exercise10

November 13, 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 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. In the introduction of their paper, the authors highlight the limitations of existing object detection systems, which typically involve generating object proposals, resampling pixels or features for each proposal, and then applying a classifier. This multi-stage process is computationally intensive and often too slow for real-time applications. They present SSD (Single Shot MultiBox Detector) as a novel approach that eliminates the need for proposal generation and subsequent resampling stages, encapsulating all computations within a single network. This design simplifies the detection pipeline, making SSD easier to train and integrate into systems requiring a detection component. The authors provide evidence of SSD's efficiency and accuracy by demonstrating its performance on datasets like PASCAL VOC, COCO, and ILSVRC, where SSD achieves competitive accuracy compared to methods utilizing additional proposal steps, while operating at significantly higher speeds.

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 base network in SSD serves as the foundational feature extractor. It processes input images to generate a series of feature maps that capture various levels of abstraction, which are essential for detecting objects at different scales and complexities. In their implementation, the authors use a truncated version of the VGG-16 network as the base network, removing the fully connected layers to maintain spatial information. This truncated VGG-16 is pretrained on the ImageNet dataset, which provides a robust starting point for feature extraction due to its extensive and diverse image content.

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? SSD's loss function comprises two main components:

1) ocalization Loss: This measures the discrepancy between the predicted bounding box coordinates and the ground truth box coordinates. By minimizing this loss, the network learns to accurately predict the positions and dimensions of objects within the image.

2) Confidence Loss: This evaluates the difference between the predicted class probabilities and the actual class labels. It ensures that the network assigns high confidence scores to correct classes and low scores to incorrect ones, thereby improving classification accuracy.

By jointly optimizing these two losses, SSD effectively reduces errors in both object localization and classification, leading to more precise object detection.

1.1.1 1. Load the SSD architecture and the pretrained weights

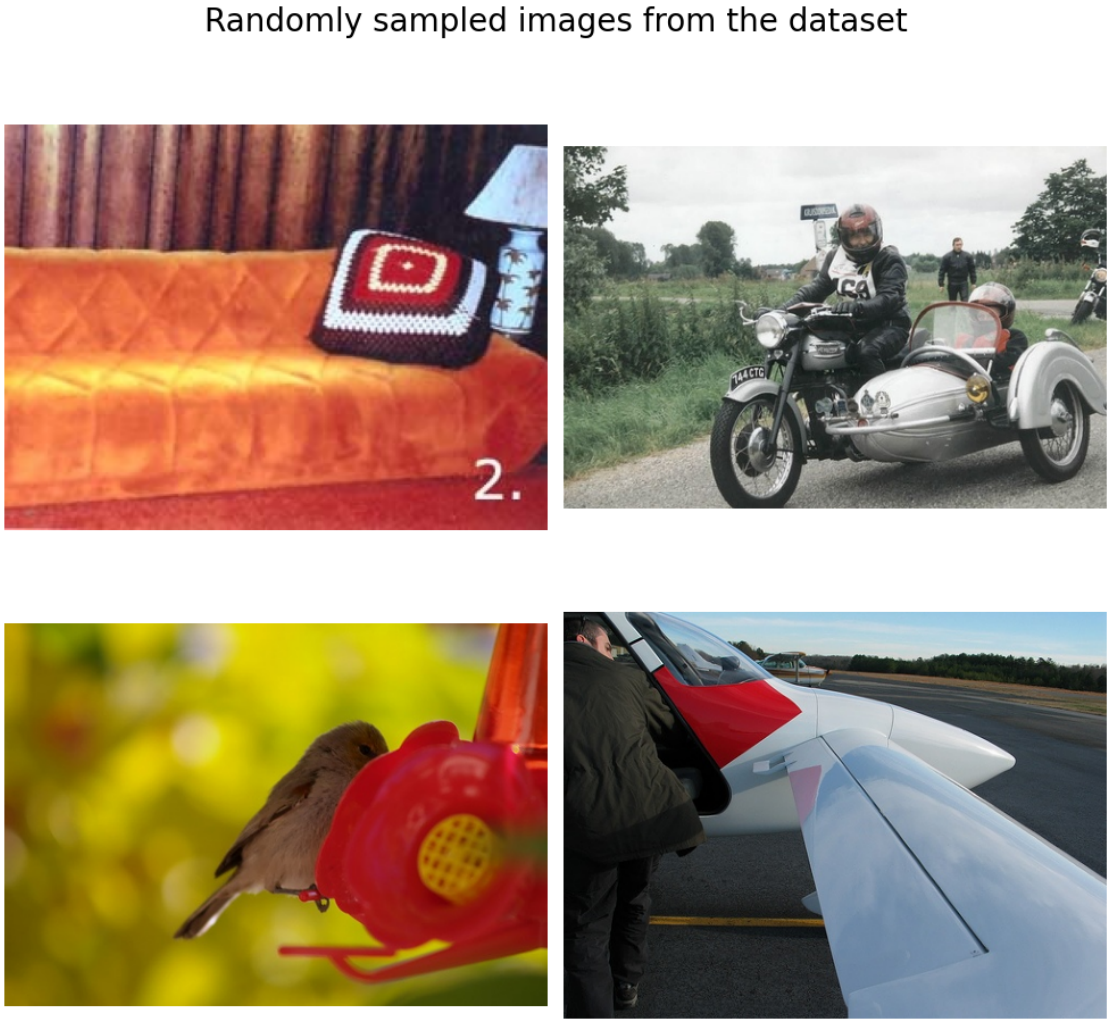
[3]:
```
net = build_ssd('test', 300, 21)    # initialize SSD
net.load_weights(data_dir+'/weights/ssd300_mAP_77.43_v2.pth')
```

```
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()
```



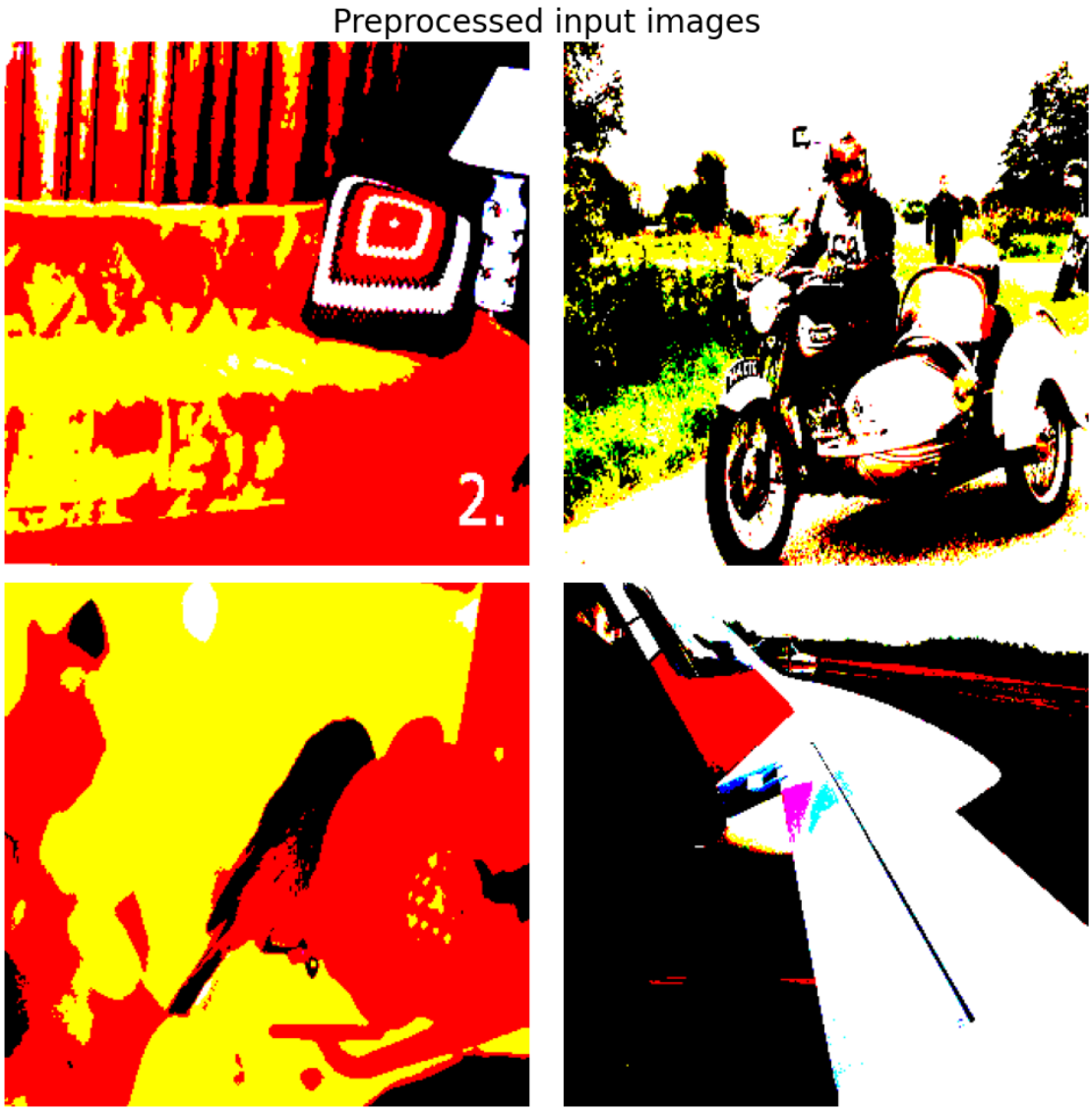
1.1.3 3 Pre-process the input images

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.

[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).
```

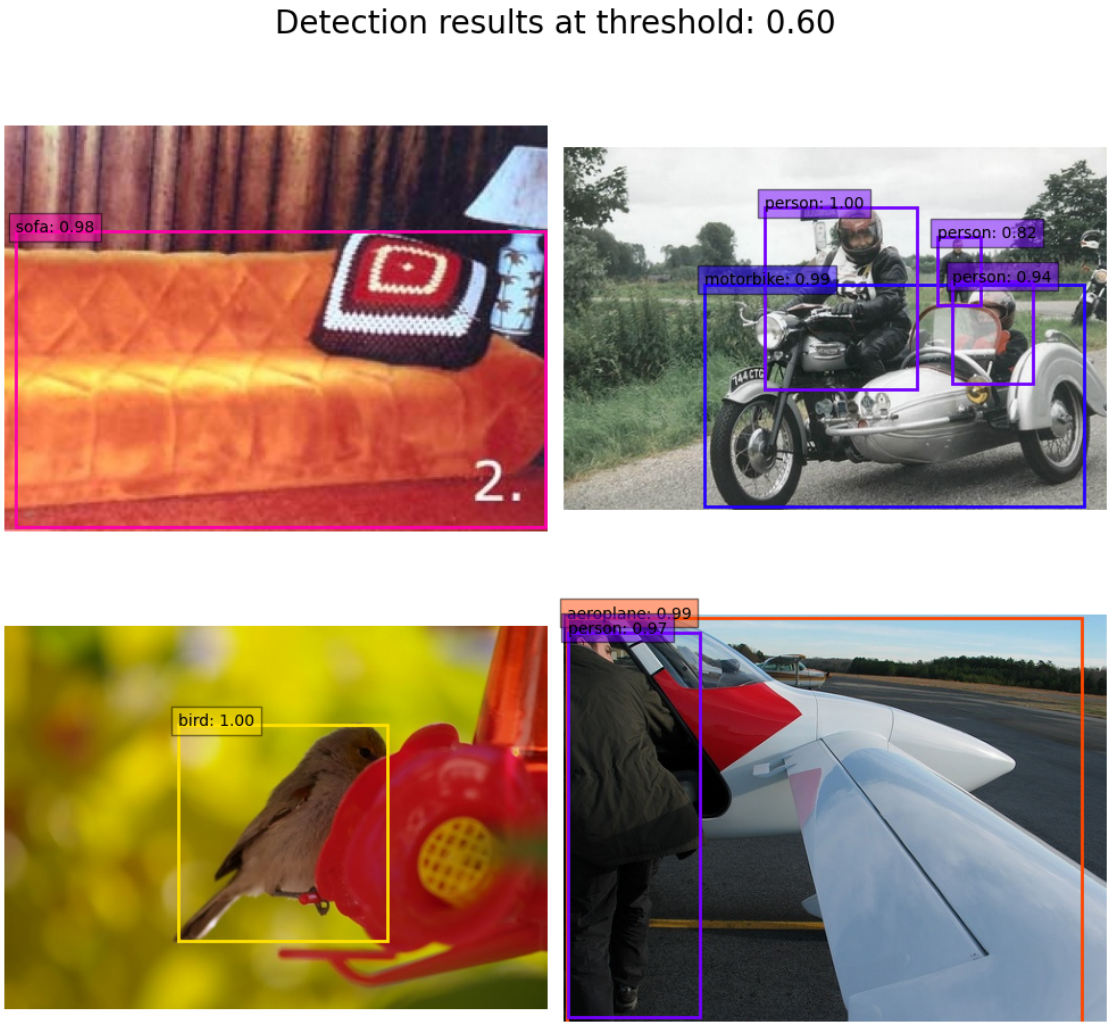


4. Run SSD on the sample images and show the results

[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: %1.2f" %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)
```



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.

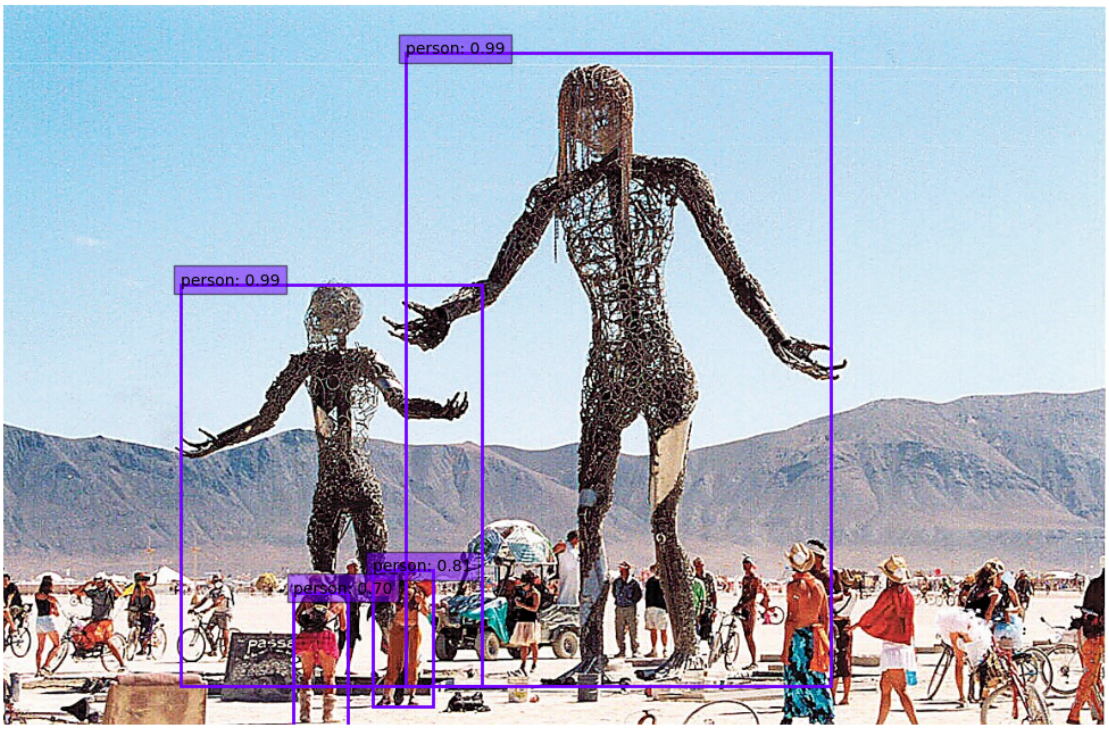
1.2.1 2.1

a) Based on the results (test images1) below what kind of objects are easier for the network to detect? Based on the detection results in the provided test image, the network seems to detect larger and more prominent objects, such as people, more accurately. This is because larger

objects typically occupy more pixels in the image, making their features easier to recognize by the network. Smaller objects or those with less distinct features tend to be more challenging for the network, as their representations in the feature map become less significant after downsampling in convolutional layers.

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 is likely to improve the detection of smaller objects. A larger input size retains more spatial detail, particularly for small objects, which helps the network to better capture their features. However, increasing the input resolution also results in higher computational costs and memory usage.Designing an object detector to use HD resolution inputs (1920x1080 or higher) could improve accuracy, particularly for detecting very small objects in large-scale images. However, this would come at a significant computational cost. It would slow down the model and require more resources, making it unsuitable for real-time applications. Thus, while higher resolutions might improve performance, it is important to balance accuracy with computational efficiency, depending on the application context.

[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)
```



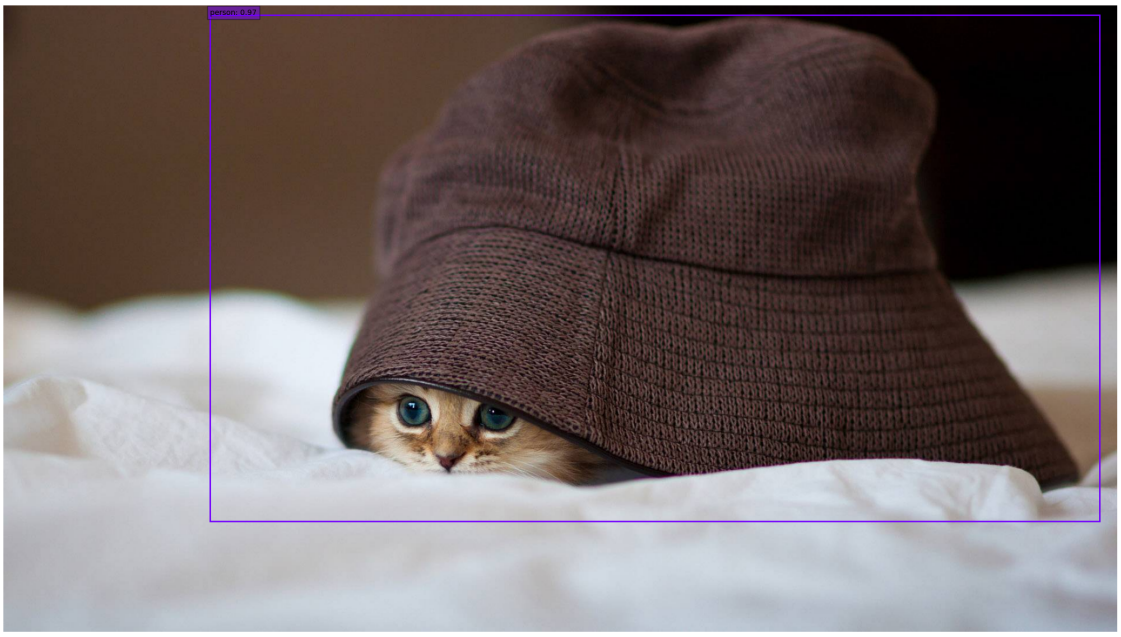
1.2.2 2.2

a) Based on the results (test images2) below elaborate why the detector has trouble detecting the objects? Based on the provided test images, the detector struggles with certain cases due to occlusion and complex object interactions. In the first image, the cat is partially hidden under a hat, which leads the model to confuse it with a "person" class. This indicates a limitation in handling partially visible objects. In the second image, the snowy background and overlapping objects (sled dogs and the person) increase the complexity of detection. Such scenarios make it difficult for the network to accurately separate individual objects and maintain high confidence scores.

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 tried to address these challenges using data augmentation techniques. These include random cropping, which helps the network learn to detect partially occluded objects, and scaling and translation, which allow the network to generalize across different object sizes and positions. Additionally, color distortion techniques were applied to improve robustness against varying lighting conditions and backgrounds. These methods collectively enhance the network's ability to handle difficult detection scenarios.

[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? In the first image, the detector struggles due to the complex background and overlapping objects. The large sculpture and the people interacting with it create a visually cluttered scene, making it harder for the detector to isolate individual objects. Additionally, the varying scales and occlusions among the people reduce the detection accuracy, especially for smaller or partially hidden objects. In the second image, while the horse is detected correctly, simpler backgrounds may lead to better performance. However, in more cluttered or dynamic scenes, the detector may struggle with distinguishing similar objects or with objects at various scales.

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 addressed these challenges through multi-scale feature maps and default boxes. Multi-scale feature maps allow the network to detect objects of different sizes by making predictions at various resolutions. Default boxes with varying aspect ratios and scales are also used to improve detection accuracy across different object shapes and sizes. Additionally, data augmentation techniques were applied to improve the model's robustness in complex scenarios, such as random cropping and scaling to simulate occlusion and scale variance.

[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)
```

1.2.4 2.4

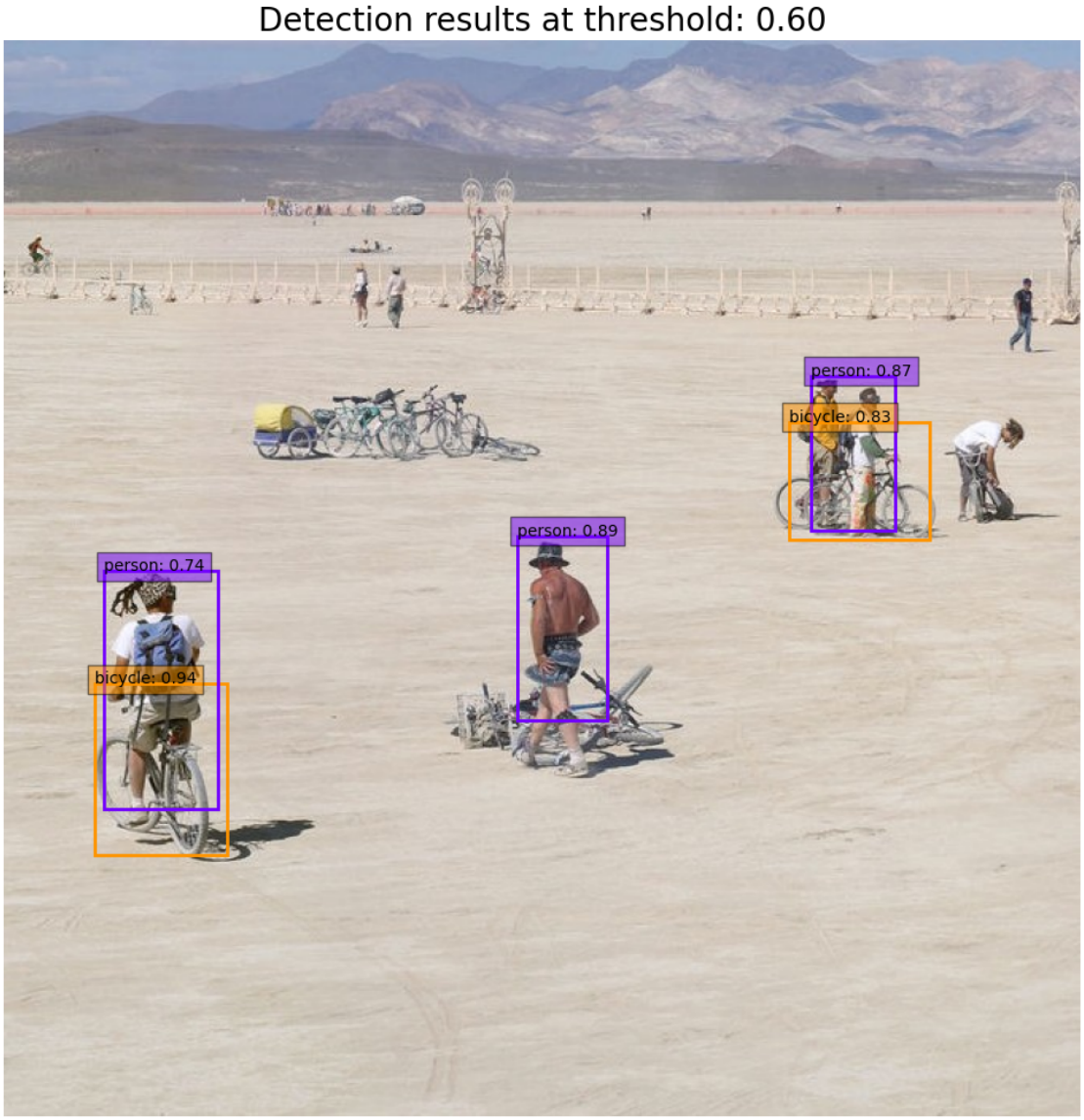
a) Based on the results (test images4) below elaborate why the detector has trouble detecting objects in the first/upper image, but is able to detect objects in the second/lower image which contains the left part of the upper image? Note: you can double click the upper image to show it in actual size. In the first image, the detector struggles due to the large-scale scene, which contains many small objects spread across a wide area. Objects such as people and bicycles appear smaller, and their features are less prominent, making them harder to detect. Additionally, the extensive background introduces more noise, reducing the detector's ability to focus on relevant objects. In contrast, the second image focuses on a smaller

portion of the scene, where objects are relatively larger and more distinguishable. This allows the detector to better capture their features and achieve higher detection accuracy.

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 addressed this issue using multi-scale feature maps and default boxes. Multi-scale feature maps help detect objects of various sizes by utilizing feature maps at different resolutions. Default boxes with varying aspect ratios improve detection performance for small objects. Additionally, data augmentation techniques such as random cropping and scaling enhance the model's robustness to different object sizes and scene complexities.

[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)
```



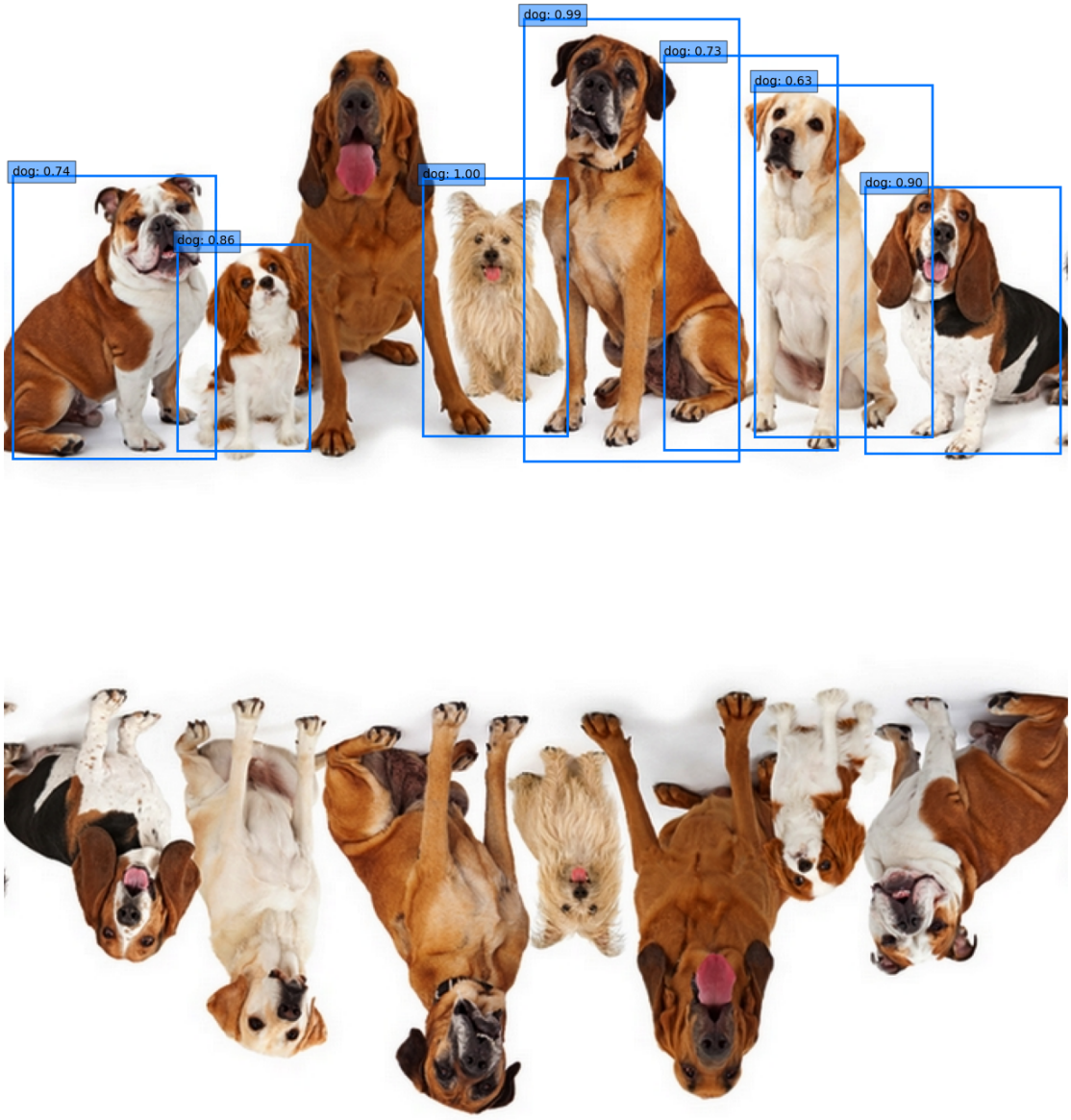
1.2.5 2.5

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 SSD publication does not specifically mention tackling the issue of significant rotations or vertical flips. However, a naive method to mitigate this problem would be to include rotation-based data augmentation during training. By introducing rotated versions of the training images, the model would learn to detect objects under varying orientations. This method might not always be necessary because real-world applications rarely involve such extreme cases, and adding rotation augmentation could

increase training time and complexity without substantial benefits. Furthermore, it may lead to overfitting for rotated objects, which are not common in the target application.

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) Yes, convolutional networks are naturally able to handle small rotations to some extent. This is due to the presence of max pooling layers, which provide some level of spatial invariance. Max pooling reduces the sensitivity to small transformations, including slight rotations, by focusing on the most prominent features within a region. However, for significant rotations, the network would struggle unless specifically trained for such scenarios through augmentation or specialized invariant designs.

[14]:
```
# 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? Lowering the confidence threshold has both advantages and disadvantages. On the upside, it increases the recall by detecting more objects, including those with low confidence. This can be useful in scenarios where missing an object could have severe consequences, such as in security or disaster response tasks. However, on the downside, it increases the number of false positives, lowering precision. This can be problematic in applications where accuracy is critical, such as medical diagnostics or autonomous driving. To measure the effect of the confidence threshold, one could use a Precision-Recall curve or calculate the F1-score, which balances precision and recall. Tasks like safety monitoring in airports or rescue missions could benefit from a lower confidence threshold, while tasks like medical imaging and self-driving vehicles require a high confidence threshold to ensure reliable decisions.

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. From the video, it's clear that YOLO V2 demonstrates incredible speed in object detection, far surpassing human perception. It efficiently detects objects across numerous images or video frames, even managing to identify objects in scenes that were challenging for me to recognize. What's particularly impressive is its ability to maintain high detection accuracy even with shaky or unstable footage, consistently capturing and classifying objects in real-time. While YOLO V2 performs exceptionally well, it does occasionally make mistakes. For instance, it sometimes misclassifies objects, such as identifying a chair as a suitcase, a window as a bench, a motorcycle as a horse, a wall as a train, a dome as an umbrella, or even the floor as a bed. However, these errors are generally minor and do not significantly impact the overall effectiveness of the detector. Considering these factors, the technology's capability remains highly impressive. On the other hand, humans excel in understanding the broader context of a scene. This allows us to detect subtle details that object detectors might overlook. Humans are also better at identifying novel objects that the detector hasn't been trained on and distinguishing between fine differences, such as a real cat versus a stuffed toy. This contextual and nuanced understanding provides a critical complement to the speed and efficiency of automated detectors like YOLO V2.

[ ]: