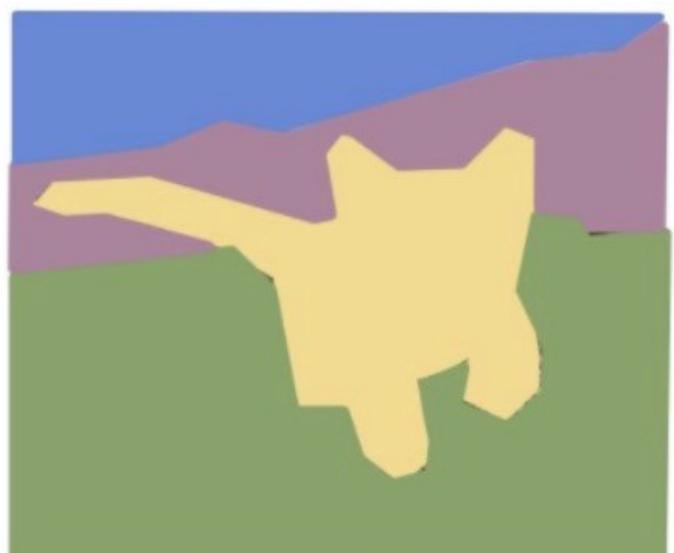


Семантическая сегментация

Семантическая сегментация — это задача анализа изображений, в которой мы классифицируем каждый пиксель изображения в класс.

На картинке ниже пример сегментации фото с котенком. Модель сегментации получает на вход изображение, а на выходе дает сегментационную карту, где каждому пикселию исходного изображения соответствует его класс.



In []:

```
1 !pip install albumentations
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://us-python.pkg.dev/colab-wheels/public/simple/ (https://us-python.pkg.dev/colab-wheels/public/simple/)

Requirement already satisfied: albumentations in /usr/local/lib/python3.8/dist-packages (1.2.1)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.8/dist-packages (from albumentations) (6.0)
Requirement already satisfied: numpy>=1.11.1 in /usr/local/lib/python3.8/dist-packages (from albumentations) (1.21.6)
Requirement already satisfied: opencv-python-headless>=4.1.1 in /usr/local/lib/python3.8/dist-packages (from albumentations) (4.6.0.66)
Requirement already satisfied: quidida>=0.0.4 in /usr/local/lib/python3.8/dist-packages (from albumentations) (0.0.4)
Requirement already satisfied: scikit-image>=0.16.1 in /usr/local/lib/python3.8/dist-packages (from albumentations) (0.18.3)
Requirement already satisfied: scipy in /usr/local/lib/python3.8/dist-packages (from albumentations) (1.7.3)
Requirement already satisfied: scikit-learn>=0.19.1 in /usr/local/lib/python3.8/dist-packages (from quidida>=0.0.4->albumentations) (1.0.2)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.8/dist-packages (from quidida>=0.0.4->albumentations) (4.1.1)
Requirement already satisfied: PyWavelets>=1.1.1 in /usr/local/lib/python3.8/dist-packages (from scikit-image>=0.16.1->albumentations) (1.3.0)
Requirement already satisfied: networkx>=2.0 in /usr/local/lib/python3.8/dist-packages (from scikit-image>=0.16.1->albumentations) (2.6.3)
Requirement already satisfied: pillow!=7.1.0,!7.1.1,>=4.3.0 in /usr/local/lib/python3.8/dist-packages (from scikit-image>=0.16.1->albumentations) (7.1.2)
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.0 in /usr/local/lib/python3.8/dist-packages (from scikit-image>=0.16.1->albumentations) (3.2.2)
Requirement already satisfied: tifffile>=2019.7.26 in /usr/local/lib/python3.8/dist-packages (from scikit-image>=0.16.1->albumentations) (2021.11.2)
Requirement already satisfied: imageio>=2.3.0 in /usr/local/lib/python3.8/dist-packages (from scikit-image>=0.16.1->albumentations) (2.9.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib!=3.0.0,>=2.0.0->scikit-image>=0.16.1->albumentations) (2.8.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-packages (from matplotlib!=3.0.0,>=2.0.0->scikit-image>=0.16.1->albumentations) (0.11.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib!=3.0.0,>=2.0.0->scikit-image>=0.16.1->albumentations) (1.4.4)
Requirement already satisfied: pyparsing!=2.0.4,!2.1.2,!2.1.6,>=2.0.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib!=3.0.0,>=2.0.0->scikit-image>=0.16.1->albumentations) (3.0.9)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (from python-dateutil>=2.1->matplotlib!=3.0.0,>=2.0.0->scikit-image>=0.16.1->albumentations) (1.15.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.8/dist-packages (from scikit-learn>=0.19.1->quidida>=0.0.4->albumen
```

```
tations) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.8/dist-packages (from scikit-learn>=0.19.1->quidida>=0.0.4->albumentations) (3.1.0)
```

In []:

```
1 # Утилиты стандартной библиотеки
2 import copy
3 from functools import lru_cache
4 import os
5 import random
6 import requests
7 import subprocess
8 import shutil
9 from tqdm import tqdm
10 from typing import Any, Callable, Dict, Iterable, List, Tuple
11 from urllib.request import urlretrieve
12
13 # Библиотеки для анализа данных
14 import numpy as np
15 import pandas as pd
16 import sklearn.preprocessing
17 from tqdm.auto import tqdm
18
19 # Библиотеки для обработки изображений
20 import cv2
21 from PIL import Image, ImageDraw, ImageEnhance
22 import torch
23 from torch import nn
24 import torchvision
25 from torch.utils.data import Dataset, DataLoader
26 from torchvision import models
27 import torchvision.transforms as transforms
28
29 # Библиотеки для визуализации
30 import ipywidgets as widgets
31 import IPython
32 import matplotlib
33 import matplotlib.pyplot as plt
34 import plotly
35 import plotly.express as px
36 import seaborn as sns
37 from torchsummary import summary
38
39 import albumentations as A
40 import albumentations.augmentations.functional as F
41 from albumentations.pytorch.transforms import ToTensorV2
```

In []:

```
1 device = "cuda" if torch.cuda.is_available() else "cpu"
2 print(device)
```

cuda

1. Upsampling

Большинство моделей семантической сегментации построено по принципу кодировщик-декодировщик. Причем кодировщик уменьшает пространственную размерность изображения за счет сверток с шагом больше 1 или пулингов. Для того, чтобы получить сегментационную карту такого же размера, что исходное изображение, необходимо преобразование *upsampling*.

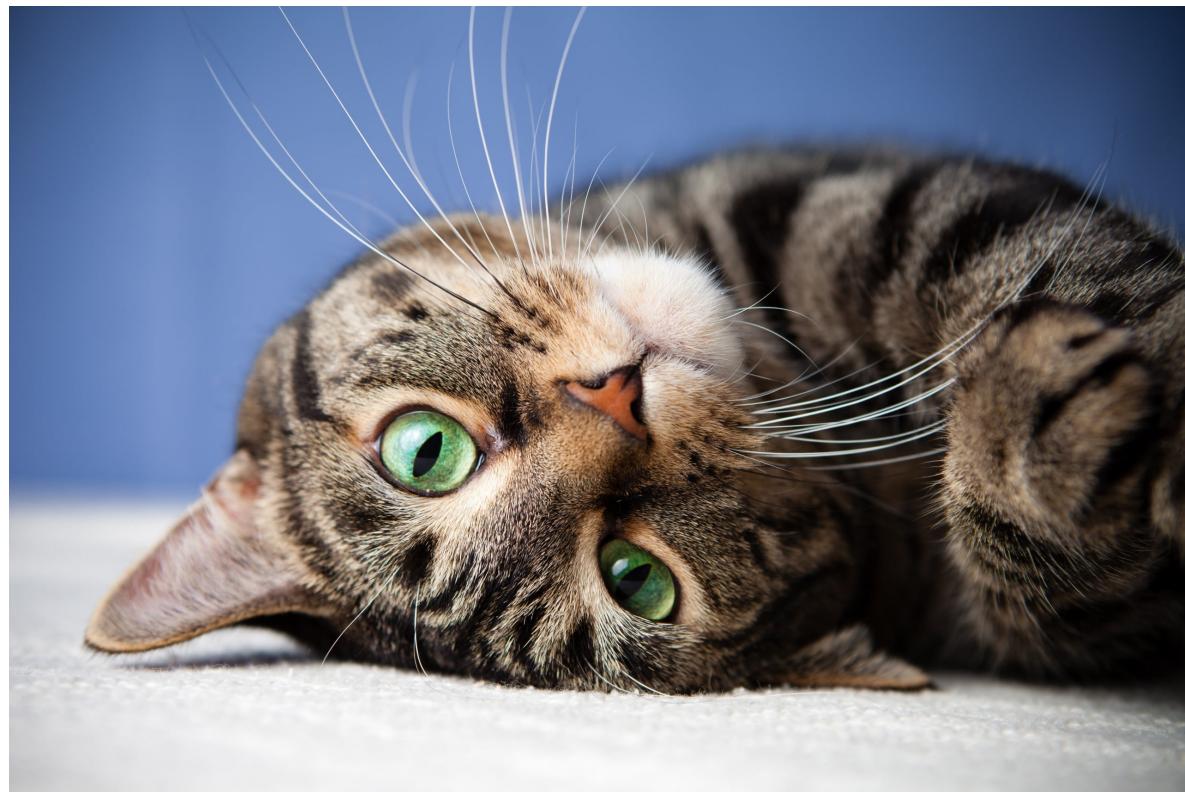
Откроем для себя еще одну библиотеку для работы с изображениями — Pillow. Для справки можете обратиться к [титориалу](https://pillow.readthedocs.io/en/stable/handbook/tutorial.html) (<https://pillow.readthedocs.io/en/stable/handbook/tutorial.html>). В качестве примера снова возьмем котика.

In []:

```
1 image = Image.open("cat.jpg")
2 print('Image size', image.size)
3 image
```

Image size (2560, 1707)

Out[168]:



Изображение получилось очень большим. Уменьшим его. Причем сделаем 2 варианта, один из которых в 2 раза меньше другого.

In []:

```
1 image1 = image.resize((50, 34))
2 image2 = image.resize((100, 68))
3 print('Размер первого изображения', image1.size)
4 print('Размер второго изображения', image2.size)
```

Размер первого изображения (50, 34)

Размер второго изображения (100, 68)

Сравним полученные изображения.

In []:

```
1 plt.figure(figsize=(10, 3))
2 plt.subplot(1, 2, 1)
3 plt.imshow(image1)
4 plt.axis('off')
5 plt.subplot(1, 2, 2)
6 plt.imshow(image2)
7 plt.axis('off')
```

Out[170]:

(-0.5, 99.5, 67.5, -0.5)



Далее мы будем проверять на примере этих двух изображений, как работают различные виды Upsampling. Точнее мы будем увеличивать первое изображение и сравнивать результат со вторым изображением.

Для начала приведем изображения к NumPy массивам.

In []:

```
1 image1 = np.array(image1)
2 image2 = np.array(image2)
```

Определим функцию для сравнения изображений. Будем считать RSMSE и MAE для увеличенного первого изображения и второго изображения.

In []:

```
1 def compare_images(img1, img2):
2     rmse = np.sqrt(((img1 - img2) ** 2).mean())
3     mae = (np.abs(img1 - img2)).mean()
4
5     fig = plt.figure(figsize=(10, 3))
6     plt.subplot(1, 2, 1)
7     plt.imshow(img1)
8     plt.axis('off')
9     plt.title(f'Изобр. 1, размер {img1.shape[0]} x {img1.shape[1]}')
10    plt.subplot(1, 2, 2)
11    plt.imshow(img2)
12    plt.axis('off')
13    plt.title(f'Изобр. 2, размер {img2.shape[0]} x {img2.shape[1]}')
14    fig.suptitle(f'RMSE={rmse:.1f}, MAE={mae:.1f}', y=1.1)
15    plt.show()
```

Мы будем пользоваться `upsampling` модулями в `torch.nn`. Для этого нужно преобразовать

изображения в батчи.

In []:

```
1 batch1 = torch.from_numpy(image1).moveaxis(2, 0).unsqueeze(0).to(torch.float32)
2 batch2 = torch.from_numpy(image2).moveaxis(2, 0).unsqueeze(0).to(torch.float32)
3 batch1.shape, batch2.shape
```

Out[173]:

```
(torch.Size([1, 3, 34, 50]), torch.Size([1, 3, 68, 100]))
```

Интерполяция по методу ближайших соседей

In []:

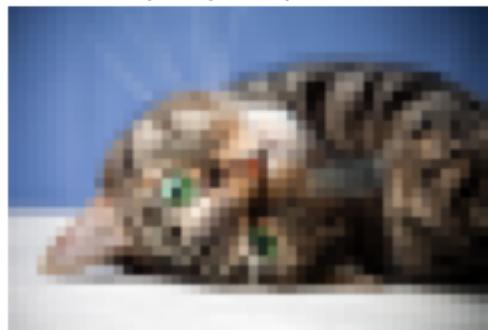
```
1 nearest_neighbour = nn.UpsamplingNearest2d(scale_factor=2)
2 image1_nn = nearest_neighbour(batch1).squeeze().moveaxis(0, 2).numpy().astype(i
```

In []:

```
1 compare_images(image1_nn, image2)
```

RMSE=16.4, MAE=9.4

Изобр. 1, размер 68 x 100



Изобр. 2, размер 68 x 100



Первое изображение визуально не поменялось, так как приведение изображений к большему / меньшему размеру в matplotlib работает как интерполяция по методу ближайших соседей.

Билинейная интерполяция

In []:

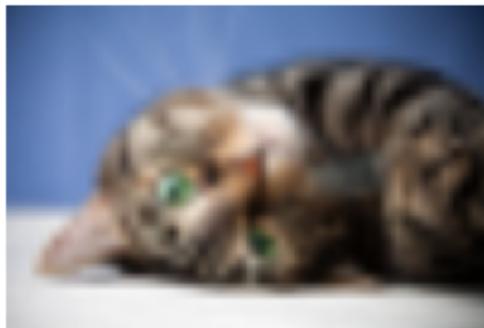
```
1 bilinear = nn.UpsamplingBilinear2d(scale_factor=2)
2 image1_bl = bilinear(batch1).squeeze().moveaxis(0, 2).numpy().astype(int)
```

In []:

```
1 compare_images(image1_bl, image2)
```

RMSE=15.9, MAE=9.4

Изобр. 1, размер 68 x 100



Изобр. 2, размер 68 x 100



Изображение получилось размытым, так как билинейная интерполяция складывает ближайшие значения с пропорциональными кооффициентами. При этом по метрикам, качество приближения улучшилось по сравнению с предыдущим методом.

MaxUnPooling

In []:

```
1 maxpool = nn.MaxPool2d(kernel_size=2, return_indices=True)
2 maxunpool = nn.MaxUnpool2d(kernel_size=2)
3 _, indices = maxpool(batch2)
4 image1_mup = maxunpool(batch1, indices).squeeze().moveaxis(0, 2).numpy().astype
```

In []:

```
1 compare_images(image1_mup, image2)
```

RMSE=118.9, MAE=91.3

Изобр. 1, размер 68 x 100



Изобр. 2, размер 68 x 100



Получились черные области, так как все кроме максимум заполняется нулями. Естественно, качество приближения такой модели хуже, чем у предыдущих. Однако данный вид upsampling может быть эффективен в нейронной сети в комбинации с другими сверточными слоями.

Обратная свертка

Проверим работу свертки со случайной инициализацией.

In []:

```
1 convt = nn.ConvTranspose2d(3, 3, kernel_size=7, padding=3, stride=2, output_padding=1)
2 image1_ct = convt(batch1).squeeze().moveaxis(0, 2).detach().numpy().astype(int)
3 image1_ct.shape
```

Out[157]:

(68, 100, 3)

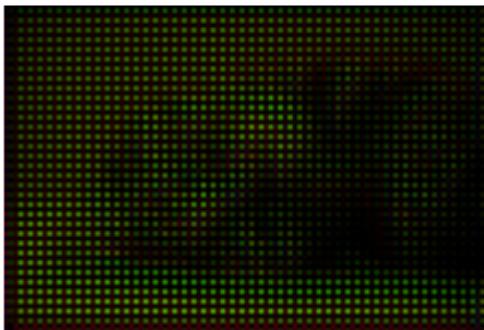
In []:

```
1 compare_images(image1_ct, image2)
```

WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

RMSE=165.4, MAE=136.0

Изобр. 1, размер 68 x 100



Изобр. 2, размер 68 x 100



Теперь обучим свертку на нашей картинке.

In []:

```
1 optimizer = torch.optim.Adam(convt.parameters(), lr=0.1)
2 criterion = nn.MSELoss()
3 for i in range(300):
4     pred = convt(batch1)
5     loss = criterion(pred, batch2)
6     loss.backward()
7     optimizer.step()
8     optimizer.zero_grad()
9     if (i + 1) % 20 == 0:
10         print(f'Iter {i}, MSE {loss.item():.2f}')
```

```
Iter 19, MSE 7858.85
Iter 39, MSE 1020.52
Iter 59, MSE 419.36
Iter 79, MSE 286.67
Iter 99, MSE 257.45
Iter 119, MSE 243.18
Iter 139, MSE 232.99
Iter 159, MSE 224.83
Iter 179, MSE 217.96
Iter 199, MSE 212.10
Iter 219, MSE 207.08
Iter 239, MSE 202.77
Iter 259, MSE 199.08
Iter 279, MSE 195.92
Iter 299, MSE 193.20
```

In []:

```
1 image1_ct = convt(batch1).squeeze().moveaxis(0, 2).detach().numpy().astype(int)
```

In []:

```
1 compare_images(image1_ct, image2)
```

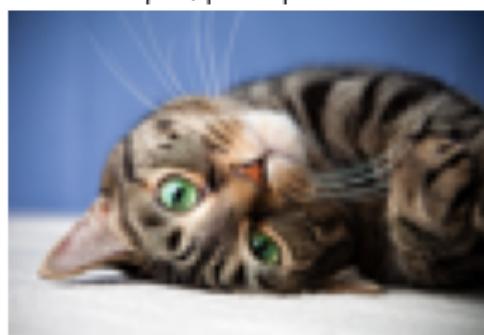
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

RMSE=13.9, MAE=9.0

Изобр. 1, размер 68 x 100



Изобр. 2, размер 68 x 100



В итоге у свертки метрики получились даже лучше чем у интерполяции.

2. Применение обученных моделей сегментации

Большая часть моделей из этого ноутбука обучена на датасете [COCO — Common Objects In Context](http://cocodataset.org/) (<http://cocodataset.org/>), а если быть точным, то на COCO 2017. Он содержит 123 287 изображений, на которых изображены 886 284 объектов, посмотреть на которые можно [по ссылке](http://cocodataset.org/#explore) (<http://cocodataset.org/#explore>). Основные достоинства этого датасета хорошо проиллюстрированы на главной странице проекта.

В целом, у COCO прекрасный сайт, что во многом объясняет его популярность. При возникновении вопросов смело обращайтесь к официальной документации, там скорее всего есть ответ.



Конкретно сегментационные модели обучены на подмножестве из 21 категории. Они перечислены в ячейке ниже.

In []:

```
1 coco_categories_segmentation = np.array([
2     '__background__',
3     'aeroplane',
4     'bicycle',
5     'bird',
6     'boat',
7     'bottle',
8     'bus',
9     'car',
10    'cat',
11    'chair',
12    'cow',
13    'diningtable',
14    'dog',
15    'horse',
16    'motorbike',
17    'person',
18    'pottedplant',
19    'sheep',
20    'sofa',
21    'train',
22    'tvmonitor'
23 ])
```

Рассмотрим предобученные модели из [Pytorch Hub \(<https://pytorch.org/hub/>\)](https://pytorch.org/hub/) — централизованного (но пока слишком сырого) репозитория с единообразным интерфейсом всех моделей. К сожалению, там есть далеко не всё, так что впоследствии придётся использовать нестандартные библиотеки. Это частая ситуация в машинном обучении: нужный функционал может быть реализован только в частном репозитории научной группы, на которую дана ссылка в статье. Уметь запускать эти репозитории — отдельное искусство, которому жизнь вас научит самостоятельно.

Также реализуем интерфейс для удобной работы с Pytorch Hub.

Добьёмся того, чтобы объекту можно было скормить картинку и получить тензор вероятностей классов для каждого пикселя. Для иллюстрации этого более чем достаточно. При желании этот интерфейс легко видоизменить таким образом, чтобы на вход метода `forward` подавала минибатч картинок. Тогда модель можно будет дообучать (в рамках этого занятия тонкости обучения и дообучения моделей не рассматриваются, т.к. занятие несёт ознакомительный характер).

In []:

```
1 class SemanticSegmentationTorchHubAPI:
2     """
3         Интерфейсный класс для моделей из Pytorch Hub, решающих
4         задачу семантической сегментации. Принимает на вход название предобученной
5         модели, бэкенд (CUDA или CPU) и изображение в формате PIL.Image. Возвращает
6         тензор вероятностей каждого из 21 классов для каждого пикселя изображения.
7         !!! Выходной тензор заранее отгружается на CPU и кастуется к np.ndarray !!!
8         """
9     def __init__(self, device: str = "cuda",
10                  torchhub_model: str = "deeplabv3_resnet101"):
11         self.device = device
12
13         # Модель из torch hub
14         self.model = torch.hub.load(
15             'pytorch/vision:v0.10.0', torchhub_model, pretrained=True).to(device)
16         self.model.eval()
17
18         # Трансформация картинки
19         self.transform = transforms.Compose([
20             # Приводим к тензору
21             transforms.ToTensor(),
22             # Нормализуем, в соответствии с нормализацией
23             # для предобученных моделей
24             transforms.Normalize(mean=[0.485, 0.456, 0.406],
25                                 std=[0.229, 0.224, 0.225]),
26         ])
27
28     def __call__(self, image: Image) -> np.ndarray:
29         return self.forward(image)
30
31     def detach(self) -> None:
32         """
33             Переводит модель на сри.
34             """
35         self.model = self.model.cpu()
36
37     def forward(self, image: Image) -> np.ndarray:
38         """
39             Получает предсказание модели.
40             """
41
42             # Предобрабатываем картинки так,
43             # чтобы на вход приходил тензорный батч
44             input_batch = self._preprocess_image(image)
45
46             # Получаем выходы модели
47             with torch.no_grad():
48                 model_outputs = self.model(input_batch)
49
50             # Модель дает на выходе тензора: out и aux
51             # out -- предсказания модели
52             # aux -- лосс, но для применения нам лосс не нужен
53
54             return model_outputs["out"][0].cpu()
55
56     def _preprocess_image(self, image) -> torch.Tensor:
57         """
58             Преобразует картинку и создает из нее батч.
59             """
60             return self.transform(image).unsqueeze(0).to(self.device)
```

Для начала будем использовать простую сегментационную модель [FCN](https://pytorch.org/hub/pytorch_vision_fcn_resnet101/) (https://pytorch.org/hub/pytorch_vision_fcn_resnet101/). Для того, чтобы заглянуть в код модели, нажмите на кнопку View on Github . Кроме того можно перейти на web-интерфейс модели в HuggingFace — кнопка Open Model Demo .

In []:

```
1 fcn = SemanticSegmentationTorchHubAPI(device=device, torchhub_model="fcn_resnet101")
```

Downloading: "https://github.com/pytorch/vision/zipball/v0.10.0" to /root/.cache/torch/hub/v0.10.0.zip
/usr/local/lib/python3.8/dist-packages/torchvision/models/_utils.py:20
8: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and will be removed in 0.15, please use 'weights' instead.
 warnings.warn(
/usr/local/lib/python3.8/dist-packages/torchvision/models/_utils.py:22
3: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and will be removed in 0.15. The current behavior is equivalent to passing `weights=FCN_ResNet101_Weights.COCO_WITH_VOC_LABELS_V1` . You can also use `weights=FCN_ResNet101_Weights.DEFAULT` to get the most up-to-date weights.
 warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/fcn_resnet101_coco-7ecb50ca.pth" to /root/.cache/torch/hub/checkpoints/fcn_resnet101_coco-7ecb50ca.pth
0%| | 0.00/208M [00:00<?, ?B/s]

ResNet-101 в названии FCN говорит о том, что в качестве Encoder'a выбрана ResNet-архитектура. Атрибут backbone — кодировщик, Feature Extractor модели. В данном случае он получен из ResNet-101.

In []:

```
1 fcn.model
```

Out[12]:

```
FCN(  
    (backbone): IntermediateLayerGetter(  
        (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)  
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        (relu): ReLU(inplace=True)  
        (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)  
        (layer1): Sequential(  
            (0): Bottleneck(  
                (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)  
                (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
                (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
                (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True).
```

Можно заметить, что здесь не модулей upsampling-а. Дело в том, что модель наследуется от `_SimpleSegmentationModel` ([код](#) (<https://github.com/pytorch/vision/blob/790f1cdcea0359619adfc9ec37b91883748d1854/torchvision/models/segme>), у которой upsampling реализован через `F.interpolate` ([код](#) (<https://github.com/pytorch/vision/blob/790f1cdcea0359619adfc9ec37b91883748d1854/torchvision/models/segme>)).

Рассмотрим работу модели на примере картинки с большим количеством объектов.

In []:

```
1 input_image = Image.open("big_bang.jpg")
2 input_image
```

Out[181]:



In []:

```
1 width, height = input_image.size
2 print(f"Ширина и высота картинки: ({width}, {height}) пикселей")
```

Ширина и высота картинки: (3000, 1994) пикселей

In []:

```
1 # Делаем предсказание
2 fcn_outputs = fcn(input_image)
3
4 # Избавляемся от градиентов и переводим модель на сри
5 fcn.detach()
```

Реализуем 2 способа приведения выходов модели к вероятностям: с помощью `softmax` и `minmax` масштабирования. `Minmax` обычно никто не пользуется, из-за того, что большое значение вероятности для данного класса может появиться из-за логитов для других классов, на которых модель дает большие

по модулю отрицательные значения. В softmax большее значение вероятности будет у данного класса только в том случае, когда у него будет достаточно больше значение логита. Однако нам minmax сможет лучше показать, какие объекты для данного класса модель точно определяет как объекты другого класса.

In []:

```
1 def outputs_to_probs(model_outputs, mode='softmax'):
2     """
3     Предсказания нужно отмасштабировать, чтобы получить ~ вероятности
4     """
5     assert mode in ('softmax', 'minmax')
6     if mode == "softmax":
7         category_probs = torch.softmax(model_outputs, dim=0).numpy()
8     elif mode == 'minmax':
9         category_preds = model_outputs.numpy()
10        category_probs = sklearn.preprocessing.MinMaxScaler()\
11            .fit_transform(np.ravel(category_preds)[:, None])\
12            .reshape(category_preds.shape)
13    else:
14        raise NotImplementedError("This mode is not implemented. Use one of ('soft'
15    return category_probs
```

Посмотрим на предсказания модели.

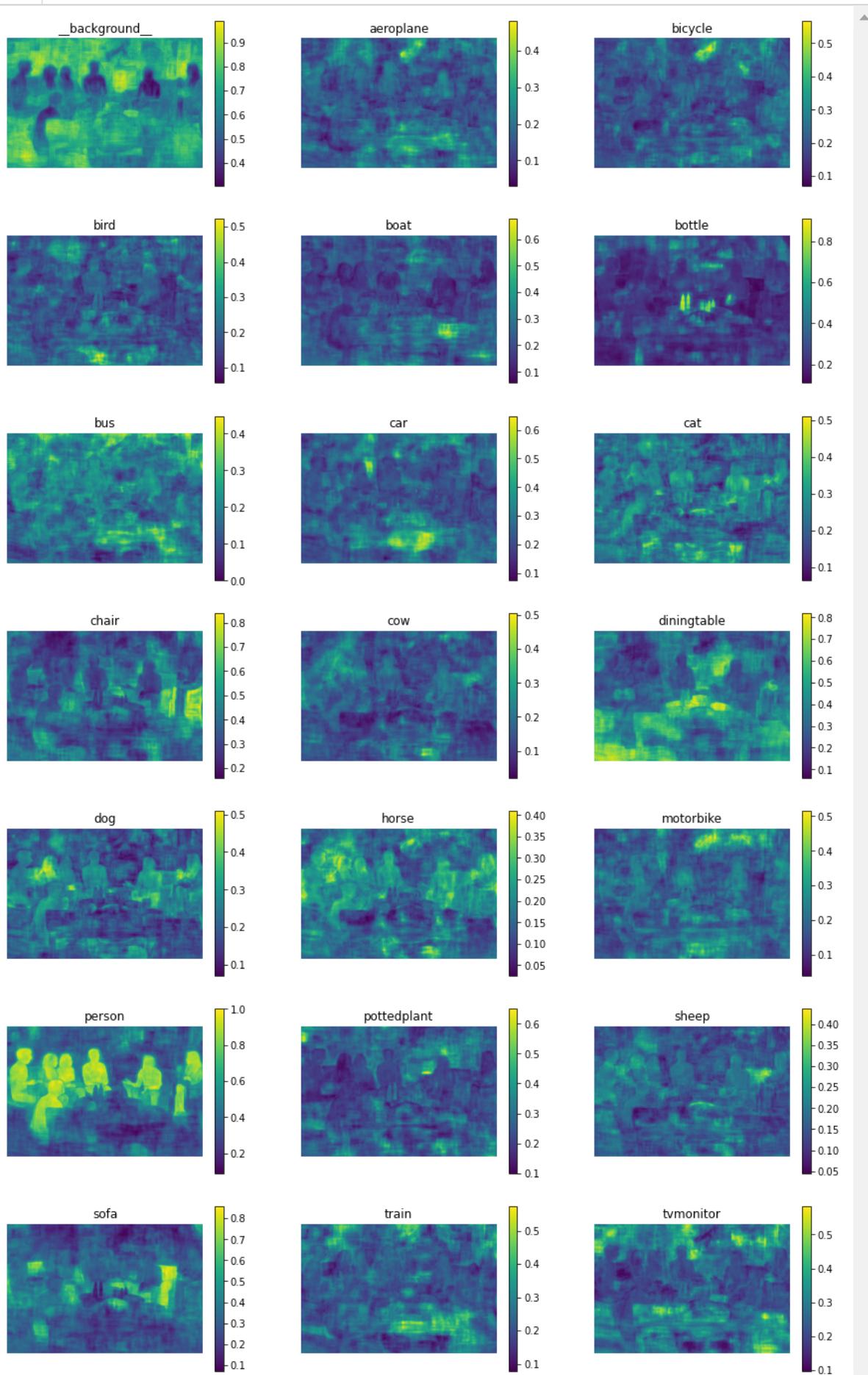
In []:

```
1 def plot_probs(probs):
2     plt.figure(figsize=(15, 25))
3     for i, name in enumerate(coco_categories_segmentation):
4         plt.subplot(7, 3, i + 1)
5         plt.imshow(probs[i])
6         plt.title(name)
7         plt.colorbar()
8         plt.axis('off')
9     plt.show()
```

С помощью minmax .

In []:

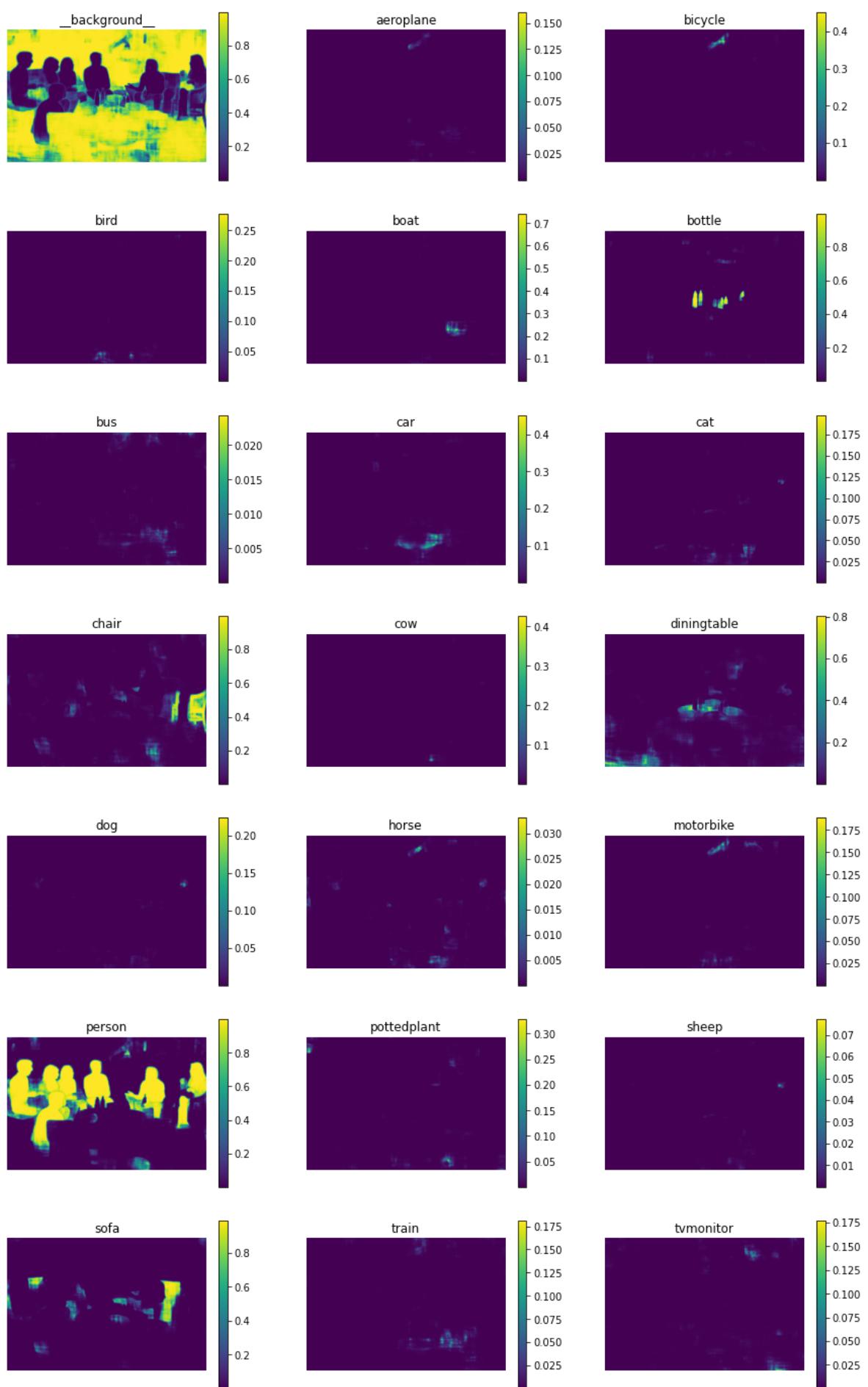
```
1 fcn_probs_mm = outputs_to_probs(fcn_outputs, mode='minmax')
2 plot_probs(fcn_probs_mm)
```



С помощью softmax.

In []:

```
1 fcn_probs_sm = outputs_to_probs(fcn_outputs, mode='softmax')
2 plot_probs(fcn_probs_sm)
```



Наибольшие значения предсказаний получились для классов:

`__background__ , bottle , chair , diningtable , person , sofa .`

Попробуем другую модель из torchhub — [deeplabv3](#) (https://pytorch.org/hub/pytorch_vision_deeplabv3_resnet101/). Все что было применимо для предыдущей модели, применимо и для этой.

In []:

```
1 deeplabv3 = SemanticSegmentationTorchHubAPI(device=device, torchhub_model="deeplabv3")
```

```
Using cache found in /root/.cache/torch/hub/pytorch_vision_v0.10.0
/usr/local/lib/python3.8/dist-packages/torchvision/models/_utils.py:20
8: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and will be removed in 0.15, please use 'weights' instead.
    warnings.warn(
/usr/local/lib/python3.8/dist-packages/torchvision/models/_utils.py:22
3: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and will be removed in 0.15. The current behavior is equivalent to passing `weights=DeepLabV3_ResNet101_Weights.COCO_WITH_VOC_LABELS_V1`. You can also use `weights=DeepLabV3_ResNet101_Weights.DEFAULT` to get the most up-to-date weights.
    warnings.warn(msg)
```

Посмотрим на устройство модели. Здесь можно увидеть dilated свертки, где выставлен параметр dilation .

In []:

```
1 deeplabv3.model
```

Out[193]:

```
DeepLabV3(
  (backbone): IntermediateLayerGetter(
    (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (layer1): Sequential(
      (0): Bottleneck(
        (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True).
```

In []:

```
1 deeplabv3_outputs = deeplabv3(input_image)
2 deeplabv3.detach()
```

In []:

```
1 deeplabv3_probs_sm = outputs_to_probs(deeplabv3_outputs, mode='softmax')
```

Определим несколько вспомогательных функций для отрисовки предсказанных масок объектов:

In []:

```
1 from typing import List
2
3 def plot_segmentation_masks(category_probs: np.ndarray,
4                             threshold: float,
5                             categories: np.ndarray,
6                             categories_of_interest: np.ndarray,
7                             ax,
8                             alpha: float,
9                             # cmap: List[np.ndarray]
10                            ) -> None:
11    """
12        Отрисовывает маски заданного подмножества категорий.
13        Принимает на вход следующие аргументы:
14        :param category_probs: – трёхмерный np.ndarray,
15        размерность – (число_категорий, *размер картинки), т.е. для каждого
16        пикселя изображения предсказывается вероятность того, что на нём
17        изображен объект из какой-то конкретной категории. Сами маски получаются
18        выбором самой вероятной категории (т.е. максимизацией правдоподобия).
19
20        :param threshold: – порог, вероятности ниже которого не учитываются
21        :param categories: – категории, на которых была обучена модель.
22        :param categories_of_interest: – категории, маски которых нужно отрисовать
23        :param ax: – холст (должен быть создан заранее)
24        :param cmap: – RGBA-палитра цветов, по цвету на категорию
25    """
26    probs = np.copy(category_probs)
27    # Выделяем только те категории, которые нам интересны
28    categories_mask = np.isin(categories, categories_of_interest)
29    probs = probs[categories_mask]
29    # Делаем маску, где каждому пикслю соответствует его категория
30    masks = probs.argmax(0) + 1
31    # Категорию фона задаем как 0
32    masks[(probs < threshold).all(axis=0)] = 0
33    sns.heatmap(masks, ax=ax, cbar=False, linewidths=0, alpha=alpha)
34
35
36
37 def visualize_semantic_segmentation(ax, # холст из matplotlib
38                                     original_image: Image,
39                                     category_probs: np.ndarray,
40                                     threshold: float,
41                                     categories: np.ndarray,
42                                     categories_of_interest: np.ndarray,
43                                     show_original: bool = True,
44                                     alpha: float = 0.5):
45    """
46        Функция для отрисовки результатов семантической сегментации.
47        Может отрисовывать маски поверх исходного изображения.
48        Позволяет настраивать прозрачность и порог достоверности.
49        См. описание некоторых параметров в функции plot_segmentation_masks
49    """
50
51    ax.imshow(original_image, alpha=1. if show_original else 0)
52    ax.set_axis_off()
53    n_categories = len(categories_of_interest)
54    plot_segmentation_masks(
55        ax=ax,
56        alpha=alpha,
57        category_probs=category_probs,
58        threshold=threshold,
59        categories=categories,
```

```
60     categories_of_interest=categories_of_interest,  
61 )
```

Для наглядного сравнения моделей снова используем интерактивный виджет:

In []:

```
1 @widgets.interact(alpha_to_plot=(0, 0.25, 0.01),  
2                     threshold=(0.05, 1.05, 0.05),  
3                     show_original=[True, False])  
4 def semantic_segmentation_widget(alpha_to_plot=0.1, threshold=0.7, show_origin  
5 fig, axes = plt.subplots(1, 2, figsize=(20,10))  
6 titles = ["Семантическая сегментация, DeepLabV3-Resnet101",  
7           "Семантическая сегментация, FCN-Resnet101"]  
8  
9 masks = [deeplabv3_probs_sm, fcn_probs_sm]  
10  
11 for i, ax in enumerate(axes):  
12     ax.set_title(titles[i], fontsize=20, fontweight="bold")  
13  
14     visualize_semantic_segmentation(  
15         original_image=input_image,  
16         category_probs=masks[i],  
17         ax=axes[i],  
18         show_original=show_original,  
19         alpha=alpha_to_plot,  
20         threshold=threshold,  
21         categories=coco_categories_segmentation,  
22         categories_of_interest=np.array(  
23             ["bottle", "chair", "diningtable", "person", "sofa"]))  
24     )  
  
interactive(children=(FloatSlider(value=0.1, description='alpha_to_plo  
t', max=0.25, step=0.01), FloatSlider(va...
```

Покажем результат применения модели: выделим студентов от фона и добавим новый.

In []:

```
1 # Порог вероятностей классов
2 threshold = 0.65
3 # Исходная картинка
4 img = np.array(input_image)
5 # Новый фон
6 background = np.array(Image.open("ussr.jpeg").resize(input_image.size))
7 # Образы студентов
8 mask = (
9     (deeplabv3_probs_sm[5] > threshold) | (deeplabv3_probs_sm[9] > threshold) |
10    (deeplabv3_probs_sm[15] > threshold)
11 )
12 objects = mask.reshape(*img.shape[:2], 1)) * img
13 # Фон без студентов
14 background = (~mask).reshape(*img.shape[:2], 1)) * background
15
16 # Визуализация
17 plt.figure(figsize=(15, 15))
18 plt.imshow(objects + background)
19 plt.axis("off")
```

Out[207]:

(-0.5, 2999.5, 1993.5, -0.5)



3. Пример обучения модели сегментации

Мы немного поигрались с предобученными моделями, теперь будем файнтюнить модель сегментации под определенную задачу. Будем находить домашних животных на фотографиях.

Определим функции для скачивания и разархивации датасета. Это можно сделать не только с помощью bash команд, но также используя функционал Python библиотек.

In []:

```
1 class TqdmUpTo(tqdm):
2     def update_to(self, b=1, bsize=1, tsize=None):
3         if tsize is not None:
4             self.total = tsize
5             self.update(b * bsize - self.n)
6
7
8     def download_url(url, filepath):
9         directory = os.path.dirname(os.path.abspath(filepath))
10        os.makedirs(directory, exist_ok=True)
11        if os.path.exists(filepath):
12            print("Dataset already exists on the disk. Skipping download.")
13            return
14
15        with TqdmUpTo(unit="B", unit_scale=True, unit_divisor=1024, miniters=1, desc=urlretrieve(url, filename=filepath, reporthook=t.update_to, data=None))
16            t.total = t.n
17
18
19
20    def extract_archive(filepath):
21        extract_dir = os.path.dirname(os.path.abspath(filepath))
22        shutil.unpack_archive(filepath, extract_dir)
```

В этой папке будет храниться датасет.

In []:

```
1 dataset_directory = "oxford-iiit-pet"
```

Скачиваем [датасет с домашними животными](https://www.robots.ox.ac.uk/~vgg/data/pets/) (<https://www.robots.ox.ac.uk/~vgg/data/pets/>) — отдельные архивы для изображений и масок.

In []:

```
1 filepath = os.path.join(dataset_directory, "images.tar.gz")
2 download_url(
3     url="https://www.robots.ox.ac.uk/~vgg/data/pets/data/images.tar.gz", filepa
4 )
5 extract_archive(filepath)
```

images.tar.gz: 0.00B [00:00, ?B/s]

In []:

```
1 filepath = os.path.join(dataset_directory, "annotations.tar.gz")
2 download_url(
3     url="https://www.robots.ox.ac.uk/~vgg/data/pets/data/annotations.tar.gz", f
4 )
5 extract_archive(filepath)
6
```

annotations.tar.gz: 0.00B [00:00, ?B/s]

Некоторые файлы в датасете испорчены. Будем использовать только те, которые можно считать.

In []:

```
1 # Задаем директории
2 root_directory = os.path.join(dataset_directory)
3 images_directory = os.path.join(root_directory, "images")
4 masks_directory = os.path.join(root_directory, "annotations", "trimaps")
5
6 # Считываем изображения
7 images_filenames = list(sorted(os.listdir(images_directory)))
8 correct_images_filenames = [
9     i for i in images_filenames
10    if cv2.imread(os.path.join(images_directory, i)) is not None
11 ]
12
13 # Делим случайнм образом датасет
14 # на тренировочную, валидационную и тестовую части
15
16 random.seed(42)
17 random.shuffle(correct_images_filenames)
18
19 train_images_filenames = correct_images_filenames[:6000]
20 val_images_filenames = correct_images_filenames[6000:-10]
21 test_images_filenames = images_filenames[-10:]
22
23 print(len(train_images_filenames), len(val_images_filenames), len(test_images_f
```

6000 1374 10

Датасет имеет необычную структуру масок. Маски кодируются тремя числами:

- 1 — животное,
- 2 — фон,
- 3 — граница животного.

Проверим, куда лучше отнести границу — к фону или к животному. Для этого зададим 2 функции преобразования и отобразим результат.

In []:

```
1 def preprocess_mask1(mask):
2     mask = mask.astype(np.float32)
3     mask[(mask == 2.0)] = 0.0
4     mask[(mask == 1.0) | (mask == 3.0)] = 1.0
5     return mask
```

In []:

```
1 def preprocess_mask2(mask):
2     mask = mask.astype(np.float32)
3     mask[(mask == 2.0) | (mask == 3.0)] = 0.0
4     mask[mask == 1.0] = 1.0
5     return mask
```

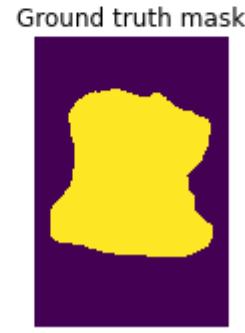
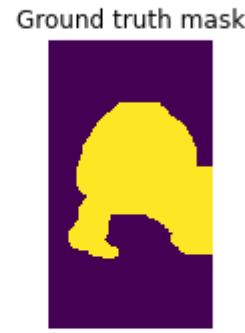
In []:

```
1 def display_image_grid(images_filenames, images_directory, masks_directory, pre
2 cols = 3 if predicted_masks else 2
3 rows = len(images_filenames)
4 figure, ax = plt.subplots(nrows=rows, ncols=cols, figsize=(10, 24))
5 for i, image_filename in enumerate(images_filenames):
6
7     # Считываем изображение с помощью библиотеки OpenCV
8     image = cv2.imread(os.path.join(images_directory, image_filename))
9     # Так как она изначально записывает каналы не в том порядке, нужно их пере
10    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
11
12    # Считываем маску
13    mask = cv2.imread(os.path.join(masks_directory, image_filename.replace(
14        image_filename, "mask.png")))
15    mask = preprocess_mask(mask)
16
17    ax[i, 0].imshow(image)
18    ax[i, 0].set_title("Image")
19    ax[i, 1].imshow(mask, interpolation="nearest")
20    ax[i, 1].set_title("Ground truth mask")
21
22    ax[i, 0].set_axis_off()
23    ax[i, 1].set_axis_off()
24
25    if predicted_masks:
26        predicted_mask = predicted_masks[i]
27        ax[i, 2].imshow(predicted_mask, interpolation="nearest")
28        ax[i, 2].set_title("Predicted mask")
29        ax[i, 2].set_axis_off()
30
31 plt.tight_layout()
32 plt.show()
```

Если относить границу к питомцу.

In []:

```
1 display_image_grid(test_images_filenames, images_directory, masks_directory, pr
```





Image



Ground truth mask



Image



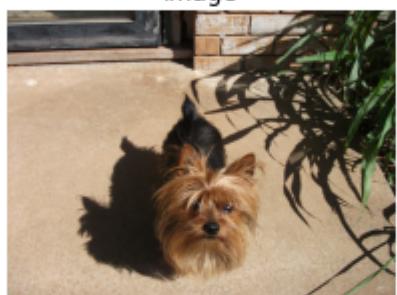
Ground truth mask



Image



Ground truth mask



Image



Ground truth mask



© Alejandro Keyes, 2010

Image



Ground truth mask

Если относить границу к фону.

In []:

```
1 display_image_grid(test_images_filenames, images_directory, masks_directory, pr
```





Image



Ground truth mask



Image



Ground truth mask



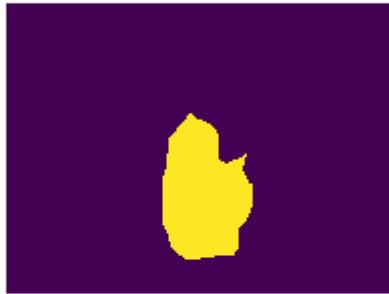
Image



Ground truth mask



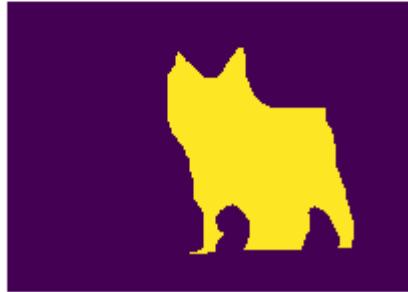
Image



Ground truth mask



© Alejandro Keyes, 2010



Ground truth mask

Кажется, что лучше немного больше взять фона, чем обрезать питомца. Будем использовать первый вариант.

Для того, чтобы эффективно загружать изображения и маски, будем использовать класс датасет, который далее будет загружаться в `Dataloader`. Для ускорения работы сохраним все изображения и маски в оперативку с помощью декоратора `lru_cache`. Он содержит параметр `maxsize`, который ограничивает максимальный размер сохраняемых данных. Если `maxsize=None`, то все данные будут созранены в оперативную память (если не будет переполнения).

In []:

```
1 class OxfordPetDataset(Dataset):
2     def __init__(self, images_filenames, images_directory, masks_directory, transform):
3         self.images_filenames = images_filenames
4         self.images_directory = images_directory
5         self.masks_directory = masks_directory
6         self.transform = transform
7
8     def __len__(self):
9         return len(self.images_filenames)
10
11    # @lru_cache(maxsize=None)
12    def _get_image_mask(self, image_filename):
13
14        # Считываем изображение
15        image = cv2.imread(os.path.join(self.images_directory, image_filename))
16        # Упорядочиваем каналы в правильном порядке
17        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
18
19        # Считываем маску
20        mask = cv2.imread(
21            os.path.join(self.masks_directory, image_filename.replace(".jpg", ".png"))
22        )
23        # Предобрабатываем маску выбранным способом
24        mask = preprocess_mask1(mask)
25
26        return image, mask
27
28    def __getitem__(self, idx):
29        image_filename = self.images_filenames[idx]
30        image, mask = self._get_image_mask(image_filename)
31
32        # Применяем трансформацию (аугментацию), если это необходимо
33        if self.transform is not None:
34            transformed = self.transform(image=image, mask=mask)
35            image = transformed["image"]
36            mask = transformed["mask"]
37
38        return image, mask
```

Зададим аугментации для обучения и валидации с помощью библиотеки [Albumentations](#) (<https://albumentations.ai/>). Она хороша тем, что позволяет делать аугментации для изображений и масок одновременно. Важно, что преобразование принимает объекты типа `PIL Image` или `numpy.ndarray`, но на прямую передать `torch.tensor` не получится:(Что делает каждая конкретная аугментация вы можете посмотреть в [документации](#) (<https://albumentations.ai/docs/>). Правда, она не очень удобная. Конкретные аугментации можно просто гуглить, либо искать в поиске на странице с документацией, либо обратиться к левой панели, перейти в раздел API Reference в подраздел Augmentations и там выбрать группу аугментаций, которая вам может помочь.

In []:

```
1 train_transform = A.Compose(
2     [
3         A.Resize(256, 256),
4         A.ShiftScaleRotate(shift_limit=0.2, scale_limit=0.2, rotate_limit=30, p
5         A.RGBShift(r_shift_limit=25, g_shift_limit=25, b_shift_limit=25, p=0.5)
6         A.RandomBrightnessContrast(brightness_limit=0.3, contrast_limit=0.3, p=
7         A.OneOf([
8             A.IAAAdditiveGaussianNoise(),
9             A.GaussNoise(),
10            ], p=0.5),
11            A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
12            ToTensorV2()
13        ]
14    )
15 train_dataset = OxfordPetDataset(train_images_filenames, images_directory, mask
16
17 val_transform = A.Compose([
18     A.Resize(256, 256),
19     A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
20     ToTensorV2(),
21 ])
22 val_dataset = OxfordPetDataset(val_images_filenames, images_directory, masks_di
23
```

```
/usr/local/lib/python3.8/dist-packages/albumentations/imgaug/transform
s.py:263: FutureWarning: IAAAdditiveGaussianNoise is deprecated. Pleas
e use GaussNoise instead
  warnings.warn("IAAAdditiveGaussianNoise is deprecated. Please use Ga
ussNoise instead", FutureWarning)
```

Посмотрим, какие преобразования получаются на тренировочных данных.

In []:

```
1 def visualize_augmentations(dataset, idx=0, samples=5):
2     # Делаем копию датасета для визуализации
3     dataset = copy.deepcopy(dataset)
4
5     # Убираем из аугментаций нормализацию и приведение к тензору, для удобства
6     dataset.transform = A.Compose([
7         t for t in dataset.transform if not isinstance(t, (A.Normalize, ToTenso
8
9     figure, ax = plt.subplots(nrows=samples, ncols=2, figsize=(10, 24))
10    for i in range(samples):
11        image, mask = dataset[idx]
12        ax[i, 0].imshow(image)
13        ax[i, 1].imshow(mask, interpolation="nearest")
14        ax[i, 0].set_title("Augmented image")
15        ax[i, 1].set_title("Augmented mask")
16        ax[i, 0].set_axis_off()
17        ax[i, 1].set_axis_off()
18    plt.tight_layout()
19    plt.show()
```

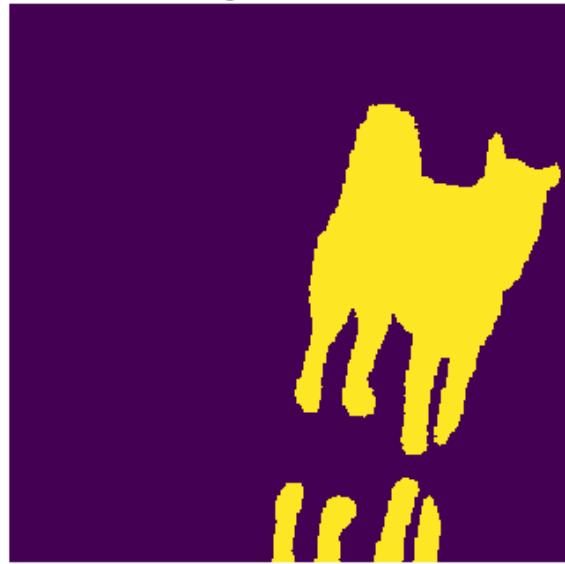
In []:

```
1 visualize_augmentations(train_dataset, idx=55)
```

Augmented image



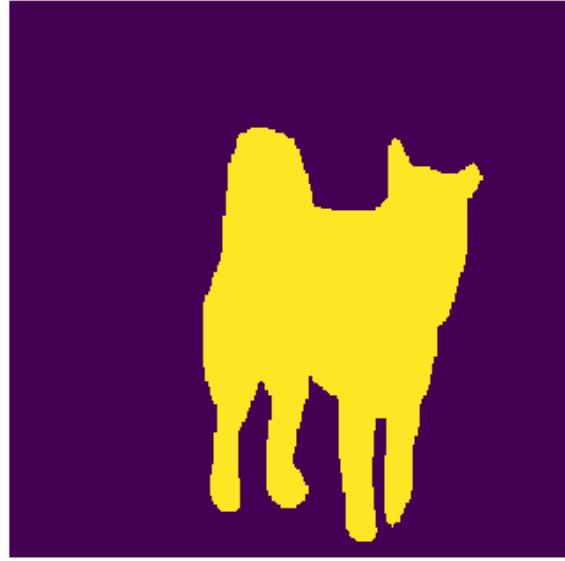
Augmented mask



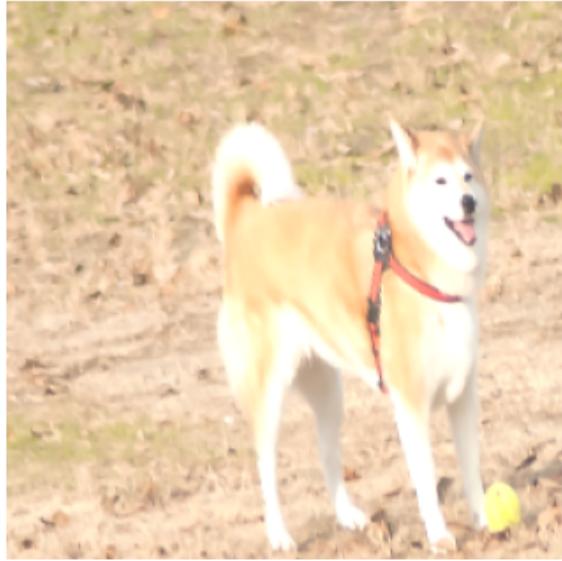
Augmented image



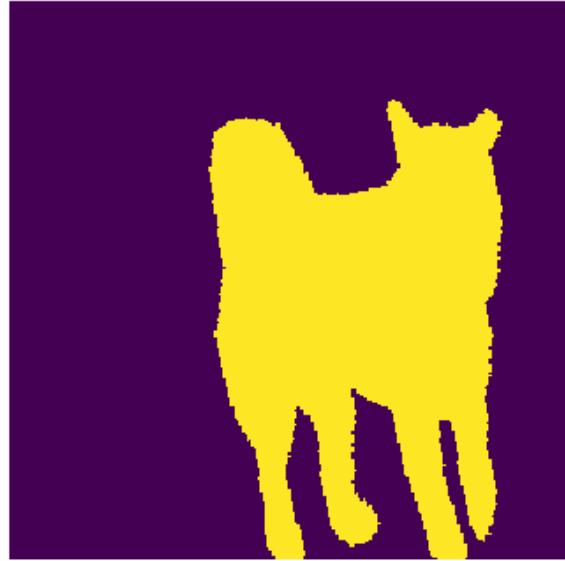
Augmented mask



Augmented image



Augmented mask



Augmented image

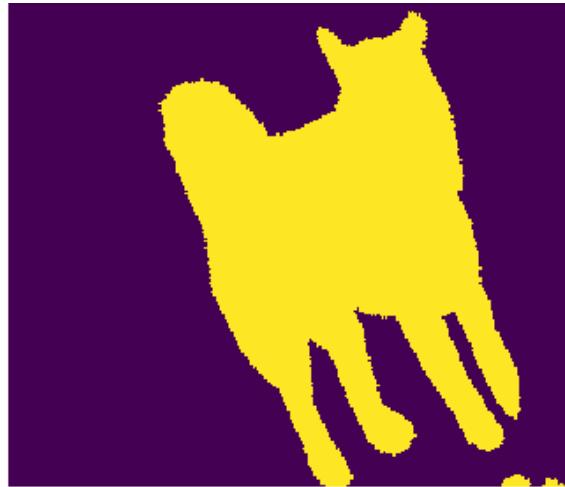


Augmented mask

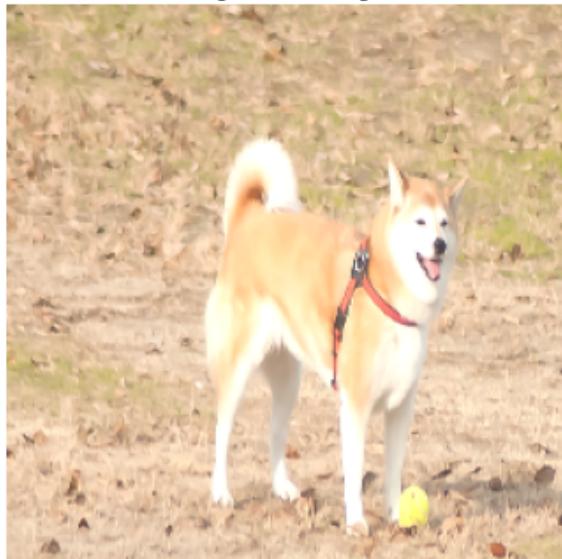




Augmented image



Augmented mask



Заведем класс для контролирования метрик.

In []:

```
1 from collections import defaultdict
2
3
4 class MetricMonitor:
5     def __init__(self, float_precision=3):
6         self.float_precision = float_precision
7         self.reset()
8
9     def reset(self):
10        self.metrics = defaultdict(lambda: {"val": 0, "count": 0, "avg": 0})
11
12    def update(self, metric_name, val):
13        metric = self.metrics[metric_name]
14
15        metric["val"] += val
16        metric["count"] += 1
17        metric["avg"] = metric["val"] / metric["count"]
18
19    def __str__(self):
20        return " | ".join(
21            [
22                "{metric_name}: {avg:.{float_precision}f}"
23                    .format(
24                        metric_name=metric_name, avg=metric["avg"], float_precision=3)
25                for (metric_name, metric) in self.metrics.items()
26            ]
27        )
```

Мы будем использовать модель без изменения последнего слоя, хоть у нас количество классов и не совпадает. Это сделано чисто из простоты реализации. Будем дообучать только один из выходных каналов модели на определение питомца. В данном случае мы выбираем канал с собаками (12), так как их больше в датасете.

Функция для обучения.

In []:

```
1 def train(train_loader, model, criterion, optimizer, epoch, params):
2     metric_monitor = MetricMonitor()
3     stream = tqdm(train_loader)
4
5     model.train()
6
7     for i, (images, target) in enumerate(stream, start=1):
8         images = images.to(params["device"], non_blocking=True)
9         target = target.to(params["device"], non_blocking=True)
10
11        output = model(images)['out'][:, 12]
12        loss = criterion(output, target)
13        metric_monitor.update("Loss", loss.item())
14        optimizer.zero_grad()
15        loss.backward()
16        optimizer.step()
17
18        stream.set_description(
19            "Epoch: {epoch}. Train. {metric_monitor}".format(epoch=epoch,
20        )
```

Функция для валидации.

In []:

```
1 def validate(val_loader, model, criterion, epoch, params):
2     metric_monitor = MetricMonitor()
3     model.eval()
4     stream = tqdm(val_loader)
5     with torch.no_grad():
6         for i, (images, target) in enumerate(stream, start=1):
7             images = images.to(params["device"], non_blocking=True)
8             target = target.to(params["device"], non_blocking=True)
9             output = model(images)['out'][:, 12]
10            loss = criterion(output, target)
11            metric_monitor.update("Loss", loss.item())
12            stream.set_description(
13                "Epoch: {epoch}. Validation. {metric_monitor}".format(epoch=epoch
14            )
```

Зададим общую функцию для обучения и валидации. Будем сохранять модель после каждой эпохи.

Вы могли обратить внимание на параметры `non_blocking=True`, при превращении `numpy` массива в `torch.tensor` и `pin_memory=True` в даталоадере. Когда эти параметры установлены вместе, то копирование данных на gpu происходит асинхронно, за счет чего уменьшается скорость вычисления.

In []:

```
1 def train_and_validate(model, train_dataset, val_dataset, params):
2     train_loader = DataLoader(
3         train_dataset,
4         batch_size=params["batch_size"],
5         shuffle=True,
6         num_workers=params["num_workers"],
7         pin_memory=True,
8     )
9     val_loader = DataLoader(
10        val_dataset,
11        batch_size=params["batch_size"],
12        shuffle=False,
13        num_workers=params["num_workers"],
14        pin_memory=True,
15    )
16     criterion = nn.BCEWithLogitsLoss().to(params["device"])
17     optimizer = torch.optim.Adam(model.parameters(), lr=params["lr"])
18     for epoch in range(1, params["epochs"] + 1):
19         train(train_loader, model, criterion, optimizer, epoch, params)
20         validate(val_loader, model, criterion, epoch, params)
21         torch.save({'model': model.state_dict(), 'epoch': epoch}, params['path'])
22     return model, optimizer
```

В качестве лосса установим бинарную кросс-энтропию. Однако есть и более удачные функции ошибки.

Также определим функцию для предсказания.

In []:

```
1 def predict(model, params, test_dataset, batch_size):
2     test_loader = DataLoader(
3         test_dataset, batch_size=batch_size, shuffle=False, num_workers=params[
4     ])
5     model.eval()
6     predictions = []
7     with torch.no_grad():
8         for images, (original_heights, original_widths) in test_loader:
9             images = images.to(params["device"], non_blocking=True)
10            output = model(images)['out'][:, 12]
11            probabilities = torch.sigmoid(output)
12            predicted_masks = (probabilities >= 0.5).float() * 1
13            predicted_masks = predicted_masks.cpu().numpy()
14            for predicted_mask, original_height, original_width in zip(
15                predicted_masks, original_heights.numpy(), original_widths.numpy(
16            )):
17                predictions.append((predicted_mask, original_height, original_w
18     return predictions
```

Рассмотрим модель из [torchvision \(<https://pytorch.org/vision/0.9/models.html#semantic-segmentation>\)](https://pytorch.org/vision/0.9/models.html#semantic-segmentation). Там представлено в целом больше моделей. Ари немножко отличается, но на практике это почти не видно. Будем использовать простую предобученную FCN из torchvision .

In []:

```
1 def create_model(params, reload=False):
2     model = models.segmentation.fcn_resnet50(pretrained=True)
3     model = model.to(params["device"])
4     if (not reload) and os.path.exists(params["path"]):
5         model.load_state_dict(torch.load(params["path"], map_location=params["d
6     return model
```

Зададим параметры обучения.

In []:

```
1 params = {
2     "device": "cuda",
3     "lr": 1e-4,
4     "batch_size": 16,
5     "num_workers": 4,
6     "epochs": 1,
7     "path": 'state.pth'
8 }
```

In []:

```
1 model = create_model(params, reload=True)
```

```
/usr/local/lib/python3.8/dist-packages/torchvision/models/_utils.py:20
8: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and will be removed in 0.15, please use 'weights' instead.
    warnings.warn(
/usr/local/lib/python3.8/dist-packages/torchvision/models/_utils.py:22
3: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and will be removed in 0.15. The current behavior is equivalent to passing `weights=FCN_ResNet50_Weights.COCO_WITH_VOC_LABELS_V1`. You can also use `weights=FCN_ResNet50_Weights.DEFAULT` to get the most up-to-date weights.
    warnings.warn(msg)
```

Посмотрим на примере одного изображения, как работает модель без файн튜нинга.

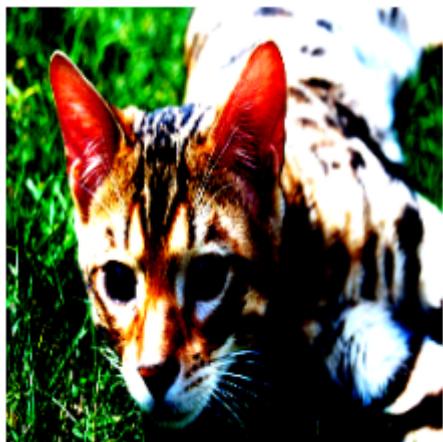
In []:

```
1 image, mask = val_dataset[0]
2 plt.imshow(image.moveaxis(0, 2))
3 plt.axis('off')
```

WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

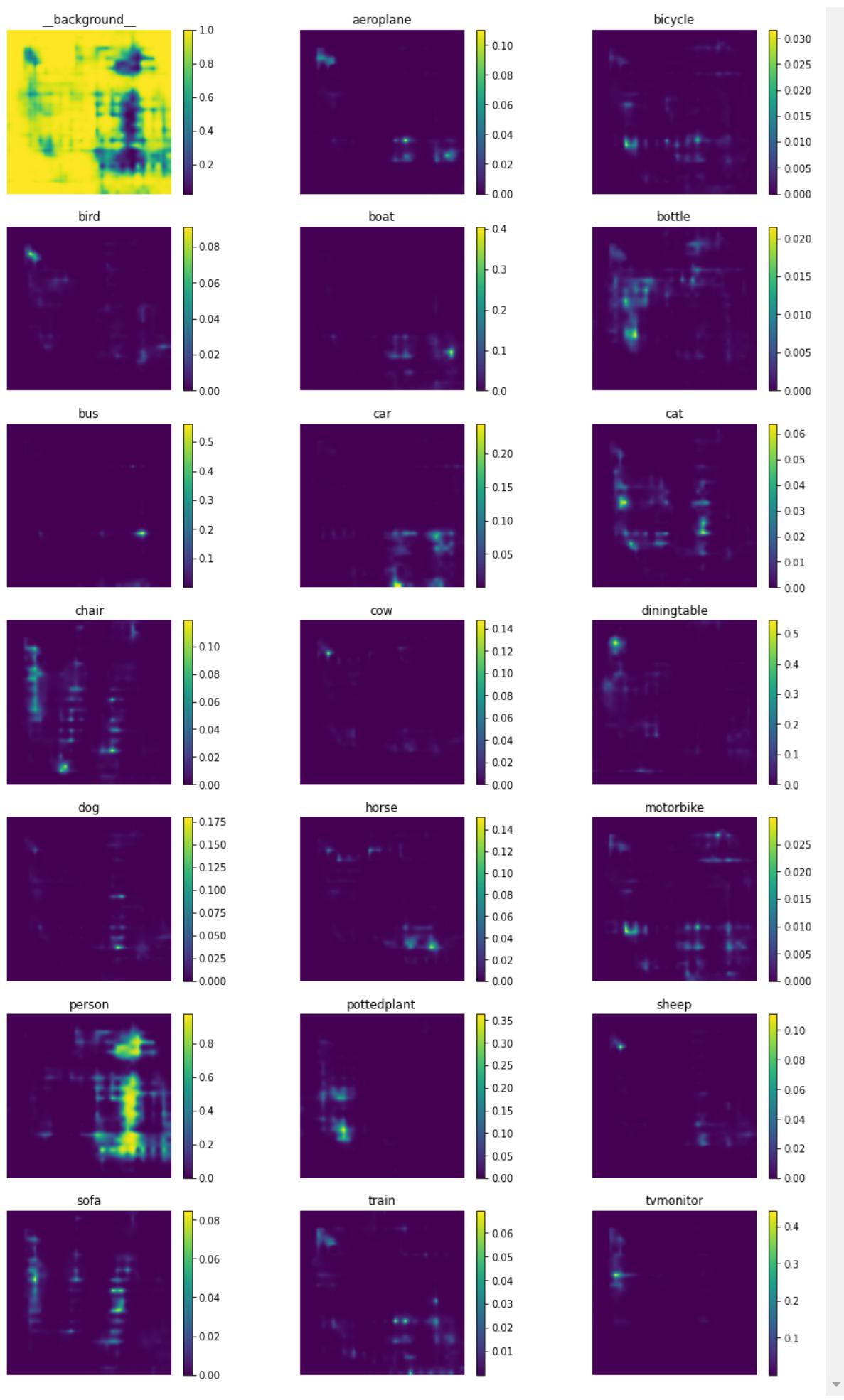
Out[116]:

```
( -0.5, 255.5, 255.5, -0.5)
```



In []:

```
1 with torch.no_grad():
2     outputs = model(image.unsqueeze(0).to(device))['out'].cpu().squeeze()
3
4 probs_sm = outputs_to_probs(outputs, mode='softmax')
5 plot_probs(probs_sm)
```



Как мы видим, кошку модель сейчас не распознала. Обучим модель за одну эпоху.

In []:

```
1 model, optimizer = train_and_validate(model, train_dataset, val_dataset, params  
0%|          | 0/375 [00:00<?, ?it/s]  
0%|          | 0/86 [00:00<?, ?it/s]
```

Проверим качество модели.

Датасет для инференса модели на teste.

In []:

```
1 class OxfordPetInferenceDataset(Dataset):  
2     def __init__(self, images_filenames, images_directory, transform=None):  
3         self.images_filenames = images_filenames  
4         self.images_directory = images_directory  
5         self.transform = transform  
6  
7     def __len__(self):  
8         return len(self.images_filenames)  
9  
10    def __getitem__(self, idx):  
11        image_filename = self.images_filenames[idx]  
12        image = cv2.imread(os.path.join(self.images_directory, image_filename))  
13        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  
14        original_size = tuple(image.shape[:2])  
15        if self.transform is not None:  
16            transformed = self.transform(image=image)  
17            image = transformed["image"]  
18        return image, original_size
```

Установим тестовые трансформации.

In []:

```
1 test_transform = A.Compose(  
2     [A.Resize(256, 256), A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.  
3 )]  
4 test_dataset = OxfordPetInferenceDataset(test_images_filenames, images_directory)
```

Получим предсказания модели.

In []:

```
1 predictions = predict(model, params, test_dataset, batch_size=16)
```

```
/usr/local/lib/python3.8/dist-packages/torch/utils/data/dataloader.py:  
563: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested max number of worker in current system is 2, which is smaller than what this DataLoader is going to create. Please be aware that excessive worker creation might get DataLoader running slow or even freeze, lower the worker number to avoid potential slowness/freeze if necessary.
```

```
    warnings.warn(_create_warning_msg(
```

In []:

```
1 predictions[0][0].shape
```

Out[122]:

```
(256, 256)
```

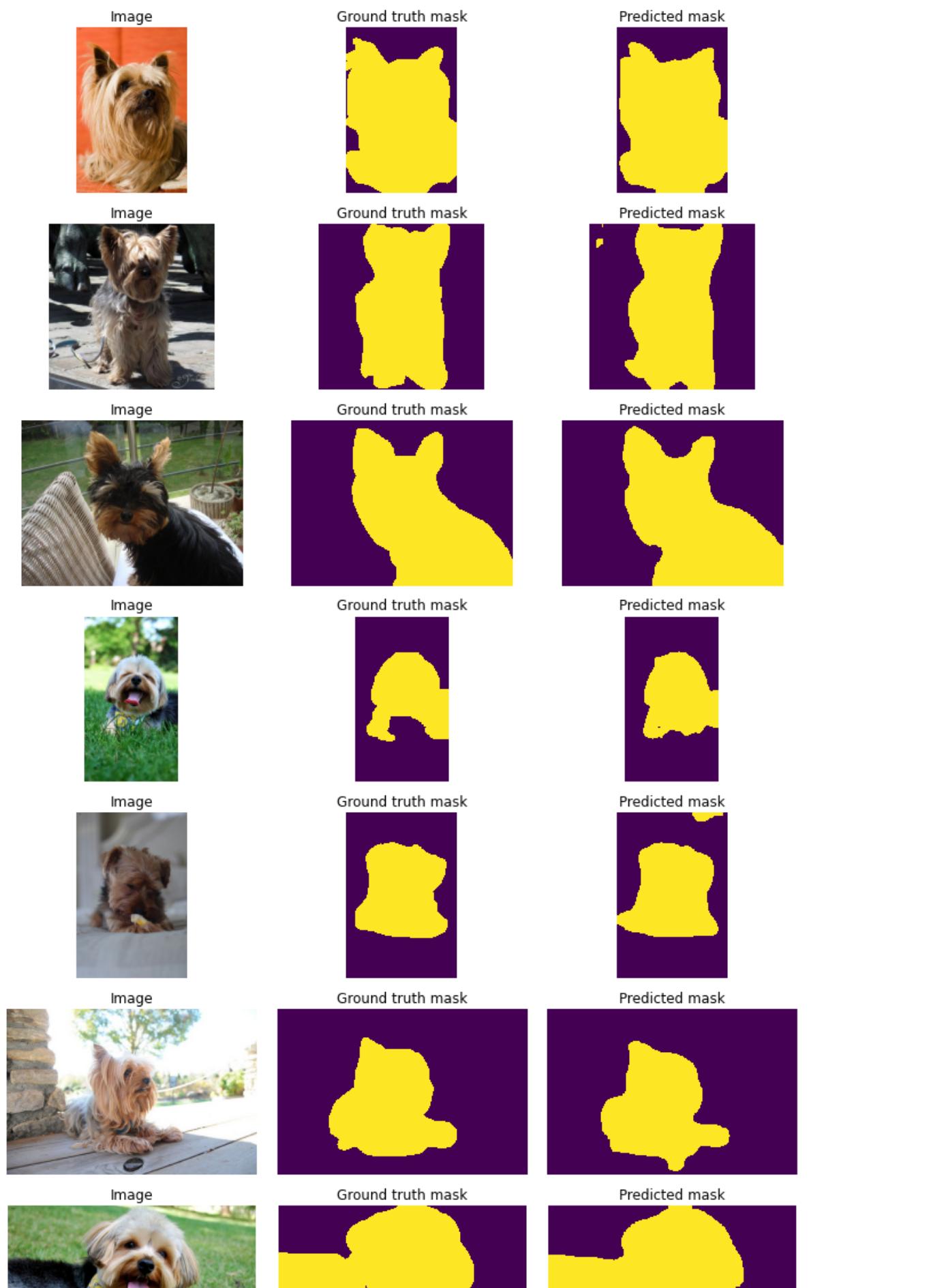
Преобразуем предсказания в маски сегментации.

In []:

```
1 predicted_masks = []  
2 for predicted_256x256_mask, original_height, original_width in predictions:  
3     full_sized_mask = cv2.resize(  
4         predicted_256x256_mask,  
5         (original_width, original_height),  
6         interpolation=cv2.INTER_NEAREST  
7     )  
8     predicted_masks.append(full_sized_mask)
```

In []:

```
1 display_image_grid(test_images_filenames, images_directory, masks_directory, pr  
2
```





За одну эпоху модель неплохо научилась определять питомцев судя по примерам. Для качественного анализа нужно считать метрики :)

Определение позы человека

Попробуем на деле один из подходов для определения позы человека — `openpose`. Он позволяет эффективно детектировать ключевые точки скелета человека на изображении и довольно хорошо справляется со сценами, где присутствуют большие группы людей. Этот метод стал победителем конкурса COCO 2016 Keypoints Challenge и пользуется популярностью благодаря своему достойному качеству и надежности в настройках для нескольких человек.

Клонируем один из репозиториев с имплеметацией OpenPose.

In []:

```
1 !git clone https://github.com/Hzzone/pytorch-openpose
```

`fatal: destination path 'pytorch-openpose' already exists and is not an empty directory.`

Перейдем в папку, где лежит код из репозитория.

In []:

```
1 %cd pytorch-openpose  
2 !ls
```

```
/content/pytorch-openpose  
basketball.jpg  demo_video.py  model          README.md  
demo_camera.py  images        notebooks      requirements.txt  
demo.py         input        pytorch-openpose  src
```

In []:

```
1 import cv2  
2 import matplotlib.pyplot as plt  
3 import copy  
4 import numpy as np  
5 import os  
6 import torch  
7  
8 from src import model, util  
9 from src.body import Body
```

Скачаем веса модели с диска.

In []:

```
1 body_pose_model_path = "./model/body_pose_model.pth"
```

Загрузим веса модели.

In []:

```
1 body_estimation = Body(body_pose_model_path)
```

In []:

```
1 !mkdir input
```

```
mkdir: cannot create directory 'input': File exists
```

Скачаем картинку баскетболиста.

In []:

```
1 test_pose_img = "./input/basketball.jpg"
```

In []:

```
1 !wget https://get.wallhere.com/photo/2048x1322-px-27-basketball-golden-NBA-stat
--2022-03-24 20:53:42-- https://get.wallhere.com/photo/2048x1322-px-2
7-basketball-golden-NBA-state-warriors-1858215.jpg (https://get.wallhe
re.com/photo/2048x1322-px-27-basketball-golden-NBA-state-warriors-1858
215.jpg)
Resolving get.wallhere.com (get.wallhere.com)... 104.26.13.130, 172.6
7.68.8, 104.26.12.130, ...
Connecting to get.wallhere.com (get.wallhere.com)|104.26.13.130|:44
3... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1693290 (1.6M) [image/jpeg]
Saving to: './input/basketball.jpg'

./input/basketball. 100%[=====] 1.61M 738KB/s in
2.2s

2022-03-24 20:53:44 (738 KB/s) - './input/basketball.jpg' saved [16932
90/1693290]
```

In []:

```
1 image = plt.imread(test_pose_img)
2 plt.figure(figsize=(10,10))
3 plt.imshow(image)
4 plt.axis('off')
5 plt.show()
```



Применим модель к картинке.

In []:

```
1 oriImg = cv2.imread(test_pose_img)
2 candidate, subset = body_estimation(oriImg)
3 canvas = copy.deepcopy(oriImg)
4 canvas = util.draw_bodypose(canvas, candidate, subset)
5 print("Количество людей:", len(subset))
6 print("Количество точек:", len(candidate))
```

Количество людей: 9

Количество точек: 145

In []:

```
1 plt.figure(figsize=(10,10))
2 plt.imshow(canvas[:, :, [2, 1, 0]])
3 plt.axis('off')
4 plt.show()
```

