

exercise10

November 14, 2024

```
[1]: # This cell is used for creating a button that hides/unhides code cells to
      ↵quickly look only the results.
# Works only with Jupyter Notebooks.

from IPython.display import HTML

HTML('''<script>
code_show=true;
function code_toggle() {
if (code_show){
$('div.input').hide();
} else {
$('div.input').show();
}
code_show = !code_show
}
$( document ).ready(code_toggle);
</script>
<form action="javascript:code_toggle()"><input type="submit" value="Click here
      ↵to toggle on/off the raw code."></form>'''')
```

```
[1]: <IPython.core.display.HTML object>
```

```
[2]: # Description:
#   Exercise10 notebook.
#
# Copyright (C) 2019 Antti Parviaainen
# Based on the SSD PyTorch implementation by Max deGroot and Ellis Brown
#
# This software is distributed under the GNU General Public
# Licence (version 2 or later);

import os
import numpy as np
from matplotlib import pyplot as plt
import cv2
```

```

import torch
from torch.autograd import Variable
from ssd import build_ssd
from data import VOCDetection, VOCAnnotationTransform
from data import VOC_CLASSES as labels

import warnings
warnings.filterwarnings("ignore")

# Select data directory
if os.path.isdir('/coursedata'):
    course_data_dir = '/coursedata'
    docker = False
elif os.path.isdir('../..//coursedata'):
    docker = True
    course_data_dir = '../..//coursedata'
else:
    # Specify course_data_dir on your machine
    docker = True
    course_data_dir = '/home/jovyan/work/coursedata/'

print('The data directory is %s' % course_data_dir)
data_dir = os.path.join(course_data_dir, 'exercise-10-data')
print('Data stored in %s' % data_dir)

```

The data directory is /coursedata
Data stored in /coursedata/exercise-10-data

1 CS-E4850 Computer Vision Exercise Round 10

The problems should be solved before the exercise session and solutions returned via MyCourses. Upload to MyCourses both: this Jupyter Notebook (.ipynb) file containing your solutions and the exported pdf version of this Notebook file. If there are both programming and pen & paper tasks kindly combine the two pdf files (your scanned/LaTeX solutions and the exported Notebook) into a single pdf and submit that with the Notebook (.ipynb) file. Note that you should be sure that everything that you need to implement works with the pictures specified in this exercise round.

Docker users: 1. One file, data file containing the weights, was again above the file size limit imposed by git. I've compressed it to get it under the limit so before you run the code you have to uncompress the file in the folder `coursedata/exercise-10-data/weights`.

1.0.1 Student Name: Aitor Urruticoechea

1.0.2 Student Id: 101444219

1.1 Exercise 1 - Object detection with SSD: Single Shot MultiBox Object Detector, in PyTorch

The goal of this task is to learn the basics of deep learning based object detection with SSD by experimenting with the provided code and by reading the original [Single Shot MultiBox Detector](#) publication by Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang, and Alexander C. Berg from 2016.

Read the research paper linked above and experiment with the provided sample code according to the instructions below. Then answer the questions a), b), c) below and return your answers. Note that scientific publications are written for domain experts and some details may be challenging to understand if necessary background information is missing. However, don't worry if you don't understand all details. You should be able to grasp the overall idea and answer the questions even if some details would be difficult to understand.

The code implements the following steps to demonstrate SSD:

1. Build the SSD300 architecture and load pretrained weights on the VOC07 trainval dataset
2. Load 4 random sample images from the VOC07 dataset
3. Preprocess the images to the correct input form and show the preprocessed images
4. Run the sample images through the SSD network, parse the detections and show the results

a) In the beginning of the publication the authors argue about the relevance and novelty of their work. Summarise their main arguments and the evidence they present to support their arguments. The main relevance point for this work is the fact that the developed neural network does *not* make use of resampling pixels or features for bounding box hypotheses; while yielding results with as accurate as those who do (see in the paper, page 2).

Since there is no need to use computational resources/time for obtaining the bounding boxes plus the subsequent resampling, this results in notable increases in speed and overall efficiency. This approach is an improvement on previous work in the same line, where by adding new convolutional filters, and multiple “prediction” layers, it results in an improvement of “accuracy on real-time detection for PASCAL VOC from 63.4% mAP for YOLO to 74.3% mAP for our SSD”.

b) SSD consists of two networks: a “truncated base network” and a network of added “convolutional feature layers to the end of the truncated base network.” What is the purpose of the base network? Which base network do the authors of the SSD publication use, and what dataset was used to train the base network? The “truncated base network” is supposed to work as the base, foundation, for the extraction of features. It consists in a standard image classification network that has had its fully connected layers removed, thus leaving only the convolutional layers for feature extraction. This is meant to allow the network to detect objects without resampling pixels, again making the process faster.

The VGG-16 network (base) was originally trained on the ILSVRC CLS-LOC dataset. This base network extracts features that are then fed into additional “convolutional feature layers” added specifically for the SSD to handle multiple scales and aspect ratios, which allegedly makes detecting objects of various sizes easier.

c) SSD has it's own loss function, defined in chapter 2.2 in the original publication. What are the two attributes this loss function observes? How are these defined (short explanation without any formulas is sufficient) and how do they help the network to minimise the object detection error? The SSD loss function has two main components: localization and confidence. - Localization loss measures how accurately the predicted bounding box aligns with the ground truth bounding box for an object (taking the differences in location, size, ratio, etc). - Confidence loss evaluates the network's confidence in classifying each bounding box as containing an object or as background. Confidence loss helps the model distinguish between various object classes and ensures it does not mistakenly identify background regions as objects.

1.1.1 1. Load the SSD architecture and the pretrained weights

```
[3]: net = build_ssd('test', 300, 21)      # initialize SSD
net.load_weights(data_dir+'/weights/ssd300_mAP_77.43_v2.pth')
```

Loading weights into state dict...
Finished!

1.1.2 2. Load Sample Images

```
[4]: images = []
for i in range(4):
    if docker:
        # if local storage use a 300 image subset of the test set
        img_id = np.random.randint(1,high = 300)
        image = cv2.imread(data_dir+'/voc_images/'+f"{img_id:06d}"+'.jpg', cv2.IMREAD_COLOR)
    else:
        # if JupyterLab use the full test set
        # here we specify year (07 or 12) and dataset ('test', 'val', 'train')
        testset = VOCDetection(data_dir+'/VOCdevkit', [('2007', 'val')], None, VOCAnnotationTransform())
        img_id = np.random.randint(0,high = len(testset))
        image = testset.pull_image(img_id)

    rgb_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    images.append(rgb_image)

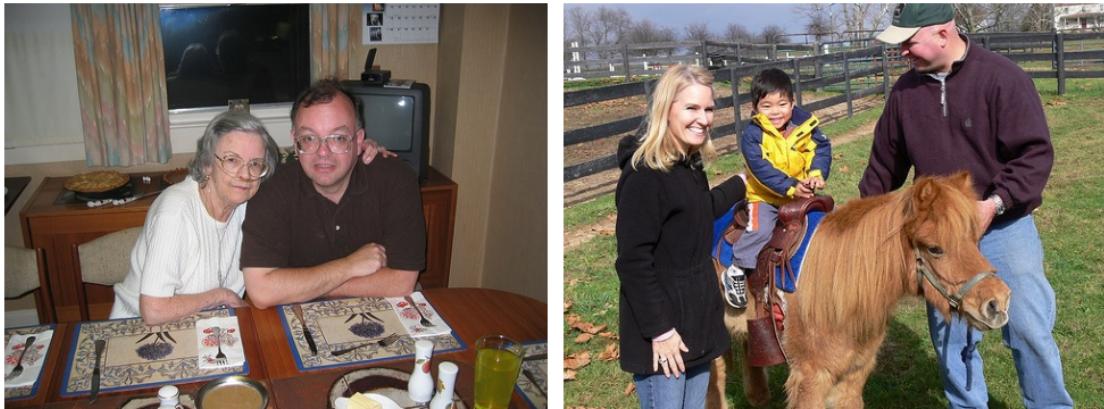
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10,10))
ax = axes.ravel()
ax[0].imshow(images[0])
ax[0].axis('off')
ax[1].imshow(images[1])
ax[1].axis('off')
ax[2].imshow(images[2])
ax[2].axis('off')
```

```

ax[3].imshow(images[3])
ax[3].axis('off')
plt.tight_layout()
plt.suptitle("Randomly sampled images from the dataset", fontsize=20)
plt.subplots_adjust(top=0.95)
plt.show()

```

Randomly sampled images from the dataset



1.1.3 3 Pre-process the input images

Using the torchvision package, we can apply multiple built-in transforms. At test time we resize our image to 300x300, subtract the dataset's mean rgb values, and swap the color channels for input to SSD300.

```

[5]: def preprocess_inputs(images):
    preprocessed_images = []
    for image in images:

```

```
x = cv2.resize(image, (300, 300)).astype(np.float32)
x -= (104.0, 117.0, 123.0)
x = x.astype(np.float32)
preprocessed_images.append(x)
return preprocessed_images
```

```
[6]: preprocessed_images = preprocess_inputs(images)

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10,10))
ax = axes.ravel()
ax[0].imshow(preprocessed_images[0])
ax[0].axis('off')
ax[1].imshow(preprocessed_images[1])
ax[1].axis('off')
ax[2].imshow(preprocessed_images[2])
ax[2].axis('off')
ax[3].imshow(preprocessed_images[3])
ax[3].axis('off')
plt.tight_layout()
plt.suptitle("Preprocessed input images", fontsize=20)
plt.subplots_adjust(top=0.95)
plt.show()
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Preprocessed input images



4. Run SSD on the sample images and show the results

```
[7]: def run_network(images, nrows, ncols, figsize = (10,10), threshold = 0.6, title=True):

    if nrows * ncols != len(images):
        print("Subgrid dimensions don't match with the number of images.")
        return

    preprocessed_images = preprocess_inputs(images)
    fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=figsize)

    if len(images) != 1:
```

```

        ax = axes.ravel()
    else:
        ax = [axes]

    for it, input_image in enumerate(images):
        # Process data for the network
        # swap color channels
        x = preprocessed_images[it][:, :, ::-1].copy()
        # change the order
        x = torch.from_numpy(x).permute(2, 0, 1)
        # wrap the image in a Variable so it is recognized by PyTorch autograd
        xx = Variable(x.unsqueeze(0))
        if torch.cuda.is_available():
            xx = xx.cuda()

        # SSD Forward Pass
        y = net(xx)
        detections = y.data

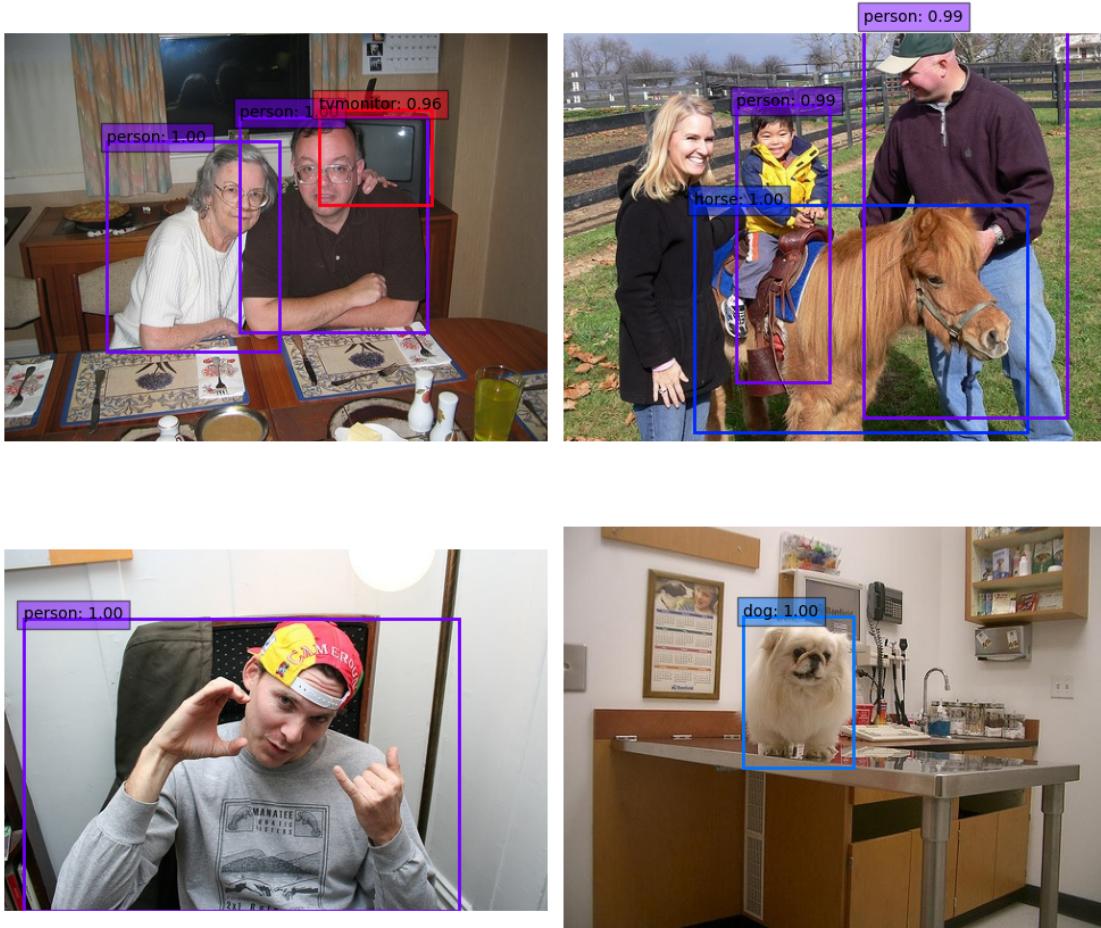
        # colormap for the bounding boxes
        colors = plt.cm.hsv(np.linspace(0, 1, 21)).tolist()

        # scale each detection back up to the image
        scale = torch.Tensor(input_image.shape[1:-1]).repeat(2)
        for i in range(detections.size(1)):
            j = 0
            while detections[0,i,j,0] >= threshold:
                score = detections[0,i,j,0]
                label_name = labels[i-1]
                display_txt = '%s: %.2f'%(label_name, score)
                pt = (detections[0,i,j,1:]*scale).cpu().numpy()
                coords = (pt[0], pt[1]), pt[2]-pt[0]+1, pt[3]-pt[1]+1
                color = colors[i]
                ax[it].add_patch(plt.Rectangle(*coords, fill=False, □
            ↵edgecolor=color, linewidth=2))
                ax[it].text(pt[0], pt[1], display_txt, bbox={'facecolor':color, □
            ↵'alpha':0.5})
                j+=1
            ax[it].imshow(images[it])
            ax[it].axis('off')
        plt.tight_layout()
        if title:
            plt.suptitle("Detection results at threshold: %.2f" %threshold, □
        ↵fontsize=20)
            plt.subplots_adjust(top=0.95)
        plt.show()

```

```
[8]: # Filter outputs with confidence scores lower than a threshold. The default
     ↪threshold is 60%.
run_network(images, nrows=2, ncols=2, figsize=(10,10), threshold=0.6)
```

Detection results at threshold: 0.60



1.2 Exercise 2 - Evaluating the SSD network on some more challenging input images

To better detect challenging inputs the authors have implemented data augmentation described in chapter 2.2 and 3.2 of the publication and in [reference 14](#). Read the chapters in the publication and selectively read the main points of reference 14 and answer the following questions.

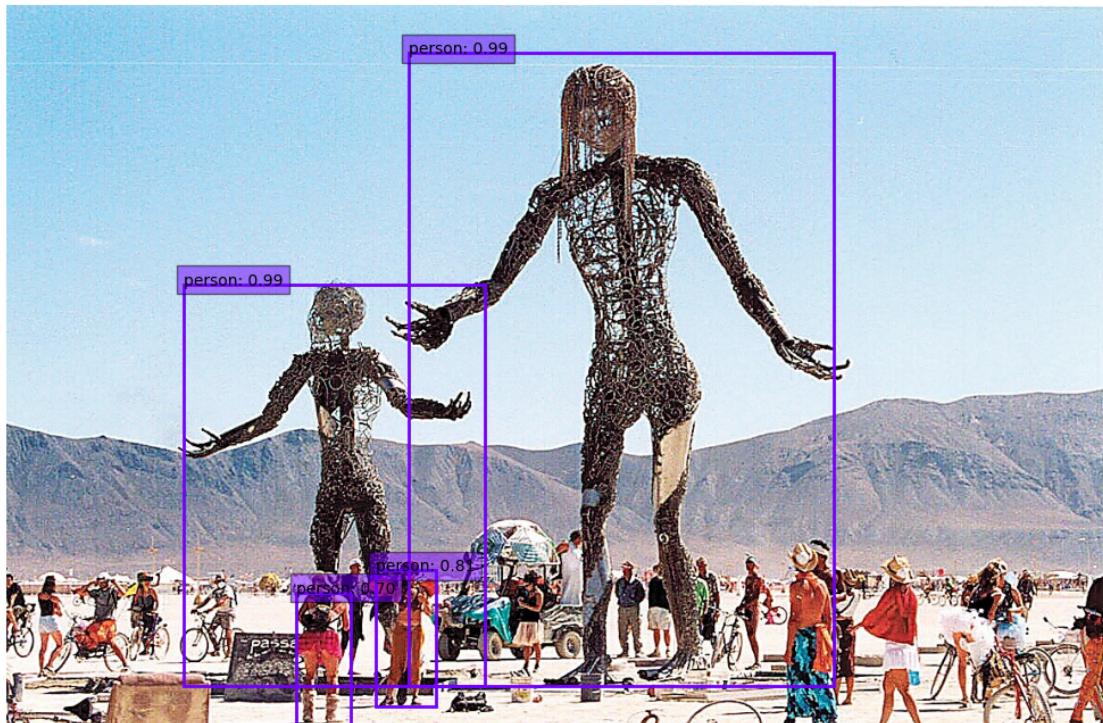
1.2.1 2.1

- a) Based on the results (test images1) below what kind of objects are easier for the network to detect? Type your answer here:

b) Would switching from input size 300x300 to 512x512 make the problem better, worse, or have no impact? Elaborate on why or why not. Would designing a object detector that uses HD resolution of 1920x1080, or even higher, images as inputs be a good idea? Elaborate on why or why not. Type your answer here:

```
[9]: # Photo credit: to burningman.org. Used under the fair use principles for  
→transformative educational purposes.  
image11 = cv2.imread(data_dir+'test_images/img11.jpg', cv2.IMREAD_COLOR)  
images1=[cv2.cvtColor(image11, cv2.COLOR_BGR2RGB)]  
run_network(images1, nrows=1, ncols=1, figsize=(10,10), threshold=0.6)
```

Detection results at threshold: 0.60



1.2.2 2.2

a) Based on the results (test images2) below elaborate why the detector has trouble detecting the objects? SSD overall struggles with partially-hidden, and overlapping or similar-looking objects because of to limitations in feature map resolution and the difficulty of distinguishing closely packed objects.

b) Have the authors of the SSD publication tried to mitigate this problem? If yes, briefly explain how. Yes, they have employed a series of strategies, like:

- Multi-scale feature map, for better chances at detecting objects at different sizes.
- Default boxes of different scales and aspect ratios.
- Hard negative mining, where not all negative boxes in the training data are used so as to avoid an imbalance between positives and negatives.
- “Classic” Data Augmentation, where training data is distorted and reused for extra sturdiness.

[10]:

```
# Photo credit: free wallpaper image. Used under the fair use principles for
# transformative educational purposes.
image21 = cv2.imread(data_dir+'/test_images/img21.jpg', cv2.IMREAD_COLOR)
# Photo credit: David Dodman, KNOM. Used under the fair use principles for
# transformative educational purposes.
image22 = cv2.imread(data_dir+'/test_images/img22.jpg', cv2.IMREAD_COLOR)
images2=[cv2.cvtColor(image21, cv2.COLOR_BGR2RGB),cv2.cvtColor(image22, cv2.
COLOR_BGR2RGB)]
run_network(images2, nrows=2, ncols=1, figsize=(25,25), threshold=0.6)
```

Detection results at threshold: 0.60



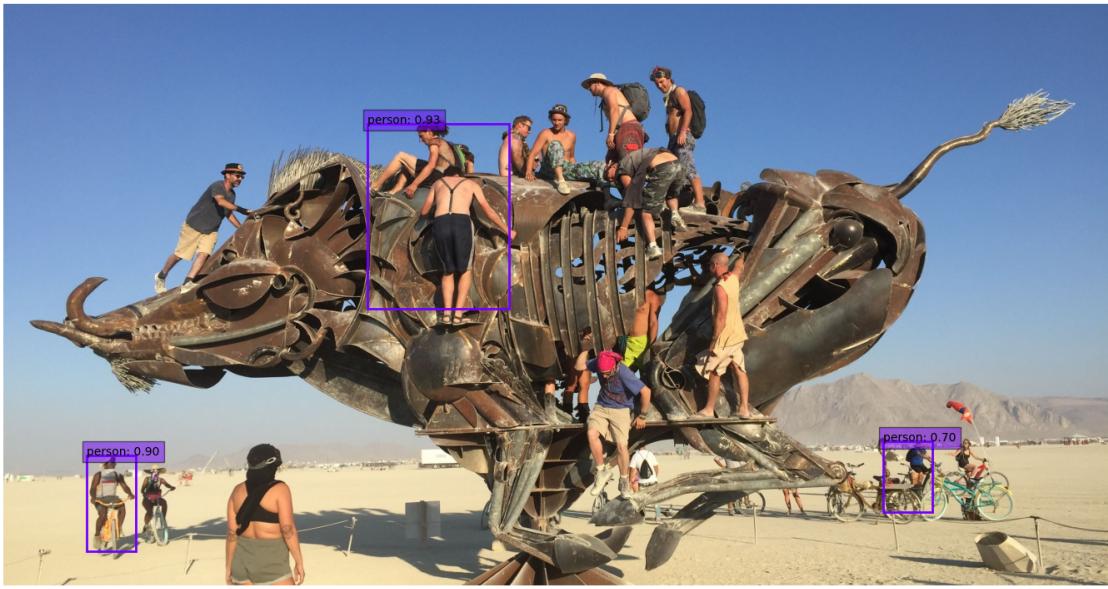
1.2.3 2.3

- a) Based on the results (test images3) below, elaborate why the detector has trouble detecting the objects? Here, the case seems to be more focused on repetition and partially overlapping cases. Two horses one “on top of” the other (in the z-axis); and a large group of people, all with very similar colours, and in many cases also overlapping.

b) Have the authors of the SSD publication tried to mitigate this problem? If yes, briefly explain how. For this kind of issues, SSD uses, again Multi-Scale feature maps and default boxes of different sizes and aspect ratios, but to this list one could also add: - Non-Maximum Suppression (NMS) filters out overlapping boxes with lower confidence scores, keeping only the most confident prediction for each object. This helps in handling multiple overlapping detections by focusing on the best bounding box for each object, at the expense of missing some cases.

```
[11]: # Photo credit: Duncan Rawlinson - Duncan.co photo from flickr. Creative Commons license. https://creativecommons.org/licenses/by-nc/2.0/
image31 = cv2.imread(data_dir+'/test_images/img31.jpg', cv2.IMREAD_COLOR)
# Photo credit: Karri Huhtanen from flickr. Creative Commons license. https://creativecommons.org/licenses/by-nc/2.0/
image32 = cv2.imread(data_dir+'/test_images/img32.jpg', cv2.IMREAD_COLOR)
images3=[cv2.cvtColor(image31, cv2.COLOR_BGR2RGB),cv2.cvtColor(image32, cv2.COLOR_BGR2RGB)]
run_network(images3, nrows=2, ncols=1, figsize=(15,15), threshold=0.6)
```

Detection results at threshold: 0.60



1.2.4 2.4

- a) Based on the results (test images4) below elaborate why the detector has trouble detecting objects in the first/upper image, but is able to detect objects in the second/lower image which contains the left part of the upper image? Note: you can double click the upper image to show it in actual size. The main issue is probably the fact that SSD processes images at a fixed input size, resizing the input image if necessary. In the case of the full image, resizing it may cause the people and bikes to become even smaller, and thus harder for SSD to detect. The cropped image, being smaller, allows objects to retain more of their detail after resizing, which helps in detection. Minimal contrast between the background (sand)

and the objects (people and bikes) also does not help.

b) Have the authors of the SSD publication tried to mitigate this problem? If yes, briefly explain how. To the already explained techniques, it is worth also adding:

- Expanded Data Augmentation (“Zoom Out” Trick), as the authors introduced an “expansion” data augmentation technique where they place the image on a larger background canvas before resizing.
- Increased Input Resolution (SSD512), where the authors also proposed a higher-resolution version of the model (SSD512) that uses a 512x512 pixel input size instead of 300x300.

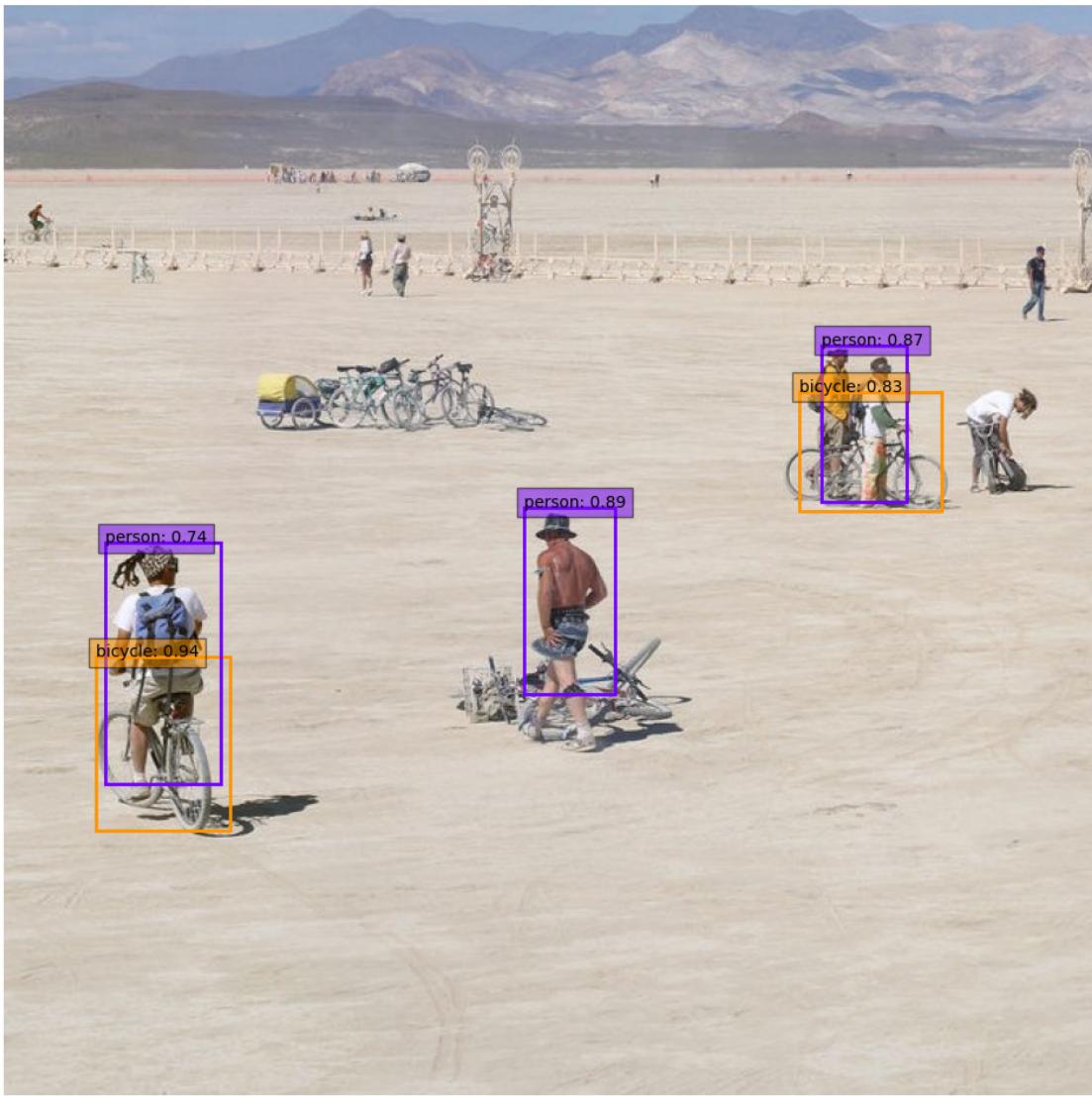
[12]: # Photo credit: Brad Templeton. Used under the fair use principles for
↳transformative educational purposes.

```
image41 = cv2.imread(data_dir+'/test_images/img41.jpg', cv2.IMREAD_COLOR)
image42 = cv2.imread(data_dir+'/test_images/img42.jpg', cv2.IMREAD_COLOR)
images4 = [cv2.cvtColor(image41, cv2.COLOR_BGR2RGB), cv2.cvtColor(image42, cv2.
    ↳COLOR_BGR2RGB)]
# run the first image without title information to show it properly
run_network([images4[0]], nrows=1, ncols=1, figsize=(100,100), threshold=0.6, ↳
    ↳title=False)
```



[13]: run_network([images4[1]], nrows=1, ncols=1, figsize=(10,10), threshold=0.6)

Detection results at threshold: 0.60



1.2.5 2.5

a) One of the test images in (images5) was perhaps accidentally flipped vertically and as can be seen the detector has trouble detecting objects if there's a significant rotation. Have the authors of the SSD publication tried to mitigate this problem? If yes, briefly explain how. If no, propose a naive method to mitigate it and explain why it usually might not be necessary or a good idea(more harm than good). SSD relies on standard convolutional layers, which are inherently translation-invariant but not rotation-invariant, meaning the model can struggle with recognizing objects if they appear at unusual orientations or rotations that differ significantly from what it saw during training. The authors did not propose anything, but one could use for this purposes techniques like the already described Data Augmentation with rotated images. Alternatively, one could also think about employing other architectures that are

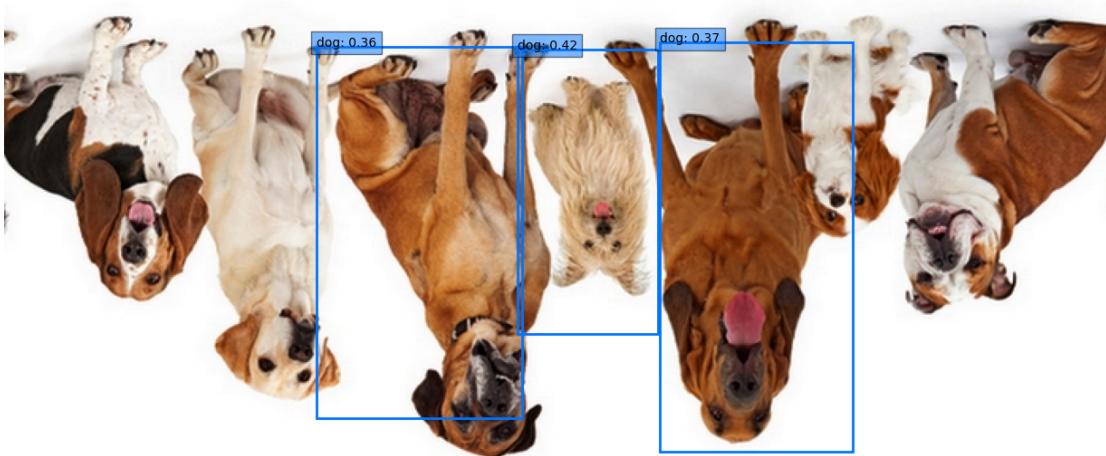
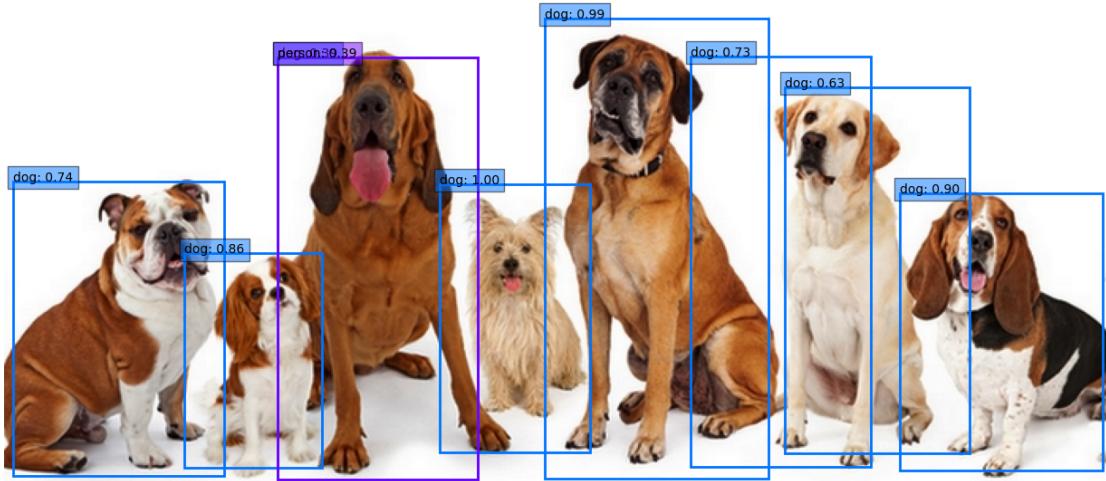
able to handle rotations.

b) Is a convolutional network naturally, without specifically addressing the issue, able to detect objects if there are small rotations? If yes, briefly explain which part of the network helps to mitigate the effects of rotation and why. (Hint: what layers are sometimes between two convolutional layers) Convolutional networks can handle small rotations to some extent even without explicitly addressing the issue. This ability is partly due to the pooling layers (such as max pooling) that are often placed between convolutional layers. By retaining only the most prominent features (e.g., the maximum value in a max-pooling window), pooling layers help the network become less sensitive to slight rotations or shifts. It is, however, limited solution as they only offer partial invariance and are primarily designed to handle minor spatial variations.

[15]: # Photo credit: Susan Schmitz / shutterstock. Used under the fair use
↳ principles for transformative educational purposes.

```
image51 = cv2.imread(data_dir+'/test_images/img51.jpg', cv2.IMREAD_COLOR)
image52 = cv2.imread(data_dir+'/test_images/img52.jpg', cv2.IMREAD_COLOR)
images5=[cv2.cvtColor(image51, cv2.COLOR_BGR2RGB),cv2.cvtColor(image52, cv2.
    ↳COLOR_BGR2RGB)]
run_network(images5, nrows=2, ncols=1, figsize=(15,15), threshold=0.2)
```

Detection results at threshold: 0.20



1.2.6 2.6

- a) Incrementally lower the detection confidence threshold, run SSD on the test images and observe the results. What are the upsides and downsides of lowering the confidence threshold? How could you measure the effect of the confidence threshold? Assume you have some object detection task that you want to apply the detector to. What kind of object detection tasks could benefit from a lower confidence threshold and what kind

of tasks need a high confidence threshold? On the upside, the network is now more able to recognize harder to detect object, has better recall, and even small objects may get a chance to get correctly tagged. On the downside, there is an increase in false positives (see the “human” detected in the second-to-last picture), it is more expensive computationally to post-process, and overall one has to be aware that the confidence in detections is, obviously, lower. I imagine tasks where you want to make sure you DO NOT miss a detection might want a lower confidence threshold (i.e. autonomous driving, you want to ensure that, even if it is a wrong detection, the car does stop before a perceived pedestrian). On the flip side, tasks where you want to be specially sure of your detection’s correctness will need a higher threshold (i.e. autonomous weapons systems, you do not want to shoot before making sure the target is the correct one, though this is also an ethical dilemma - why are you even fully automating such a decision?).

b) Watch the following YouTube video of a detector that is similar to SSD called YOLO V2 (You Only Look Once) and use the knowledge from previous tasks to answer the following questions: In what object detection tasks is a state of the art object detector especially useful and able to perform better than a human? In what object detection tasks is a human better than a state of the art object detector? Give examples of both. Humans are remarkable at understanding context and novel objects, which state-of-the-art object detectors still are unable to properly handle. Similarly, cases with extreme occlusion, specially poor contrast, and other factors that traditionally make it difficult for a machine are handled quite well by humans. State-of-the-art object detectors, nonetheless, are surprisingly capable to when a large enough data set has been used. Even in dynamic environments, diverse lighting conditions, but specially in controlled environments they yield results quicker and more accurately in many human cases.

[]: