

In []:

```
# This cell is used for creating a button that hides/unhides code cells to quickly
# 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 t'''
```

In [1]:

```
# 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

CS-E4850 Computer Vision Exercise Round 10

Student: Eugeniu Vezeteu - 886240

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`.

2. Unfortunately PyTorch was not included in the Docker image I created for this years course and therefore this assignment will not work with your docker container based on that image. For this exercise the patented parts of opencv are not needed so it's still easy to work on the exercise locally by creating a Python environment and installing the correct packages. For example using Conda:

```
conda create --name cv2019
conda activate cv2019
conda install -c pytorch pytorch
conda install -c conda-forge opencv, jupyterlab, matplotlib
jupyter lab
```

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](#) (<http://arxiv.org/abs/1512.02325>) 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.

Type your answer here:

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?

Type your answer here:

c) SSD has its 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?

Type your answer here:

1. Load the SSD architecture and the pretrained weights

In [2]:

```
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!

2. Load Sample Images

In [3]:

```
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, VOCAnnotator)
        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



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.

In [4]:

```
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
```

In [5]:

```
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

In [6]:

```
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

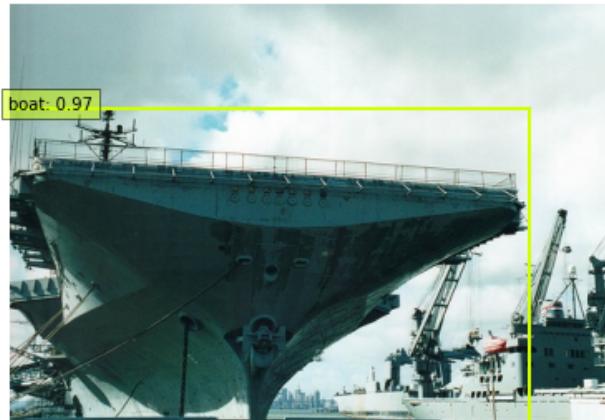
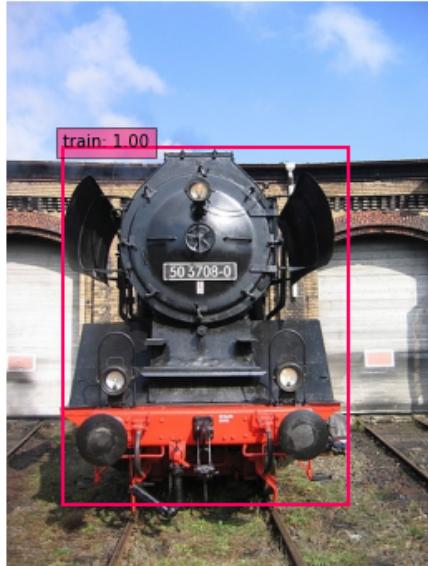
        # colormap for the bounding boxes
        colors = plt.cm.hsv(np.linspace(0, 1, 21)).tolist()

        # 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))
                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()
```

In [7]:

```
# Filter outputs with confidence scores lower than a threshold. The default threshold
run_network(images, nrows=2, ncols=2, figsize=(10,10), threshold=0.6)
```

Detection results at threshold: 0.60



Exercise 1.a)

The authors present deep network-based object detector that does not re-sample pixels or features for bounding box hypotheses and is as accurate as approaches that do. By eliminating the bounding box proposal and the subsequent pixel or feature resampling stage, the model gains speed improvement. They replace the bounding box proposal with a small convolutional filter to predict object categories and offsets in bounding box locations. These filters are applied for different aspect ratio and multiple feature maps, this allow to detect the object at multiple scales. This outperform YOLO and other versions of R-CNN, being faster at 59 fps, and achieving mAP 74.3%

Exercise 1.b)

The purpose of the base network is to extract the image features, it's a standard architecture used for high-quality image classification. The base network used in the publication is VGG-16, which was pre-trained on the ILSVRC CLS-LOC dataset.

Exercise 1.c)

The loss function is composed of localization loss and confidence loss. The localization loss is a smooth L1 loss between the predicted box and the true box values. In the loss function, the localization loss is weighted by some parameter alfa, which was selected by cross-validation to be 1. The confidence loss is the softmax loss over multiple classes confidences. The sum of the above 2 losses are divided by the number of matched default boxes, to form the final loss.

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 \(https://arxiv.org/abs/1312.5402\)](https://arxiv.org/abs/1312.5402). Read the chapters in the publication and selectively read the main points of reference 14 and answer the following questions.

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: Based on the test images1 below, it is easier to detect the objects that are full size are not too big or too small compared to the whole image. We notice that the two mannequins have the optimal size to be detected as people, while only 2-3 real people are detected in front, the others are not detected at all.

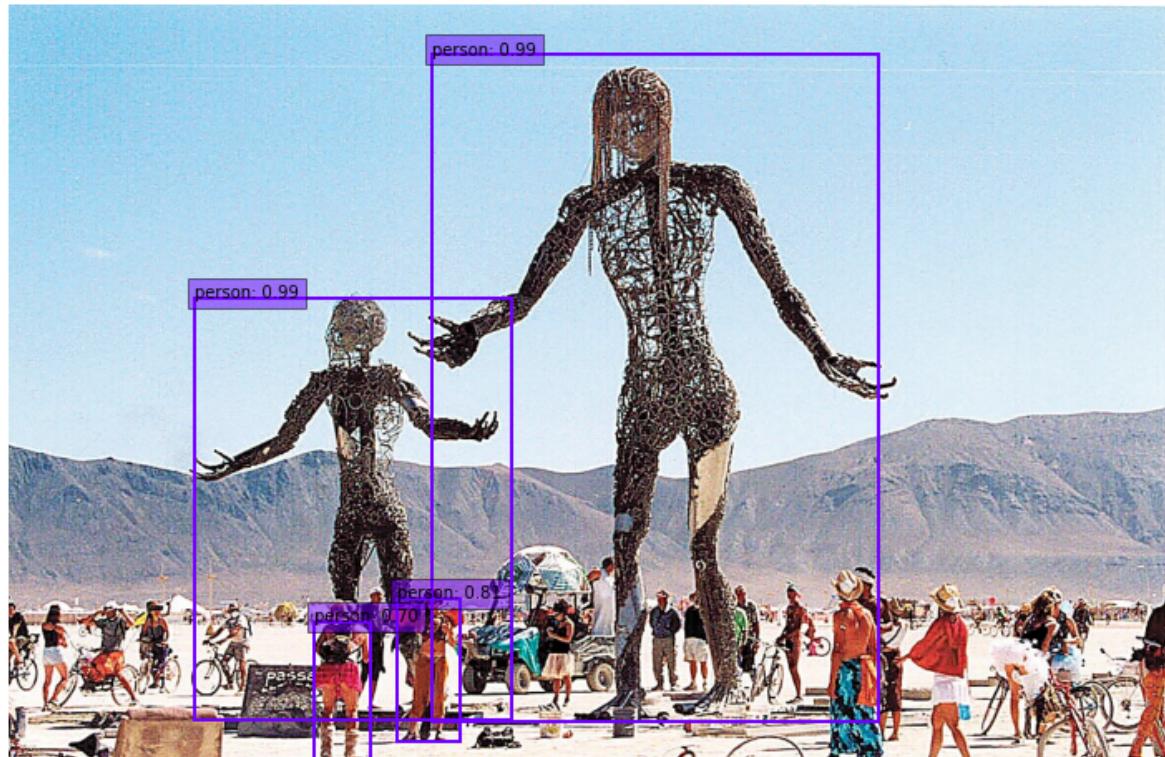
b) Would switching from input size 300x300 to 512x512 make the problem better, worse, or have no impact? Elaborate on why or why not. Would designing a object detector that uses HD resolution of 1920x1080, or even higher, images as inputs be a good idea? Elaborate on why or why not.

Type your answer here: As stated in the original paper, larger input size leads to better results, and more data always helps. Using image size of 512x512 instead of 300x300 improves the classification performance, since it used multiple layers for prediction at different scales. It is not a good idea to use the HD image resolution for the network, since it will require a lot of computational power, and here we have the tradeoff between the accuracy and the speed. Using HD resolution, may not be possible to run it on real-time.

In [8]:

```
# Photo credit: to burningman.org. Used under the fair use principles for transform
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



2.2

- a) Based on the results (test images2) below elaborate why the detector has trouble detecting the objects?

Type your answer here: The detector has problems to detect objects that overlaps. This may happen due to non-max suppression, when most of the bounding boxes are thrown away, using the smaller threshold for non-max suppression may result in multiple bounding boxes for the same object. Also, the cat in the image was detected as a person, this is because the overall shape of the detected object is more similar to the human head, then to a cat. This may be a way to fool the detector, also I think we should include this kind of images in the training dataset.

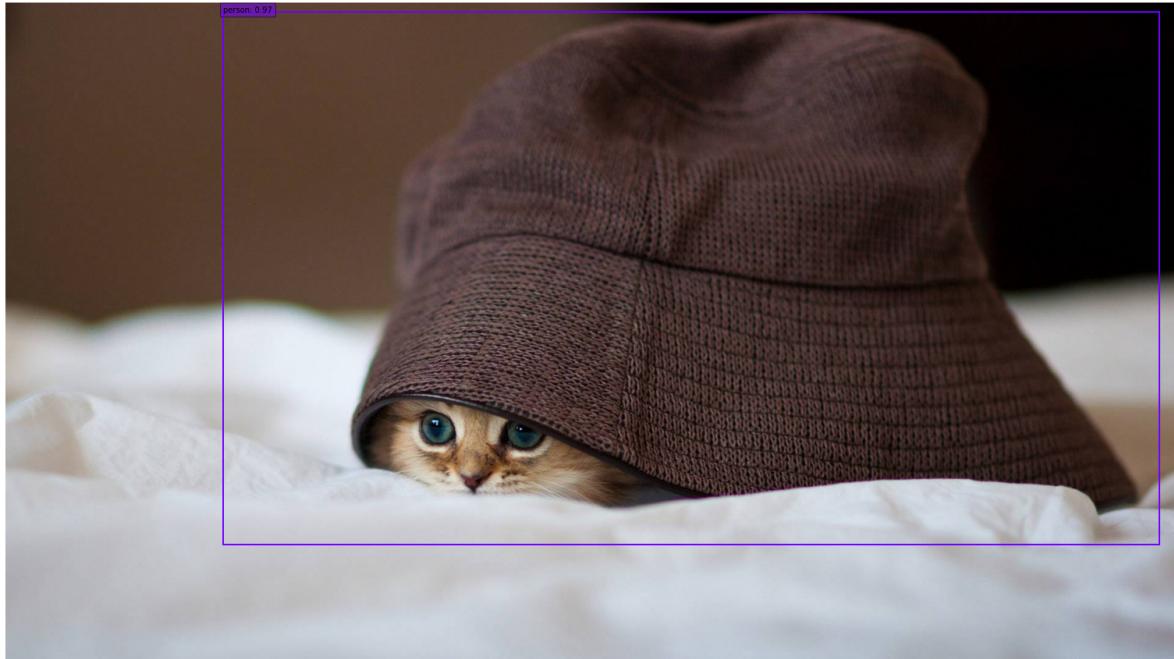
b) Have the authors of the SSD publication tried to mitigate this problem? If yes, briefly explain how.

Type your answer here: To handle the detection of various object sizes and shapes the authors proposed following data augmentation techniques: a) Use the entire original input image b) Sample a patch so that the minimum Jaccard overlap with the objects is 0.1, 0.3, 0.5, 0.7, or 0.9. c) Randomly sample a patch. \ After the aforementioned sampling step, each sampled patch is resized to fixed size and is horizontally flipped with probability of 0.5

In [9]:

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

Detection results at threshold: 0.60



2.3

a) Based on the results (test images3) below, elaborate why the detector has trouble detecting the objects?

Type your answer here: Based on the images below, the model has problems to detect small objects in the scene and object occlusions. In the case of the horses, the smaller one wasn't detected since it is already included in the bigger bounding box, which correspond to the bigger horse. In the case of humans, the problem

is as stated before, small size and also the background bay cause some issues

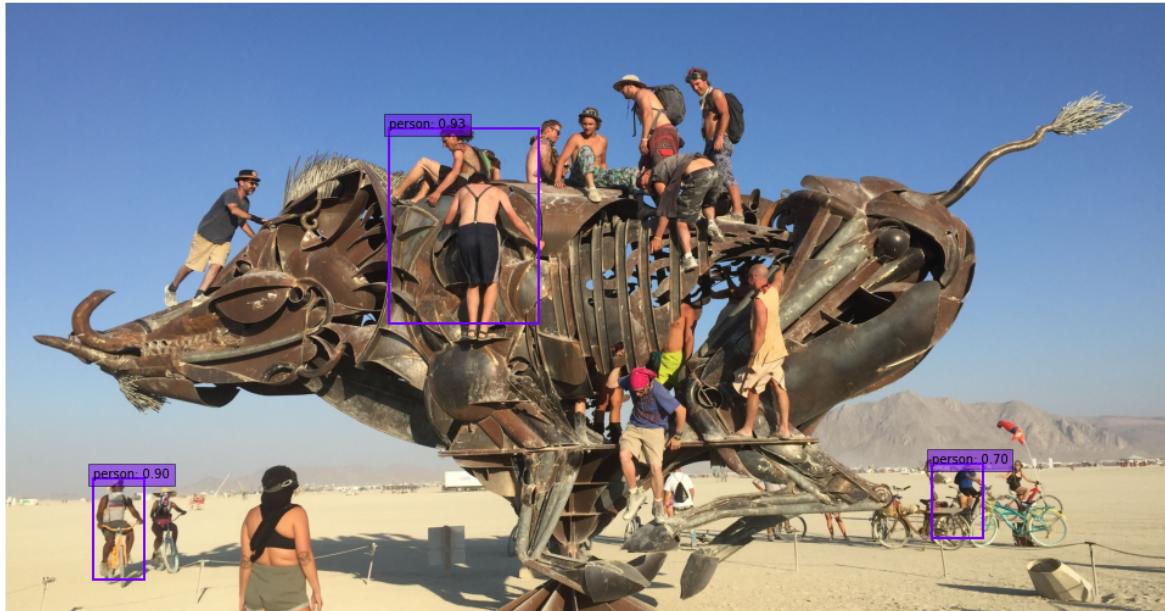
b) Have the authors of the SSD publication tried to mitigate this problem? If yes, briefly explain how.

Type your answer here: The authors present the data augmentation techniques to handle all these issues.
 Solutions presented above in section 2.2.b)

In [10]:

```
# Photo credit: Duncan Rawlinson - Duncan.co photo from flickr. Creative Commons license.
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/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



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.

Type your answer here: As stated before, the model has problems to detect small size objects, also based on the images4 we have a scaling issue, the lower image is a scaled version of the above one, and it contains the object with the optimal sizes, to be detected.

b) Have the authors of the SSD publication tried to mitigate this problem? If yes, briefly explain how.

Type your answer here: The authors propose the tiling of default boxes so that specific feature maps learn to be responsive to particular scales of the objects. The scale of the default box for each feature map is computed individually using: $s_k = s_{\text{min}} + \frac{s_{\text{max}} - s_{\text{min}}}{m-1} \cdot (k-1)$, where $k \in [1, m]$, and m is the number of feature maps.

In [11]:

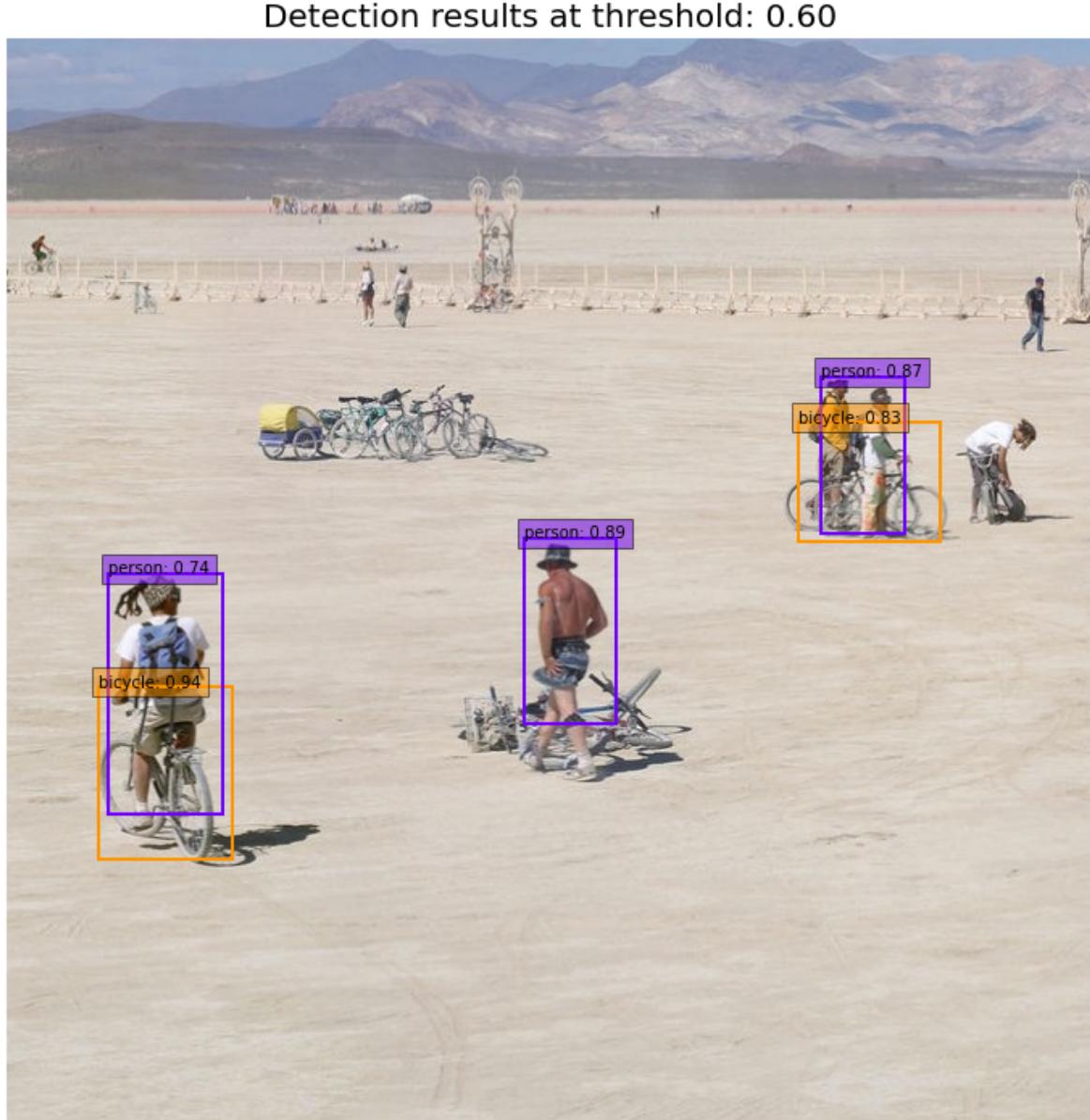
```
# Photo credit: Brad Templeton. Used under the fair use principles for transformation
image41 = cv2.imread(data_dir+'/test_images/img41.jpg', cv2.IMREAD_COLOR)
image42 = cv2.imread(data_dir+'/test_images/img42.jpg', cv2.IMREAD_COLOR)
images4 = [cv2.cvtColor(image41, cv2.COLOR_BGR2RGB), cv2.cvtColor(image42, 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='')


```



In [12]:

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



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).

Type your answer here:IN the original paper only horizontal flip was used. To solve the issue the images can be, flipped vertically, not only horizontally, rotated or subsampled with different crops and scales. Also, noise can be added to the input images.

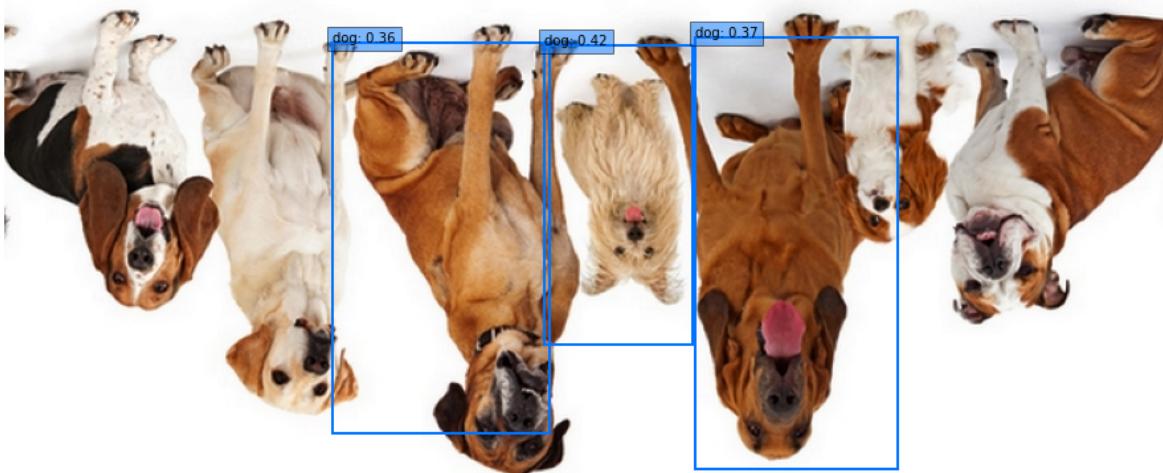
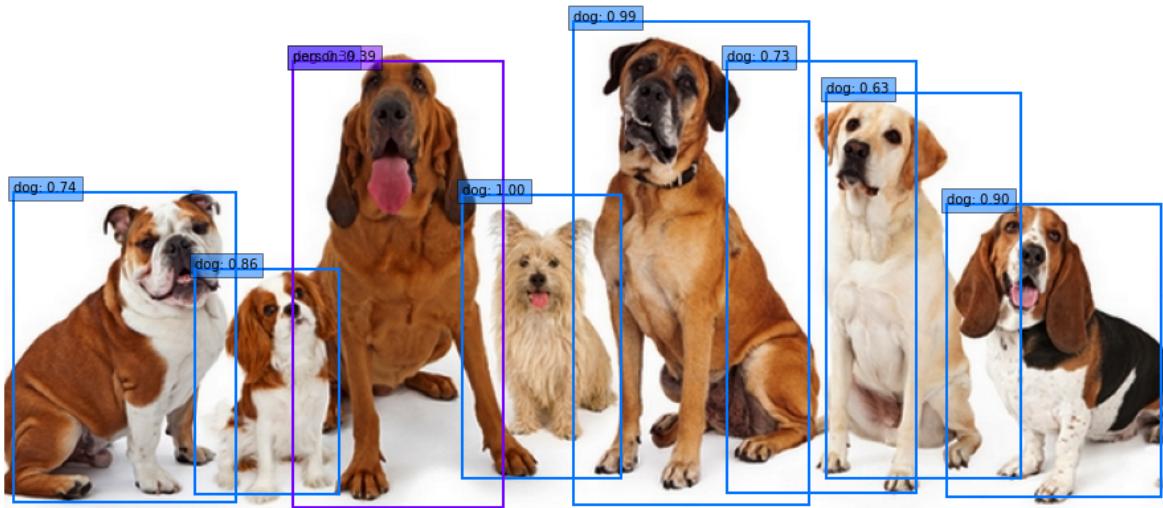
b) Is a convolutional network naturally, without specifically addressing the issue, able to detect objects if there are small rotations? If yes, briefly explain which part of the network helps to mitigate the effects of rotation and why. (Hint: what layers are sometimes between two convolutional layers)

Type your answer here:The max-pooling layers may be insensitive to small shifts and rotations, but not that much. If we want our model to be invariant to such transformations, we shoult use spatial transformers.

In [14]:

```
# Photo credit: Susan Schmitz / shutterstock. Used under the fair use principles for
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



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?

Type your answer here: The model confidence and Intersection over Union (IoU) are used to predict detection is True Positive or a False Positive. Lowering the threshold allows the model to predict more objects, but there might be FP and TN detections. The higher the threshold is, the more severe the detection. For example, if I have an object detection problem, and am interested in detecting a single class (eg car), I will leave a lower threshold to detect all cars, otherwise if there are multiple classes (eg 20 classes VOC dataset) I will increase threshold to the recommended 0.5-0.6

b) Watch the following YouTube video of a detector that is similar to SSD called [YOLO V2 \(You Only Look Once\)](https://www.youtube.com/watch?v=VOC3huqHrss) (<https://www.youtube.com/watch?v=VOC3huqHrss>) 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.

Type your answer here: The human visual system is incredibly adapted to "seeing" in a 3D world. Everything that has to do with optical illusions is difficult to emulate on a machine. In these cases, human outperform the state of the art object detectors, overlapping objects, absurd cases (for example: horse coloured like a zebra, etc.), all kinds of transformations, rotation, scaling, are more easily detected by man.\

State-of-the-art object detectors outperform human detection in terms of speed (for examining object counts) or detect objects that have not been seen so often by humans (e.g., bacteria under a microscope)

In []:

