exercise10

November 16, 2023

[2]:
```
# 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>''')
```

[2]:
```
<IPython.core.display.HTML object>
```

[3]:
```
# Description:
#   Exercise10 notebook.
#
# Copyright (C) 2019 Antti Parviainen
# 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`.

1.1 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 authors' contributions in introducing SSD (Single Shot MultiBox Detector) can be summarized as follows:

- Faster and More Accurate Detection: SSD is presented as a single-shot detector for multiple categories that surpasses the speed of previous state-of-the-art single-shot detectors like YOLO. Importantly, it achieves comparable accuracy to slower techniques that involve explicit region proposals and pooling, including Faster R-CNN.
- Core Architecture: The fundamental aspect of SSD is its prediction mechanism, where it forecasts category scores and box offsets for a predefined set of default bounding boxes. This is accomplished using small convolutional filters applied to feature maps.
- Multi-Scale Predictions with Aspect Ratio Separation: SSD enhances detection accuracy by generating predictions at different scales from feature maps of varying scales. Additionally, the authors explicitly separate predictions based on aspect ratio, contributing to the model's ability to handle objects of diverse shapes.
- End-to-End Training and High Accuracy: The design choices in SSD lead to a straightforward end-to-end training process. Despite working well on low-resolution input images, SSD maintains high accuracy, thus improving the trade-off between speed and accuracy.
- Comprehensive Experiments: The authors conducted experiments involving timing and accuracy analyses on models with different input sizes. These evaluations were performed on widely recognized datasets, including PASCAL VOC, COCO, and ILSVRC. The results were compared against a range of recent state-of-the-art approaches in object detection.

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 purpose of the base network in SSD is to provide a foundational architecture for high-quality image classification. It serves as the early layers of the network and is truncated before any classification layers. The base network is responsible for extracting hierarchical features from input images, which can then be used for subsequent object detection tasks. The authors use VGG16 as their truncated base network and they pre-train it using the ILSVRC CLS-LOC dataset.

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 loss function in SSD incorporates two essential attributes: localization loss (loc) and confidence loss (conf). The localization loss is a Smooth L1 loss between predicted bounding box parameters and ground truth box parameters, facilitating accurate object localization. Meanwhile, the confidence loss is the softmax loss over multiple classes confidences, ensuring precise object classification through softmax loss. Together, these attributes form the overarching objective loss function in SSD, guiding the network to minimize errors in both bounding box prediction and object classification during the training process.

1.1.1 1. Load the SSD architecture and the pretrained weights

[4]:
```
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

[5]:
```
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()
```

1.1.3 3 Pre-process the input images

Using the torchvision package, we can apply multiple built-in transorms. 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.

[6]:
```
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
```

[7]:
```
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).
```

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

[8]:
```
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: %1.2f" %threshold,␣
↪fontsize=20)
        plt.subplots_adjust(top=0.95)
    plt.show()
```

[9]:
```
# 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)
```

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.

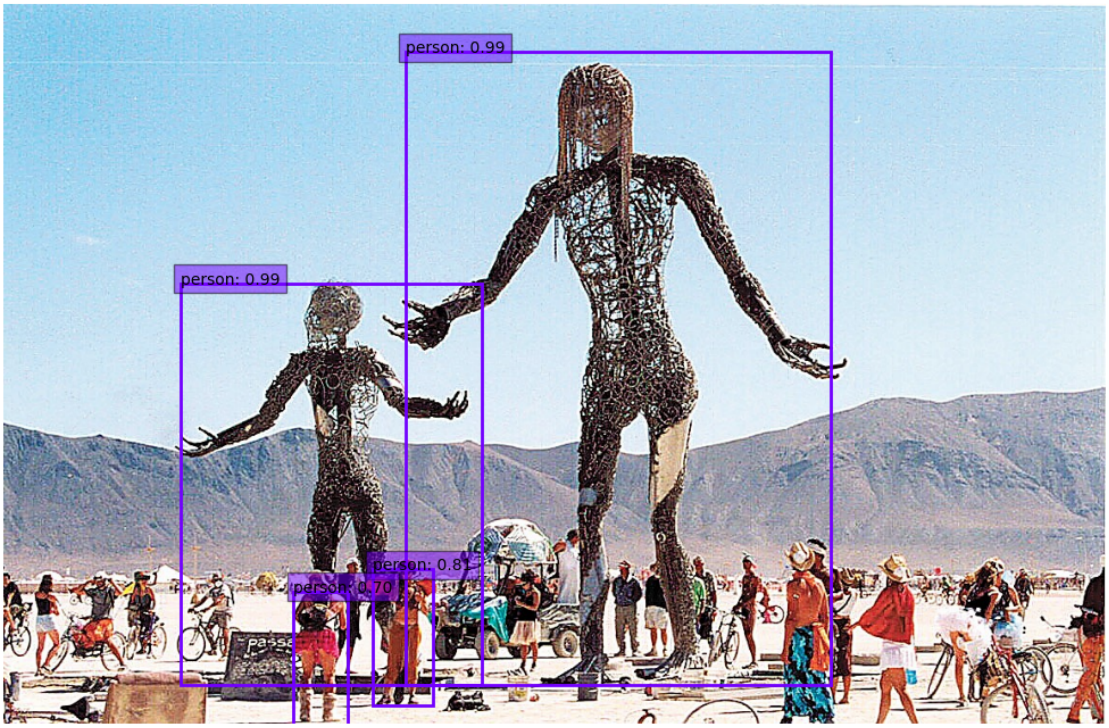
1.2.1 2.1

a) Based on the results (test images1) below what kind of objects are easier for the network to detect? The provided results suggest that the network performs better in detecting bigger objects that are surrounded by fewer elements or distractions.

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. Switching from an input size of 300x300 to 512x512 could have several implications on the performance of the object detection model. Generally, increasing the input resolution might lead to better accuracy, especially in detecting smaller objects and capturing finer details. However, this improvement might cost the increase of the computational complexity and slower processing speed. The authors mention that for a 512x512 input, SSD achieves 76.8% mAP, whereas for 300x300 it achieves only 74.3%, suggesting a positive impact on accuracy.

As for designing an object detector that uses HD resolution (1920x1080 or higher) images as inputs could have negative implications on the performance of the model. While higher resolution images can potentially provide more detailed information for accurate object detection, they also significantly increase computational demands. Processing HD images requires more powerful hardware, and real-time applications may face challenges in achieving desirable speed.

[10]:
```
# 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)
```



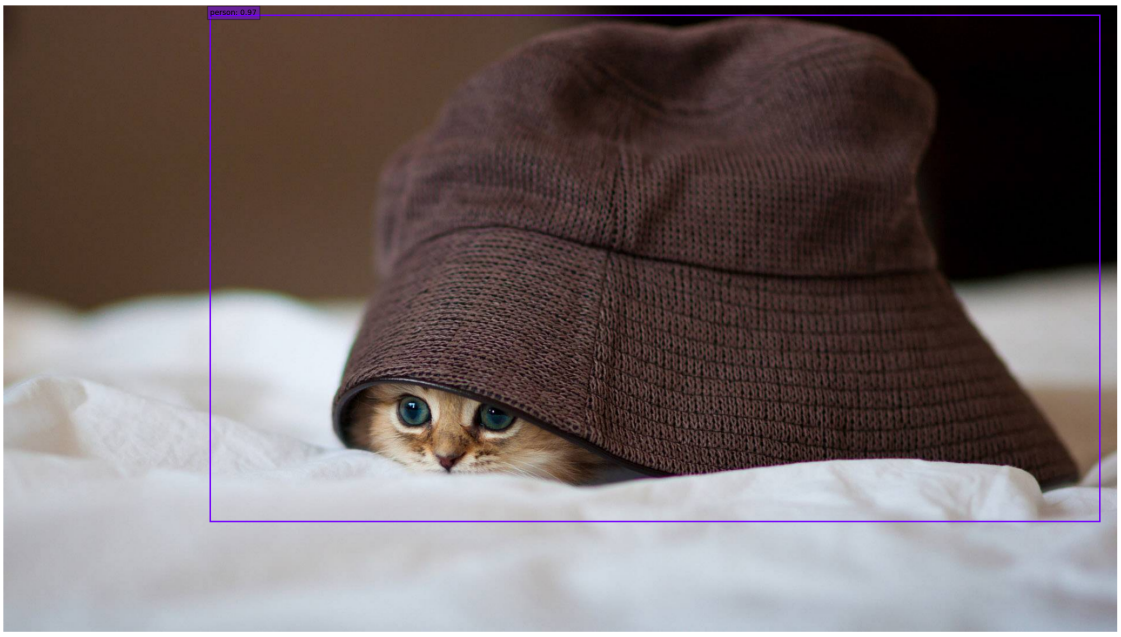
1.2.2 2.2

a) Based on the results (test images2) below elaborate why the detector has trouble detecting the objects? The detector seems to struggle when two or more objects overlap, such as the cat and hat or the group of dogs. This issue is particularly evident when objects are close together or when they share common boundaries, making it challenging for the detector to distinguish and accurately localize individual objects within the overlaid regions.

b) Have the authors of the SSD publication tried to mitigate this problem? If yes, briefly explain how. Yes, the authors of the SSD publication have attempted to address this problem. They implemented a data augmentation strategy described in Section 2.2, which involves generating random crops of the images. These random crops act as a "zoom in" operation, providing larger training examples and aiding in the detection of small objects. Additionally, the authors introduced a "zoom out" operation, where the image is placed on a canvas of 16× the original image size before undergoing any random crop operation. This augmentation trick, referred to as "expansion," increases the number of small training examples and has shown consistent improvements of 2%-3% mAP across multiple datasets.

[11]:
```
# 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? Again, the detector appears to face challenges in detecting objects that overlap with each other in the test images3.

b) Have the authors of the SSD publication tried to mitigate this problem? If yes, briefly explain how. Yes, the authors of the SSD publication have implemented data augmentation techniques to address the challenge of overlapping objects. During training, each image is randomly sampled using various options, including using the entire original input image, sampling patches with different minimum Jaccard overlaps (0.1, 0.3, 0.5, 0.7, or 0.9), and randomly sampling patches. The size of each sampled patch is a fraction of the original image size, and the aspect ratio is between 1/2 and 2. After sampling, each patch is resized to a fixed size and may be horizontally flipped with a probability of 0.5. These augmentation strategies aim to expose the model to diverse scenarios, including overlapping objects, making it more robust to such challenging situations.

[12]:
```
# 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)
```

1.2.4 2.4

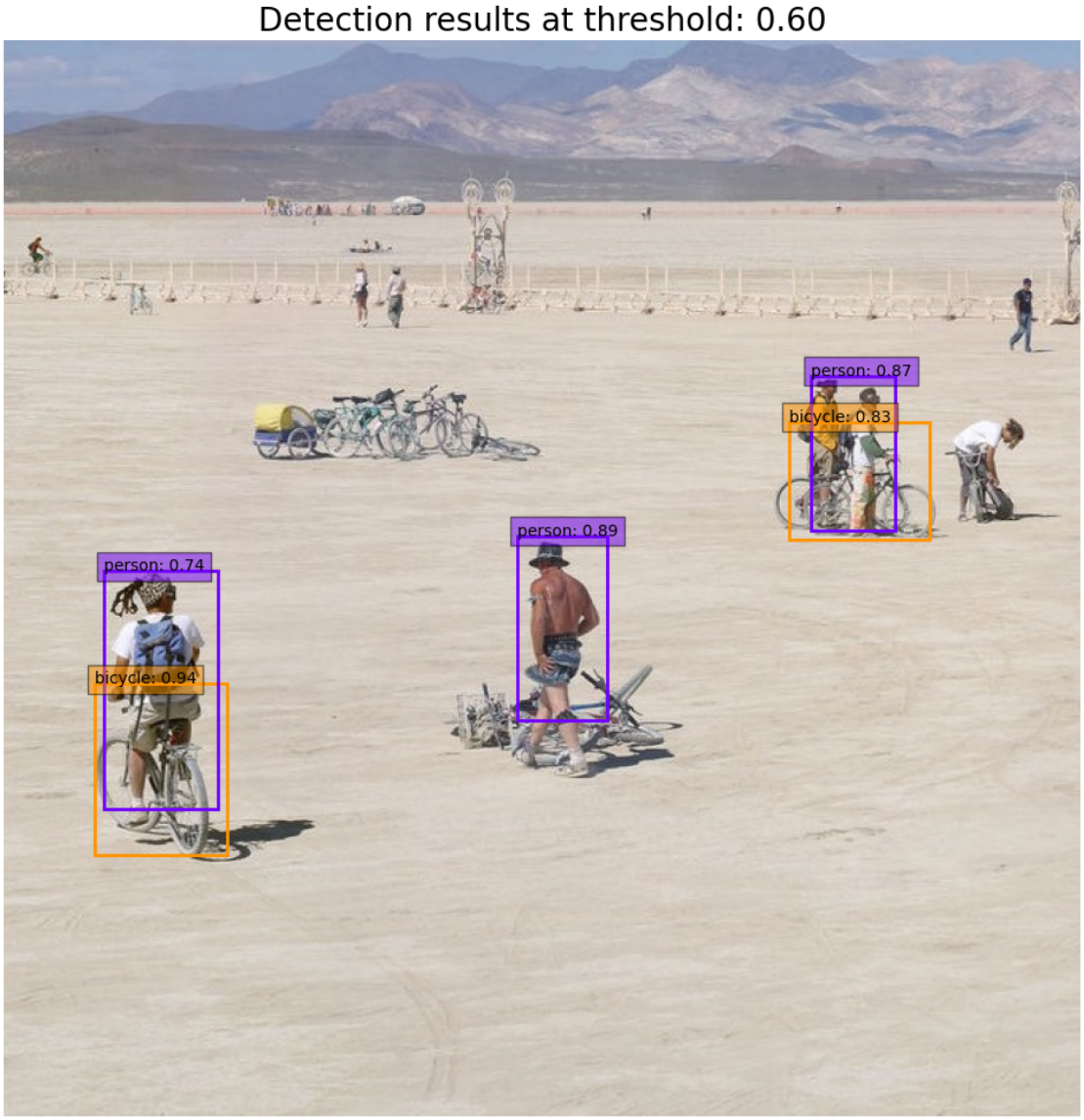
a) Based on the results (test images4) below elaborate why the detector has trouble detecting objects in the first/upper image, but is able to detect objects in the second/lower image which contains the left part of the upper image? Note: you can double click the upper image to show it in actual size. In the first image (upper), the detector faces challenges in detecting objects likely due to their small size and close proximity to each other. These factors make it difficult for the model to accurately distinguish and localize individual objects in such a crowded and compact arrangement. In contrast, the second image (lower), which contains a zoomed-in section of the first image, allows for larger and more separated objects. This

makes it comparatively easier for the detector to identify and detect the objects.

b) Have the authors of the SSD publication tried to mitigate this problem? If yes, briefly explain how. Yes, the authors have attempted to address the challenge of detecting small and closely spaced objects by implementing a data augmentation technique using "zoom in" and "zoom out" operations. The "zoom in" operation is achieved through random cropping, generating larger training examples to improve the model's performance, especially on small objects. Conversely, the "zoom out" operation involves randomly placing an image on a canvas of 16 times the original image size before cropping, creating more small training examples. This data augmentation strategy helps enhance the model's accuracy, particularly in scenarios with small and densely packed objects.

[13]:
```
# 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)
```

[14]:
```
run_network([images4[1]], nrows=1, ncols=1, figsize=(10,10), threshold=0.6)
```



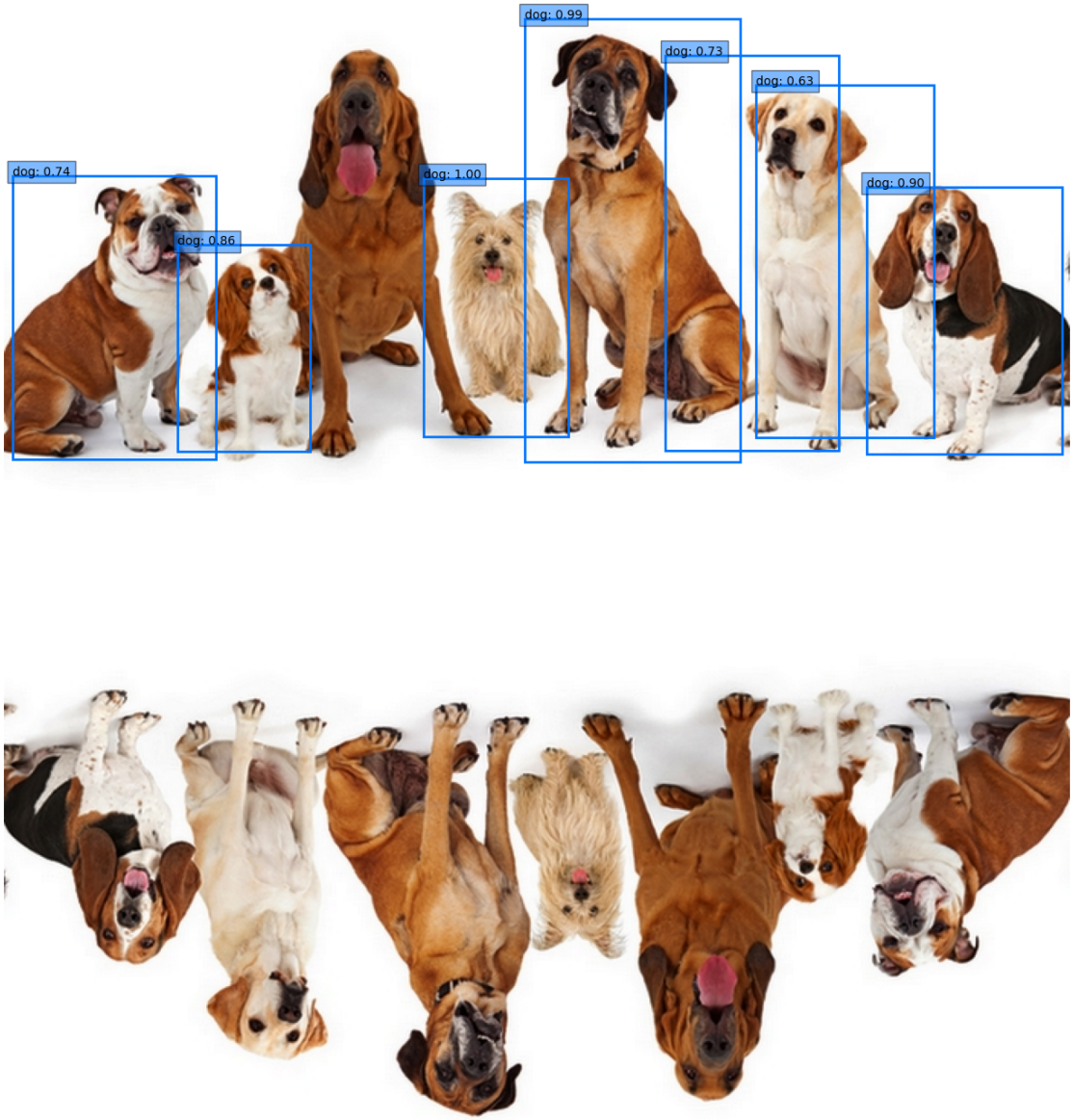
1.2.5 2.5

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). The authors of the SSD publication considered horizontal flips during data augmentation but did not explicitly mention addressing vertical flips or rotations. Given this, a naive method to mitigate rotation issues could involve extending the data augmentation strategy to include random rotations. This would entail generating additional training samples by applying random rotations to the original images.

However, the decision to include or exclude certain augmentation techniques, such as vertical flips or rotations, depends on the specific characteristics of the dataset and the requirements of the object detection task. In some cases, objects may have a consistent orientation in real-world scenarios, making random rotations unnecessary or potentially detrimental. For instance, if the objects of interest are typically upright, introducing random rotations might introduce unrealistic variations that could hinder the model's ability to generalize.

b) Is a convolutional network naturally, without specifically adressing 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 neural networks are generally capable of handling small rotations to some extent due to their inherent spatial hierarchies. The use of max-pooling layers between convolutional layers contributes to the network's ability to achieve a degree of rotational invariance. Max-pooling layers downsample the spatial dimensions, capturing the most salient features in a local region and providing a level of translation invariance. This property makes CNNs more robust to small rotations, as the network can still recognize patterns and features even if they are slightly tilted or rotated.

[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=0.6)
```



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 one hand, lowering the confidence threshold allows the detector to be more permissive in accepting predictions, which can increase the recall. This means the detector is more likely to detect objects, including those with lower confidence scores. As a consequence, objects that are partially visible or have weaker features may still be detected with a lower confidence threshold.

On the other hand, lowering the confidence threshold may lead to an increase in false positive detections, where the model identifies objects that do not actually exist. At the same time, more detections mean additional computational resources are required for processing and analysis.

For scenarios requiring a lower confidence threshold, such as exploratory analysis tasks, it becomes acceptable to capture more potential objects, minimizing the risk of missing critical detections. For example, applications like medical imaging demand a lower confidence threshold to prioritize safety and reduce false negatives. In these cases, it's crucial to identify as many objects as possible, even if it means subjecting the results to human review. In contrast, tasks necessitating real-time responses, like autonomous driving, benefit from a higher confidence threshold to optimize computational efficiency during processing and analysis.

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. State-of-the-art object detectors are especially useful in tasks that require speed, scalability, and consistency, particularly in environments with large datasets. . For instance, in surveillance systems, these detectors are invaluable for tirelessly monitoring video feeds, ensuring constant vigilance, and swiftly identifying potential threats without succumbing to the limitations of human attention spans. Likewise, object detectors can play a crucial role in the field of autonomous driving, where rapid and accurate identification of objects in the vehicle's surroundings is essential for real-time decision-making. They contribute to enhancing safety by surpassing human reaction times and providing continuous awareness.

However, there are certain object detection tasks where humans still outperform state-of-the-art detectors, especially in tasks requiring nuanced understanding, contextual reasoning, and adaptability. For example, in fine-grained object recognition, humans excel at recognizing subtle differences in objects, particularly in scenarios where detailed understanding is crucial, such as identifying specific species of plants or animal breeds. Similarly, in situations where understanding the broader context is vital, like interpreting intricate scenes or situations requiring common sense reasoning, humans demonstrate superior comprehension compared to current object detectors.

[ ]:

[ ]: