MP3_P2

April 6, 2022

[45]:
```
import zipfile
with zipfile.ZipFile("TestImages.zip","r") as zip_ref:
    zip_ref.extractall()
```

[1]:
```
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
```

0.1 Initialization

[2]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

[3]:
```
device
```

[3]:
```
device(type='cuda', index=0)
```

[4]:
```
# 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.

[5]:
```
load_network_path = 'checkpoints/best_detector.pth' #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)
```

```
Loading saved network from checkpoints/best_detector.pth
```

[6]:
```
learning_rate = 0.001
num_epochs = 10 #note that the network was ran for 50 epochs and best losses␣
 ↪saved.
#This is the sesond running for 10 epochs
batch_size = 24

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

0.2 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.

[8]:
```
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=Tr
↪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
```

[9]:
```
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=Fals
↪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
```

[10]:
```
data = train_dataset[0]
```

0.3 Set up training tools

[11]:
```
criterion = YoloLoss(S, B, lambda_coord, lambda_noobj)

optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9,␣
↪weight_decay=5e-4)

#optimizer = torch.optim.Adam(net.parameters(), lr= learning_rate)
```

0.4 Train detector

[22]:
```
best_test_loss = np.inf
learning_rate = 1e-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()
        torch.autograd.set_detect_anomaly(True)
        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 1 / 10
Learning Rate for this epoch: 0.001
Epoch [1/10], Iter [50/209], Loss: total=1.937, reg=5.006,
containing_obj=11.784, no_obj=9.213, cls=5.067
Epoch [1/10], Iter [100/209], Loss: total=1.990, reg=5.115,
containing_obj=12.203, no_obj=9.560, cls=5.201
Epoch [1/10], Iter [150/209], Loss: total=1.979, reg=5.058,
containing_obj=12.301, no_obj=9.655, cls=5.085
Epoch [1/10], Iter [200/209], Loss: total=1.968, reg=5.001,
containing_obj=12.349, no_obj=9.696, cls=5.039
Updating best test loss: 2.93192


Starting epoch 2 / 10
Learning Rate for this epoch: 0.001
Epoch [2/10], Iter [50/209], Loss: total=1.935, reg=4.833,
containing_obj=12.318, no_obj=10.475, cls=4.717
Epoch [2/10], Iter [100/209], Loss: total=1.915, reg=4.812,
containing_obj=12.187, no_obj=10.455, cls=4.476
Epoch [2/10], Iter [150/209], Loss: total=1.949, reg=4.906,
containing_obj=12.576, no_obj=10.374, cls=4.476
Epoch [2/10], Iter [200/209], Loss: total=1.944, reg=4.859,
```

```
containing_obj=12.544, no_obj=10.460, cls=4.591


Starting epoch 3 / 10
Learning Rate for this epoch: 0.001
Epoch [3/10], Iter [50/209], Loss: total=1.906, reg=4.736,
containing_obj=12.073, no_obj=11.083, cls=4.439
Epoch [3/10], Iter [100/209], Loss: total=1.940, reg=4.846,
containing_obj=12.502, no_obj=10.644, cls=4.514
Epoch [3/10], Iter [150/209], Loss: total=1.949, reg=4.861,
containing_obj=12.458, no_obj=10.662, cls=4.678
Epoch [3/10], Iter [200/209], Loss: total=1.938, reg=4.839,
containing_obj=12.373, no_obj=10.551, cls=4.658


Starting epoch 4 / 10
Learning Rate for this epoch: 0.001
Epoch [4/10], Iter [50/209], Loss: total=1.951, reg=4.839,
containing_obj=12.921, no_obj=10.716, cls=4.354
Epoch [4/10], Iter [100/209], Loss: total=1.912, reg=4.746,
containing_obj=12.468, no_obj=10.544, cls=4.406
Epoch [4/10], Iter [150/209], Loss: total=1.887, reg=4.656,
containing_obj=12.420, no_obj=10.508, cls=4.329
Epoch [4/10], Iter [200/209], Loss: total=1.890, reg=4.665,
containing_obj=12.389, no_obj=10.478, cls=4.395


Starting epoch 5 / 10
Learning Rate for this epoch: 0.001
Epoch [5/10], Iter [50/209], Loss: total=1.938, reg=4.732,
containing_obj=12.757, no_obj=10.691, cls=4.749
Epoch [5/10], Iter [100/209], Loss: total=1.899, reg=4.650,
containing_obj=12.483, no_obj=10.575, cls=4.557
Epoch [5/10], Iter [150/209], Loss: total=1.868, reg=4.561,
containing_obj=12.456, no_obj=10.554, cls=4.303
Epoch [5/10], Iter [200/209], Loss: total=1.862, reg=4.546,
containing_obj=12.371, no_obj=10.509, cls=4.323
---Evaluate model on test samples---

100%|                                        |
4950/4950 [04:46<00:00, 17.30it/s]

---class aeroplane ap 0.5545325711954947---
---class bicycle ap 0.6511538390208185---
---class bird ap 0.5290218909950416---
---class boat ap 0.2679847749536596---
---class bottle ap 0.25152677979898697---
---class bus ap 0.609987226834106---
---class car ap 0.6996934596153546---
```

```
---class cat ap 0.7130208566768952---
---class chair ap 0.33759562702282137---
---class cow ap 0.5663174793194221---
---class diningtable ap 0.35490848168178724---
---class dog ap 0.6835951689673284---
---class horse ap 0.7120147044683036---
---class motorbike ap 0.5853930164231931---
---class person ap 0.5929755994645363---
---class pottedplant ap 0.21663588364717618---
---class sheep ap 0.5143568689819437---
---class sofa ap 0.5055179023056386---
---class train ap 0.6090999188074084---
---class tvmonitor ap 0.4919421293343803---
---map 0.5223637089757149---
4 [0.5545325711954947, 0.6511538390208185, 0.5290218909950416,
0.2679847749536596, 0.25152677979898697, 0.609987226834106, 0.6996934596153546,
0.7130208566768952, 0.33759562702282137, 0.5663174793194221,
0.35490848168178724, 0.6835951689673284, 0.7120147044683036, 0.5853930164231931,
0.5929755994645363, 0.21663588364717618, 0.5143568689819437, 0.5055179023056386,
0.6090999188074084, 0.4919421293343803]
Updating best test loss: 2.90610


Starting epoch 6 / 10
Learning Rate for this epoch: 0.001
Epoch [6/10], Iter [50/209], Loss: total=1.843, reg=4.576,
containing_obj=12.089, no_obj=10.450, cls=4.044
Epoch [6/10], Iter [100/209], Loss: total=1.849, reg=4.518,
containing_obj=12.283, no_obj=10.557, cls=4.238
Epoch [6/10], Iter [150/209], Loss: total=1.837, reg=4.507,
containing_obj=12.182, no_obj=10.497, cls=4.129
Epoch [6/10], Iter [200/209], Loss: total=1.840, reg=4.516,
containing_obj=12.184, no_obj=10.510, cls=4.134
Updating best test loss: 2.90269


Starting epoch 7 / 10
Learning Rate for this epoch: 0.001
Epoch [7/10], Iter [50/209], Loss: total=1.705, reg=4.104,
containing_obj=11.284, no_obj=10.913, cls=3.653
Epoch [7/10], Iter [100/209], Loss: total=1.736, reg=4.202,
containing_obj=11.600, no_obj=10.661, cls=3.712
Epoch [7/10], Iter [150/209], Loss: total=1.774, reg=4.280,
containing_obj=11.854, no_obj=10.645, cls=4.002
Epoch [7/10], Iter [200/209], Loss: total=1.788, reg=4.336,
containing_obj=11.923, no_obj=10.632, cls=3.995
```

```
Starting epoch 8 / 10
Learning Rate for this epoch: 0.001
Epoch [8/10], Iter [50/209], Loss: total=1.762, reg=4.241,
containing_obj=12.002, no_obj=10.451, cls=3.853
Epoch [8/10], Iter [100/209], Loss: total=1.757, reg=4.246,
containing_obj=11.871, no_obj=10.277, cls=3.941
Epoch [8/10], Iter [150/209], Loss: total=1.794, reg=4.371,
containing_obj=11.975, no_obj=10.341, cls=4.050
Epoch [8/10], Iter [200/209], Loss: total=1.792, reg=4.388,
containing_obj=11.850, no_obj=10.390, cls=4.011
Updating best test loss: 2.87515


Starting epoch 9 / 10
Learning Rate for this epoch: 0.001
Epoch [9/10], Iter [50/209], Loss: total=1.732, reg=4.254,
containing_obj=11.522, no_obj=10.632, cls=3.449
Epoch [9/10], Iter [100/209], Loss: total=1.735, reg=4.223,
containing_obj=11.644, no_obj=10.568, cls=3.586
Epoch [9/10], Iter [150/209], Loss: total=1.746, reg=4.269,
containing_obj=11.569, no_obj=10.460, cls=3.751
Epoch [9/10], Iter [200/209], Loss: total=1.760, reg=4.288,
containing_obj=11.674, no_obj=10.488, cls=3.892


Starting epoch 10 / 10
Learning Rate for this epoch: 0.001
Epoch [10/10], Iter [50/209], Loss: total=1.716, reg=4.133,
containing_obj=11.375, no_obj=10.381, cls=3.949
Epoch [10/10], Iter [100/209], Loss: total=1.714, reg=4.128,
containing_obj=11.575, no_obj=10.456, cls=3.694
Epoch [10/10], Iter [150/209], Loss: total=1.693, reg=4.046,
containing_obj=11.354, no_obj=10.419, cls=3.833
Epoch [10/10], Iter [200/209], Loss: total=1.720, reg=4.144,
containing_obj=11.525, no_obj=10.421, cls=3.832
---Evaluate model on test samples---

100%|                                       |
4950/4950 [04:47<00:00, 17.22it/s]

---class aeroplane ap 0.5287083512441363---
---class bicycle ap 0.6794630662933854---
---class bird ap 0.6065462017333644---
---class boat ap 0.38775833040091595---
---class bottle ap 0.20614022283064154---
---class bus ap 0.5958759227863629---
---class car ap 0.7033132876517104---
---class cat ap 0.7118655564270318---
---class chair ap 0.309649642552588---
```

```
---class cow ap 0.5384747355890704---
---class diningtable ap 0.40990832507238756---
---class dog ap 0.6503395851651028---
---class horse ap 0.7163792946499739---
---class motorbike ap 0.5763539266357915---
---class person ap 0.5909687913971275---
---class pottedplant ap 0.19129799562290956---
---class sheep ap 0.5436582303114936---
---class sofa ap 0.4751108748101551---
---class train ap 0.6634858184258798---
---class tvmonitor ap 0.5465783044934311---
---map 0.5315938232046731---
9 [0.5287083512441363, 0.6794630662933854, 0.6065462017333644,
0.38775833040091595, 0.20614022283064154, 0.5958759227863629,
0.7033132876517104, 0.7118655564270318, 0.309649642552588, 0.5384747355890704,
0.40990832507238756, 0.6503395851651028, 0.7163792946499739, 0.5763539266357915,
0.5909687913971275, 0.19129799562290956, 0.5436582303114936, 0.4751108748101551,
0.6634858184258798, 0.5465783044934311]
```

1 View example predictions

[39]:
```python
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…
```

[39]: <matplotlib.image.AxesImage at 0x7f1fcf76d390>

[46]:
```
#File root test from YOLO video on Youtube for EXTRA CREDIT
#I am using 20 screenshots from the video and i will be showing 5 tests under␣
↪this cell

file_root_test_YoutubeData = 'data/VOCdevkit_2007/VOC2007test/TestImages/'
annotation_file_test_new = 'data/yoloTest.txt'

test_dataset_new =␣
↪VocDetectorDataset(root_img_dir=file_root_test_YoutubeData,dataset_file=annotation_file_tes
↪S=S)
test_loader =␣
↪DataLoader(test_dataset_new,batch_size=batch_size,shuffle=False,num_workers=2)
print('Loaded %d test images' % len(test_dataset_new))
```

```
Initializing dataset
Loaded 20 test images
```

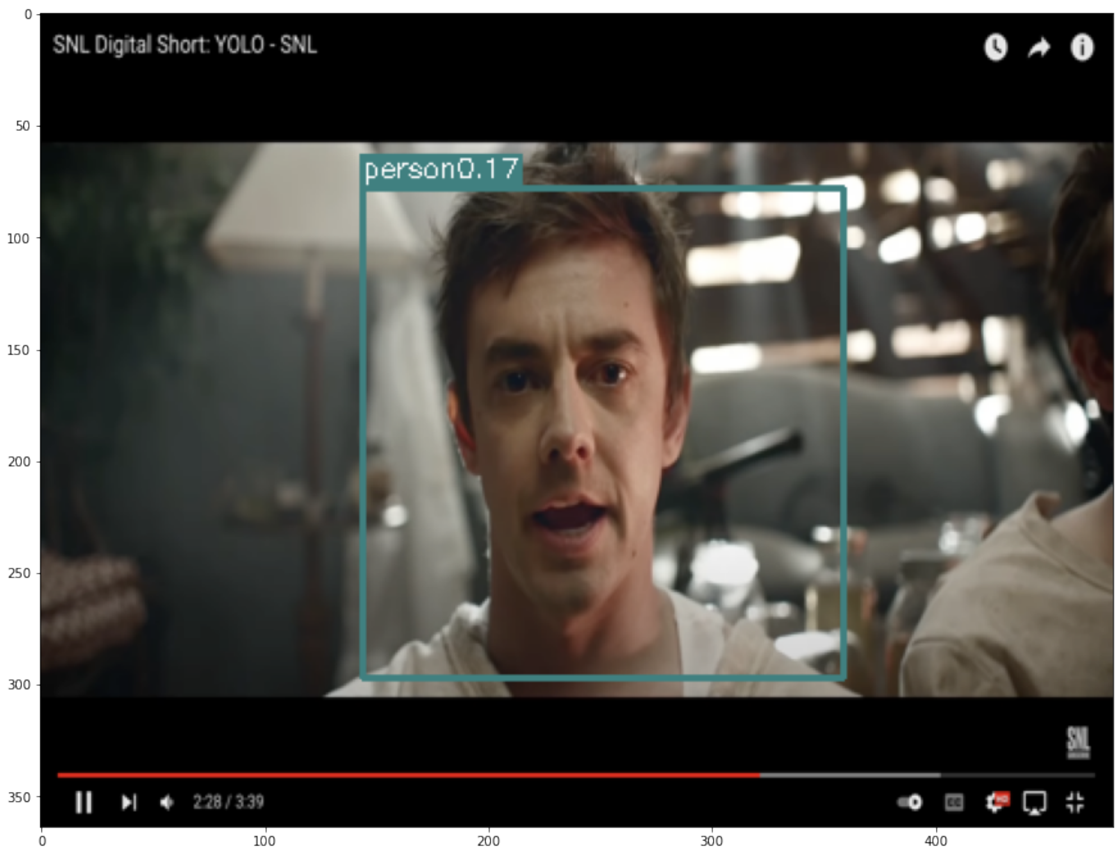
[50]:
```
#Shwoing first prediction from the YOLO Youtube video

net.eval()
```

```python
# select random image from test set
image_name = random.choice(test_dataset_new.fnames)
image = cv2.imread(os.path.join(file_root_test_YoutubeData, image_name))
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

print('predicting...')
result = predict_image(net, image_name,␣
 ↪root_img_directory=file_root_test_YoutubeData)
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…
```

[50]:

```
<matplotlib.image.AxesImage at 0x7f21945d1f10>
```



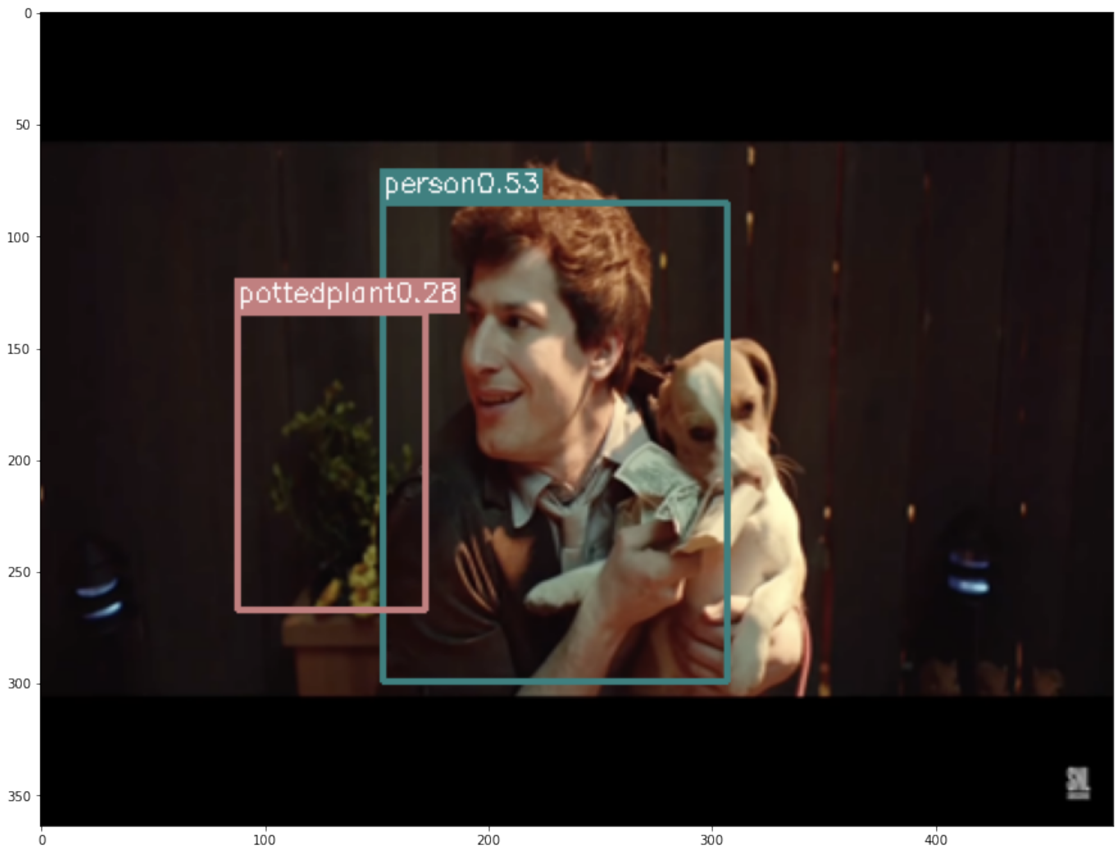
[52]:

```
#Shwoing second prediction from the YOLO Youtube video
net.eval()

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

print('predicting...')
result = predict_image(net, image_name, root_img_directory=file_root_test_YoutubeData)
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…

[52]: <matplotlib.image.AxesImage at 0x7f21944c7c90>

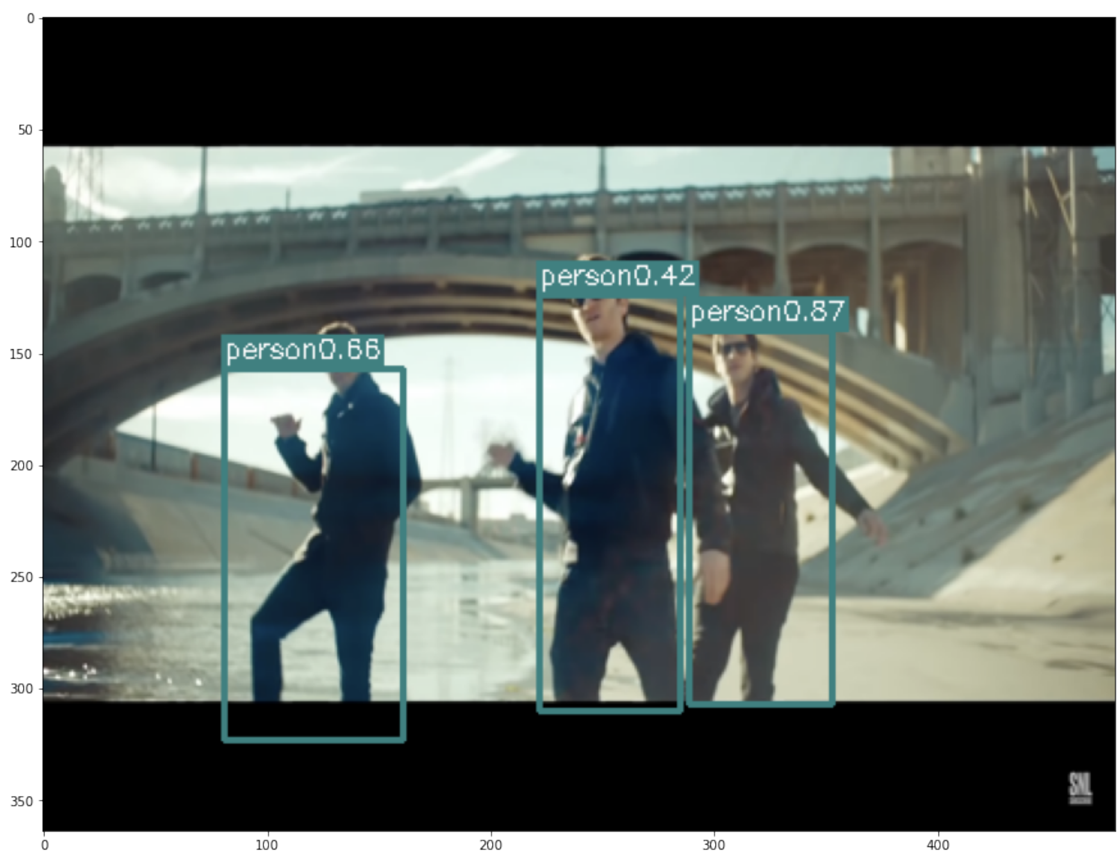


[53]:
```
#Shwoing third prediction from the YOLO Youtube video
net.eval()

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

print('predicting...')
```

```python
result = predict_image(net, image_name,␣
↪root_img_directory=file_root_test_YoutubeData)
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…

[53]: <matplotlib.image.AxesImage at 0x7f219443de50>

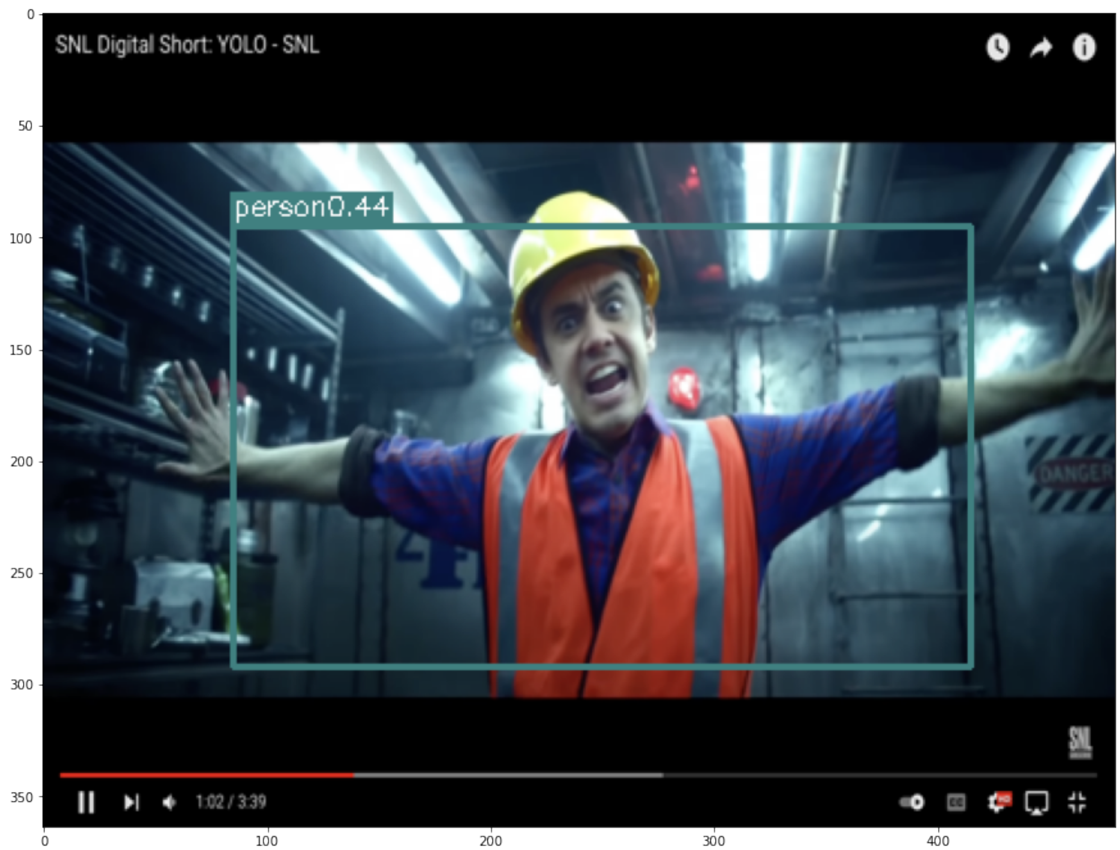


[61]:

```
#Shwoing 4th prediction from the YOLO Youtube video
net.eval()

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

print('predicting...')
result = predict_image(net, image_name,␣
↪root_img_directory=file_root_test_YoutubeData)
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…

[61]: <matplotlib.image.AxesImage at 0x7f21940b8c50>



[63]:
```
#Shwoing 5th prediction from the YOLO Youtube video
net.eval()

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

print('predicting...')
result = predict_image(net, image_name, root_img_directory=file_root_test_YoutubeData)
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…
```

[63]:
```
<matplotlib.image.AxesImage at 0x7f2193fb7ed0>
```

1.1 Evaluate on Test

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

[25]:
```
test_aps = evaluate(net, test_dataset_file=annotation_file_test,␣
↪img_root=file_root_test)
```

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

```
100%|                                        |
```

```
4950/4950 [04:46<00:00, 17.26it/s]

---class aeroplane ap 0.5287083512441363---
---class bicycle ap 0.6794630662933854---
---class bird ap 0.6065462017333644---
---class boat ap 0.38775833040091595---
---class bottle ap 0.20614022283064154---
---class bus ap 0.5958759227863629---
---class car ap 0.7033132876517104---
---class cat ap 0.7118655564270318---
---class chair ap 0.309649642552588---
---class cow ap 0.5384747355890704---
---class diningtable ap 0.40990832507238756---
---class dog ap 0.6503395851651028---
---class horse ap 0.7163792946499739---
---class motorbike ap 0.5763539266357915---
---class person ap 0.5909687913971275---
---class pottedplant ap 0.19129799562290956---
---class sheep ap 0.5436582303114936---
---class sofa ap 0.4751108748101551---
---class train ap 0.6634858184258798---
---class tvmonitor ap 0.5465783044934311---
---map 0.5315938232046731---
```

1.1.1 Cell added to get intermediate mAP values for students

[27]:

```
#network paths = ['checkpoints/detector _epoch_d.pth'
#checkpoints/best_detector.pth
#network_paths = ['checkpoints/best_detector.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)
```

[28]:

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

[ ]:

```

```