
      (shortcut_55): EmptyLayer()
    )
    (56): Sequential(
      (conv_56): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1
), bias=False)
      (batch_norm_56): BatchNorm2d(256, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_56): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (57): Sequential(
      (conv_57): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1
), padding=(1, 1), bias=False)
      (batch_norm_57): BatchNorm2d(512, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_57): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (58): Sequential(
      (shortcut_58): EmptyLayer()
    )
    (59): Sequential(
      (conv_59): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1
), bias=False)
      (batch_norm_59): BatchNorm2d(256, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_59): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (60): Sequential(
      (conv_60): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1
), padding=(1, 1), bias=False)
      (batch_norm_60): BatchNorm2d(512, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_60): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (61): Sequential(
      (shortcut_61): EmptyLayer()
    )
    (62): Sequential(
      (conv_62): Conv2d(512, 1024, kernel_size=(3, 3), stride=(2,
2), padding=(1, 1), bias=False)
      (batch_norm_62): BatchNorm2d(1024, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
      (leaky_62): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (63): Sequential(
      (conv_63): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1,
1), bias=False)
      (batch_norm_63): BatchNorm2d(512, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_63): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (64): Sequential(
      (conv_64): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1,
1), padding=(1, 1), bias=False)
```

```
      (batch_norm_64): BatchNorm2d(1024, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
      (leaky_64): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (65): Sequential(
      (shortcut_65): EmptyLayer()
    )
    (66): Sequential(
      (conv_66): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1,
1), bias=False)
      (batch_norm_66): BatchNorm2d(512, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_66): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (67): Sequential(
      (conv_67): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1,
1), padding=(1, 1), bias=False)
      (batch_norm_67): BatchNorm2d(1024, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
      (leaky_67): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (68): Sequential(
      (shortcut_68): EmptyLayer()
    )
    (69): Sequential(
      (conv_69): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1,
1), bias=False)
      (batch_norm_69): BatchNorm2d(512, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_69): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (70): Sequential(
      (conv_70): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1,
1), padding=(1, 1), bias=False)
      (batch_norm_70): BatchNorm2d(1024, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
      (leaky_70): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (71): Sequential(
      (shortcut_71): EmptyLayer()
    )
    (72): Sequential(
      (conv_72): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1,
1), bias=False)
      (batch_norm_72): BatchNorm2d(512, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_72): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (73): Sequential(
      (conv_73): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1,
1), padding=(1, 1), bias=False)
      (batch_norm_73): BatchNorm2d(1024, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
```

```
      (leaky_73): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (74): Sequential(
      (shortcut_74): EmptyLayer()
    )
    (75): Sequential(
      (conv_75): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1,
1), bias=False)
      (batch_norm_75): BatchNorm2d(512, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_75): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (76): Sequential(
      (conv_76): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1,
1), padding=(1, 1), bias=False)
      (batch_norm_76): BatchNorm2d(1024, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
      (leaky_76): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (77): Sequential(
      (conv_77): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1,
1), bias=False)
      (batch_norm_77): BatchNorm2d(512, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_77): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (78): Sequential(
      (conv_78): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1,
1), padding=(1, 1), bias=False)
      (batch_norm_78): BatchNorm2d(1024, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
      (leaky_78): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (79): Sequential(
      (conv_79): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1,
1), bias=False)
      (batch_norm_79): BatchNorm2d(512, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_79): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (80): Sequential(
      (conv_80): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1,
1), padding=(1, 1), bias=False)
      (batch_norm_80): BatchNorm2d(1024, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
      (leaky_80): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (81): Sequential(
      (conv_81): Conv2d(1024, 255, kernel_size=(1, 1), stride=(1,
1))
    )
    (82): Sequential(
      (Detection_82): DetectionLayer()
```

```
    )
    (83): Sequential(
      (route_83): EmptyLayer()
    )
    (84): Sequential(
      (conv_84): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1
), bias=False)
      (batch_norm_84): BatchNorm2d(256, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_84): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (85): Sequential(
      (upsample_85): Upsample(scale_factor=2.0, mode=bilinear)
    )
    (86): Sequential(
      (route_86): EmptyLayer()
    )
    (87): Sequential(
      (conv_87): Conv2d(768, 256, kernel_size=(1, 1), stride=(1, 1
), bias=False)
      (batch_norm_87): BatchNorm2d(256, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_87): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (88): Sequential(
      (conv_88): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1
), padding=(1, 1), bias=False)
      (batch_norm_88): BatchNorm2d(512, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_88): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (89): Sequential(
      (conv_89): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1
), bias=False)
      (batch_norm_89): BatchNorm2d(256, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_89): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (90): Sequential(
      (conv_90): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1
), padding=(1, 1), bias=False)
      (batch_norm_90): BatchNorm2d(512, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_90): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (91): Sequential(
      (conv_91): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1
), bias=False)
      (batch_norm_91): BatchNorm2d(256, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_91): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (92): Sequential(
```

```
      (conv_92): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1
), padding=(1, 1), bias=False)
      (batch_norm_92): BatchNorm2d(512, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_92): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (93): Sequential(
      (conv_93): Conv2d(512, 255, kernel_size=(1, 1), stride=(1, 1
))
    )
    (94): Sequential(
      (Detection_94): DetectionLayer()
    )
    (95): Sequential(
      (route_95): EmptyLayer()
    )
    (96): Sequential(
      (conv_96): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1
), bias=False)
      (batch_norm_96): BatchNorm2d(128, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_96): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (97): Sequential(
      (upsample_97): Upsample(scale_factor=2.0, mode=bilinear)
    )
    (98): Sequential(
      (route_98): EmptyLayer()
    )
    (99): Sequential(
      (conv_99): Conv2d(384, 128, kernel_size=(1, 1), stride=(1, 1
), bias=False)
      (batch_norm_99): BatchNorm2d(128, eps=1e-05, momentum=0.1, a
ffine=True, track_running_stats=True)
      (leaky_99): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (100): Sequential(
      (conv_100): Conv2d(128, 256, kernel_size=(3, 3), stride=(1,
1), padding=(1, 1), bias=False)
      (batch_norm_100): BatchNorm2d(256, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
      (leaky_100): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (101): Sequential(
      (conv_101): Conv2d(256, 128, kernel_size=(1, 1), stride=(1,
1), bias=False)
      (batch_norm_101): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
      (leaky_101): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (102): Sequential(
      (conv_102): Conv2d(128, 256, kernel_size=(3, 3), stride=(1,
1), padding=(1, 1), bias=False)
```

```
      (batch_norm_102): BatchNorm2d(256, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
      (leaky_102): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (103): Sequential(
      (conv_103): Conv2d(256, 128, kernel_size=(1, 1), stride=(1,
1), bias=False)
      (batch_norm_103): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
      (leaky_103): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (104): Sequential(
      (conv_104): Conv2d(128, 256, kernel_size=(3, 3), stride=(1,
1), padding=(1, 1), bias=False)
      (batch_norm_104): BatchNorm2d(256, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
      (leaky_104): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (105): Sequential(
      (conv_105): Conv2d(256, 255, kernel_size=(1, 1), stride=(1,
1))
    )
    (106): Sequential(
      (Detection_106): DetectionLayer()
    )
  )
)
```

In [22]:
```
read_dir = time.time()
#Detection phase
try:
    imlist = [osp.join(osp.realpath('.'), img_dir, img) for img in
os.listdir(img_dir) if img.find("png") > -1 ]
except NotADirectoryError:
    imlist = []
    imlist.append(osp.join(osp.realpath('.'), img_dir))
except FileNotFoundError:
    print ("No file or directory with the name {}".format(img_dir))
    exit()
```

In [23]:
```
txt = '/Users/chanho/Documents/GitHub/keras-YOLOv3/img/frame0700.pn
g'
print(txt.find("png") > -1)
```

```
True
```

In [24]:
```
print(imlist)
```

```
['/Users/chanho/Documents/GitLab/niceface/models/img/frame0900.png
', '/Users/chanho/Documents/GitLab/niceface/models/img/zebra.png',
'/Users/chanho/Documents/GitLab/niceface/models/img/frame1700.png'
, '/Users/chanho/Documents/GitLab/niceface/models/img/frame1900.pn
g', '/Users/chanho/Documents/GitLab/niceface/models/img/frame1309.
png', '/Users/chanho/Documents/GitLab/niceface/models/img/frame060
0.png', '/Users/chanho/Documents/GitLab/niceface/models/img/frame1
291.png', '/Users/chanho/Documents/GitLab/niceface/models/img/fram
e1441.png', '/Users/chanho/Documents/GitLab/niceface/models/img/do
g-cycle-car.png', '/Users/chanho/Documents/GitLab/niceface/models/
img/frame0700.png']
```

In [25]:
```
if not os.path.exists(det_dir):
    os.makedirs(det_dir)
```

In [26]:
```
load_batch = time.time()
loaded_ims = [cv2.imread(x) for x in imlist]
```

In [27]:
```
def letterbox_image(img, inp_dim):
    '''resize image with unchanged aspect ratio using padding'''
    img_w, img_h = img.shape[1], img.shape[0]
    w, h = inp_dim
    new_w = int(img_w * min(w/img_w, h/img_h))
    new_h = int(img_h * min(w/img_w, h/img_h))
    resized_image = cv2.resize(img, (new_w,new_h), interpolation =
cv2.INTER_CUBIC)

    canvas = np.full((inp_dim[1], inp_dim[0], 3), 128)

    canvas[(h-new_h)//2:(h-new_h)//2 + new_h,(w-new_w)//2:(w-new_w)
//2 + new_w,  :] = resized_image

    return canvas
```

In [28]:
```
def prep_image(img, inp_dim):
    """
    Prepare image for inputting to the neural network.

    Returns a Variable
    """
    img = (letterbox_image(img, (inp_dim, inp_dim)))
    img = img[:,:,::-1].transpose((2,0,1)).copy()
    img = torch.from_numpy(img).float().div(255.0).unsqueeze(0)
    return img
```

In [29]:
```
#PyTorch Variables for images
im_batches = list(map(prep_image, loaded_ims, [inp_dim for x in ran
ge(len(imlist))]))

#List containing dimensions of original images
im_dim_list = [(x.shape[1], x.shape[0]) for x in loaded_ims]
im_dim_list = torch.FloatTensor(im_dim_list).repeat(1,2)

if CUDA:
    im_dim_list = im_dim_list.cuda()
```

In [30]:
```
leftover = 0
if (len(im_dim_list) % batch_size):
   leftover = 1

if batch_size != 1:
   num_batches = len(imlist) // batch_size + leftover
   im_batches = [torch.cat((im_batches[i*batch_size : min((i +  1)*
batch_size,
                        len(im_batches))]))  for i in range(num_batc
hes)]
```

In [31]:
```
def unique(tensor):
    tensor_np = tensor.cpu().numpy()
    unique_np = np.unique(tensor_np)
    unique_tensor = torch.from_numpy(unique_np)

    tensor_res = tensor.new(unique_tensor.shape)
    tensor_res.copy_(unique_tensor)
    return tensor_res
```

In [32]:
```
write = 0
start_det_loop = time.time()
for i, batch in enumerate(im_batches):
    #load the image
    start = time.time()
    if CUDA:
        batch = batch.cuda()

    prediction = model(Variable(batch, volatile = True), CUDA)

    prediction = write_results(prediction, confidence, num_classes,
nms_conf = nms_thesh)

    end = time.time()

    if type(prediction) == int:

        for im_num, image in enumerate(imlist[i*batch_size: min((i
+  1)*batch_size, len(imlist))]):
            im_id = i*batch_size + im_num
            print("{0:20s} predicted in {1:6.3f} seconds".format(im
age.split("/")[-1], (end - start)/batch_size))
            print("{0:20s} {1:s}".format("Objects Detected:", ""))
            print("----------------------------------------------
----------")
        continue

    prediction[:,0] += i*batch_size    #transform the atribute from
index in batch to index in imlist

    if not write:                      #If we have't initialised ou
tput
        output = prediction
        write = 1
    else:
        output = torch.cat((output,prediction))

    for im_num, image in enumerate(imlist[i*batch_size: min((i +  1
)*batch_size, len(imlist))]):
        im_id = i*batch_size + im_num
        objs = [classes[int(x[-1])] for x in output if int(x[0]) ==
im_id]
        print("{0:20s} predicted in {1:6.3f} seconds".format(image.
split("/")[-1], (end - start)/batch_size))
        print("{0:20s} {1:s}".format("Objects Detected:", " ".join(
objs)))
        print("----------------------------------------------------
------")

    if CUDA:
        torch.cuda.synchronize()
```

```
<ipython-input-32-e5724478fd54>:9: UserWarning: volatile was remov
ed and now has no effect. Use `with torch.no_grad():` instead.
  prediction = model(Variable(batch, volatile = True), CUDA)

frame0900.png        predicted in  0.863 seconds
Objects Detected:    person person person person person person per
son person
----------------------------------------------------------
zebra.png            predicted in  0.801 seconds
Objects Detected:    zebra zebra zebra
----------------------------------------------------------
frame1700.png        predicted in  0.782 seconds
Objects Detected:    person person person person person
----------------------------------------------------------
frame1900.png        predicted in  0.774 seconds
Objects Detected:    person person person person person person per
son person person
----------------------------------------------------------
frame1309.png        predicted in  0.771 seconds
Objects Detected:    person person person person person person
----------------------------------------------------------
frame0600.png        predicted in  0.795 seconds
Objects Detected:    person person person person person person per
son train backpack
----------------------------------------------------------
frame1291.png        predicted in  0.808 seconds
Objects Detected:    person person person
----------------------------------------------------------
frame1441.png        predicted in  0.813 seconds
Objects Detected:    person person person person person
----------------------------------------------------------
dog-cycle-car.png    predicted in  0.830 seconds
Objects Detected:    bicycle truck dog
----------------------------------------------------------
frame0700.png        predicted in  0.780 seconds
Objects Detected:    person person person person
----------------------------------------------------------
```

In [33]:

```
try:
    output
except NameError:
    print ("No detections were made")
    exit()
```

In [34]:
```
im_dim_list = torch.index_select(im_dim_list, 0, output[:,0].long()
)

scaling_factor = torch.min(inp_dim/im_dim_list,1)[0].view(-1,1)


output[:,[1,3]] -= (inp_dim - scaling_factor*im_dim_list[:,0].view(
-1,1))/2
output[:,[2,4]] -= (inp_dim - scaling_factor*im_dim_list[:,1].view(
-1,1))/2

output[:,1:5] /= scaling_factor
```

In [35]:
```
for i in range(output.shape[0]):
    output[i, [1,3]] = torch.clamp(output[i, [1,3]], 0.0, im_dim_li
st[i,0])
    output[i, [2,4]] = torch.clamp(output[i, [2,4]], 0.0, im_dim_li
st[i,1])
```

In [36]:
```
class_load = time.time()
colors = pkl.load(open("data/pallete", "rb"))
```

In [37]:
```
draw = time.time()

def write(x, results, color):
    c1 = tuple(x[1:3].int())
    c2 = tuple(x[3:5].int())
    img = results[int(x[0])]
    cls = int(x[-1])
    label = "{0}".format(classes[cls])
    cv2.rectangle(img, c1, c2,color, 1)
    t_size = cv2.getTextSize(label, cv2.FONT_HERSHEY_PLAIN, 1 , 1)[
0]
    c2 = c1[0] + t_size[0] + 3, c1[1] + t_size[1] + 4
    cv2.rectangle(img, c1, c2,color, -1)
    cv2.putText(img, label, (c1[0], c1[1] + t_size[1] + 4), cv2.FON
T_HERSHEY_PLAIN, 1, [225,255,255], 1);
    return img
```

In [38]:
```
list(map(lambda x: write(x, loaded_ims, colors[0]), output))
det_names = pd.Series(imlist).apply(lambda x: "{}/det_{}".format(de
t_dir,x.split("/")[-1]))
list(map(cv2.imwrite, det_names, loaded_ims))
end = time.time()
```

Here is a sample image of results.