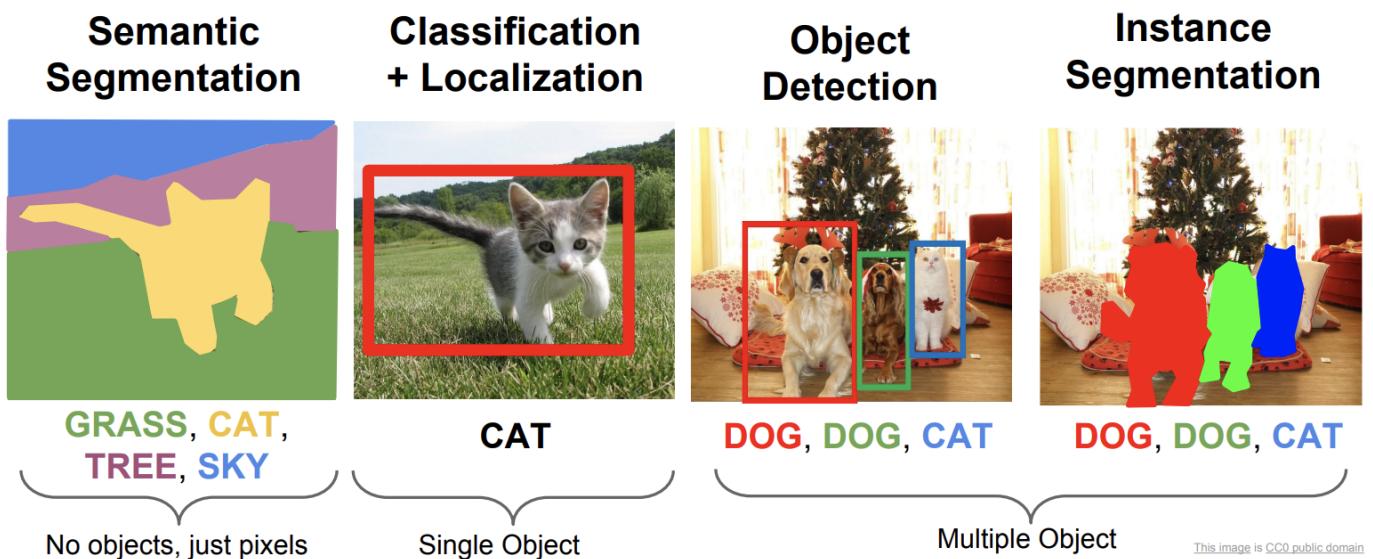


Объектная сегментация / Instance segmentation

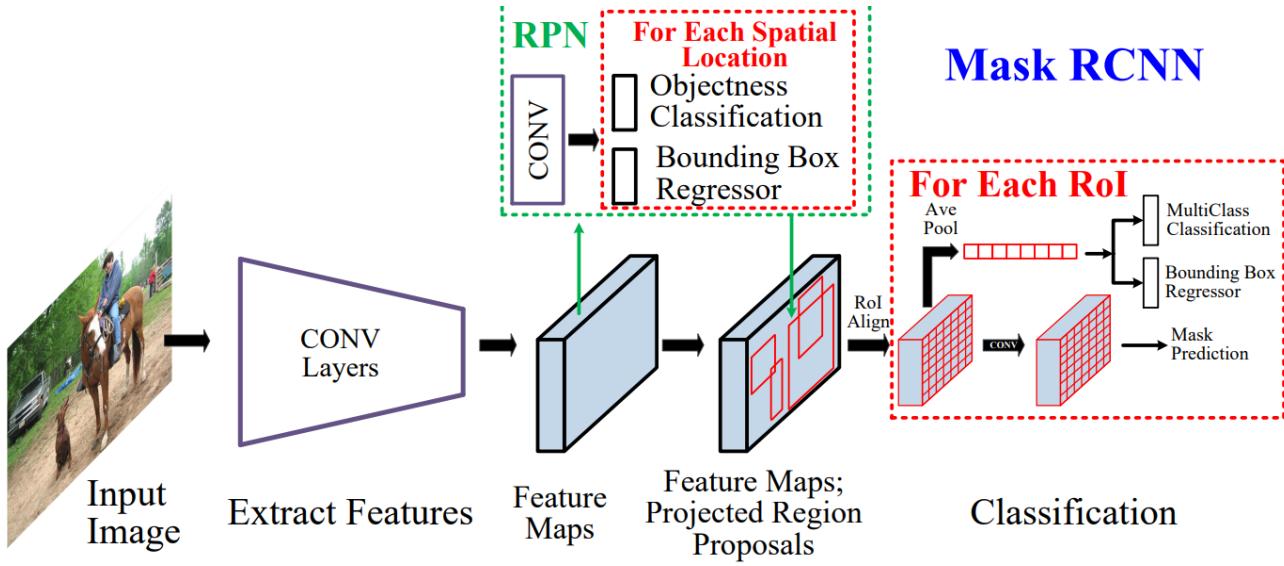
Задача объектной сегментации заключается в том, чтобы найти все объекты заданных классов на изображении и определить их границы, то есть найти все пиксели, принадлежащие объекту. Таким образом задачу инстанс сегментации можно разбить на две части: детекция объектов и сегментация найденных объектов.



1. Архитектура модели

Для решения этой задачи будем использовать модель **Mask-RCNN**. Напомним, что модель состоит из следующих компонентов.

1. Извлечения признаков из картинки с помощью Backbone-сети.
2. Генерации поверх полученного Feature Map'a регионов интереса.
3. Приведения предсказанных регионов к одному размеру с помощью ROI Pooling (в Mask-RCNN используется улучшенная версия ROI Pooling -- ROI Align).
4. Предсказания поверх полученного тензора из выровненных регионов интереса трех вещей (multi-task):
 - координат боксов
 - классов внутри боксов
 - масок внутри боксов



2. Библиотеки

В данном ноутбуке мы сравним работу двух реализаций Mask RCNN: в [torchvision](https://pytorch.org/vision/main/models/mask_rcnn.html) (https://pytorch.org/vision/main/models/mask_rcnn.html) и [detectron2](https://github.com/facebookresearch/detectron2/blob/main/MODEL_ZOO.md#coco-instance-segmentation-baselines-with-mask-r-cnn) (https://github.com/facebookresearch/detectron2/blob/main/MODEL_ZOO.md#coco-instance-segmentation-baselines-with-mask-r-cnn).

Torchvision

С библиотекой [torchvision](https://pytorch.org/vision/stable/index.html) (<https://pytorch.org/vision/stable/index.html>) мы уже знакомились ранее при решении задач классификации и семантической сегментации изображений. Ее интерфейс для моделей детекции не сильно отличается от рассмотренного ранее интерфейса для семантической сегментации.

Detectron2

[Detectron2](https://github.com/facebookresearch/detectron2) (<https://github.com/facebookresearch/detectron2>) — это библиотека от Meta AI Research (запрещена в России), одной из самых известных и опытных лабораторий по компьютерному зрению и глубокому обучению в мире. Они очень часто публикуются, и, например, Mask-RCNN — это изобретение FAIR. Они также часто вкладывают в open-source, например, тот же Detectron2 .

Эта библиотека хороша продуманным интерфейсом и визуализацией предсказаний. При реализации своих моделей детекции стоит ориентироваться на этот репозиторий и даже наследоваться от тамошних классов, чтобы перенести часть технической работы на библиотеку. В долгосрочной перспективе это оправдывает себя.

In [1]:

```
1 import sys, os, distutils.core
```

In [2]:

```
1 !python -m pip install pyyaml==5.1
2
3 # Быстрый способ установить detectron2 в Colab, однако он немного ограничивает
4 # Смотри https://detectron2.readthedocs.io/tutorials/install.html для инструкции
5 !git clone 'https://github.com/facebookresearch/detectron2'
6 dist = distutils.core.run_setup("./detectron2/setup.py")
7 !python -m pip install {''.join(['f"'{x}'" for x in dist.install_requires])}
8 sys.path.insert(0, os.path.abspath('./detectron2'))
9
10 # Правильный способ установки detectron2.
11 # Только, пожалуйста, не устанавливайте библиотеку двумя разными способами сразу
12 # !python -m pip install 'git+https://github.com/facebookresearch/detectron2.gi
```

```
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/)
Collecting pyyaml==5.1
  Downloading PyYAML-5.1.tar.gz (274 kB)
    274.2/274.2 kB 5.6 MB/s
eta 0:00:00
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: pyyaml
  Building wheel for pyyaml (setup.py) ... done
  Created wheel for pyyaml: filename=PyYAML-5.1-cp38-cp38-linux_x86_64.whl size=44089 sha256=85fda56d92f0209fc5511edb2d6b02235f0cfa899705188fe2fd271422c4f64f
  Stored in directory: /root/.cache/pip/wheels/52/dd/2b/10ff8b0ac81b93946bb5fb9e6749bae2dac246506c8774e6cf
Successfully built pyyaml
Installing collected packages: pyyaml
  Attempting uninstall: pyyaml
    Found existing installation: PyYAML 6.0
      Uninstalling PyYAML 6.0...
```

In [3]:

```
1 !pip install torchmetrics
```

```
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/)
Collecting torchmetrics
  Downloading torchmetrics-0.11.0-py3-none-any.whl (512 kB)
    512.4/512.4 kB 22.3 MB/s
eta 0:00:00
Requirement already satisfied: torch>=1.8.1 in /usr/local/lib/python3.8/dist-packages (from torchmetrics) (1.13.0+cu116)
Requirement already satisfied: numpy>=1.17.2 in /usr/local/lib/python3.8/dist-packages (from torchmetrics) (1.21.6)
Requirement already satisfied: packaging in /usr/local/lib/python3.8/dist-packages (from torchmetrics) (21.3)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.8/dist-packages (from torchmetrics) (4.4.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.8/dist-packages (from packaging->torchmetrics) (3.0.9)
Installing collected packages: torchmetrics
Successfully installed torchmetrics-0.11.0
```

Typesetting math: 100%

In [4]:

```
1 from collections import defaultdict
2 import os, json, random
3 import typing as tp
4
5 from IPython.display import display
6 import ipywidgets as widgets
7
8 from matplotlib import patches
9 import matplotlib.pyplot as plt
10 import seaborn as sns
11
12 import numpy as np
13 import pandas as pd
14
15 from PIL import Image
16 import cv2
17
18 import torch
19 import torchvision
20 from torchmetrics.detection.mean_ap import MeanAveragePrecision
21
22 import detectron2
23 from detectron2 import model_zoo
24 from detectron2.engine import DefaultPredictor, DefaultTrainer
25 from detectron2.config import get_cfg
26 from detectron2.utils.visualizer import ColorMode, Visualizer
27 from detectron2.data import MetadataCatalog, DatasetCatalog
28 from detectron2.structures import BoxMode
29 from detectron2.utils.logger import setup_logger
30 setup_logger()
```

Out[4]:

```
<Logger detectron2 (DEBUG)>
```

In [5]:

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

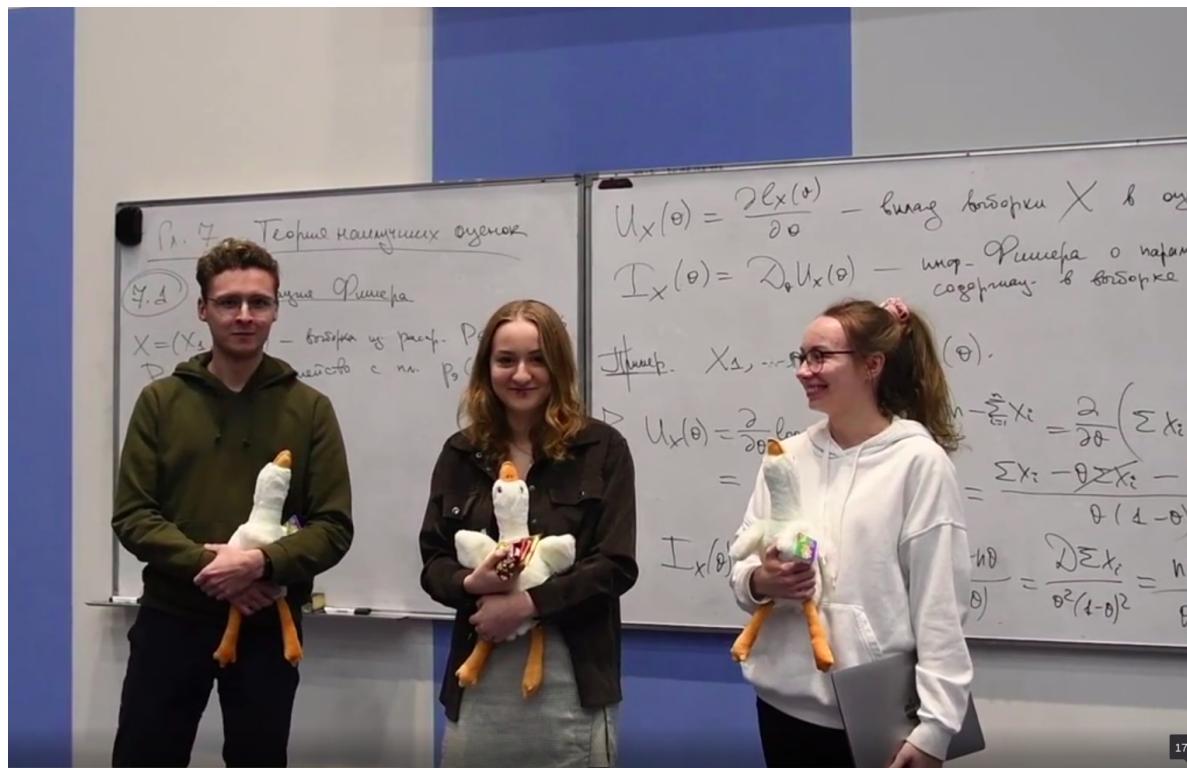
3. Использование предобученных моделей

Для начала проверим, как работают модели на нашем изображении.

In [7]:

```
1 input_image = Image.open('students.png')
2 input_image
```

Out[7]:



3.1 Torchvision

Загрузим модель с предобученными весами.

In [8]:

```
1 model = torchvision.models.detection.maskrcnn_resnet50_fpn(pretrained=True)
2 model.to(device)
3 model.eval()

/usr/local/lib/python3.8/dist-packages/torchvision/models/_utils.py:
208: UserWarning: The parameter 'pretrained' is deprecated since 0.1
3 and may be removed in the future, please use 'weights' instead.
    warnings.warn(
/usr/local/lib/python3.8/dist-packages/torchvision/models/_utils.py:
223: UserWarning: Arguments other than a weight enum or `None` for
'weights' are deprecated since 0.13 and may be removed in the futur
e. The current behavior is equivalent to passing `weights=MaskRCNN_R
esNet50_FPN_Weights.COCO_V1`. You can also use `weights=MaskRCNN_Res
Net50_FPN_Weights.DEFAULT` to get the most up-to-date weights.
    warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/maskrcnn_resnet50_
fpn_coco-bf2d0c1e.pth" to /root/.cache/torch/hub/checkpoints/maskrcn
n_resnet50_fpn_coco-bf2d0c1e.pth

0% | 0.00/170M [00:00<?, ?B/s]
```

Сделаем предсказание моделью.

In [9]:

```
1 # Подготовка батча для модели
2 preprocess = torchvision.transforms.ToTensor()
3 input_tensor = preprocess(input_image)
4 input_batch = input_tensor[:3].unsqueeze(0).to(device)
5
6 # Применение модели
7 with torch.no_grad():
8     maskrcnn_outputs = model(input_batch)[0]
```

Предсказание представляют собой словарик с ограничивающими прямоугольниками объектов, их лейблами, оценками уверенности в объектах и масками объектов.

In [10]:

```
1 maskrcnn_outputs.keys()
```

Out[10]:

```
dict_keys(['boxes', 'labels', 'scores', 'masks'])
```

Обработаем предсказания.

In [11]:

```
1 # Переводим все в numpy массивы
2 for key in maskrcnn_outputs.keys():
3     maskrcnn_outputs[key] = maskrcnn_outputs[key].cpu().numpy()
4
5 # Разделяем выход модели по переменным
6 maskrcnn_category_probs, maskrcnn_labels, maskrcnn_scores, maskrcnn_boxes = [
7     maskrcnn_outputs[key] for key in ["masks", "labels", "scores", "boxes"]
8 ]
9 maskrcnn_category_probs = maskrcnn_category_probs.squeeze(1)
```

In [12]:

```
1 maskrcnn_category_probs.shape, maskrcnn_labels.shape, maskrcnn_scores.shape, ma
```

Out[12]:

```
((29, 982, 1521), (29,), (29,), (29, 4))
```

Загрузим классы, на которые была обучена модель. Они нам понадобятся для визуализации работы модели.

In [13]:

```
1 def load_coco_categories() -> np.array:
2     coco_categories = pd.read_csv(
3         "https://raw.githubusercontent.com/nightrome/cocostuff/master/labels.txt",
4         sep=" ", usecols=[1]
5     ).values.ravel()
6     coco_categories = np.hstack(([__background__], coco_categories))
7     return coco_categories
8
9
10 coco_categories = load_coco_categories()
11 coco_categories
```

Out[13]:

```
array(['__background__', 'person', 'bicycle', 'car', 'motorcycle',
       'airplane', 'bus', 'train', 'truck', 'boat', 'traffic', 'fire',
       'street', 'stop', 'parking', 'bench', 'bird', 'cat', 'dog',
       'horse', 'sheep', 'cow', 'elephant', 'bear', 'zebra', 'giraffe',
       'hat', 'backpack', 'umbrella', 'shoe', 'eye', 'handbag', 'tie',
       'suitcase', 'frisbee', 'skis', 'snowboard', 'sports', 'kite',
       'baseball', 'baseball', 'skateboard', 'surfboard', 'tennis',
       'bottle', 'plate', 'wine', 'cup', 'fork', 'knife', 'spoon',
       'bowl', 'banana', 'apple', 'sandwich', 'orange', 'broccoli', 'carrot',
       'hot', 'pizza', 'donut', 'cake', 'chair', 'couch', 'potted',
       'bed', 'mirror', 'dining', 'window', 'desk', 'toilet', 'door', 'tv',
       'laptop', 'mouse', 'remote', 'keyboard', 'cell', 'microwave',
       'oven', 'toaster', 'sink', 'refrigerator', 'blender', 'book',
       'clock', 'vase', 'scissors', 'teddy', 'hair', 'toothbrush',
       'hair', 'banner', 'blanket', 'branch', 'bridge', 'building-other',
       'bush', 'cabinet', 'cage', 'cardboard', 'carpet', 'ceiling-other',
       'ceiling-tile', 'cloth', 'clothes', 'clouds', 'counter',
       'cupboard', 'curtain', 'desk-stuff', 'dirt', 'door-stuff',
       'fence', 'floor-marble', 'floor-other', 'floor-stone', 'floor-tile',
       'floor-wood', 'flower', 'fog', 'food-other', 'fruit',
       'furniture-other', 'grass', 'gravel', 'ground-other', 'hill',
       'house', 'leaves', 'light', 'mat', 'metal', 'mirror-stuff',
       'moss', 'mountain', 'mud', 'napkin', 'net', 'paper', 'pavement',
       'pillow', 'plant-other', 'plastic', 'platform', 'playingfield',
       'railings', 'railroad', 'river', 'road', 'rock', 'roof', 'rug', 'salad',
       'sand', 'sea', 'shelf', 'sky-other', 'skyscraper', 'snow',
       'solid-other', 'stairs', 'stone', 'straw', 'structural-other',
       'table', 'tent', 'textile-other', 'towel', 'tree', 'vegetable',
       'wall-brick', 'wall-concrete', 'wall-other', 'wall-panel',
       'wall-stone', 'wall-tile', 'wall-wood', 'water-other'],
      Typesetting math at 100%
```

```
'waterdrops', 'window-blind', 'window-other', 'wood'], dtype=object)
```

Функционал для визуализации сегментации.

In [14]:

```
1 def plot_segmentation_masks(
2     category_probs: np.ndarray,
3     scores: np.ndarray,
4     threshold_probs: float,
5     threshold_scores: float,
6     ax,
7     alpha: float
8 ) -> None:
9 """
10    Функция для отрисовки маски сегментации.
11    Использует heatmap из seaborn.
12 """
13     probs = np.copy(category_probs)
14     # Выделяем маски только тех объектов, которые мы оставили
15     probs = probs[scores >= threshold_scores]
16     # Делаем маску, где каждому пикслю соответствует его категория
17     masks = probs.argmax(0) + 1
18     # Категорию фона задаем как 0
19     masks[(probs < threshold_probs).all(axis=0)] = 0
20     sns.heatmap(masks, ax=ax, cbar=False, linewidths=0, alpha=alpha)
21
22
23 def plot_segmentation(
24     ax,
25     original_image: Image,
26     category_probs: np.ndarray,
27     scores: np.ndarray,
28     threshold_probs: float,
29     threshold_scores: float,
30     show_original: bool = True,
31     alpha: float = 0.5
32 ):
33 """
34    Функция для отрисовки результата сегментации для изображения.
35 """
36
37     ax.imshow(original_image, alpha=1. if show_original else 0)
38     # класс с минимальным номером это класс фона, он будет прозрачным
39     # добиться этого можно, правильно сформировав палитру
40     plot_segmentation_masks(
41         ax=ax,
42         category_probs=category_probs,
43         scores=scores,
44         threshold_probs=threshold_probs,
45         threshold_scores=threshold_scores,
46         alpha=alpha
47     )
```

Функционал для визуализации детекции.

In [15]:

```
1 def plot_boxes(boxes: np.ndarray,
2                 labels: np.ndarray,
3                 scores: np.ndarray,
4                 threshold: float,
5                 categories: np.ndarray,
6                 ax,
7                 annot_kwargs: tp.Dict[str, tp.Any] = None,
8                 box_kwargs: tp.Dict[str, tp.Any] = None):
9
10    """
11        Функция отрисовки bounding box-ов в задаче детекции
12        Большая часть параметров вам уже знакома по предыдущим
13        функциям такого рода. При помощи параметров annot_kwargs
14        и box_kwargs вы можете контролировать отрисовку меток классов
15        и самих bounding box-ов. Пример – см. виджет ячейкой ниже.
16    """
17
18    if box_kwargs is None:
19        box_kwargs = dict(linewidth=1, fill=False,
20                           edgecolor='r', facecolor='none')
21    if annot_kwargs is None:
22        annot_kwargs = dict()
23
24    for i, box in enumerate(boxes):
25        if scores[i] < threshold:
26            continue
27        left_x, top_y, right_x, bottom_y = box
28        xy_bottom_left = (left_x, top_y)
29        width = right_x - left_x + 1
30        height = bottom_y - top_y + 1
31        box_patch = patches.Rectangle(
32            xy_bottom_left, width=width, height=height,
33            **box_kwargs
34        )
35        ax.add_patch(box_patch)
36        ax.text(left_x, top_y,
37                "{}, {:.2f}".format(categories[labels[i]], scores[i]),
38                **annot_kwargs)
```

Визуализируем результат.

In [16]:

```
1 @widgets.interact(
2     alpha=(0.05, 0.5, 0.05),
3     threshold_scores=(0., 1, 0.1),
4     threshold_probs=(0., 1, 0.1),
5     show_original=[False, True]
6 )
7 def object_detection_widget(
8     alpha: float = 0.1,
9     threshold_scores=0.5,
10    threshold_probs=0.5,
11    show_original=True,
12 ):
13     fig, ax = plt.subplots(figsize=(20, 10))
14     ax.set_title("Объектная сегментация, Mask-RCNN",
15                  fontsize=25, fontweight="bold")
16     ax.imshow(input_image, alpha=1. if show_original else 0)
17     ax.set_axis_off()
18     plot_boxes(
19         ax=ax,
20         boxes=maskrcnn_boxes,
21         labels=maskrcnn_labels,
22         scores=maskrcnn_scores,
23         threshold=threshold_scores,
24         categories=coco_categories,
25         annot_kwargs=dict(color="white", fontsize=10)
26     )
27     plot_segmentation(
28         ax=ax,
29         original_image=input_image,
30         category_probs=maskrcnn_category_probs,
31         scores=maskrcnn_scores,
32         threshold_probs=threshold_probs,
33         threshold_scores=threshold_scores,
34         alpha=alpha,
35         show_original=show_original,
36     )
```

```
interactive(children=(FloatSlider(value=0.1, description='alpha', max=
0.5, min=0.05, step=0.05), FloatSlider(v...
```

3.2 Detectron2

Определим модель для предсказания.

In [17]:

```
1 # Определим дефолтную конфигурацию параметров
2 cfg = get_cfg()
3
4 # Установим конфигурацию параметров, необходимую для нашей модели Mask RCNN
5 cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_r
6
7 # Скачиваем веса модели
8 cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-InstanceSegmentation/mas
9
10 # Устанавливаем порог для оценок боксов
11 cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5
12
13 # Получим модель предсказания на основе заданной конфигурации
14 predictor = DefaultPredictor(cfg)
```

```
[01/04 16:19:15 d2.checkpoint.detection_checkpoint]: [DetectionCheckpo
inter] Loading from https://dl.fbaipublicfiles.com/detectron2/COCO-InstanceSegmentatio
n/mask\_rcnn\_R\_50\_FPN\_3x/137849600/model\_final\_f10217.pkl ...
```

```
model_final_f10217.pkl: 178MB [00:08, 21.2MB/s]
```

Модель работает с изображениями в том формате, который получается при считывании его с помощью OpenCV. Считаем изображение в таком формате.

In [18]:

```
1 input_image_cv2 = cv2.imread('students.png')
2 input_image_cv2.shape
```

Out[18]:

```
(982, 1521, 3)
```

Сделаем предсказание моделью.

In [19]:

```
1 outputs = predictor(input_image_cv2)
2 outputs.keys()
```

```
/usr/local/lib/python3.8/dist-packages/torch/functional.py:504: UserWa
rning: torch.meshgrid: in an upcoming release, it will be required to
pass the indexing argument. (Triggered internally at ../aten/src/ATen/
native/TensorShape.cpp:3190.)
    return _VF.meshgrid(tensors, **kwargs) # type: ignore[attr-defined]
```

Out[19]:

```
dict_keys(['instances'])
```

Как и реализация модели в `torchvision` реализация в `detectron` дает предсказание для объектов в виде ограничивающих прямоугольников `pred_boxes`, лейблов `pred_classes`, оценок уверенности в обнаружении `pred_scores` и масками объектов `pred_masks`.

In [20]:

```
1 outputs['instances']
```

Out[20]:

```
Instances(num_instances=3, image_height=982, image_width=1521, fields=[pred_boxes: Boxes(tensor([[ 926.9436, 362.3972, 1256.6064, 977.1041], [ 129.2217, 290.6291, 447.8856, 971.8853], [ 526.4768, 377.4456, 801.6569, 975.9445]], device='cuda:0')), scores: tensor([0.9994, 0.9993, 0.9988], device='cuda:0'), pred_classes: tensor([0, 0, 0], device='cuda:0'), pred_masks: tensor([[[[False, False, False, ..., False, False, False], [False, False, False, ..., False, False, False], [False, False, False, ..., False, False, False], ..., [False, False, False, ..., False, False, False], [False, False, False, ..., False, False, False], [False, False, False, ..., False, False, False]], [[False, False, False, ..., False, False, False], [False, False, False, ..., False, False, False], [False, False, False, ..., False, False, False], ..., [False, False, False, ..., False, False, False], [False, False, False, ..., False, False, False], [False, False, False, ..., False, False, False]], [[False, False, False, ..., False, False, False], [False, False, False, ..., False, False, False], [False, False, False, ..., False, False, False], ..., [False, False, False, ..., False, False, False], [False, False, False, ..., False, False, False], [False, False, False, ..., False, False, False]]], device='cuda:0')])
```

В detectron2 есть удобный визуализатор предсказаний. Проверим его на нашем изображении.

In [21]:

```
1 visualizer = Visualizer(  
2     input_image_cv2[:, :, ::-1],  
3     MetadataCatalog.get(cfg.DATASETS.TRAIN[0]),  
4     scale=1.2  
5 )  
6 out = visualizer.draw_instance_predictions(outputs["instances"].to("cpu"))  
7 Image.fromarray(out.get_image())
```

Out[21]:



4. Fine-tuning моделей

Попробуем адаптюнить модели на датасете с шарами. Этот датасет имеет только один класс для инстнас-сегментации — воздушный шар.

Typesetting math: 100%

In [22]:

```
1 # download, decompress the data
2 !wget https://github.com/matterport/Mask_RCNN/releases/download/v2.1/balloon_da
3 !unzip balloon_dataset.zip > /dev/null
```

```
--2023-01-04 16:19:33-- https://github.com/matterport/Mask_RCNN/relea
ses/download/v2.1/balloon_dataset.zip (https://github.com/matterport/M
ask_RCNN/releases/download/v2.1/balloon_dataset.zip)
Resolving github.com (github.com)... 140.82.121.3
Connecting to github.com (github.com)|140.82.121.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-rele
ase-asset-2e65be/107595270/737339e2-2b83-11e8-856a-188034eb3468?X-Amz-
Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F202
30104%2Fs-east-1%2Fs3%2Faws4_request&X-Amz-Date=20230104T161933Z&X-Am
z-Expires=300&X-Amz-Signature=c66d5784ab2cbab9fc793ae7916fa9e590052a3
0f247f0f49ec30016968799e&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&
repo_id=107595270&response-content-disposition=attachment%3B%20filename%3Dba
lloon_dataset.zip&response-content-type=application%2Foctet-stream) [f
ollowing]
--2023-01-04 16:19:33-- https://objects.githubusercontent.com/github-
production-release-asset-2e65be/107595270/737339e2-2b83-11e8-856a-1880
34eb3468?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX
4CSVEH53A%2F20230104%2Fs-east-1%2Fs3%2Faws4_request&X-Amz-Date=202301
04T161933Z&X-Amz-Expires=300&X-Amz-Signature=c66d5784ab2cbab9fc793ae7
916fa9e590052a30f247f0f49ec30016968799e&X-Amz-SignedHeaders=host&actor
_id=0&key_id=0&repo_id=107595270&response-content-disposition=attachme
nt%3B%20filename%3Dballoon_dataset.zip&response-content-type=application%
2Foctet-stream (https://objects.githubusercontent.com/github-produ
ction-release-asset-2e65be/107595270/737339e2-2b83-11e8-856a-188034eb34
68?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH
53A%2F20230104%2Fs-east-1%2Fs3%2Faws4_request&X-Amz-Date=20230104T161
933Z&X-Amz-Expires=300&X-Amz-Signature=c66d5784ab2cbab9fc793ae7916fa9
e590052a30f247f0f49ec30016968799e&X-Amz-SignedHeaders=host&actor_id=0&
key_id=0&repo_id=107595270&response-content-disposition=attachment%3B%
20filename%3Dballoon_dataset.zip&response-content-type=application%2Fo
ctet-stream)
Resolving objects.githubusercontent.com (objects.githubusercontent.co
m)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to objects.githubusercontent.com (objects.githubusercontentcom)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 38741381 (37M) [application/octet-stream]
Saving to: 'balloon_dataset.zip'

balloon_dataset.zip 100%[=====] 36.95M 17.7MB/s in
2.1s
```

```
2023-01-04 16:19:36 (17.7 MB/s) - 'balloon_dataset.zip' saved [3874138
1/38741381]
```

Typesetting math: 100%

4.1 Detectron2

Сначала проверим работу detectron2 . Реализуем функцию, которая для каждого изображения получает разметку в виде границ ограничивающих прямоугольников и маски сегментации.

In [23]:

```
1 def get_balloon_dicts(img_dir):
2
3     # Считываем файл с разметкой
4     json_file = os.path.join(img_dir, "via_region_data.json")
5     with open(json_file) as f:
6         imgs_anns = json.load(f)
7
8     dataset_dicts = []
9     for idx, v in enumerate(imgs_anns.values()):
10         record = {}
11
12         # Получаем характеристики самого изображения
13         filename = os.path.join(img_dir, v["filename"])
14         height, width = cv2.imread(filename).shape[:2]
15
16         record["file_name"] = filename
17         record["image_id"] = idx
18         record["height"] = height
19         record["width"] = width
20
21         # Получаем разметку для изображения
22         annos = v["regions"]
23         objs = []
24         for _, anno in annos.items():
25             assert not anno["region_attributes"]
26             anno = anno["shape_attributes"]
27
28             # Разметка задана попиксельно
29             px = anno["all_points_x"]
30             py = anno["all_points_y"]
31             poly = [(x + 0.5, y + 0.5) for x, y in zip(px, py)]
32             # Получаем вектор их центров пикселей сегментации
33             poly = [p for x in poly for p in x]
34
35             obj = {
36                 "bbox": [np.min(px), np.min(py), np.max(px), np.max(py)], # pa
37                 "bbox_mode": BoxMode.XYXY_ABS, # необходимо для detectron2
38                 "segmentation": [poly], # сегментационная разметка
39                 "category_id": 0, # только одна категория в датасете
40             }
41             objs.append(obj)
42             record["annotations"] = objs
43             dataset_dicts.append(record)
44     return dataset_dicts
```

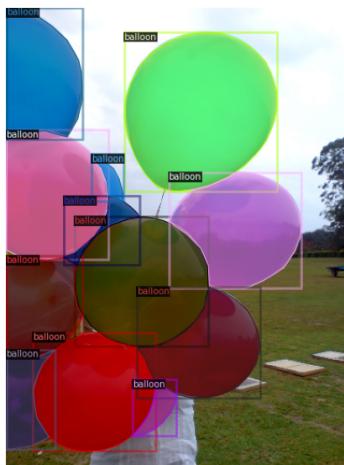
Регистрируем набор данных в DatasetCatalog и устанавливаем множество используемых классов через MetadataCatalog .

In [24]:

```
1 for d in ["train", "val"]:  
2     DatasetCatalog.register("balloon_" + d, lambda d=d: get_balloon_dicts("ball  
3         MetadataCatalog.get("balloon_" + d).set(thing_classes=["balloon"])  
4 balloon_metadata = MetadataCatalog.get("balloon_train")
```

In [25]:

```
1 dataset_dicts = get_balloon_dicts("balloon/train")  
2 fig, axs = plt.subplots(1, 3, figsize=(25, 10))  
3 for i, d in enumerate(random.sample(dataset_dicts, 3)):  
4     img = cv2.imread(d["file_name"])  
5     visualizer = Visualizer(img[:, :, ::-1], metadata=balloon_metadata, scale=0  
6     out = visualizer.draw_dataset_dict(d)  
7     axs[i].imshow(out.get_image())  
8     axs[i].axis('off')
```



Установим конфигурацию модели, которую мы будем обучать.

In [26]:

```
1 cfg = get_cfg()  
2 cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_r  
3 cfg.DATASETS.TRAIN = ("balloon_train",)  
4 cfg.DATASETS.TEST = ()  
5 cfg.DATALOADER.NUM_WORKERS = 2  
6 cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-InstanceSegmentation/mas  
7 cfg.SOLVER.IMS_PER_BATCH = 2 # This is the real "batch size" commonly known to  
8 cfg.SOLVER.BASE_LR = 0.00025 # pick a good LR  
9 cfg.SOLVER.MAX_ITER = 300 # 300 iterations seems good enough for this toy da  
10 cfg.SOLVER.STEPS = [] # do not decay learning rate  
11 cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128 # The "RoIHead batch size". 12  
12 cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1 # only has one class (balloon). (see https:  
# NOTE: this config means the number of classes, but a few popular unofficial t
```

Обучим модель.

In [27]:

```
1 os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
2 trainer = DefaultTrainer(cfg)
3 trainer.resume_or_load(resume=False)
4 trainer.train()
```

```
[01/04 16:19:45 d2.engine.defaults]: Model:
GeneralizedRCNN(
    (backbone): FPN(
        (fpn_lateral2): Conv2d(256, 256, kernel_size=(1, 1), stride=(1,
1))
        (fpn_output2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1,
1), padding=(1, 1))
        (fpn_lateral3): Conv2d(512, 256, kernel_size=(1, 1), stride=(1,
1))
        (fpn_output3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1,
1), padding=(1, 1))
        (fpn_lateral4): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1,
1))
        (fpn_output4): Conv2d(256, 256, kernel_size=(3, 3), stride=(1,
1), padding=(1, 1))
        (fpn_lateral5): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1,
1))
        (fpn_output5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1,
1), padding=(1, 1))
    )
)
```

Мы можем посмотреть на то, как менялись метрики в [tensorboard](https://www.tensorflow.org/tensorboard) (<https://www.tensorflow.org/tensorboard>) с помощью следующих команд.

In [28]:

```
1 %load_ext tensorboard
2 %tensorboard --logdir output
```

<IPython.core.display.Javascript object>

Теперь определим модель для предсказания. Она естественно будет использовать веса нашей обученной модели.

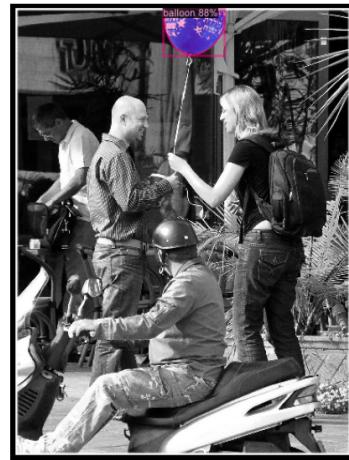
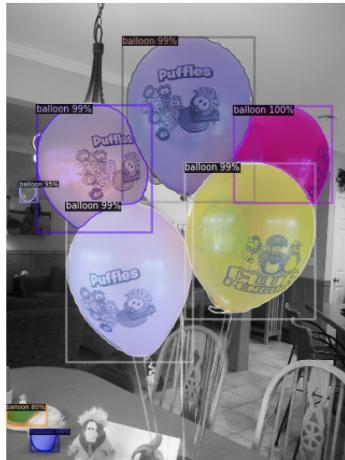
In [29]:

```
1 cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
2 cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.7
3 predictor = DefaultPredictor(cfg)
```

```
[01/04 16:22:25 d2.checkpoint.detection_checkpoint]: [DetectionCheckpo
inter] Loading from ./output/model_final.pth ...
```

In [30]:

```
1 dataset_dicts = get_balloon_dicts("balloon/val")
2 fig, axs = plt.subplots(1, 3, figsize=(25, 10))
3 for i, d in enumerate(random.sample(dataset_dicts, 3)):
4     im = cv2.imread(d["file_name"])
5     outputs = predictor(im)
6     v = Visualizer(im[:, :, ::-1],
7                     metadata=balloon_metadata,
8                     scale=0.5,
9                     instance_mode=ColorMode.IMAGE_BW    # Убираем цвет для несегм
10    )
11    out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
12    axs[i].imshow(out.get_image())
13    axs[i].axis('off')
```



Посчитаем качество предсказаний модели на валидационной части данных. В качестве метрики будем использовать mAP или Mean Average Precision на ограничивающих прямоугольниках. Для этого будет удобно применить ее [реализацию](#) (https://torchmetrics.readthedocs.io/en/stable/detection/mean_average_precision.html) в библиотеке [torchmetrics](#) (<https://torchmetrics.rtfd.io/en/latest/>). Для этого необходимо собрать все предсказания и разметку в два отдельных списка, в которых для каждого объекта будут собраны ограничивающие прямоугольники и лейблы (для предсказаний также оценка для каждого бокса) в виде тензоров.

In [31]:

```
1 preds, targets = [], []
2 for d in dataset_dicts:
3     target = defaultdict(list)
4     for obj in d['annotations']:
5         target['boxes'].append(obj['bbox'])
6         target['labels'].append(0)
7     target['boxes'] = torch.tensor(target['boxes'])
8     target['labels'] = torch.tensor(target['labels'])
9     targets.append(target)
10
11 im = cv2.imread(d["file_name"])
12 outputs = predictor(im)
13
14 pred = dict()
15 pred['boxes'] = outputs['instances'].pred_boxes.tensor.detach().cpu()
16 pred['scores'] = outputs['instances'].scores.detach().cpu()
17 pred['labels'] = outputs['instances'].pred_classes.detach().cpu()
18 preds.append(pred)
```

Наконец посчитаем метрики. Данная реализация метрики включает в себя не только стандартную Mean Average Precision, но также ее же модификации.

In [32]:

```
1 metric = MeanAveragePrecision(iou_type="bbox")
2 metric.update(preds, targets)
3 metric.compute()
```

Out[32]:

```
{'map': tensor(0.7308),
'map_50': tensor(0.8285),
'map_75': tensor(0.8182),
'map_small': tensor(0.),
'map_medium': tensor(0.5267),
'map_large': tensor(0.8331),
'mar_1': tensor(0.2380),
'mar_10': tensor(0.7520),
'mar_100': tensor(0.7520),
'mar_small': tensor(0.),
'mar_medium': tensor(0.5667),
'mar_large': tensor(0.8556),
'map_per_class': tensor(-1.),
'mar_100_per_class': tensor(-1.)}
```

In [32]:

1	
---	--