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

```
Mounted at /content/drive
```

```
import os
datadir = '/content/drive/My Drive/assignment3_part2/'
if not os.path.exists(datadir):
  !ln -s '/content/drive/My Drive/assignment3_part2/' $datadir
os.chdir(datadir)
!pwd
```

```
/content/drive/My Drive/assignment3_part2
```

```
import os
import random

import cv2
import numpy as np

import torch
from torch.utils.data import DataLoader
from torchvision import models

from src.resnet_yolo import resnet50
from yolo_loss import YoloLoss
from src.dataset import VocDetectorDataset
from src.eval_voc import evaluate
from src.predict import predict_image
from src.config import VOC_CLASSES, COLORS
from kaggle_submission import output_submission_csv

import matplotlib.pyplot as plt
import collections

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

Initialization

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

```
# 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)
```

To implement Yolo we will rely on a pretrained classifier as the backbone for our detection network. PyTorch offers a variety of models which are pretrained on ImageNet in the `torchvision.models` package. In particular, we will use the ResNet50 architecture as a base for our detector. This is different from the base architecture in the Yolo paper and also results in a different output grid size (14x14 instead of 7x7).

Models are typically pretrained on ImageNet since the dataset is very large (> 1 million images) and widely used. The pretrained model provides a very useful weight initialization for our detector, so that the network is able to learn quickly and effectively.

```
load_network_path = None #'checkpoints/best_detector.pth'
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
```

```
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
```

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.

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 datasets since the bounding box annotations must be kept consistent throughout the transformations.

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

```
file_root_train = 'data/VOCdevkit_2007/VOC2007/JPEGImages/'
annotation_file_train = 'data/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=2)
print('Loaded %d train images' % len(train_dataset))
```

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

```
file_root_test = 'data/VOCdevkit_2007/VOC2007test/JPEGImages/'
annotation_file_test = 'data/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=2)
print('Loaded %d test images' % len(test_dataset))
```

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

```
data = train_dataset[0]
```

```
!unzip data.zip
```

```
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009446.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009458.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009464.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009527.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009533.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009550.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009562.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009580.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009597.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009605.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009618.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009649.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009654.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009655.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009684.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009687.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009691.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009706.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009709.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009724.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009756.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009764.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009794.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009807.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009832.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009841.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009897.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009911.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009923.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009938.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009947.png
    inflating: data/VOCdevkit_2007/VOC2007/SegmentationObject/009950.png
```

```
type(data[0])
print(len(data))
```

```
4
```

Set up training tools

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

Train detector

```
best_test_loss = np.inf
learning_rate = 2e-3
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 = collections.defaultdict(int)

    for i, data in enumerate(train_loader) :
        data = (item.to(device) for item in data)
        images, target_boxes, target_cls, has_object_map = data
        pred = net(images)
        loss_dict = criterion(pred, target_boxes, target_cls, has_object_map)
        for key in loss_dict:
            total_loss[key] += loss_dict[key].item()

        optimizer.zero_grad()
        loss_dict['total_loss'].backward()
        optimizer.step()

        if (i+1) % 50 == 0:
            outstring = 'Epoch [%d/%d], Iter [%d/%d], Loss: ' % ((epoch+1, num_epochs, i+1, len(train_loader)))
            outstring += ', '.join( "%s=%.3f" % (key[:-5], val / (i+1)) for key, val in total_loss.items() )
            print(outstring)
```

```
# evaluate the network on the test data
if (epoch + 1) % 5 == 0:
    test_aps = evaluate(net, test_dataset_file=annotation_file_test, img_root=file_root_test)
    print(epoch, test_aps)
with torch.no_grad():
    test_loss = 0.0
    net.eval()
    for i, data in enumerate(test_loader):
        data = (item.to(device) for item in data)
        images, target_boxes, target_cls, has_object_map = data

        pred = net(images)
        loss_dict = criterion(pred, target_boxes, target_cls, has_object_map)
        test_loss += loss_dict['total_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(),'checkpoints/best_detector.pth')

if (epoch+1) in [5, 10, 20, 30, 40]:
    torch.save(net.state_dict(),'checkpoints/detector_epoch_%d.pth' % (epoch+1))

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

```
Starting epoch 47 / 50
Learning Rate for this epoch: 2e-05
Epoch [47/50], Iter [50/209], Loss: total=1.701, reg=0.792, containing_obj=0.563, no_obj=0.207, cls=0.138
Epoch [47/50], Iter [100/209], Loss: total=1.701, reg=0.801, containing_obj=0.558, no_obj=0.204, cls=0.137
Epoch [47/50], Iter [150/209], Loss: total=1.669, reg=0.780, containing_obj=0.552, no_obj=0.204, cls=0.134
Epoch [47/50], Iter [200/209], Loss: total=1.647, reg=0.766, containing_obj=0.546, no_obj=0.203, cls=0.132


Starting epoch 48 / 50
Learning Rate for this epoch: 2e-05
Epoch [48/50], Iter [50/209], Loss: total=1.565, reg=0.749, containing_obj=0.502, no_obj=0.207, cls=0.107
Epoch [48/50], Iter [100/209], Loss: total=1.596, reg=0.752, containing_obj=0.526, no_obj=0.200, cls=0.117
Epoch [48/50], Iter [150/209], Loss: total=1.595, reg=0.751, containing_obj=0.521, no_obj=0.201, cls=0.122
Epoch [48/50], Iter [200/209], Loss: total=1.617, reg=0.764, containing_obj=0.527, no_obj=0.201, cls=0.126
Updating best test loss: 2.56105


Starting epoch 49 / 50
Learning Rate for this epoch: 2e-05
Epoch [49/50], Iter [50/209], Loss: total=1.578, reg=0.730, containing_obj=0.517, no_obj=0.199, cls=0.132
Epoch [49/50], Iter [100/209], Loss: total=1.576, reg=0.734, containing_obj=0.514, no_obj=0.201, cls=0.126
Epoch [49/50], Iter [150/209], Loss: total=1.582, reg=0.741, containing_obj=0.516, no_obj=0.200, cls=0.125
Epoch [49/50], Iter [200/209], Loss: total=1.625, reg=0.762, containing_obj=0.535, no_obj=0.198, cls=0.130


Starting epoch 50 / 50
Learning Rate for this epoch: 2e-05
Epoch [50/50], Iter [50/209], Loss: total=1.579, reg=0.727, containing_obj=0.533, no_obj=0.203, cls=0.116
Epoch [50/50], Iter [100/209], Loss: total=1.592, reg=0.744, containing_obj=0.525, no_obj=0.200, cls=0.123
Epoch [50/50], Iter [150/209], Loss: total=1.613, reg=0.750, containing_obj=0.536, no_obj=0.198, cls=0.128
Epoch [50/50], Iter [200/209], Loss: total=1.609, reg=0.747, containing_obj=0.534, no_obj=0.201, cls=0.127
---Evaluate model on test samples---
100%|██████████| 4950/4950 [02:18<00:00, 35.84it/s]
---class aeroplane ap 0.5494678944469458---
---class bicycle ap 0.6036336165666508---
---class bird ap 0.5139082438414576---
---class boat ap 0.3465606443857522---
---class bottle ap 0.21638629569269344---
---class bus ap 0.6117935928088871---
---class car ap 0.6849521638825125---
---class cat ap 0.7096037072546938---
---class chair ap 0.32862583362528436---
---class cow ap 0.5491099781674855---
---class diningtable ap 0.3454889502918649---
---class dog ap 0.6459411856636212---
---class horse ap 0.7210845735067598---
---class motorbike ap 0.588970691874612---
---class person ap 0.5580267048438299---
---class pottedplant ap 0.19098228353353233---
---class sheep ap 0.5020023570445826---
---class sofa ap 0.5176211724512401---
---class train ap 0.6762971184140794---
---class tvmonitor ap 0.513587650555432---
---map 0.5187022329425959---
49 [0.5494678944469458, 0.6036336165666508, 0.5139082438414576, 0.3465606443857522, 0.21638629569269344, 0.6117935928088871, 0.68495216
```

View example predictions

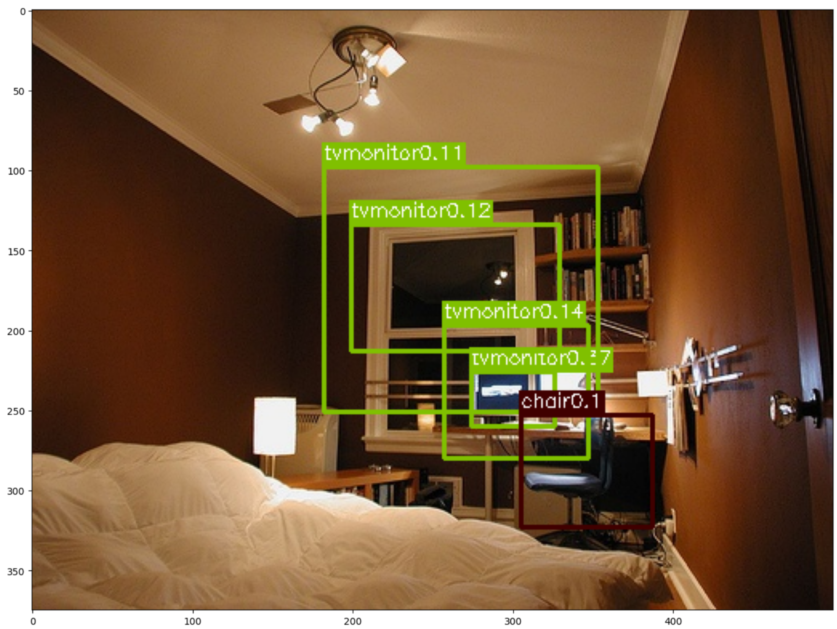
```
net.eval()

# select random image from test set
image_name = random.choice(test_dataset.fnames)
image = cv2.imread(os.path.join(file_root_test, image_name))
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

print('predicting...')
result = predict_image(net, image_name, root_img_directory=file_root_test)
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...
<matplotlib.image.AxesImage at 0x7ff149845310>
```



Evaluate on Test

To evaluate detection results we use mAP (mean of average precision over each class)

```
test_aps = evaluate(net, test_dataset_file=annotation_file_test, img_root=file_root_test)
```

```
---Evaluate model on test samples---
100%|██████████| 4950/4950 [02:18<00:00, 35.75it/s]
---class aeroplane ap 0.5494678944469458---
---class bicycle ap 0.6036336165666508---
---class bird ap 0.5139082438414576---
---class boat ap 0.3465606443857522---
---class bottle ap 0.21638629569269344---
---class bus ap 0.6117935928088871---
---class car ap 0.6849521638825125---
---class cat ap 0.7096037072546938---
---class chair ap 0.32862583362528436---
---class cow ap 0.5491099781674855---
---class diningtable ap 0.3454889502918649---
---class dog ap 0.6459411856636212---
---class horse ap 0.7210845735067598---
---class motorbike ap 0.588970691874612---
---class person ap 0.5580267048438299---
---class pottedplant ap 0.19098228353353233---
---class sheep ap 0.5020023570445826---
---class sofa ap 0.5176211724512401---
---class train ap 0.6762971184140794---
---class tvmonitor ap 0.513587650555432---
---map 0.5187022329425959---
```

Cell added to get intermediate mAP values for students

```
network_paths = ['detector_epoch_%d.pth' % epoch for epoch in [5, 10, 20, 30, 40]]+['detector.pth']
for load_network_path in network_paths:
    print('Loading saved network from {}'.format(load_network_path))
    net_loaded =  resnet50().to(device)
    net_loaded.load_state_dict(torch.load(load_network_path))
    evaluate(net_loaded, test_dataset_file=annotation_file_test)
```

```
output_submission_csv('my_new_solution.csv', test_aps)
```

✓ 0s completed at 19:02