Lab04

In this lab, yolov4 was implimented using DarkNet as its backbone. Since the given config file as well as the architechture of the DarkNet were specifically for yolov3, a new config file and some modifications to the DarkNet architechture were necessary. The config file for yolov4 and the pretrained weights were taken from https://github.com/AlexeyAB/darknet/blob/master/cfg/yolov4-csp.cfg and https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.weight respectively.

Apart from the those, some modifications of the DarkNet architechture are as follows:

Maxpool (Ref: Prof. Matt)

```
(in forward:)
    if module_type == "convolutional" or module_type == "upsample" or module_type == "maxpool":
        x = self.module_listi

(in create_modules:)
    elif x["type"] == "maxpool":
        stride = int(x["stride"])
        size = int(x["size"])
        assert size % 2
        maxpool = nn.MaxPool2d(kernel_size=size, stride=stride, padding=size // 2)
        module.addmodule("maxpool{0}".format(index), maxpool)
```

Mish activation

```
class Mish(nn.Module):
    def init(self):
        super().init()
    def forward(self, x):
        return x * torch.tanh(F.softplus(x))

In create module
    elif activation == "mish":
        activn = Mish()
        module.addmodule("mish{0}".format(index), activn)
```

The input image size from 416 x 416 to 608 x 608

The route (Ref: Prof. Matt)

```
    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: # 1 item in layer

            x = outputs[i + (layers[0])]
        else:# more than 1 item in layer
            if len(layers) == 4:# 4 items in layer

                if (layers[1]) > 0:
                    layers[1] = layers[1] - i
                if (layers[2]) > 0:
                    layers[2] = layers[2] - i
                if (layers[3]) > 0:
                    layers[3] = layers[3] - i
                map1 = outputs[i + layers[0]]
                map2 = outputs[i + layers[1]]
                map3 = outputs[i + layers[2]]
                map4 = outputs[i + layers[3]]
                x = torch.cat((map1, map2, map3, map4), 1)
            else: # 2 items in layer

                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)
```

Conversion of colors in prep_images

```
    def prep_image(img, inp_dim):
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = letterbox_image(img, (inp_dim, inp_dim))
        return torch.from_numpy(img.transpose(2, 0, 1)).float().div(255.0).unsqueeze(0)
```

As for the inference, I have tested yolov4 with some COCO images, the results can be seen below.
Note that I have also converted the images from RGB back to BGR since during the prep_image the images were converted from BGR to RGB.

In [2]:

```
from __future__ import division
import time
import torch
import torch.nn as nn
from torch.autograd import Variable
import numpy as np
import cv2
from util import *
import argparse
import os
import os.path as osp
from darknet import Darknet
import pickle as pkl
import pandas as pd
```

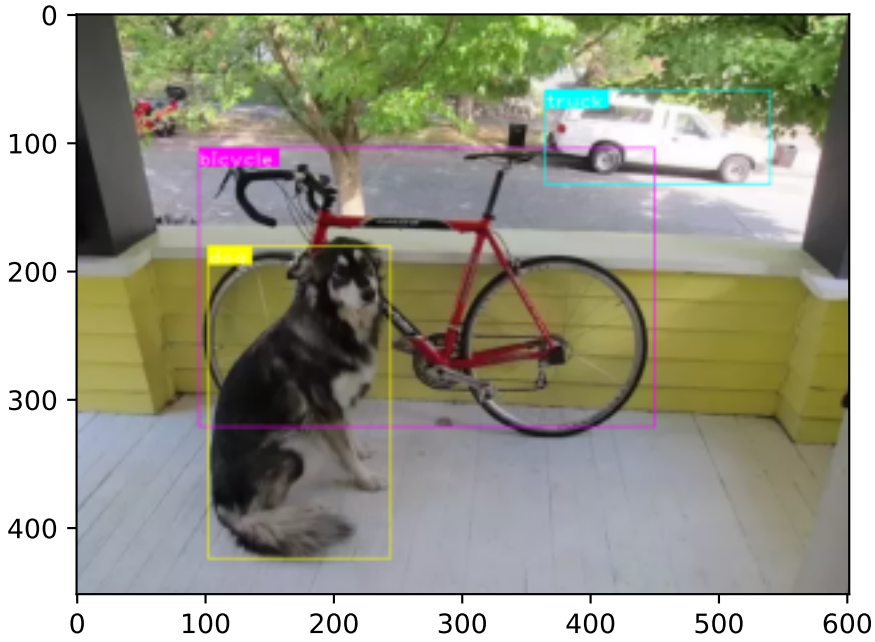
```
import random

images = "cocoimages"
batch_size = 4
confidence = 0.5
nms_thesh = 0.4
start = 0
CUDA = torch.cuda.is_available()

num_classes = 80
classes = load_classes("data/coco.names")

#Set up the neural network

print("Loading network.....",file=open("output_coco.txt", "a"))
model = Darknet("cfg/yolov4.cfg")
model.module_list[114].conv_114 = nn.Conv2d(2048, 512, kernel_size=(1, 1), st
model.load_weights("yolov4.weights")
print("Network successfully loaded",file=open("output_coco.txt", "a"))

model.net_info["height"] = 416
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()

read_dir = time.time()

# Detection phase

try:
    imlist = [osp.join(osp.realpath('.'), images, img) for img in os.listdir(
except NotADirectoryError:
    imlist = []
    imlist.append(osp.join(osp.realpath('.'), images))
except FileNotFoundError:
    print ("No file or directory with the name {}".format(images),file=open("
    exit()

if not os.path.exists("des"):
    os.makedirs("des")

load_batch = time.time()
loaded_ims = [cv2.imread(x) for x in imlist]

im_batches = list(map(prep_image, loaded_ims, [inp_dim for x in range(len(iml
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)


leftover = 0
if (len(im_dim_list) % batch_size):
    leftover = 1

if batch_size != 1:
    num_batches = len(imlist) // batch_size + leftover
    im_batches = [torch.cat((im_batches[i*batch_size : min((i +  1)*batch_siz
```

```
                        len(im_batches))]))  for i in range(num_batches)]

write = 0

if CUDA:
    im_dim_list = im_dim_list.cuda()

start_det_loop = time.time()
for i, batch in enumerate(im_batches):
    # Load the image
    start = time.time()
    if CUDA:
        batch = batch.cuda()
    with torch.no_grad():
        prediction = model(Variable(batch), CUDA)

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

    end = time.time()

    if type(prediction) == int:

        for im_num, image in enumerate(imlist[i*batch_size: min((i +  1)*batch
            im_id = i*batch_size + im_num
            print("{0:20s} predicted in {1:6.3f} seconds".format(image.split(
            print("{0:20s} {1:s}".format("Objects Detected:", ""),file=open("
            print("----------------------------------------------------------
        continue

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

    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_si
        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("/")
        print("{0:20s} {1:s}".format("Objects Detected:", " ".join(objs)),fil
        print("----------------------------------------------------------",fi

    if CUDA:
        torch.cuda.synchronize()
try:
    output
except NameError:
    print ("No detections were made",file=open("output_coco.txt", "a"))
    exit()

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

scaling_factor = torch.min(416/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

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

```
output_recast = time.time()
class_load = time.time()
colors = [[255, 0, 0], [255, 0, 0], [255, 255, 0], [0, 255, 0], [0, 255, 255]

draw = time.time()

def write(x, results):
    c1 = tuple(x[1:3].int())
    c2 = tuple(x[3:5].int())
    img = results[int(x[0])]
    cls = int(x[-1])
    color = random.choice(colors)
    label = "{0}".format(classes[cls])
    cv2.rectangle(img, c1, c2,color, 1)
    t_size = cv2.getTextSize(label, cv2.FONT_HERSHEY_PLAIN, 1 , 1)[0]
    c2 = c1[0] + t_size[0] + 3, c1[1] + t_size[1] + 4
    cv2.rectangle(img, c1, c2,color, -1)
    cv2.putText(img, label, (c1[0], c1[1] + t_size[1] + 4), cv2.FONT_HERSHEY_
    return img


list(map(lambda x: write(x, loaded_ims), output))

det_names = pd.Series(imlist).apply(lambda x: "{}/det_{}".format("des",x.spli

list(map(cv2.imwrite, det_names, loaded_ims))

end = time.time()

print("SUMMARY",file=open("output_coco.txt", "a"))
print("----------------------------------------------------------",file=open(
print("{:25s}: {}".format("Task", "Time Taken (in seconds)"),file=open("outpu
print()
print("{:25s}: {:2.3f}".format("Reading addresses", load_batch - read_dir),fi
print("{:25s}: {:2.3f}".format("Loading batch", start_det_loop - load_batch),
print("{:25s}: {:2.3f}".format("Detection (" + str(len(imlist)) +  " images)"
print("{:25s}: {:2.3f}".format("Output Processing", class_load - output_recas
print("{:25s}: {:2.3f}".format("Drawing Boxes", end - draw),file=open("output_
print("{:25s}: {:2.3f}".format("Average time_per_img", (end - load_batch)/len
print("----------------------------------------------------------",file=open(


torch.cuda.empty_cache()
```

In [18]:

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
image = mpimg.imread("des/det_dog-cycle-car.png")
plt.imshow(image)
plt.show()
```



Inference on COCO dataset

In [1]:

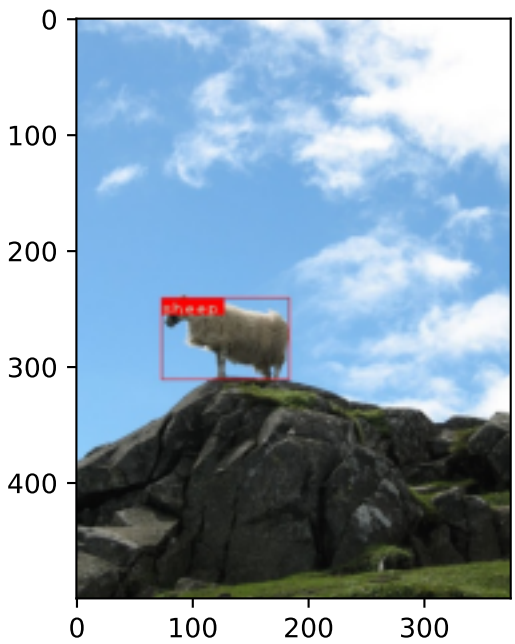
```
results_COCO = open("output_coco.txt", "r")
print(results_COCO.read())
```

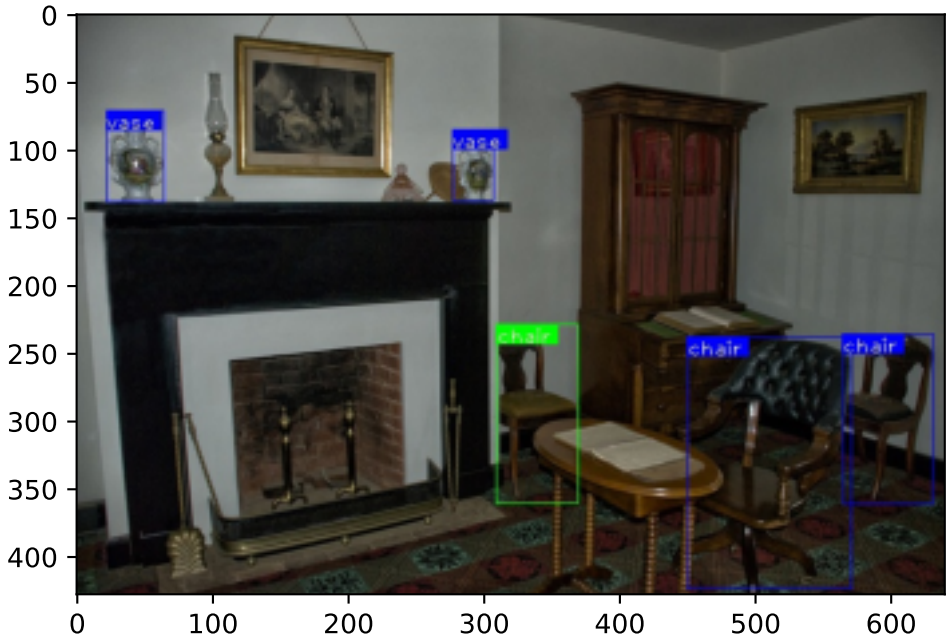
```
Loading network.....
Network successfully loaded
dog-cycle-car.png     predicted in  0.015 seconds
Objects Detected:     bicycle truck dog
-----------------------------------------------------------
000000016502.jpg      predicted in  0.015 seconds
Objects Detected:     sheep
-----------------------------------------------------------
000000016598.jpg      predicted in  0.015 seconds
Objects Detected:     person tie cell phone
-----------------------------------------------------------
000000016958.jpg      predicted in  0.015 seconds
Objects Detected:     chair chair chair vase vase
-----------------------------------------------------------
000000017029.jpg      predicted in  0.015 seconds
Objects Detected:     car car dog frisbee
-----------------------------------------------------------
000000017031.jpg      predicted in  0.015 seconds
Objects Detected:     person giraffe
-----------------------------------------------------------
000000017115.jpg      predicted in  0.015 seconds
Objects Detected:     zebra zebra
-----------------------------------------------------------
000000017178.jpg      predicted in  0.015 seconds
Objects Detected:     car horse horse horse
-----------------------------------------------------------
000000017182.jpg      predicted in  0.015 seconds
Objects Detected:     apple chair book book book
-----------------------------------------------------------
000000017207.jpg      predicted in  0.015 seconds
Objects Detected:     person car motorbike bus truck
-----------------------------------------------------------
000000017379.jpg      predicted in  0.015 seconds
Objects Detected:     person tvmonitor sink sink
-----------------------------------------------------------
000000017436.jpg      predicted in  0.015 seconds
Objects Detected:     person bench
-----------------------------------------------------------
000000017714.jpg      predicted in  0.016 seconds
```

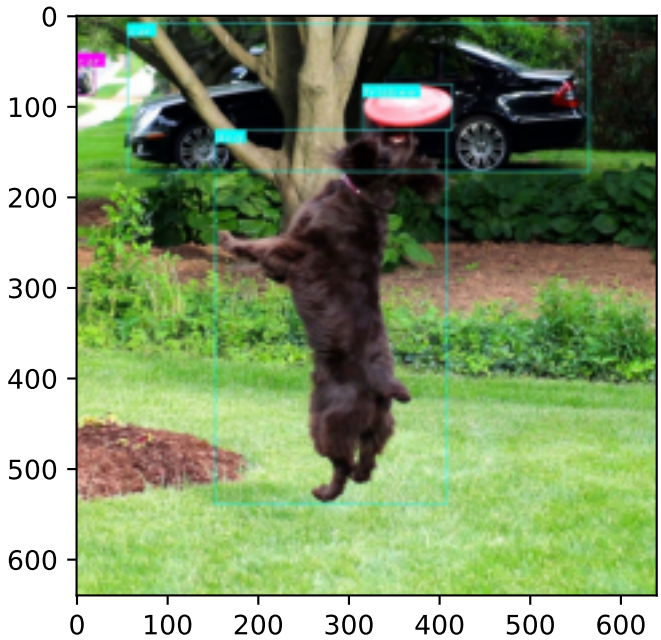
```
Objects Detected:    cup cup fork fork knife bowl banana banana diningtable
------------------------------------------------------------
000000017899.jpg     predicted in  0.016 seconds
Objects Detected:    person cup cup bowl sandwich sandwich chair chair sofa di
ningtable
------------------------------------------------------------
000000017905.jpg     predicted in  0.016 seconds
Objects Detected:    person traffic light
------------------------------------------------------------
000000017959.jpg     predicted in  0.016 seconds
Objects Detected:    person person kite kite kite kite
------------------------------------------------------------
000000018150.jpg     predicted in  0.009 seconds
Objects Detected:    person person bottle pizza
------------------------------------------------------------
000000018193.jpg     predicted in  0.009 seconds
Objects Detected:    person donut chair
------------------------------------------------------------
SUMMARY
------------------------------------------------------------
Task                      : Time Taken (in seconds)
Reading addresses         : 0.001
Loading batch             : 0.123
Detection (18 images)     : 0.478
Output Processing         : 0.000
Drawing Boxes             : 0.142
Average time_per_img      : 0.041
```

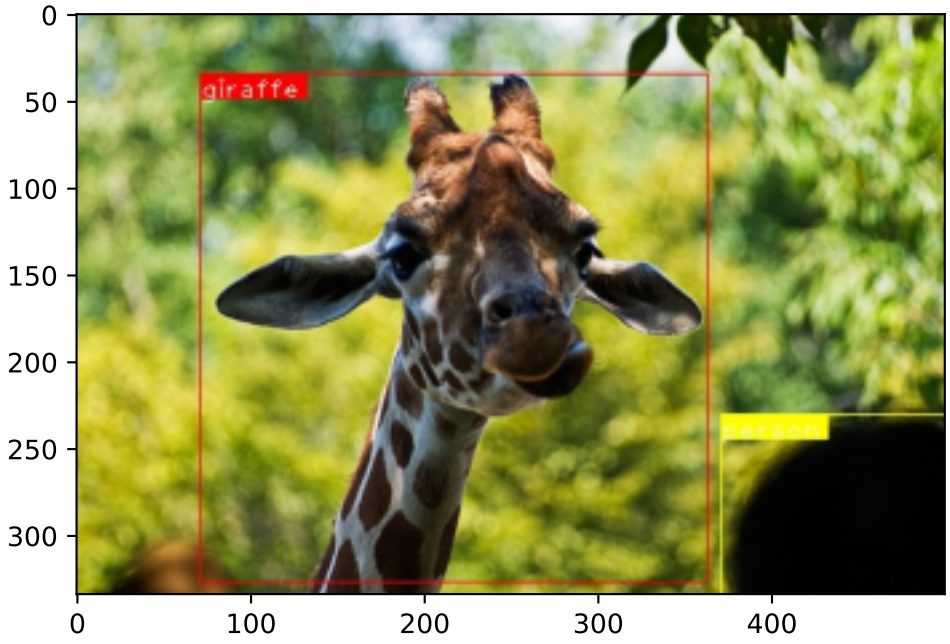
In [8]:

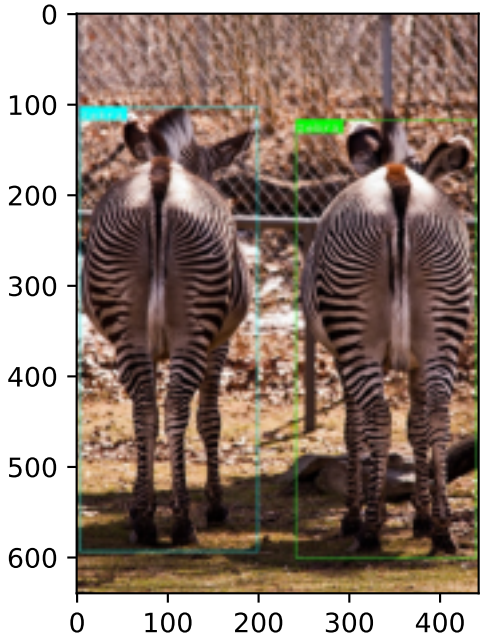
```
import cv2
import matplotlib.pyplot as plt
import glob
for img in glob.glob("des/*.jpg"):
    cv_img = cv2.imread(img)
    cv_img = cv2.cvtColor(cv_img, cv2.COLOR_RGB2BGR)
    plt.imshow(cv_img)
    plt.show()
```



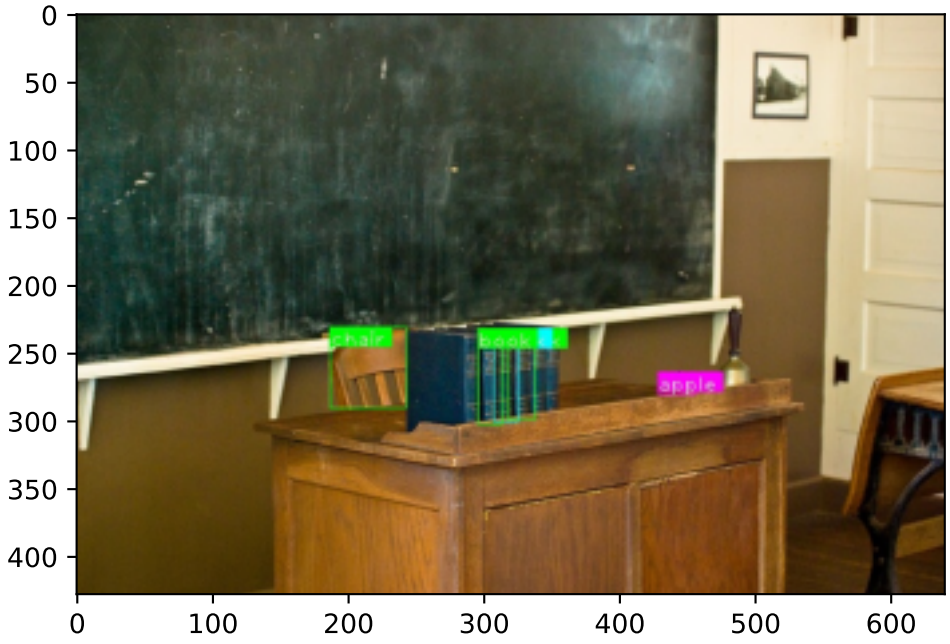

vase
vase
chair
chair
chair



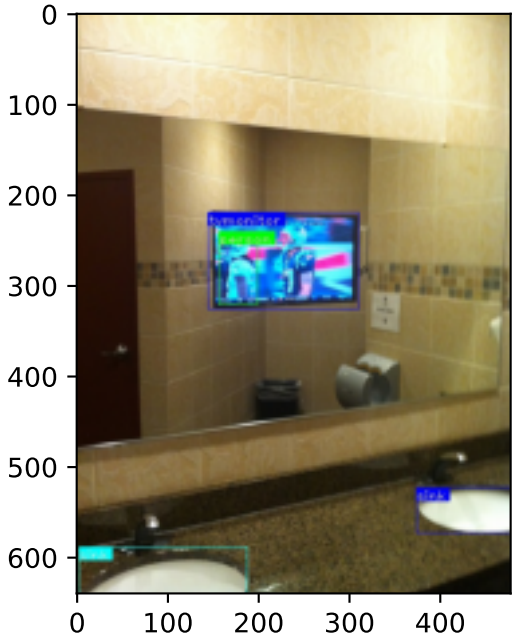

giraffe

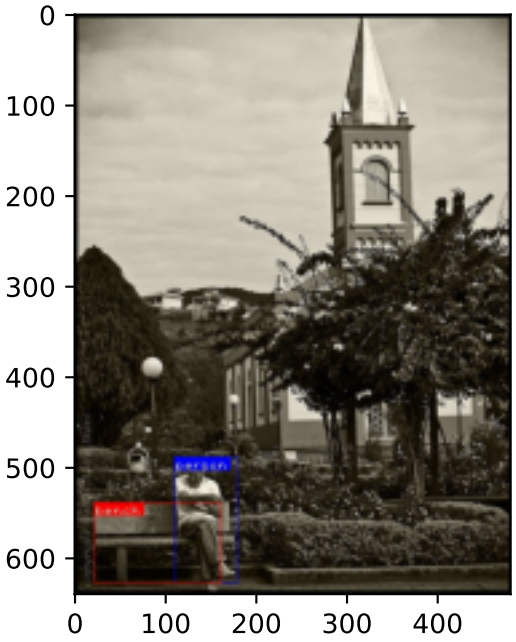


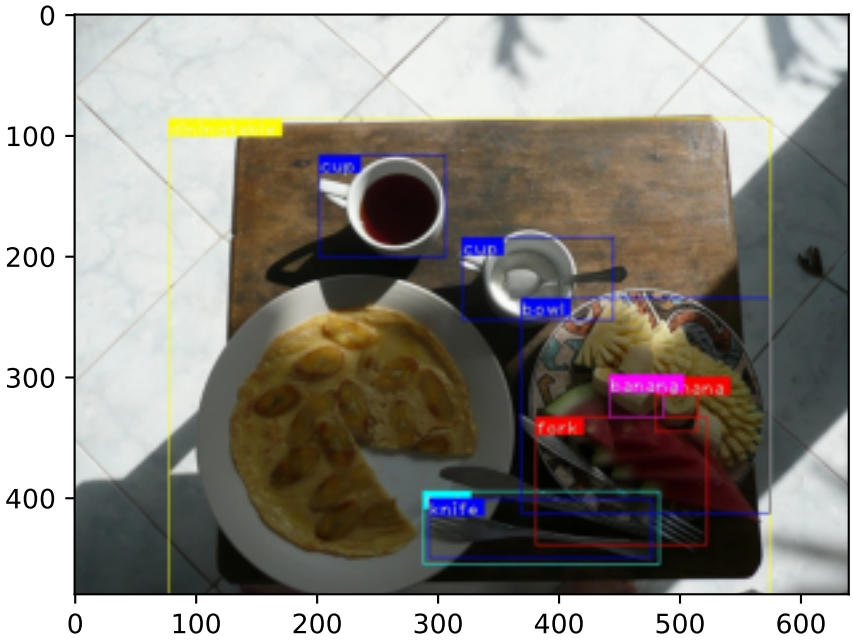

car
horse
horse
horse


chair
book
apple

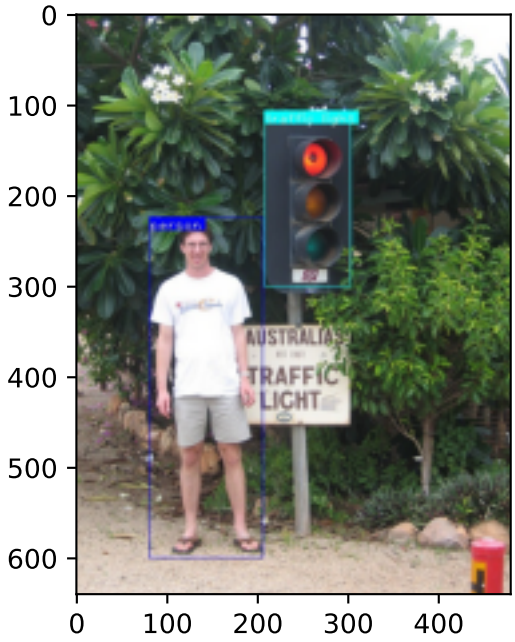

truck

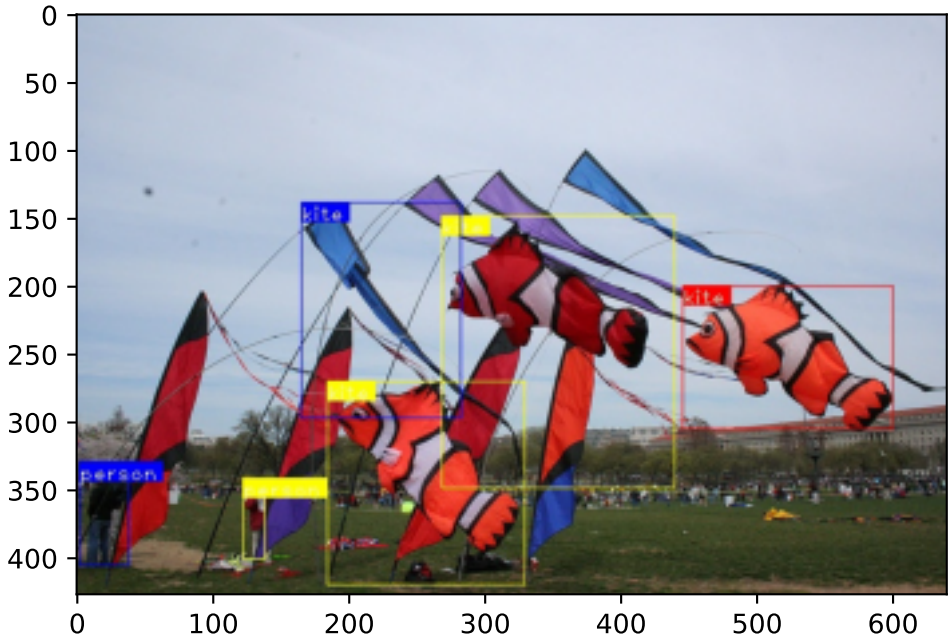


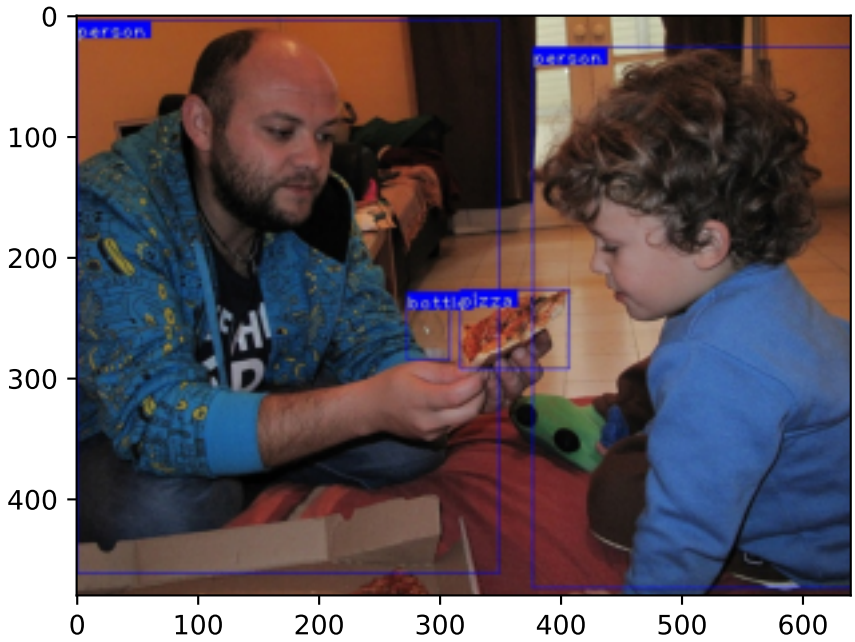

0
100
200
300
400
500
600
0
100
200
300
400

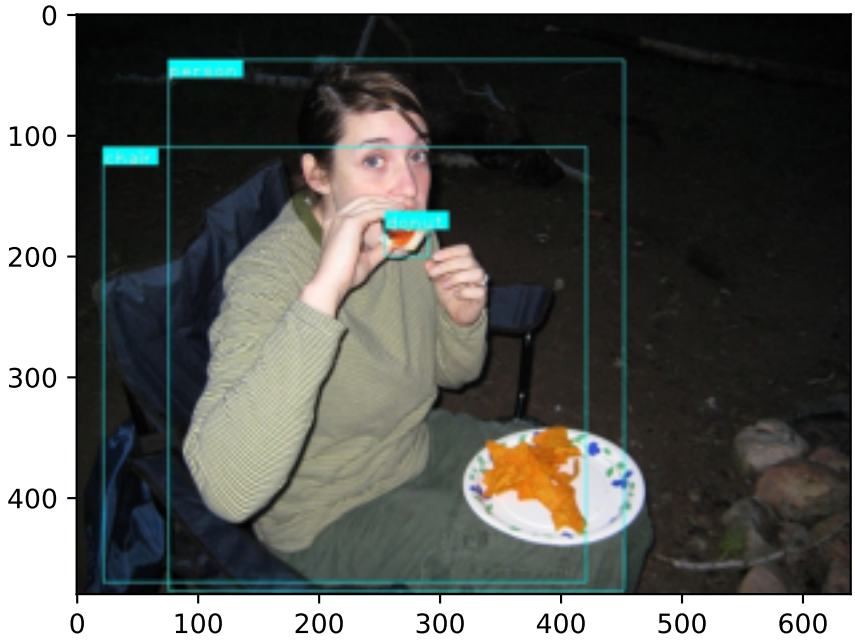

0
100
200
300
400
0
100
200
300
400
500
600
cup
cup
bowl
fork
knife


0
100
200
300
400
500
600
0
100
200
300
400


0
100
200
300
400
500
600
0
100
200
300
400
TRAFFIC
LIGHT


0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
kite
kite
person


0
100
200
300
400
0
100
200
300
400
500
600
person
person


0
100
200
300
400
0
100
200
300
400
500
600