Assignment_3_0_0

October 17, 2020

1 ECE-6524 / CS-6524 Deep Learning

2 Assignment 3 [100 pts]

In this assignment, **you need to complete the Yolo loss function, and train an object detector. Yay!**

2.1 Submission guideline for the coding part (Jupyter Notebook)

1. Click the Save button at the top of the Jupyter Notebook
2. Please make sure to have entered your Virginia Tech PID below
3. Once you've completed everything (make sure output for all cells are visible), select File -> Download as -> PDF via LaTeX
4. Look at the PDF file and make sure all your solutions are displayed correctly there
5. Zip all the files along with this notebook (Please don't include the data). Name it as Assignment_3_Code_[YOUR PID NUMBER].zip
6. Name your PDF file as Assignment_2_NB_[YOUR PID NUMBER].pdf
7. **Submit your zipped file and the PDF SEPARATELY**

Note: if facing issues with step 3 refer: https://pypi.org/project/notebook-as-pdf/

2.2 Submission guideline for the coding part (Google Colab)

1. Click the Save button at the top of the Notebook
2. Please make sure to have entered your Virginia Tech PID below
3. Follow last two cells in this notebook for guidelines to download pdf file of this notebook
4. Look at the PDF file and make sure all your solutions are displayed correctly there
5. Zip all the files along with this notebook (Please don't include the data). Name it as Assignment_2_Code_[YOUR PID NUMBER].zip
6. Name your PDF file as Assignment_2_NB_[YOUR PID NUMBER].pdf
7. **Submit your zipped file and the PDF SEPARATELY**

While you are encouraged to discuss with your peers, all work submitted is expected to be your own. If you use any information from other resources (e.g. online materials), you are required to cite it below you VT PID. Any violation will result in a 0 mark for the assignment.

2.2.1 Please Write Your VT PID Here: 906213559

2.2.2 Reference (if any):

https://pjreddie.com/media/files/papers/yolo_1.pdf

https://stats.stackexchange.com/questions/287486/yolo-loss-function-explanation

https://medium.com/adventures-with-deep-learning/yolo-v1-part-1-cfb47135f81f

https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

https://github.com/x-hou/yolo-pytorch

https://github.com/xiongzihua/pytorch-YOLO-v1

In this homework, you would need to use **Python 3.6+** along with the following packages:

```
1. pytorch 1.2
2. torchvision
3. numpy
4. matplotlib
5. tqdm (for better, cuter progress bar. Yay!)
```

To install pytorch, please follow the instructions on the Official website. In addition, the official document could be very helpful when you want to find certain functionalities.

Note that, on a high-end GPU, it sill takes 3-4 hours to train. **SO START EARLY. IT'S IMPOSSIBLE TO FINISH IT AT THE LAST MINUTE!**

2.2.3 Colab Setup

2.2.4 To select GPU in Google Colab:

- go to **Edit -> Notebook settings -> Hardware accelerator -> GPU**

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[ ]:
```
import sys
# modify "customized_path_to_homework", path of folder in drive, where you␣
↪uploaded your homework
customized_path_to_homework = "/content/drive/My Drive/DL_Fall_2020/Assignment_3/
↪"
sys.path.append(customized_path_to_homework)
```

[ ]:
```
# run this to download dataset, give path to the download.sh file from your drive
!sh /content/drive/My\ Drive/DL_Fall_2020/Assignment_3/download_data.sh
```

```
--2020-10-16 00:57:05--
http://pjreddie.com/media/files/VOCtrainval_06-Nov-2007.tar
Resolving pjreddie.com (pjreddie.com)... 128.208.4.108
Connecting to pjreddie.com (pjreddie.com)|128.208.4.108|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://pjreddie.com/media/files/VOCtrainval_06-Nov-2007.tar
[following]
--2020-10-16 00:57:05--
https://pjreddie.com/media/files/VOCtrainval_06-Nov-2007.tar
Connecting to pjreddie.com (pjreddie.com)|128.208.4.108|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 460032000 (439M) [application/octet-stream]
Saving to: ‘VOCtrainval_06-Nov-2007.tar’

VOCtrainval_06-Nov- 100%[===================>] 438.72M   319KB/s    in 18m 7s

2020-10-16 01:15:13 (413 KB/s) - ‘VOCtrainval_06-Nov-2007.tar’ saved
[460032000/460032000]

URL transformed to HTTPS due to an HSTS policy
--2020-10-16 01:15:14--
https://pjreddie.com/media/files/VOCtest_06-Nov-2007.tar
Resolving pjreddie.com (pjreddie.com)... 128.208.4.108
Connecting to pjreddie.com (pjreddie.com)|128.208.4.108|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 451020800 (430M) [application/octet-stream]
Saving to: ‘VOCtest_06-Nov-2007.tar’

VOCtest_06-Nov-2007 100%[===================>] 430.13M   475KB/s    in 22m 43s

2020-10-16 01:37:59 (323 KB/s) - ‘VOCtest_06-Nov-2007.tar’ saved
[451020800/451020800]
```

[ ]:
```
# copy and place downloaded dataset to your drive. To access dataset multiple␣
↪times, no need to download everytime you open colab.
!cp -r /content/VOCdevkit_2007 /content/drive/My\ Drive/DL_Fall_2020/
↪Assignment_3/
```

[ ]:
```
import os
import random
import cv2
import numpy as np
import torch
from torch.utils.data import DataLoader
from torchvision import models
```

```
from resnet_yolo import resnet50
from dataset import VocDetectorDataset
from eval_voc import evaluate
from predict import predict_image
from config import VOC_CLASSES, COLORS
import matplotlib.pyplot as plt
from tqdm import tqdm

%matplotlib inline
%load_ext autoreload
%autoreload 2
```

2.3 Initialization

[ ]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
#device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(torch.cuda.get_device_name()) # GPU name
```

```
Tesla P100-PCIE-16GB
```

3 You Only Look Once: Unified, Real-Time Object Detection

In this assignment, you need to implement the loss function and train the **YOLO object detector** (specfically, YOLO-v1). Here we provide a list of recommend readings for you: - YOLO original paper (recommended) - Object detection methods (Slides) - Great post about YOLO on Medium - Differences between YOLO, YOLOv2 and YOLOv3 - Great explanation of the Yolo Loss function

We adopt a variant of YOLO, which: 1. Use pretrained ResNet50 classifier as detector backbone. The pretrained model is offered in `torchvision.models`. 2. Instead of using a 7×7 detection grid, we use 14×14 to get a more finegrained detection.

In general, the backbone models are usually pretrained on ImageNet dataset (> 1 million images) with numerous classes. As a result, having these pretrained backbone can greatly shorten the required training time, as well as improve the performance. **But still, it takes at least 3-4 hours to train, not to mention that you might need to debug after one training run. So START EARLY, DON'T GO #YOLO!**

You are supposed to get a reasonable detector (like the … above?) after training the model correctly.

[ ]:
```
# YOLO network hyperparameters
B = 2  # number of bounding box predictions per cell
S = 14  # width/height of network output grid (larger than 7x7 from paper since␣
↪we use a different network)
```

3.1 Load the pretrained ResNet classifier

Load the pretrained classifier. By default, it would use the pretrained model provided by `Pytorch`.

[ ]:

```
load_network_path = None
pretrained = True

# use to load a previously trained network
if load_network_path is not None:
    print('Loading saved network from {}'.format(load_network_path))
    net = resnet50().to(device)
    net.load_state_dict(torch.load(load_network_path))
else:
    print('Load pre-trained model')
    net = resnet50(pretrained=pretrained).to(device)
```

```
Load pre-trained model
```

Some basic hyperparameter settings that you probably don't have to tune.

[ ]:

```
learning_rate = 0.001
num_epochs = 50
batch_size = 24

# Yolo loss component coefficients (as given in Yolo v1 paper)
lambda_coord = 5
lambda_noobj = 0.5
```

3.2 Implement the YOLO-v1 loss [80 pts]

Now, you have to implement the `YoloLoss` for training your object detector. Please read closely to the YOLO original paper so that you can implement it.

In general, there are 4 components in the YOLO loss. Consider that we have our prediction grid of size$(N, S, S, 5B + c)$ ((x, y, w, h, C) for each bounding box, and c is the number of classes), where N is the batch size, S is the grid size, B is the number of bounding boxes. We have : 1. Bounding box regression loss on the bounding box(x, y, w, h) - $l_{coord} = \sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}\left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2\right]$ + $\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}\left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2\right]$ - $\mathbb{1}_{ij}^{obj}$: equals to 1 when object appears in cell i, and the bounding box j is responsible for the prediction. 0 otherwise. 2. Contain object loss on the confidence prediction c (only calculate for those boxes that actually have objects) - $l_{contain} = \sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}(C_i - \hat{C}_i)^2$ - C_i the predicted confidence score for cell i from predicted box j - For each grid cell, you only calculate the contain object loss for the predicted bounding box that has maximum overlap (iou) with the gruond truth box. - We say that this predicted box with maximum iou is **responsible** for the prediction. 3. No object loss on the confidence prediction c (only calculate for those boxes that don't have objects) - $l_{noobj} = \sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{noobj}(C_i - \hat{C}_i)^2$ - $\mathbb{1}_{ij}^{obj}$: equals to 1 when **no object appears** in cell i. 4. Classification error loss. - $l_{class} = \sum_{i=0}^{S^2}\mathbb{1}_{i}^{obj}\sum_{c \in classes}(p_i(c) - \hat{p}_i(c))^2$ - $p_i(c)$ is the predicted score for class c

Putting them together, we get the yolo loss:

$$yolo = \lambda_{coord} l_{coord} + l_{contain} + \lambda_{noobj} l_{noobj} + l_{class} \tag{1}$$

where λ are hyperparameters. We have provided detailed comments to guide you through implementing the loss. So now, please complete the YoloLoss in the code block below. **If you have any problem with regard to implementation, post and discuss it on Piazza.**

[ ]:
```
import torch.nn as nn
import torch.nn.functional as F
from torch.autograd import Variable

class YoloLoss(nn.Module):
    def __init__(self,S,B,l_coord,l_noobj):
        super(YoloLoss,self).__init__()
        self.S = S
        self.B = B
        self.l_coord = l_coord
        self.l_noobj = l_noobj

    def compute_iou(self, box1, box2):
        """Compute the intersection over union of two set of boxes, each box is
 ↪[x1,y1,x2,y2].
        Args:
          box1: (tensor) bounding boxes, sized [N,4].
          box2: (tensor) bounding boxes, sized [M,4].
        Return:
          (tensor) iou, sized [N,M].
        """
        N = box1.size(0)
        M = box2.size(0)

        lt = torch.max(
            box1[:,:2].unsqueeze(1).expand(N,M,2),  # [N,2] -> [N,1,2] -> [N,M,2]
            box2[:,:2].unsqueeze(0).expand(N,M,2),  # [M,2] -> [1,M,2] -> [N,M,2]
        )

        rb = torch.min(
            box1[:,2:].unsqueeze(1).expand(N,M,2),  # [N,2] -> [N,1,2] -> [N,M,2]
            box2[:,2:].unsqueeze(0).expand(N,M,2),  # [M,2] -> [1,M,2] -> [N,M,2]
        )

        wh = rb - lt  # [N,M,2]
        wh[wh<0] = 0  # clip at 0
        inter = wh[:,:,0] * wh[:,:,1]  # [N,M]

        area1 = (box1[:,2]-box1[:,0]) * (box1[:,3]-box1[:,1])  # [N,]
        area2 = (box2[:,2]-box2[:,0]) * (box2[:,3]-box2[:,1])  # [M,]
        area1 = area1.unsqueeze(1).expand_as(inter)  # [N,] -> [N,1] -> [N,M]
        area2 = area2.unsqueeze(0).expand_as(inter)  # [M,] -> [1,M] -> [N,M]
```

```
        iou = inter / (area1 + area2 - inter)
        return iou

    def get_class_prediction_loss(self, classes_pred, classes_target):
        """
        Parameters:
        classes_pred : (tensor) size (batch_size, S, S, 20)                                ␣
↪                                                                                          ␣
↪
        classes_target : (tensor) size (batch_size, S, S, 20)

        Returns:
        class_loss : scalar
        """
        ##### CODE #####
        class_loss = F.mse_loss(classes_pred, classes_target, reduction='sum')
        ##### CODE #####
        return class_loss


    def get_regression_loss(self, box_pred_response, box_target_response):
        """
        Parameters:
        box_pred_response : (tensor) size (-1, 5)
        box_target_response : (tensor) size (-1, 5)
        Note : -1 corresponds to ravels the tensor into the dimension specified
        See : https://pytorch.org/docs/stable/tensors.html#torch.Tensor.view_as

        Returns:
        reg_loss : scalar
        """
        ##### CODE #####
        loss_xy = F.mse_loss(box_pred_response[:, :2], box_target_response[:, :
↪2], reduction='sum')
        loss_wh = F.mse_loss(torch.sqrt(box_pred_response[:, 2:4]), torch.
↪sqrt(box_target_response[:, 2:4]), reduction='sum')
        reg_loss = loss_xy + loss_wh
        ##### CODE #####
        return reg_loss

    def get_contain_object_loss(self, box_pred_response,␣
↪box_target_response_iou):
        """
        Parameters:
        box_pred_response : (tensor) size ( -1 , 5)
        box_target_response_iou : (tensor) size ( -1 , 5)
        Note : -1 corresponds to ravels the tensor into the dimension specified
```

```
        See : https://pytorch.org/docs/stable/tensors.html#torch.Tensor.view_as

        Returns:
        contain_loss : scalar
        """
        ##### CODE #####
        contain_loss = F.mse_loss(box_pred_response[:, 4],␣
↪box_target_response_iou[:, 4], reduction = 'sum')
        ##### CODE #####
        return contain_loss

    def get_no_object_loss(self, target_tensor, pred_tensor, no_object_mask):
        """                                                                     ␣
↪                                                                               ␣
↪
        Parameters:
        target_tensor : (tensor) size (batch_size, S , S, 30)
        pred_tensor : (tensor) size (batch_size, S , S, 30)
        no_object_mask : (tensor) size (batch_size, S , S)

        Returns:
        no_object_loss : scalar

        Hints:
        1) Create 2 tensors no_object_prediction and no_object_target which only␣
↪have the
        values which have no object.
        2) Have another tensor no_object_prediction_mask of the same size such␣
↪that
        mask with respect to both confidences of bounding boxes set to 1.
        3) Create 2 tensors which are extracted from no_object_prediction and␣
↪no_object_target using
        the mask created above to find the loss.
        """
        ##### CODE #####
        no_object_pred = pred_tensor[no_object_mask].view(-1, self.B*5+20)
        no_object_target = target_tensor[no_object_mask].view(-1, self.B*5+20)
        no_object_confidence_mask = torch.cuda.ByteTensor(no_object_pred.size()).
↪fill_(0)

        for b in range(self.B):
          no_object_confidence_mask[:, 4 + 5*b] = 1

        target = no_object_target[no_object_confidence_mask]
        predict = no_object_pred[no_object_confidence_mask]
        no_object_loss = F.mse_loss(predict, target, reduction='sum')
```

```
        ##### CODE #####
        return no_object_loss

    def find_best_iou_boxes(self, box_target, box_pred):
        """
        Parameters:
        box_target : (tensor)  size (-1, 5)
        box_pred : (tensor) size (-1, 5)
        Note : -1 corresponds to ravels the tensor into the dimension specified
        See : https://pytorch.org/docs/stable/tensors.html#torch.Tensor.view_as

        Returns:
        box_target_iou: (tensor)
        contains_object_response_mask : (tensor)

        Hints:
        1) Find the iou's of each of the 2 bounding boxes of each grid cell of␣
↪each image.
        2) Set the corresponding contains_object_response_mask of the bounding␣
↪box with the max iou
        of the 2 bounding boxes of each grid cell to 1.
        3) For finding iou's use the compute_iou function
        4) Before using compute preprocess the bounding box coordinates in such␣
↪a way that
        if for a Box b the coordinates are represented by [x, y, w, h] then
        x, y = x/S - 0.5*w, y/S - 0.5*h ; w, h = x/S + 0.5*w, y/S + 0.5*h
        Note: Over here initially x, y are the center of the box and w,h are␣
↪width and height.
        We perform this transformation to convert the correct coordinates into␣
↪bounding box coordinates.
        5) Set the confidence of the box_target_iou of the bounding box to the␣
↪maximum iou
        """
        ##### CODE #####
        B = self.B
        coo_response_mask = torch.cuda.ByteTensor(box_target.size()).fill_(0)
        box_target_iou = torch.zeros(box_target.size()).cuda()

        for i in range(0, box_target.size(0), B):
          pred = box_pred[i:i+B]
          pred_xy = Variable(torch.FloatTensor(pred.size()))
          # target = box_target[i]
          target = box_target[i].view(-1, 5)
          target_xy = Variable(torch.FloatTensor(target.size()))

          pred_xy[:, :2] = pred[:, :2]/float(S)-0.5*pred[:, 2:4]
```

```
            pred_xy[:,2:4] = pred[:, :2]/float(S)+0.5*pred[:, 2:4]
            target_xy[:, :2] = target[:, :2]/float(S)-0.5*target[:, 2:4]
            target_xy[:, 2:4] = target[:, :2]/float(S)+0.5*target[:, 2:4]

            iou = self.compute_iou(pred_xy[:, :4], target_xy[:, :4])
            max_iou, max_index = iou.max(0)
            max_index = max_index.data.cuda()
            coo_response_mask[i+max_index] = 1
            box_target_iou[i+max_index, torch.LongTensor([4]).cuda()] = (max_iou).
↪data.cuda()

        box_target_iou = Variable(box_target_iou).cuda()
        ##### CODE #####
        return box_target_iou, coo_response_mask

    def forward(self, pred_tensor,target_tensor):
        '''
        pred_tensor: (tensor) size(batchsize,S,S,Bx5+20=30)
                      where B - number of bounding boxes this grid cell is a␣
↪part of = 2
                            5 - number of bounding box values corresponding to␣
↪[x, y, w, h, c]
                                where x - x_coord, y - y_coord, w - width, h -␣
↪height, c - confidence of having an object
                            20 - number of classes

        target_tensor: (tensor) size(batchsize,S,S,30)

        Returns:
        Total Loss
        '''
        N = pred_tensor.size(0)

        total_loss = None
        # Create 2 tensors contains_object_mask and no_object_mask
        # of size (Batch_size, S, S) such that each value corresponds to if the␣
↪confidence of having
        # an object > 0 in the target tensor.

        ##### CODE #####
        contains_object_mask = target_tensor[:, :, :, 4] > 0
        contains_object_mask = contains_object_mask.unsqueeze(-1).
↪expand_as(target_tensor)

        no_object_mask = target_tensor[:, :, :, 4] == 0
        no_object_mask = no_object_mask.unsqueeze(-1).expand_as(target_tensor)
```

```
        ##### CODE #####
        """
        Create a tensor contains_object_pred that corresponds to
        to all the predictions which seem to confidence > 0 for having an object
        Then, split this tensor into 2 tensors :                               ␣
↪                                                                              ␣
↪
        1) bounding_box_pred : Contains all the Bounding box predictions (x, y,␣
↪w, h, c) of all grid
                                cells of all images
        2) classes_pred : Contains all the class predictions for each grid cell␣
↪of each image
        Hint : Use contains_object_mask
        """
        ##### CODE #####
        contains_object_pred = pred_tensor[contains_object_mask].view(-1, 5*self.
↪B+20)
        bounding_box_pred = contains_object_pred[:, :5*self.B].contiguous().
↪view(-1, 5)
        classes_pred = contains_object_pred[:, 5*self.B:]
        ##### CODE #####
        """
        # Similarly, create 2 tensors bounding_box_target and classes_target
        # using the contains_object_mask.
        """
        ##### CODE #####
        contains_object_target = target_tensor[contains_object_mask].view(-1,␣
↪5*self.B+20)
        bounding_box_target = contains_object_target[:, :5*self.B].contiguous().
↪view(-1, 5)
        classes_target = contains_object_target[:, 5*self.B:]
        ##### CODE #####

        #Compute the No object loss here
        # Instruction: finish your get_no_object_loss
        ##### CODE #####
        loss_noobj = self.get_no_object_loss(target_tensor, pred_tensor,␣
↪no_object_mask)
        ##### CODE #####

        """
        # Compute the iou's of all bounding boxes and the mask for which␣
↪bounding box
        # of 2 has the maximum iou the bounding boxes for each grid cell of each␣
↪image.
        # Instruction: finish your find_best_iou_boxes and use it.
```

```
        """
        ##### CODE #####
        box_target_iou, coo_response_mask = self.
↪find_best_iou_boxes(bounding_box_target, bounding_box_pred)
        ##### CODE #####
        """
        # Create 3 tensors :
        # 1) box_prediction_response - bounding box predictions for each grid␣
↪cell which has the maximum iou
        # 2) box_target_response_iou - bounding box target ious for each grid␣
↪cell which has the maximum iou
        # 3) box_target_response -  bounding box targets for each grid cell␣
↪which has the maximum iou
        # Hint : Use coo_response_mask
        """
        ##### CODE #####
        box_prediction_response = bounding_box_pred[coo_response_mask].view(-1,␣
↪5)
        box_target_response_iou = box_target_iou[coo_response_mask].view(-1, 5)
        box_target_response = bounding_box_target[coo_response_mask].view(-1, 5)
        ##### CODE #####
        """
        # Find the class_loss, containing object loss and regression loss
        """
        ##### CODE #####
        loss_class = self.get_class_prediction_loss(classes_pred, classes_target)
        loss_obj = self.get_contain_object_loss(box_prediction_response,␣
↪box_target_response_iou)
        loss_regression = self.get_regression_loss(box_prediction_response,␣
↪box_target_response)
        total_loss = self.l_coord*loss_regression+loss_obj+self.
↪l_noobj*loss_noobj+loss_class
        ##### CODE #####
        return total_loss / N
```

[ ]:
```
criterion = YoloLoss(S, B, lambda_coord, lambda_noobj)
optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9,␣
↪weight_decay=5e-4)
```

3.3 Reading Pascal Data

Since Pascal is a small dataset (5000 in train+val) we have combined the train and val splits to train our detector. This is not typically a good practice, but we will make an exception in this case to be able to get reasonable detection results with a comparatively small object detection dataset. Use `download_data.sh` to download the dataset.

The train dataset loader also using a variety of data augmentation techniques including random shift, scaling, crop, and flips. Data augmentation is slightly more complicated for detection dataset since the bounding box annotations must be kept consistent through the transformations.

Since the output of the dector network we train is a $(S, S, 5B + c)$ tensor, we use an encoder to convert the original bounding box coordinates into relative grid bounding box coordinates corresponding to the the expected output. We also use a decoder which allows us to convert the opposite direction into image coordinate bounding boxes.

[ ]:
```
file_root_train = customized_path_to_homework + '/VOCdevkit_2007/VOC2007/
 ↪JPEGImages/'
annotation_file_train = customized_path_to_homework + '/voc2007.txt'

train_dataset =␣
 ↪VocDetectorDataset(root_img_dir=file_root_train,dataset_file=annotation_file_train,train=True
 ↪S=S)
train_loader =␣
 ↪DataLoader(train_dataset,batch_size=batch_size,shuffle=True,num_workers=4)
print('Loaded %d train images' % len(train_dataset))
#print(len(train_loader))
```

```
Initializing dataset
Loaded 5011 train images
```

[ ]:
```
file_root_test = customized_path_to_homework + '/VOCdevkit_2007/VOC2007test/
 ↪JPEGImages/'
annotation_file_test = customized_path_to_homework + '/voc2007test.txt'

test_dataset =␣
 ↪VocDetectorDataset(root_img_dir=file_root_test,dataset_file=annotation_file_test,train=False,
 ↪S=S)
test_loader =␣
 ↪DataLoader(test_dataset,batch_size=batch_size,shuffle=False,num_workers=4)
print('Loaded %d test images' % len(test_dataset))
```

```
Initializing dataset
Loaded 4950 test images
```

3.4 Train detector

Now, train your detector.

[ ]:
```
#%debug
best_test_loss = np.inf

for epoch in range(num_epochs):
    net.train()

    # Update learning rate late in training
```

```
    if epoch == 30 or epoch == 40:
        learning_rate /= 10.0

    for param_group in optimizer.param_groups:
        param_group['lr'] = learning_rate

    print('\n\nStarting epoch %d / %d' % (epoch + 1, num_epochs))
    print('Learning Rate for this epoch: {}'.format(learning_rate))

    total_loss = 0.

    for i, (images, target) in enumerate(tqdm(train_loader,␣
↪total=len(train_loader))):
        images, target = images.to(device), target.to(device)

        pred = net(images)
        loss = criterion(pred,target)
        total_loss += loss.item()

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    print('Epoch [%d/%d], average_loss: %.4f'
          % (epoch+1, num_epochs, total_loss / (i+1)))

    # evaluate the network on the test data
    with torch.no_grad():
        test_loss = 0.0
        net.eval()
        for i, (images, target) in enumerate(tqdm(test_loader,␣
↪total=len(test_loader))):
            images, target = images.to(device), target.to(device)

            pred = net(images)
            loss = criterion(pred,target)
            test_loss += loss.item()
        test_loss /= len(test_loader)

    if best_test_loss > test_loss:
        best_test_loss = test_loss
        print('Updating best test loss: %.5f' % best_test_loss)
        torch.save(net.state_dict(),'best_detector.pth')

    torch.save(net.state_dict(),'detector.pth')
```

```
0%|          | 0/209 [00:00<?, ?it/s]
```

```
Starting epoch 1 / 50
Learning Rate for this epoch: 0.001

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:122: UserWarning:
indexing with dtype torch.uint8 is now deprecated, please use a dtype torch.bool
instead. (Triggered internally at
/pytorch/aten/src/ATen/native/IndexingUtils.h:20.)
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:123: UserWarning:
indexing with dtype torch.uint8 is now deprecated, please use a dtype torch.bool
instead. (Triggered internally at
/pytorch/aten/src/ATen/native/IndexingUtils.h:20.)
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:259: UserWarning:
indexing with dtype torch.uint8 is now deprecated, please use a dtype torch.bool
instead. (Triggered internally at
/pytorch/aten/src/ATen/native/IndexingUtils.h:20.)
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:260: UserWarning:
indexing with dtype torch.uint8 is now deprecated, please use a dtype torch.bool
instead. (Triggered internally at
/pytorch/aten/src/ATen/native/IndexingUtils.h:20.)
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:261: UserWarning:
indexing with dtype torch.uint8 is now deprecated, please use a dtype torch.bool
instead. (Triggered internally at
/pytorch/aten/src/ATen/native/IndexingUtils.h:20.)
100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [1/50], average_loss: 10.3459

100%|| 207/207 [01:19<00:00,  2.60it/s]

Updating best test loss: 5.20049

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 2 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:25<00:00,  1.43it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [2/50], average_loss: 4.7801

100%|| 207/207 [01:17<00:00,  2.66it/s]

Updating best test loss: 4.52127

  0%|          | 0/209 [00:00<?, ?it/s]
```

```
Starting epoch 3 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [3/50], average_loss: 4.2596

100%|| 207/207 [01:18<00:00,  2.63it/s]

Updating best test loss: 4.17145

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 4 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:31<00:00,  1.38it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [4/50], average_loss: 3.9533

100%|| 207/207 [01:18<00:00,  2.65it/s]

Updating best test loss: 3.92383

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 5 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:27<00:00,  1.42it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [5/50], average_loss: 3.7049

100%|| 207/207 [01:18<00:00,  2.63it/s]

Updating best test loss: 3.81524

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 6 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [6/50], average_loss: 3.4720

100%|| 207/207 [01:18<00:00,  2.63it/s]

Updating best test loss: 3.66781
```

```
  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 7 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:26<00:00,  1.43it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [7/50], average_loss: 3.3610

100%|| 207/207 [01:19<00:00,  2.59it/s]

Updating best test loss: 3.49631

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 8 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:25<00:00,  1.43it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [8/50], average_loss: 3.2200

100%|| 207/207 [01:18<00:00,  2.65it/s]

Updating best test loss: 3.39329

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 9 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [9/50], average_loss: 3.0984

100%|| 207/207 [01:17<00:00,  2.68it/s]

Updating best test loss: 3.28102

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 10 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]
```

```
Epoch [10/50], average_loss: 2.9562

100%|| 207/207 [01:18<00:00,  2.64it/s]

Updating best test loss: 3.24382

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 11 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [11/50], average_loss: 2.8911

100%|| 207/207 [01:18<00:00,  2.65it/s]

Updating best test loss: 3.18133

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 12 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [12/50], average_loss: 2.7876

100%|| 207/207 [01:16<00:00,  2.70it/s]

Updating best test loss: 3.15862

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 13 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:26<00:00,  1.43it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [13/50], average_loss: 2.7280

100%|| 207/207 [01:16<00:00,  2.70it/s]

Updating best test loss: 3.11271

  0%|          | 0/209 [00:00<?, ?it/s]
```

```
Starting epoch 14 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [14/50], average_loss: 2.6482

100%|| 207/207 [01:17<00:00,  2.67it/s]

Updating best test loss: 3.09734

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 15 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [15/50], average_loss: 2.6008

100%|| 207/207 [01:16<00:00,  2.69it/s]

Updating best test loss: 3.04292

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 16 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [16/50], average_loss: 2.5422

100%|| 207/207 [01:17<00:00,  2.67it/s]

Updating best test loss: 2.97164

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 17 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:25<00:00,  1.43it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [17/50], average_loss: 2.4825

100%|| 207/207 [01:17<00:00,  2.66it/s]
  0%|          | 0/209 [00:00<?, ?it/s]
```

```
Starting epoch 18 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [18/50], average_loss: 2.4441

100%|| 207/207 [01:18<00:00,  2.65it/s]

Updating best test loss: 2.96073

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 19 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:25<00:00,  1.43it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [19/50], average_loss: 2.3992

100%|| 207/207 [01:17<00:00,  2.67it/s]

Updating best test loss: 2.93470

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 20 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:26<00:00,  1.43it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [20/50], average_loss: 2.3502

100%|| 207/207 [01:18<00:00,  2.64it/s]
  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 21 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:25<00:00,  1.43it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [21/50], average_loss: 2.3184

100%|| 207/207 [01:18<00:00,  2.65it/s]

Updating best test loss: 2.92890
```

```
  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 22 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [22/50], average_loss: 2.2834

100%|| 207/207 [01:17<00:00,  2.66it/s]

Updating best test loss: 2.90533

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 23 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [23/50], average_loss: 2.1947

100%|| 207/207 [01:17<00:00,  2.66it/s]

Updating best test loss: 2.88942

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 24 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [24/50], average_loss: 2.1986

100%|| 207/207 [01:17<00:00,  2.68it/s]

Updating best test loss: 2.86331

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 25 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]
```

```
Epoch [25/50], average_loss: 2.1397

100%|| 207/207 [01:17<00:00,  2.66it/s]
  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 26 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [26/50], average_loss: 2.1328

100%|| 207/207 [01:17<00:00,  2.67it/s]

Updating best test loss: 2.84554

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 27 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [27/50], average_loss: 2.1212

100%|| 207/207 [01:17<00:00,  2.66it/s]
  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 28 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [28/50], average_loss: 2.0937

100%|| 207/207 [01:18<00:00,  2.64it/s]

Updating best test loss: 2.79059

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 29 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:25<00:00,  1.43it/s]
  0%|          | 0/207 [00:00<?, ?it/s]
```

```
Epoch [29/50], average_loss: 2.0312

100%|| 207/207 [01:18<00:00,  2.64it/s]
  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 30 / 50
Learning Rate for this epoch: 0.001

100%|| 209/209 [02:26<00:00,  1.43it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [30/50], average_loss: 2.0333

100%|| 207/207 [01:18<00:00,  2.63it/s]
  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 31 / 50
Learning Rate for this epoch: 0.0001

100%|| 209/209 [02:25<00:00,  1.43it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [31/50], average_loss: 1.9408

100%|| 207/207 [01:18<00:00,  2.65it/s]

Updating best test loss: 2.75277

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 32 / 50
Learning Rate for this epoch: 0.0001

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [32/50], average_loss: 1.8872

100%|| 207/207 [01:18<00:00,  2.64it/s]

Updating best test loss: 2.73873

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 33 / 50
Learning Rate for this epoch: 0.0001

100%|| 209/209 [02:25<00:00,  1.43it/s]
  0%|          | 0/207 [00:00<?, ?it/s]
```

```
Epoch [33/50], average_loss: 1.8556

100%|| 207/207 [01:17<00:00,  2.68it/s]

Updating best test loss: 2.73123

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 34 / 50
Learning Rate for this epoch: 0.0001

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [34/50], average_loss: 1.8356

100%|| 207/207 [01:17<00:00,  2.68it/s]

Updating best test loss: 2.73014

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 35 / 50
Learning Rate for this epoch: 0.0001

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [35/50], average_loss: 1.8448

100%|| 207/207 [01:18<00:00,  2.64it/s]

Updating best test loss: 2.72601

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 36 / 50
Learning Rate for this epoch: 0.0001

100%|| 209/209 [02:26<00:00,  1.43it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [36/50], average_loss: 1.8344

100%|| 207/207 [01:17<00:00,  2.67it/s]
  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 37 / 50
Learning Rate for this epoch: 0.0001
```

```
100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [37/50], average_loss: 1.7979

100%|| 207/207 [01:18<00:00,  2.64it/s]

Updating best test loss: 2.72210

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 38 / 50
Learning Rate for this epoch: 0.0001

100%|| 209/209 [02:26<00:00,  1.43it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [38/50], average_loss: 1.7953

100%|| 207/207 [01:17<00:00,  2.66it/s]

Updating best test loss: 2.71421

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 39 / 50
Learning Rate for this epoch: 0.0001

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [39/50], average_loss: 1.8025

100%|| 207/207 [01:16<00:00,  2.71it/s]
  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 40 / 50
Learning Rate for this epoch: 0.0001

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [40/50], average_loss: 1.7795

100%|| 207/207 [01:15<00:00,  2.73it/s]
  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 41 / 50
Learning Rate for this epoch: 1e-05
```

```
100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [41/50], average_loss: 1.7761

100%|| 207/207 [01:16<00:00,  2.72it/s]
  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 42 / 50
Learning Rate for this epoch: 1e-05

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [42/50], average_loss: 1.7786

100%|| 207/207 [01:16<00:00,  2.71it/s]
  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 43 / 50
Learning Rate for this epoch: 1e-05

100%|| 209/209 [02:25<00:00,  1.43it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [43/50], average_loss: 1.7619

100%|| 207/207 [01:16<00:00,  2.70it/s]
  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 44 / 50
Learning Rate for this epoch: 1e-05

100%|| 209/209 [02:25<00:00,  1.43it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [44/50], average_loss: 1.7853

100%|| 207/207 [01:16<00:00,  2.71it/s]
  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 45 / 50
Learning Rate for this epoch: 1e-05

100%|| 209/209 [02:26<00:00,  1.43it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [45/50], average_loss: 1.7508
```

```
100%|| 207/207 [01:16<00:00,  2.70it/s]
  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 46 / 50
Learning Rate for this epoch: 1e-05

100%|| 209/209 [02:25<00:00,  1.43it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [46/50], average_loss: 1.7854

100%|| 207/207 [01:17<00:00,  2.69it/s]
  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 47 / 50
Learning Rate for this epoch: 1e-05

100%|| 209/209 [02:26<00:00,  1.43it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [47/50], average_loss: 1.7907

100%|| 207/207 [01:17<00:00,  2.68it/s]
  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 48 / 50
Learning Rate for this epoch: 1e-05

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [48/50], average_loss: 1.7586

100%|| 207/207 [01:16<00:00,  2.70it/s]

Updating best test loss: 2.71180

  0%|          | 0/209 [00:00<?, ?it/s]


Starting epoch 49 / 50
Learning Rate for this epoch: 1e-05

100%|| 209/209 [02:25<00:00,  1.44it/s]
  0%|          | 0/207 [00:00<?, ?it/s]

Epoch [49/50], average_loss: 1.7766

100%|| 207/207 [01:16<00:00,  2.69it/s]
  0%|          | 0/209 [00:00<?, ?it/s]
```

```
Starting epoch 50 / 50
Learning Rate for this epoch: 1e-05
```

```
100%|| 209/209 [02:25<00:00,  1.43it/s]
  0%|          | 0/207 [00:00<?, ?it/s]
```

```
Epoch [50/50], average_loss: 1.7663
```

```
100%|| 207/207 [01:16<00:00,  2.70it/s]
```

Now, take a glance at how your detector works:

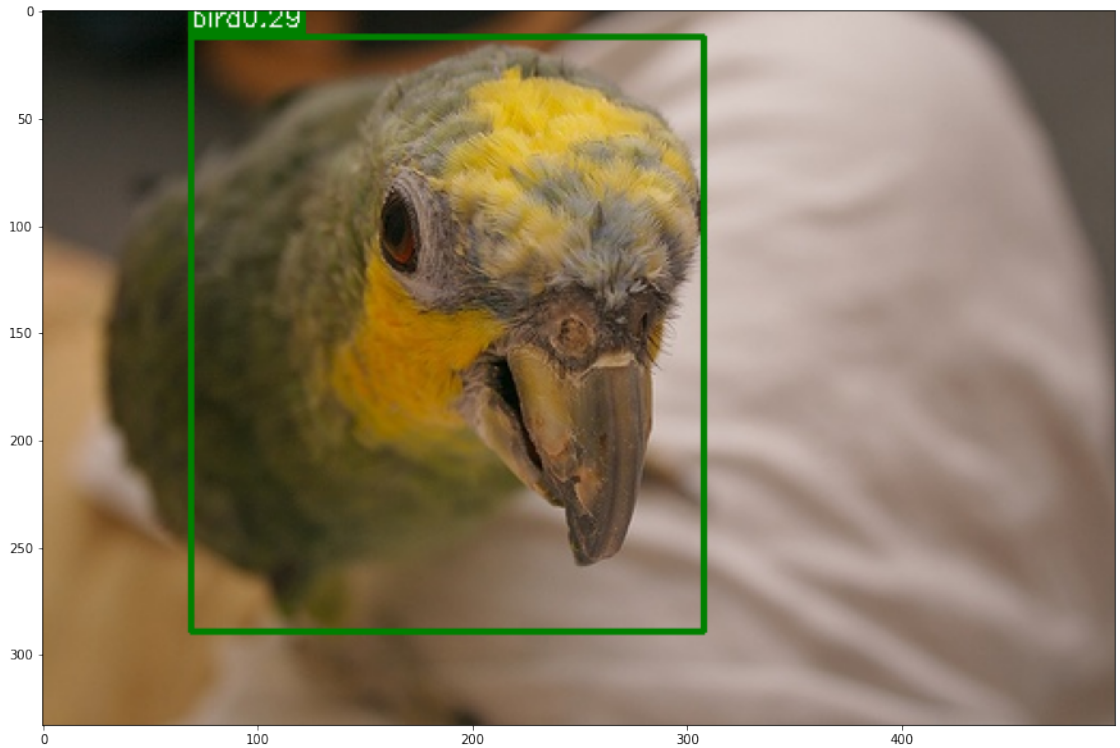
```
[ ]: net.eval()
net.load_state_dict(torch.load('best_detector.pth'))
# select random image from train set
image_name = random.choice(train_dataset.fnames)
image = cv2.imread(os.path.join(file_root_train, image_name))
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
threshold = 0.1
print('predicting...')
print(image.shape)
result = predict_image(net, image_name, root_img_directory=file_root_train,␣
 ↪threshold=threshold)
for left_up, right_bottom, class_name, _, prob in result:
    color = COLORS[VOC_CLASSES.index(class_name)]
    cv2.rectangle(image, left_up, right_bottom, color, 2)
    label = class_name + str(round(prob, 2))
    text_size, baseline = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.4,␣
 ↪1)
    p1 = (left_up[0], left_up[1] - text_size[1])
    cv2.rectangle(image, (p1[0] - 2 // 2, p1[1] - 2 - baseline), (p1[0] +␣
 ↪text_size[0], p1[1] + text_size[1]),
                  color, -1)
    cv2.putText(image, label, (p1[0], p1[1] + baseline), cv2.
 ↪FONT_HERSHEY_SIMPLEX, 0.4, (255, 255, 255), 1, 8)

plt.figure(figsize = (15,15))
plt.imshow(image)
```

```
predicting...
(333, 500, 3)
```

```
/content/drive/My Drive/DL_Fall_2020/Assignment_3/predict.py:99: UserWarning:
This overload of nonzero is deprecated:
        nonzero()
Consider using one of the following signatures instead:
        nonzero(*, bool as_tuple) (Triggered internally at
/pytorch/torch/csrc/utils/python_arg_parser.cpp:766.)
```

```
ids = (ovr<=threshold).nonzero().squeeze()
```

[ ]: <matplotlib.image.AxesImage at 0x7f0c5efc6710>



3.5 Evaluate on Test [20 pts]

To evaluate detection results we use mAP (mean of average precision over each class), You are expected to get an map of at least 49.

[ ]:
```
test_aps = evaluate(net, test_dataset_file=annotation_file_test,␣
↪threshold=threshold)
```

```
---Evaluate model on test samples---

100%|| 4950/4950 [02:57<00:00, 27.89it/s]

---class aeroplane ap 0.5367103438577185---
---class bicycle ap 0.622378036052192---
---class bird ap 0.4631083921693554---
---class boat ap 0.27583423213577085---
---class bottle ap 0.22441527413667484---
---class bus ap 0.5751149356057058---
---class car ap 0.6652098227982262---
---class cat ap 0.7335162700396873---
---class chair ap 0.3008076892882541---
```

```
---class cow ap 0.5112586761153028---
---class diningtable ap 0.3168475486167006---
---class dog ap 0.6098020243552785---
---class horse ap 0.6539583349825462---
---class motorbike ap 0.5875314385145219---
---class person ap 0.5296722219000412---
---class pottedplant ap 0.18127928192377749---
---class sheep ap 0.4996816985315411---
---class sofa ap 0.5100543175183547---
---class train ap 0.6913178761475878---
---class tvmonitor ap 0.4869931534028143---
---map 0.4987745784046026---
```

3.5.1 Guidelines for Downloading PDF in Google Colab

- Run below cells only in Google Colab, Comment out in case of Jupyter notebook

[ ]:
```
#Run below two lines (in google colab), installation steps to get .pdf of the␣
↪notebook

!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
!pip install pypandoc

# After installation, comment above two lines and run again to remove␣
↪installation comments from the notebook.
```

[ ]:
```
# Find path to your notebook file in drive and enter in below line

!jupyter nbconvert --to PDF "/content/drive/My Drive/DL_Fall_2020/Assignment_3/
↪Assignment_3.ipynb"

#Example: "/content/drive/My Drive/DL_Fall_2020/Assignment_2/DL_Assignment_2.
↪ipynb"
```