Base Model

In [1]:
```
%load_ext autoreload
%autoreload 2
```

In [2]:
```
import os
import sys
from shutil import copyfile

import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import torch
import torch.optim as optim
from torch.utils.data import TensorDataset, DataLoader
from torch_snippets import Report
import time
```

```
2021-07-11 20:45:47.858 | WARNING  | torch_snippets.torch_loader:<module>:233 - Not importing Lightning Report
2021-07-11 20:45:48.150 | WARNING  | torch_snippets:<module>:13 - sklearn is not found. Skipping relevant imports from submodule `sklegos`
Exception: No module named 'sklego'
```

In [46]:
```
base_model_path = '../src/base-model/'

module_path = os.path.abspath(os.path.join('../src/base-model/'))
if module_path not in sys.path:
    sys.path.append(module_path)

import utils
import rcnn
```

In [47]:
```
## sample of 135 images
## resized by a factor of 4 from 2048x1024 to 512x256
imgs_path = '../data/sample-dataset/'
```

In [48]:
```
## load annotations of 135 images
anno_dict = np.load('../data/anno-big.npy', allow_pickle='TRUE').item()
imgs_person = list(anno_dict.keys())
len(imgs_person)
```

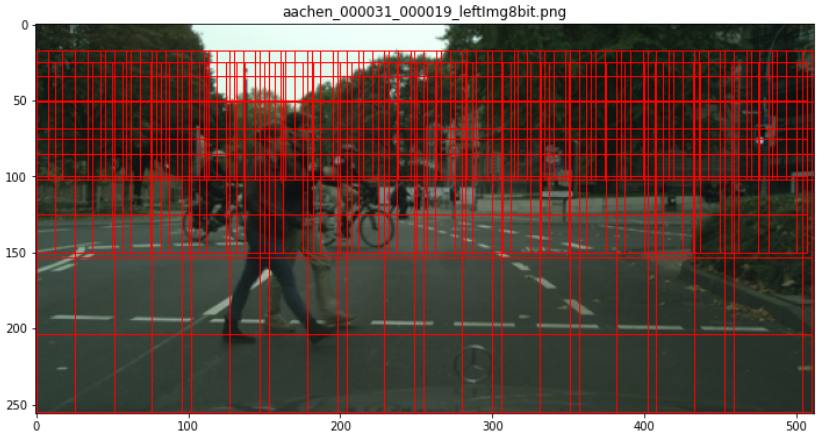
Out[48]: 135

In [49]:
```
## resize annotations by factor SCALE
SCALE = 4
res_anno_dict = {}
for img_name in imgs_person:
    res_bboxes = []
    for bbox in anno_dict[img_name]:
        res_bbox = bbox / SCALE
        res_bbox = res_bbox.astype('int32')
        res_bboxes.append(res_bbox)
    res_anno_dict[img_name] = res_bboxes
anno_dict = res_anno_dict # keep only the resized dict
```

In [50]:
```
plt.rcParams['figure.figsize'] = [12, 8]
```

In [51]:
```
## get regions
regions = utils.get_regions()
print('%d regions:' % len(regions))

## check result
i = 0
img_name = imgs_person[i]
img_path = imgs_path + img_name
img = Image.open(img_path)
fig, ax = plt.subplots()
ax.imshow(img);

for bbox in regions:
    rect = patches.Rectangle(
        (bbox[0], bbox[1]), bbox[2], bbox[3],
        linewidth=1, edgecolor='red', facecolor='none')
    ax.add_patch(rect)

plt.title(img_name)
plt.show()
```

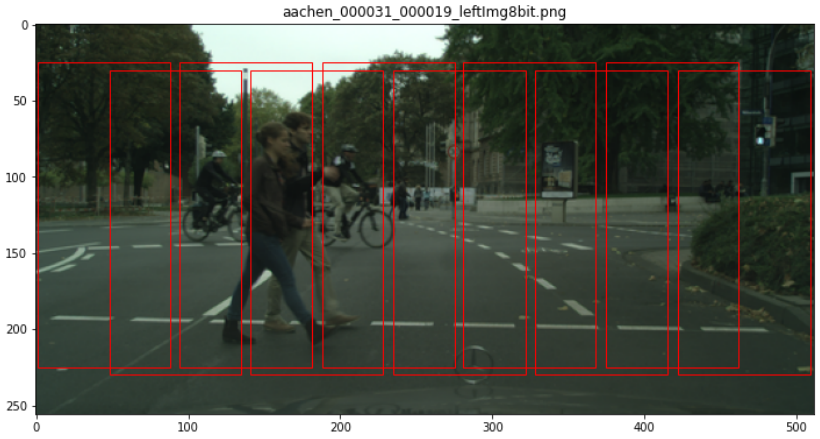
```
351 regions:
```



In [52]:

```
## get simple regions
regions = utils.get_regions(simple=True, scale=SCALE)
print('%d regions:' % len(regions))

## check result
i = 0
img_name = imgs_person[i]
img_path = imgs_path + img_name
img = Image.open(img_path)
fig, ax = plt.subplots()
ax.imshow(img);

for bbox in regions:
    rect = patches.Rectangle(
        (bbox[0], bbox[1]), bbox[2], bbox[3],
        linewidth=1, edgecolor='red', facecolor='none')
    ax.add_patch(rect)

plt.title(img_name)
plt.show()
```

```
10 regions:
```



In [53]:

```
i = 0
img_name = imgs_person[i]
img_path = imgs_path + img_name
img = Image.open(img_path)
bboxes = anno_dict[img_name]
```

In [54]:

```
fig, ax = plt.subplots()
ax.imshow(img)

for bbox in bboxes:
    rect = patches.Rectangle(
        (bbox[0], bbox[1]), bbox[2], bbox[3],
        linewidth=1, edgecolor='r', facecolor='none')

    ax.add_patch(rect)

plt.title(img_name)
plt.show()
```

In [55]:
```
fig, ax = plt.subplots()
ax.imshow(img)

for bbox in bboxes:
    rect = patches.Rectangle(
        (bbox[0], bbox[1]), bbox[2], bbox[3],
        linewidth=1, edgecolor='orange', facecolor='none')
    ax.add_patch(rect)

for bbox in regions:
    rect = patches.Rectangle(
        (bbox[0], bbox[1]), bbox[2], bbox[3],
        linewidth=1, edgecolor='red', facecolor='none')
    ax.add_patch(rect)

plt.title(img_name)
plt.show()
```

In [56]:
```
img_np = np.array(img)
H, W = img_np.shape[:2]
```

Prepare data

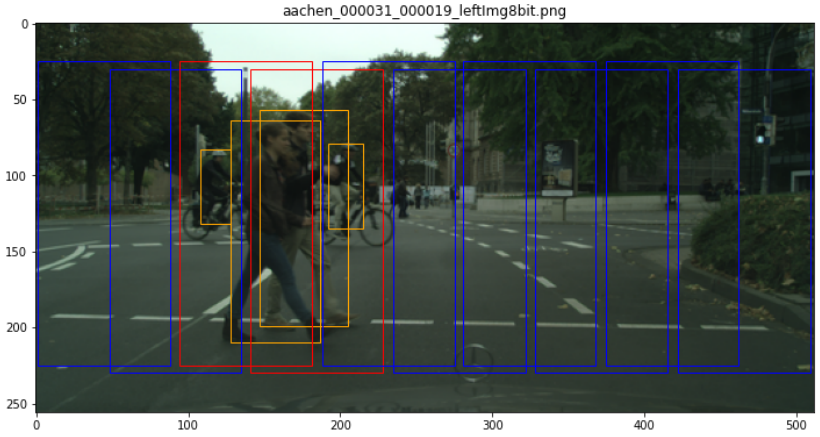
In [57]:
```
all_img_names, all_labels, all_diffs, all_rois, all_bboxes = utils.get_data(
    imgs_person, anno_dict, regions, W, H)
```

In [58]:
```
len(all_img_names), len(all_labels)
```

Out[58]: (135, 135)

In [59]:
```
## check results
i = 0
img_name = all_img_names[i]
bboxes = all_bboxes[i]

img_path = imgs_path + img_name
img = Image.open(img_path)

fig, ax = plt.subplots()
ax.imshow(img)

labels = all_labels[i]
colors = ['b', 'r']

for i, bbox in enumerate(bboxes):
    rect = patches.Rectangle(
        (bbox[0], bbox[1]), bbox[2], bbox[3],
        linewidth=1, edgecolor='orange', facecolor='none')
    ax.add_patch(rect)

for i, bbox in enumerate(regions):
    rect = patches.Rectangle(
        (bbox[0], bbox[1]), bbox[2], bbox[3],
        linewidth=1, edgecolor=colors[labels[i]], facecolor='none')
    ax.add_patch(rect)
```

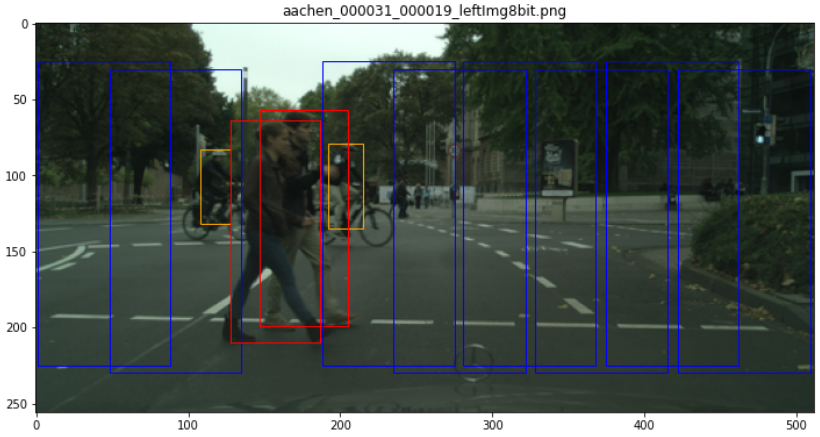
```
plt.title(img_name)
plt.show()
```



In [60]:
```
img_region = np.array([W, H, W, H]) # for scaling
diffs0 = all_diffs[0] * img_region
diffs0
```

Out[60]:
```
array([[-146.,  -32.,   29.,   58.],
       [ -60.,  -53.,   67.,  151.],
       [ -34.,  -39.,   28.,   54.],
       [  -6.,  -27.,   29.,   58.],
       [  41.,  -32.,   29.,   58.],
       [  88.,  -27.,   29.,   58.],
       [ 134.,  -32.,   29.,   58.],
       [ 181.,  -27.,   29.,   58.],
       [ 228.,  -32.,   29.,   58.],
       [ 275.,  -27.,   29.,   58.]])
```

In [61]:
```
i = 0
img_name = all_img_names[i]
bboxes = all_bboxes[i]
img_path = imgs_path + img_name
img = Image.open(img_path)

fig, ax = plt.subplots()
ax.imshow(img);

labels = all_labels[i]
colors = ['b', 'r']

# bbox = [x, y, w, h]
for i, bbox in enumerate(bboxes):
    rect = patches.Rectangle(
        (bbox[0], bbox[1]), bbox[2], bbox[3],
        linewidth=1, edgecolor='orange', facecolor='none')
    ax.add_patch(rect)

for i, bbox in enumerate(regions):
    if labels[i] == 1:
        bbox = bbox - diffs0[i] # add the diffs

    rect = patches.Rectangle(
        (bbox[0], bbox[1]), bbox[2], bbox[3],
        linewidth=1, edgecolor=colors[labels[i]], facecolor='none')
    ax.add_patch(rect)

plt.title(img_name)
plt.show()
```



Prepare train and test sets

In [62]:
```
img_paths = [(imgs_path + img_name) for img_name in all_img_names]
n_train = 8 * len(img_paths) // 10
n_train
```

Out[62]:
```
108
```

In [63]:
```
## img_paths, rois, labels, diffs, bboxes
train_set = rcnn.Dataset(
    img_paths[:n_train], all_rois[:n_train], all_labels[:n_train],
    all_diffs[:n_train], all_bboxes[:n_train])

test_set = rcnn.Dataset(
    img_paths[n_train:], all_rois[n_train:], all_labels[n_train:],
    all_diffs[n_train:], all_bboxes[n_train:])
```

In [64]:
```
## check the results
image, crops, roi_bboxes, labels, diffs, bboxes, img_path = train_set[0]


plt.rcParams['figure.figsize'] = [5, 3]
fig, ax = plt.subplots()
ax.imshow(crops[3])
plt.show()

print(labels[3])
```

1

In [65]:
```
train_loader = DataLoader(
    train_set, batch_size=2, collate_fn=train_set.collate_fn, drop_last=True)
```

In [66]:
```
test_loader = DataLoader(
    test_set, batch_size=2, collate_fn=test_set.collate_fn, drop_last=True)
```

Define the backbone

In [24]:
```
backbone = utils.get_backbone()
```

In [25]:
```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
device
```

Out[25]:
```
'cpu'
```

In [26]:
```
backbone.eval().to(device)
```

Out[26]:
```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential()
)
```

Train the model

In [27]:
```
model = rcnn.BaseModel(backbone).to(device)
criterion = model.calc_loss
optimizer = optim.SGD(model.parameters(), lr=1e-3)
```

In [28]:
```
n_epochs = 5

## log for plot training and validation metrics
log = Report(n_epochs)
tstart = time.time()
for epoch in range(n_epochs):
```

```
    _n = len(train_loader)
    for i, inputs in enumerate(train_loader):
        loss = rcnn.train_batch(inputs, model, optimizer, criterion)
        pos = (epoch + (i + 1)/_n)
        log.record(pos,
                   trn_loss=loss.item(),
                   end='\r')

    _n = len(test_loader)
    for i, inputs in enumerate(test_loader):
        loss = rcnn.validate_batch(inputs, model, criterion)
        pos = (epoch + (i + 1)/_n)
        log.record(
            pos,
            val_loss=loss.item(),
            end='\r')

tend = time.time()
```

```
EPOCH: 5.000    val_loss: 0.029 (767.11s - 0.00s remaining))))
```

Training on 8th Gen Intel i5 cpu (i5-9500T CPU @ 2.20GHz) took about:

In [29]:

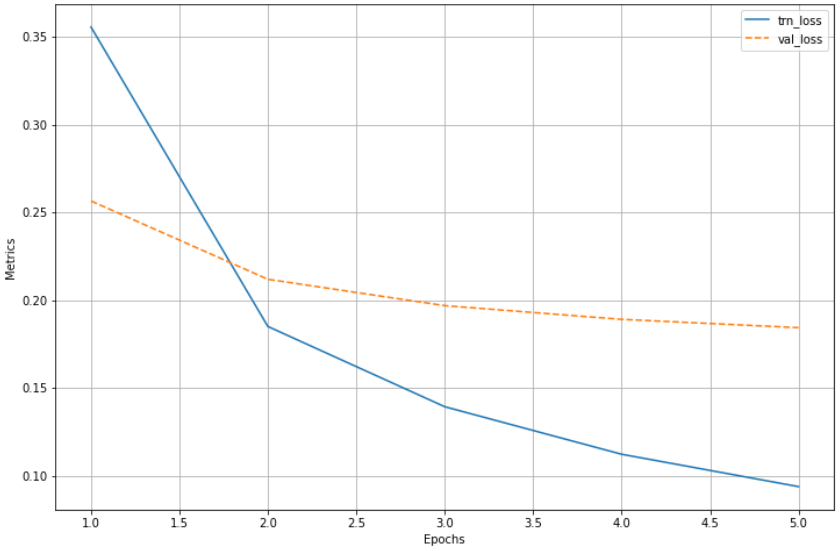
```
print('Time elapsed = %.2f min' % ((tend - tstart)/60))
```

```
Time elapsed = 12.79 min
```

In [30]:

```
plt.rcParams['figure.figsize'] = [12, 8]
fig, ax = plt.subplots()
log.plot_epochs(ax=ax)
```

```
100%|██████████| 106/106 [00:00<00:00, 6104.24it/s]
```



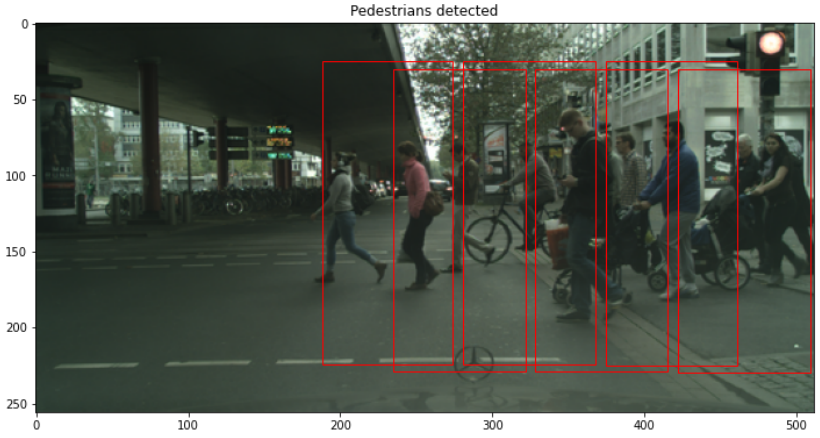
In [114…

```
def detect(img, img_name='test.png', scale=4,
           show=True, save=False, save_dir=''):
    np_image = np.array(img)

    ## reverse the code from before
    regions = utils.get_regions(simple=True, scale=SCALE)
    input = []
    crops = []
    for region in regions:
        x, y, w, h = region
        x0, y0, x1, y1 = x, y, int(x + w), int(y + h)
        crop = np_image[y0:y1, x0:x1]
        crops.append(crop)

    newsize = (224, 224)
    crops = [Image.fromarray(crop, 'RGB') for crop in crops]
    crops = [crop.resize(newsize) for crop in crops]
    crops = [utils.preprocess_image(crop)[None] for crop in crops]

    input = torch.cat(crops).to(device)
    with torch.no_grad():
        model.eval()
        probs, diffs = model(input)
        probs = torch.nn.functional.softmax(probs, -1)
        confs, classes = torch.max(probs, -1)

    regions = np.array(regions)
    confs, classes, probs, diffs = [
        tensor.detach().cpu().numpy() for tensor in [confs, classes, probs, diffs]]

    ## TODO: use nms to lower the recall

    ## adding predicted diffs
    detected_bboxes = (regions + diffs).astype(np.uint16)

    if show:
        fig, ax = plt.subplots()
        if img_name != 'test.png':
            plt.title(img_name)
        else:
            plt.title('Pedestrians detected')
        ax.imshow(img);

        classes = classes.tolist()
        for i, bbox in enumerate(detected_bboxes):
            if classes[i] == 1:
                rect = patches.Rectangle(
```

```
                    (bbox[0], bbox[1]), bbox[2], bbox[3],
                    linewidth=1, edgecolor='r', facecolor='none')
                ax.add_patch(rect)
        plt.show()

        if save:
            fig.savefig(save_dir + img_name)

    return {'confs': confs, 'classes': classes}
```

In [77]:
```
def detect_wrap(i):
    img_name = all_img_names[i]
    img_path = imgs_path + img_name
    img = Image.open(img_path)
    return detect(img, show=True)
```

In [78]:
```
plt.rcParams['figure.figsize'] = [12, 8]
for i in range(3):
    detect_wrap(i)
```



Using more anchor boxes, we get larger training set and tighter detections. TODO: show example for 20 and then 2000.

Video

Let's take some frames from KITTI-17 https://motchallenge.net/vis/KITTI-17

In [79]:
```
frames_path = '../data/sample-video-frames/'
```

In [80]:
```
## crop and scale frames
crops = []
SCALE2 = 1.4
for k in range(1, 19):
    img_name = 'KITTI-17-raw_' + str(k) + '.png'
```

```
    img = Image.open(frames_path + img_name)

    h, w = (456, 412)
    np_image = np.array(img)
    x0, y0 = 190, 40
    x1, y1 = x0 + int(w // SCALE2), y0 + int(h // SCALE2)
    crop = np_image[y0:y1, x0:x1]
    crop = Image.fromarray(crop, 'RGB')
    crops.append(crop)

## resize crops to W x H
newsize = (W, H)
crops = [crop.resize(newsize) for crop in crops]
```

In [82]:

```
for k in [1,2, len(crops)-1]:
    if k > 2:
        print('...')
    detect(crops[k])
```

...

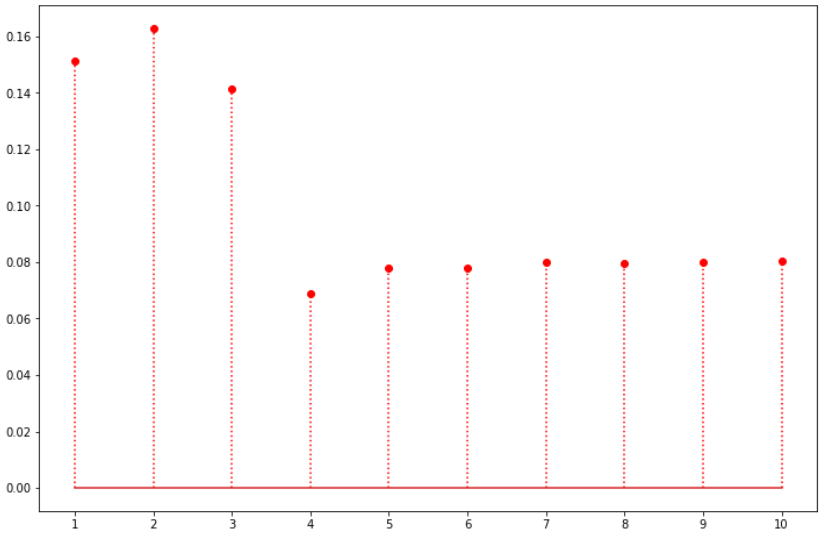
In [94]:

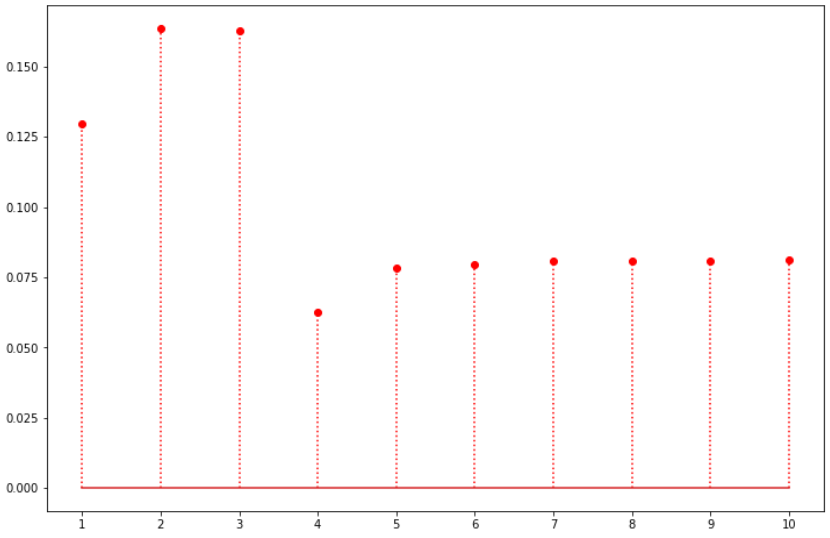
```
for k in range(len(crops)):
    preds = detect(crops[k], img_name=str(k+1),
                   show=False, save=True,
                   save_dir='/home/marko/data/video/frames-detections-original/')
```

In [88]:

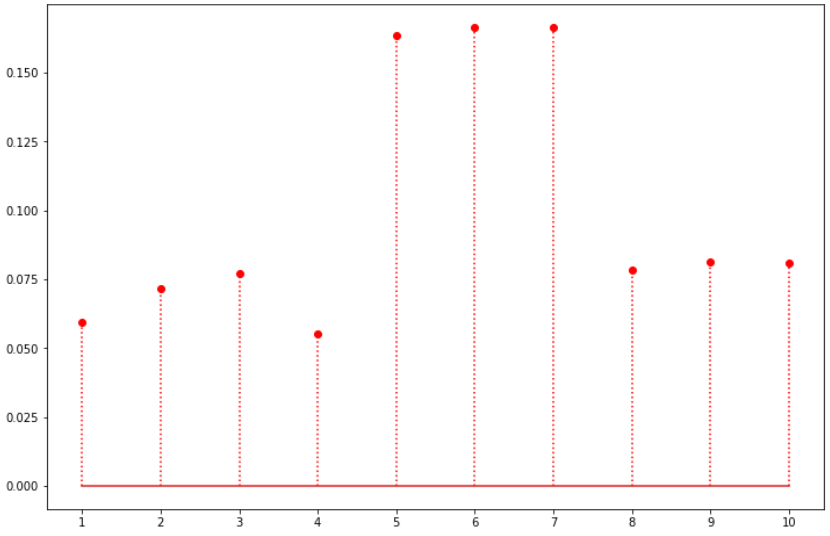
```
for k in [1,2, len(crops)-1]:
    if k > 2:
        print('...')

    d = detect(crops[k], show=False)
    d['confs'][np.where(d['classes'])] += 1

    rks = np.arange(1, 11)
    yks = d['confs'] / sum(d['confs']) # normalize
    markerline, stemlines, baseline = plt.stem(
```

```
        rks, yks, markerfmt='ro')

    plt.setp(stemlines, 'color', plt.getp(markerline,'color'))
    plt.setp(stemlines, 'linestyle', 'dotted')
    plt.xticks(rks, [str(rk) for rk in rks])
    plt.show()
```





. . .



Extension

In the figure below are 3 consecutive frames from the testing video: https://youtu.be/M0r1JTfh9fE where I run the model on all frames of Pedestrian Challenge video from https://motchallenge.net/vis/MOT17-13.

Say. we are detecting pedestrians at time t represented by the first image in the figure below. At time t, because of occlusions, brightness variations or for some other reasons, the model can fail to detect a pedestrian. But perhaps the model detected the same person in one or more of the previous frames. The idea is to use this information which has accumulated over time (at frames $t-1, t-2$, ...) in order to achieve better detection at time t.

My proposed solution is a bit similar to moving-average idea. For a chosen number of time steps, for simplicity say 2 we do the following. We first solve the classic matching problem of computer vision and match region proposals at time t to the region proposals at previous 2 time frames. This way, we know which detecting bounding boxes should

represent the same person trough time. Second, we take a mean of corresponding scores (estimated likelihoods that the region contains a person). And if this mean is greater that some threshold, we show the bounding box at time t.

I tried different variations of this idea with different number of time steps with interesting results. In the figure below, we can see the car approaching a pedestrian on the left. Here I colored the proposed regions with red for current frame, with green the previous frame and blue the previos frame at time $t-1$ for each time t. For example, we see that at time $t-1$ only the region proposed at time $t-1$ was detected as pedestrian. By thresholding the mean of all scores, we can detect the pedestrians we otherwise wouldn't. And vice versa, if there was, say a bicycle in one region proposal that received a high score, but it didn't receive a high score for previous 2 times, the mean would be smaller that threshould and we wouldn't falsely detect it as a pedestrian at time t.

For details, see the scripts: `bus-video-pred.py`, `bus-video-dets.py` and `bus-video-dets-ma3.py` in `../scripts/`.

In [14]:
```
Image.open('../figures/extension.png')
```

Out[14]:

Frame at time t. Frame at time $t-1$. Frame at time $t-2$.

In [ ]: