predict_YOLOv3

May 1, 2020

0.1 This code is adapted from How to implement a YOLO (v3) object detector from scratch in PyTorch and its implementation, Github repo.

1 1. Import

1.1 1) General

[1]:
```
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
```

1.2 2) Model

[2]:
```
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.blocks)

    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 == "upsample":
```

```
            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_classes, CUDA)
            if not write:              #if no collector has been 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_normalize"])
                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 weights.
                    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.running_mean)
                    bn_running_var = bn_running_var.view_as(bn.running_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 dims of the␣
↪model weights
                    conv_biases = conv_biases.view_as(conv.bias.data)

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

                #Let us load the weights for the Convolutional layers
                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.data)
                conv.weight.data.copy_(conv_weights)
```

[3]:
```
def predict_transform(prediction, inp_dim, anchors, num_classes, CUDA = 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*num_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_anchors).
↪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
```

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

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

[5]:
```
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
```

```
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, stride, 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), bn)

            #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 = "bilinear")
            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_filters[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), detection)
        module_list.append(module)
        prev_filters = filters
        output_filters.append(filters)
    return (net_info, module_list)
```

1.3 3) etc

[6]:
```
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], box1[:,3]
    b2_x1, b2_y1, b2_x2, b2_y2 = box2[:,0], box2[:,1], box2[:,2], box2[:,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
```

[7]:
```
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 the 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 maximum objectness
            #confidence is at the top
            conf_sort_index = torch.sort(image_pred_class[:,4], descending =␣
↪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 > treshhold
                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].view(-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
```

[8]:
```
def load_classes(namesfile):
    fp = open(namesfile, "r")
    names = fp.read().split("\n")[:-1]
    return names
```

2 2. Run

[9]:
```
img_dir = 'testset-img'
det_dir = 'predicted_boxes'
batch_size = 1
confidence = 0.5
nms_thesh = 0.4
cfgfile = 'cfg/yolov3.cfg'
weightsfile = 'weights/yolov3.weights'
start = 0
reso = 416
CUDA = torch.cuda.is_available()
num_classes = 80    #For COCO
classes = load_classes("data/coco.names")
```

[10]:
```
#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
```

[10]:
```
Darknet(
  (module_list): ModuleList(
    (0): Sequential(
      (conv_0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (batch_norm_0): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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, affine=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()
    )
  )
)
```

[11]:
```
from os import listdir
from os.path import isfile, join

filename = [f for f in listdir(img_dir)]
imlist = [join(img_dir, f) for f in listdir(img_dir) if isfile(join(img_dir,␣
 ↪f)) and f.endswith('.jpg')]
loaded_ims = [cv2.imread(x) for x in imlist]
```

[12]:
```
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

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

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
```

[13]:
```
im_batches = list(map(prep_image, loaded_ims, [inp_dim for x in␣
 ↪range(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)
```

[14]:
```
p = model(Variable(im_batches[0], volatile = True), CUDA)
```

```
<ipython-input-14-17ae41444f36>:1: UserWarning: volatile was removed and now has
no effect. Use `with torch.no_grad():` instead.
  p = model(Variable(im_batches[0], volatile = True), CUDA)
/Users/chanho/miniconda3/envs/eval/lib/python3.8/site-
packages/torch/nn/functional.py:2970: 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 "
```

[15]:
```
write = 0
start_det_loop = time.time()
for i, batch in enumerate(im_batches):
#     print(i)
    start = time.time()
    prediction = model(Variable(batch, volatile = True), CUDA)
    prediction = write_results(prediction, confidence, num_classes, nms_conf =␣
 ↪nms_thesh)
    end = time.time()
#     print(prediction)

    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(image.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 output
        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("----------------------------------------------------------")
```

```
<ipython-input-15-19b1c4e5e187>:6: UserWarning: volatile was removed and now has
no effect. Use `with torch.no_grad():` instead.
  prediction = model(Variable(batch, volatile = True), CUDA)
/Users/distiller/project/conda/conda-
bld/pytorch_1587428077867/work/torch/csrc/utils/python_arg_parser.cpp:756:
UserWarning: This overload of nonzero is deprecated:
        nonzero(Tensor input, *, Tensor out)
Consider using one of the following signatures instead:
        nonzero(Tensor input, *, bool as_tuple)

frame0080.jpg        predicted in  0.732 seconds
Objects Detected:    person person
----------------------------------------------------------
frame0094.jpg        predicted in  0.754 seconds
Objects Detected:    person person person person person truck
----------------------------------------------------------
frame0043.jpg        predicted in  0.724 seconds
Objects Detected:    person person person person person person person person
person
----------------------------------------------------------
frame0057.jpg        predicted in  0.757 seconds
Objects Detected:    person person car truck
----------------------------------------------------------
frame0056.jpg        predicted in  0.734 seconds
Objects Detected:    car car
----------------------------------------------------------
frame0042.jpg        predicted in  0.719 seconds
```

```
Objects Detected:    person car car
------------------------------------------------------------
frame0095.jpg        predicted in  0.717 seconds
Objects Detected:    person person person person
------------------------------------------------------------
frame0081.jpg        predicted in  0.722 seconds
Objects Detected:    person
------------------------------------------------------------
frame0097.jpg        predicted in  0.723 seconds
Objects Detected:    person person person person person
------------------------------------------------------------
frame0083.jpg        predicted in  0.713 seconds
Objects Detected:    person
------------------------------------------------------------
frame0068.jpg        predicted in  0.715 seconds
Objects Detected:
------------------------------------------------------------
frame0054.jpg        predicted in  0.713 seconds
Objects Detected:    person person car car
------------------------------------------------------------
frame0040.jpg        predicted in  0.714 seconds
Objects Detected:    person person person person person person person car
------------------------------------------------------------
frame0041.jpg        predicted in  0.745 seconds
Objects Detected:    person person person person person person person person
person
------------------------------------------------------------
frame0055.jpg        predicted in  0.719 seconds
Objects Detected:    person person person
------------------------------------------------------------
frame0069.jpg        predicted in  0.703 seconds
Objects Detected:
------------------------------------------------------------
frame0082.jpg        predicted in  0.701 seconds
Objects Detected:    car truck
------------------------------------------------------------
frame0096.jpg        predicted in  0.710 seconds
Objects Detected:    person person person person person person backpack
------------------------------------------------------------
frame0092.jpg        predicted in  0.704 seconds
Objects Detected:    person person person
------------------------------------------------------------
frame0086.jpg        predicted in  0.712 seconds
Objects Detected:    person person person car
------------------------------------------------------------
frame0051.jpg        predicted in  0.715 seconds
Objects Detected:    person person person handbag
------------------------------------------------------------
```

```
frame0045.jpg        predicted in  0.713 seconds
Objects Detected:    person person person person
------------------------------------------------------------
frame0079.jpg        predicted in  0.701 seconds
Objects Detected:    person person
------------------------------------------------------------
frame0078.jpg        predicted in  0.709 seconds
Objects Detected:
------------------------------------------------------------
frame0044.jpg        predicted in  0.706 seconds
Objects Detected:    person person person person person person person
------------------------------------------------------------
frame0050.jpg        predicted in  0.707 seconds
Objects Detected:    person person person person person person person handbag
------------------------------------------------------------
frame0087.jpg        predicted in  0.700 seconds
Objects Detected:    person person person person
------------------------------------------------------------
frame0093.jpg        predicted in  0.703 seconds
Objects Detected:
------------------------------------------------------------
frame0085.jpg        predicted in  0.703 seconds
Objects Detected:
------------------------------------------------------------
frame0091.jpg        predicted in  0.702 seconds
Objects Detected:
------------------------------------------------------------
frame0046.jpg        predicted in  0.704 seconds
Objects Detected:    person person person person person person person
------------------------------------------------------------
frame0052.jpg        predicted in  0.712 seconds
Objects Detected:    person person person person person person
------------------------------------------------------------
frame0053.jpg        predicted in  0.730 seconds
Objects Detected:    person person person person person person person person
person person
------------------------------------------------------------
frame0047.jpg        predicted in  0.735 seconds
Objects Detected:    person person person person person person bus
------------------------------------------------------------
frame0090.jpg        predicted in  0.758 seconds
Objects Detected:    person person person person car car
------------------------------------------------------------
frame0084.jpg        predicted in  0.721 seconds
Objects Detected:
------------------------------------------------------------
frame0008.jpg        predicted in  0.719 seconds
Objects Detected:    person person person person person person person person
```

```
person person person
-----------------------------------------------------------
frame0020.jpg        predicted in  0.717 seconds
Objects Detected:    car car
-----------------------------------------------------------
frame0034.jpg        predicted in  0.712 seconds
Objects Detected:    person person person person person person person
-----------------------------------------------------------
frame0035.jpg        predicted in  0.711 seconds
Objects Detected:    person person person person
-----------------------------------------------------------
frame0021.jpg        predicted in  0.708 seconds
Objects Detected:
-----------------------------------------------------------
frame0009.jpg        predicted in  0.704 seconds
Objects Detected:
-----------------------------------------------------------
frame0037.jpg        predicted in  0.704 seconds
Objects Detected:    person person person person person person person
-----------------------------------------------------------
frame0023.jpg        predicted in  0.714 seconds
Objects Detected:    person person person person person person person car car
truck
-----------------------------------------------------------
frame0022.jpg        predicted in  0.713 seconds
Objects Detected:    person person person person person person person person
-----------------------------------------------------------
frame0036.jpg        predicted in  0.707 seconds
Objects Detected:    car
-----------------------------------------------------------
frame0032.jpg        predicted in  0.730 seconds
Objects Detected:    person person person person person person person handbag
-----------------------------------------------------------
frame0026.jpg        predicted in  0.734 seconds
Objects Detected:    person person person person handbag
-----------------------------------------------------------
frame0027.jpg        predicted in  0.706 seconds
Objects Detected:    person car
-----------------------------------------------------------
frame0033.jpg        predicted in  0.712 seconds
Objects Detected:    person person person
-----------------------------------------------------------
frame0025.jpg        predicted in  0.727 seconds
Objects Detected:    person person person person person person person person
person person person person person
-----------------------------------------------------------
frame0031.jpg        predicted in  0.717 seconds
Objects Detected:
```

```
------------------------------------------------------------
frame0019.jpg        predicted in  0.767 seconds
Objects Detected:    person person person person person person person person
------------------------------------------------------------
frame0018.jpg        predicted in  0.763 seconds
Objects Detected:    person person person person person person car truck
------------------------------------------------------------
frame0030.jpg        predicted in  0.723 seconds
Objects Detected:    car car
------------------------------------------------------------
frame0024.jpg        predicted in  0.713 seconds
Objects Detected:    car
------------------------------------------------------------
frame0029.jpg        predicted in  0.715 seconds
Objects Detected:    person person car truck
------------------------------------------------------------
frame0001.jpg        predicted in  0.716 seconds
Objects Detected:    person person person person person person person
------------------------------------------------------------
frame0015.jpg        predicted in  0.719 seconds
Objects Detected:    person person person person person person person
------------------------------------------------------------
frame0014.jpg        predicted in  0.710 seconds
Objects Detected:    person car car
------------------------------------------------------------
frame0000.jpg        predicted in  0.709 seconds
Objects Detected:    person person person person person person bicycle
------------------------------------------------------------
frame0028.jpg        predicted in  0.724 seconds
Objects Detected:    person person person person person person person car
------------------------------------------------------------
frame0016.jpg        predicted in  0.716 seconds
Objects Detected:    person person person
------------------------------------------------------------
frame0002.jpg        predicted in  0.728 seconds
Objects Detected:    person person person person person person person person
------------------------------------------------------------
frame0003.jpg        predicted in  0.727 seconds
Objects Detected:    person person person person person person person person
person person person person
------------------------------------------------------------
frame0017.jpg        predicted in  0.730 seconds
Objects Detected:    person person car car
------------------------------------------------------------
frame0013.jpg        predicted in  0.718 seconds
Objects Detected:    person person person person person car
------------------------------------------------------------
frame0007.jpg        predicted in  0.720 seconds
```

```
Objects Detected:    person person person person person person person
----------------------------------------------------------
frame0006.jpg        predicted in  0.706 seconds
Objects Detected:    person car
----------------------------------------------------------
frame0012.jpg        predicted in  0.707 seconds
Objects Detected:    person car
----------------------------------------------------------
frame0004.jpg        predicted in  0.705 seconds
Objects Detected:    person person person person person person person person
----------------------------------------------------------
frame0010.jpg        predicted in  0.714 seconds
Objects Detected:    person person person backpack
----------------------------------------------------------
frame0038.jpg        predicted in  0.715 seconds
Objects Detected:    person person person
----------------------------------------------------------
frame0039.jpg        predicted in  0.713 seconds
Objects Detected:    person person person person person person
----------------------------------------------------------
frame0011.jpg        predicted in  0.705 seconds
Objects Detected:    person person person
----------------------------------------------------------
frame0005.jpg        predicted in  0.724 seconds
Objects Detected:    person person
----------------------------------------------------------
frame0089.jpg        predicted in  0.732 seconds
Objects Detected:    person person car truck
----------------------------------------------------------
frame0062.jpg        predicted in  0.746 seconds
Objects Detected:    person person person person person person person person
----------------------------------------------------------
frame0076.jpg        predicted in  0.726 seconds
Objects Detected:    person person person person person car
----------------------------------------------------------
frame0077.jpg        predicted in  0.731 seconds
Objects Detected:    person person person person person person
----------------------------------------------------------
frame0063.jpg        predicted in  0.741 seconds
Objects Detected:    person person person person person
----------------------------------------------------------
frame0088.jpg        predicted in  0.725 seconds
Objects Detected:    person person person person person bus truck
----------------------------------------------------------
frame0049.jpg        predicted in  0.735 seconds
Objects Detected:    person person person person person person person car
----------------------------------------------------------
frame0075.jpg        predicted in  0.737 seconds
```

```
Objects Detected:
-------------------------------------------------------
frame0061.jpg        predicted in  0.733 seconds
Objects Detected:    person person person person person person person
-------------------------------------------------------
frame0060.jpg        predicted in  0.734 seconds
Objects Detected:    person person person person person person
-------------------------------------------------------
frame0074.jpg        predicted in  0.732 seconds
Objects Detected:    person person person person person
-------------------------------------------------------
frame0048.jpg        predicted in  0.723 seconds
Objects Detected:    person person person
-------------------------------------------------------
frame0070.jpg        predicted in  0.733 seconds
Objects Detected:    person person person
-------------------------------------------------------
frame0064.jpg        predicted in  0.747 seconds
Objects Detected:    person person person
-------------------------------------------------------
frame0058.jpg        predicted in  0.755 seconds
Objects Detected:    person person
-------------------------------------------------------
frame0059.jpg        predicted in  0.762 seconds
Objects Detected:    person person person person person person
-------------------------------------------------------
frame0065.jpg        predicted in  0.830 seconds
Objects Detected:    person person person person person person person person
person person
-------------------------------------------------------
frame0071.jpg        predicted in  0.750 seconds
Objects Detected:    car car truck
-------------------------------------------------------
frame0098.jpg        predicted in  0.719 seconds
Objects Detected:    person person person person person person person
-------------------------------------------------------
frame0067.jpg        predicted in  0.706 seconds
Objects Detected:    person person person person person person person person
-------------------------------------------------------
frame0073.jpg        predicted in  0.707 seconds
Objects Detected:    person person person
-------------------------------------------------------
frame0072.jpg        predicted in  0.721 seconds
Objects Detected:    person person
-------------------------------------------------------
frame0066.jpg        predicted in  0.714 seconds
Objects Detected:    person truck
-------------------------------------------------------
```

```
frame0099.jpg         predicted in  0.726 seconds
Objects Detected:     person person person
-----------------------------------------------------------
```

[16]:
```
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
```

[17]:
```
def write(x):
    img_id = int(x[0])
    xmin, ymin, xmax, ymax = int(x[1]), int(x[2]), int(x[3]), int(x[4])
    conf_score = float(x[6])
    cls = int(x[7])
    label = "{0}".format(classes[cls])
    return filename[img_id], xmin, ymin, xmax, ymax, conf_score, label
```

[18]:
```
df = pd.DataFrame(list(map(write, output)), columns=['fn', 'xmin', 'ymin',␣
↪'xmax', 'ymax', 'conf_score', 'label'])
df.head()
```

[18]:

	fn	xmin	ymin	xmax	ymax	conf_score	label
0	frame0080.jpg	512	345	534	398	0.993586	person
1	frame0080.jpg	755	320	789	393	0.987500	person
2	frame0094.jpg	812	294	852	399	0.999899	person
3	frame0094.jpg	955	312	1005	394	0.999720	person
4	frame0094.jpg	533	235	594	392	0.999767	person

[19]:
```
import copy

class_dict = {}
for l in classes:
    class_dict[l] = {}
    for f in filename:
        df_l_f = df[(df.label == l) & (df.fn == f)].values.tolist()
        if df_l_f:
            class_dict[l][f] = {}
            class_dict[l][f]['boxes'] = []
            class_dict[l][f]['scores'] = []

            for d in df_l_f:
                class_dict[l][f]['boxes'].append([d[1], d[2], d[3], d[4]])
                class_dict[l][f]['scores'].append(d[5])
```

[22]:
```
import json

for l in ['person', 'car']:
    with open(det_dir+'/predicted_boxes-YOLOv3-'+l+'.json', 'w') as fp:
        json.dump(class_dict[l], fp)
```