В данном ноутбуке подготовим модели для детектирования и сегментации людей на картинках для дальнейшего использования в веб демо детектора. За основу возьмём ноутбук с туториалом с pytorch.

```
%%shell

pip install cython
pip install -U 'git+https://github.com/cocodataset/cocoapi.git#subdirectory=PythonAPI'
```

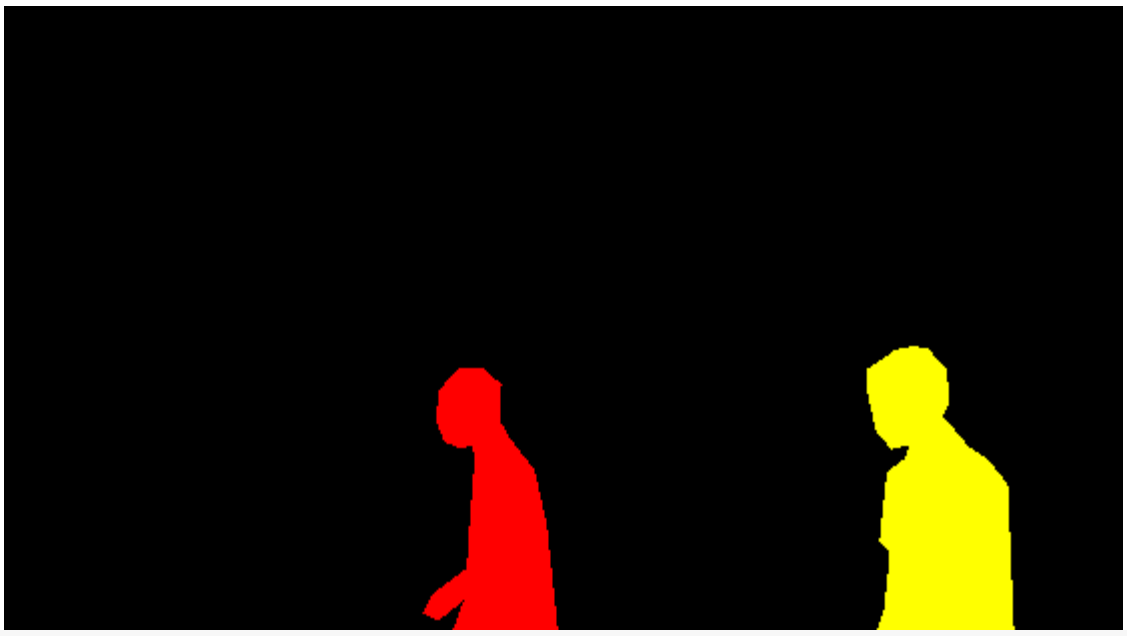
```
! wget https://www.cis.upenn.edu/~jshi/ped_html/PennFudanPed.zip
```

```
--2020-06-24 19:09:47--  https://www.cis.upenn.edu/~jshi/ped_html/PennFudanPed.zip
Resolving www.cis.upenn.edu (www.cis.upenn.edu)... 158.130.69.163, 2607:f470:8:64:5ea5::d
Connecting to www.cis.upenn.edu (www.cis.upenn.edu)|158.130.69.163|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 53723336 (51M) [application/zip]
Saving to: ‘PennFudanPed.zip’

PennFudanPed.zip    100%[===================>]  51.23M  1009KB/s    in 48s

2020-06-24 19:10:36 (1.06 MB/s) - ‘PennFudanPed.zip’ saved [53723336/53723336]
```

```
import zipfile
with zipfile.ZipFile('PennFudanPed.zip', 'r') as zip_ref:
    zip_ref.extractall()
```

```
from PIL import Image
Image.open('PennFudanPed/PNGImages/FudanPed00001.png')
```

```
mask = Image.open('PennFudanPed/PedMasks/FudanPed00001_mask.png')

mask
```



```
import os
import numpy as np
import torch
import torch.utils.data
from PIL import Image


class PennFudanDataset(torch.utils.data.Dataset):
    def __init__(self, root, transforms=None):
        self.root = root
        self.transforms = transforms
        # Загрузим данные и упорядочим их
        self.imgs = list(sorted(os.listdir(os.path.join(root, "PNGImages"))))
        self.masks = list(sorted(os.listdir(os.path.join(root, "PedMasks"))))

    def __getitem__(self, idx):
        img_path = os.path.join(self.root, "PNGImages", self.imgs[idx])
        mask_path = os.path.join(self.root, "PedMasks", self.masks[idx])
        img = Image.open(img_path).convert("RGB")
        mask = Image.open(mask_path)

        mask = np.array(mask)
        # каждый объект на картинке маркеруется своим цветом
        obj_ids = np.unique(mask)
        # уберём фон из списко классов сегментации
        obj_ids = obj_ids[1:]

        masks = mask == obj_ids[:, None, None]

        #Кординаты боксов для каждой маски
        num_objs = len(obj_ids)
        boxes = []
        for i in range(num_objs):
            pos = np.where(masks[i])
            xmin = np.min(pos[1])
            xmax = np.max(pos[1])
            ymin = np.min(pos[0])
            ymax = np.max(pos[0])
            boxes.append([xmin, ymin, xmax, ymax])

        boxes = torch.as_tensor(boxes, dtype=torch.float32)
        labels = torch.ones((num_objs,), dtype=torch.int64)
        masks = torch.as_tensor(masks, dtype=torch.uint8)

        image_id = torch.tensor([idx])
        area = (boxes[:, 3] - boxes[:, 1]) * (boxes[:, 2] - boxes[:, 0])
        iscrowd = torch.zeros((num_objs,), dtype=torch.int64)

        target = {}
        target["boxes"] = boxes
        target["labels"] = labels
        target["masks"] = masks
        target["image_id"] = image_id
        target["area"] = area
        target["iscrowd"] = iscrowd

        if self.transforms is not None:
            img, target = self.transforms(img, target)

        return img, target

    def __len__(self):
```

```
        return len(self.imgs)
```

```
dataset = PennFudanDataset('PennFudanPed/')
dataset[0]
```

```
(<PIL.Image.Image image mode=RGB size=559x536 at 0x7FEBD86452E8>,
 {'area': tensor([35358., 36225.]), 'boxes': tensor([[159., 181., 301., 430.],
         [419., 170., 534., 485.]]), 'image_id': tensor([0]), 'iscrowd': tensor([0, 0]), 'labels': tensor([1, 1]), 'masks': te
          [0, 0, 0,  ..., 0, 0, 0],
          [0, 0, 0,  ..., 0, 0, 0],
          ...,
          [0, 0, 0,  ..., 0, 0, 0],
          [0, 0, 0,  ..., 0, 0, 0],
          [0, 0, 0,  ..., 0, 0, 0]],

         [[0, 0, 0,  ..., 0, 0, 0],
          [0, 0, 0,  ..., 0, 0, 0],
          [0, 0, 0,  ..., 0, 0, 0],
          ...,
          [0, 0, 0,  ..., 0, 0, 0],
          [0, 0, 0,  ..., 0, 0, 0],
          [0, 0, 0,  ..., 0, 0, 0]]], dtype=torch.uint8)})
```

```
%%shell

# Скачаем репозиторий torchvision
# для использования некоторых методов, отсутствующих в сборке через pip
# references/detection
git clone https://github.com/pytorch/vision.git
cd vision
git checkout v0.3.0

cp references/detection/utils.py ../
cp references/detection/transforms.py ../
cp references/detection/coco_eval.py ../
cp references/detection/engine.py ../
cp references/detection/coco_utils.py ../
```

```
fatal: destination path 'vision' already exists and is not an empty directory.
HEAD is now at be37608 version check against PyTorch's CUDA version
```

```
from engine import train_one_epoch, evaluate
import utils as utils
import transforms as T
import engine

import torchvision
import torch
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
```

```
import torchvision
from torchvision.models.detection import FasterRCNN
from torchvision.models.detection.rpn import AnchorGenerator

#Загрузим обученный классификатор
backbone = torchvision.models.mobilenet_v2(pretrained=True).features
backbone.out_channels = 1280

# Создадим генератор боксов с различными размерами,
# так как люди могут встретиться не картинках в различном масштабе
anchor_generator = AnchorGenerator(sizes=((32, 64, 128, 256, 512),),
                                   aspect_ratios=((0.5, 1.0, 2.0),))

roi_pooler = torchvision.ops.MultiScaleRoIAlign(featmap_names=[0],
                                                output_size=7,
                                                sampling_ratio=2)

# Сконструируем из объявленных составляющих FasterRCNN
model = FasterRCNN(backbone,
                   num_classes=2,
                   rpn_anchor_generator=anchor_generator,
                   box_roi_pool=roi_pooler)
```

```
# дополним модель сегментацией дя построения MaskRCNN
import torchvision
```

```
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.models.detection.mask_rcnn import MaskRCNNPredictor


def get_model_instance_segmentation(num_classes):
    # Загрузим модель сегментации
    model = torchvision.models.detection.maskrcnn_resnet50_fpn(pretrained=True)

    # Количество входов нв классификатор
    in_features = model.roi_heads.box_predictor.cls_score.in_features
    # replace the pre-trained head with a new one
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

    # now get the number of input features for the mask classifier
    in_features_mask = model.roi_heads.mask_predictor.conv5_mask.in_channels
    hidden_layer = 256
    # and replace the mask predictor with a new one
    model.roi_heads.mask_predictor = MaskRCNNPredictor(in_features_mask,
                                                       hidden_layer,
                                                       num_classes)

    return model
```

```
def get_transform(train):
    transforms = []
    transforms.append(T.ToTensor())
    if train:
        transforms.append(T.RandomHorizontalFlip(0.5))
    return T.Compose(transforms)
```

```
dataset = PennFudanDataset('PennFudanPed', get_transform(train=True))
dataset_test = PennFudanDataset('PennFudanPed', get_transform(train=False))

# Разделим данные на тренировку и тест
torch.manual_seed(1)
indices = torch.randperm(len(dataset)).tolist()
dataset = torch.utils.data.Subset(dataset, indices[:-50])
dataset_test = torch.utils.data.Subset(dataset_test, indices[-50:])

# Определим генераторы соответствующих картинок
data_loader = torch.utils.data.DataLoader(
    dataset, batch_size=2, shuffle=True, num_workers=4,
    collate_fn=utils.collate_fn)

data_loader_test = torch.utils.data.DataLoader(
    dataset_test, batch_size=1, shuffle=False, num_workers=4,
    collate_fn=utils.collate_fn)

data_loader_end = torch.utils.data.DataLoader(
    dataset_test, batch_size=100, shuffle=False, num_workers=4,
    collate_fn=utils.collate_fn)
```

```
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
```

```
# зададим количество классов
num_classes = 2 # люди и фон


model = get_model_instance_segmentation(num_classes)
model.to(device)


params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=0.005,
                            momentum=0.9, weight_decay=0.0005)

lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                               step_size=3,
                                               gamma=0.1)
```

```
# тренировка модели
num_epochs = 50

for epoch in range(num_epochs):
    train_one_epoch(model, optimizer, data_loader, device, epoch, print_freq=10)
```

```
    lr_scheduler.step()
    evaluate(model, data_loader_test, device=device)
```

```
torch.save(model, 'drive/My Drive/Colab Notebooks/weights/maskRCNN2')
```

```
# import torch
# model = torch.load('drive/My Drive/Colab Notebooks/weights/maskRCNN.pt')
# model.eval()
```

```
import cv2
import matplotlib.pyplot as plt
```

далее напишем функции для создания итоговых изображений. (отрисовка коробок детекции на картинках, и объединение всех масок людей на картинке в одну с наложением на картинку для сегментации).

```
def plot_preds(img, preds):
  numpy_img = (np.rollaxis(img.numpy(), 0, 3)* 255).astype(np.uint8)
  boxes = preds['boxes'].detach().cpu().numpy()
  for box in boxes:
    numpy_img = cv2.rectangle(
        numpy_img,
        (box[0],box[1]),
        (box[2],box[3]),
        255,
        3,
    )
  return numpy_img.get()
```

```
def plot_masks(img, preds):
  numpy_img = (np.rollaxis(img.numpy(), 0, 3)* 255).astype(np.uint8)
  # numpy_img[numpy_img>0] = 254
  masks = preds['masks'].detach().cpu().numpy()
  for mask in masks:
     mask = (np.rollaxis(mask, 0, 3)* 255).astype(np.float32)
     numpy_img = numpy_img - mask
     numpy_img[numpy_img>255] = 255
     numpy_img[numpy_img<0] = 0
  return numpy_img
```

Проверим работу модели на серии тестовых картинок:

```
model = model.to(device)
```

```
img1,_ = next(iter(data_loader_end))
X = [x.to(device) for x in list(img1[:3])]

model = model.eval()
with torch.no_grad():
    predictions = model(X)
```

```
/usr/local/lib/python3.6/dist-packages/torch/nn/functional.py:2854: UserWarning: The default behavior for interpolate/upsample w
  warnings.warn("The default behavior for interpolate/upsample with float scale_factor will change "
```

```
X[0].shape
```

```
torch.Size([3, 349, 292])
```

```
from PIL import Image, ImageDraw
```

```
image_with_boxes_list = []
image_with_masks_list = []

for i in range(len(predictions)):
```

```
  CONF_THRESH = 0.7
  boxes = predictions[i]['boxes'][predictions[i]['scores'] > CONF_THRESH]
  masks = predictions[i]['masks'][predictions[i]['scores'] > CONF_THRESH]
  boxes_dict = {}
  masks_dict = {}
  boxes_dict['boxes'] = boxes
  masks_dict['masks'] = masks
  img_with_boxes = plot_preds(img1[i], boxes_dict)
  img_with_masks = plot_masks(img1[i], masks_dict)
  print(img_with_boxes.shape)
  image_with_boxes_list.append(img_with_boxes)
  image_with_masks_list.append(img_with_masks)
```

```
(349, 292, 3)
(376, 508, 3)
(348, 473, 3)
```

```
img.shape
```

```
torch.Size([3, 349, 292])
```

```
import gc
torch.cuda.empty_cache()
gc.collect()
```

```
8957
```

```
import matplotlib.pyplot as plt
from IPython.display import clear_output

m = 0
plt.figure(figsize=(18, 6))
for i in range(2):
    plt.subplot(2, 2, i+1)
    plt.axis("off")
    plt.title('detection')
    plt.imshow(image_with_boxes_list[i+m])

    plt.subplot(2, 2, i+3)
    plt.axis("off")
    plt.title('segmentation')
    plt.imshow(image_with_masks_list[i+m])
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).
```


```
predictions[0]
```

```
{'boxes': tensor([[ 63.7770,  43.8986, 195.5872, 327.0247],
        [276.0391,  22.7177, 290.9091,  73.2007]], device='cuda:0'),
 'labels': tensor([1, 1], device='cuda:0'),
 'masks': tensor([[[[0., 0., 0.,  ..., 0., 0., 0.],
          [0., 0., 0.,  ..., 0., 0., 0.],
          [0., 0., 0.,  ..., 0., 0., 0.],
          ...,
          [0., 0., 0.,  ..., 0., 0., 0.],
          [0., 0., 0.,  ..., 0., 0., 0.],
          [0., 0., 0.,  ..., 0., 0., 0.]]],
```

Модель готова для использования в веб демо детектора.

```
          ...,
          [0., 0., 0.,  ..., 0., 0., 0.],
          [0., 0., 0.,  ..., 0., 0., 0.],
          [0., 0., 0.,  ..., 0., 0., 0.]]]], device='cuda:0'),
 'scores': tensor([0.9994, 0.7440], device='cuda:0')}
```