Распознавание категорий яблок с помощью глубокого обучения

1 Описание задачи и данных

Постановка задачи

Разработать пайплайн из нейросетевых моделей, которые будет использовать робот для сбора яблок.

Робот оборудован камерой, с помощью которой смотрит на окружающую среду, а также инструментом для сбора плодов. Он будет распознавать отдельные яблоки, собирать их с дерева, и в зависимости от качества плода, класть в соответствующие контейнеры.

Входные данные:

- Изображение с камеры робота

Выходные данные:

- Найденные на изображении яблоки и их принадлежность одному из трех классов:
  - normal (нормальные)
  - parsha (пораженные)
  - mechanic (с механическими повреждениями)

  А именно координаты прямоугольной области и лейбл для каждого яблока. Это задача многоклассовой детекции.

Набор данных: датасет YablokiAll

Папка	Кол-во картинок	Описание
red	63	Картинки со средним количеством яблок
red_old	4	Картинки с большим количеством яблок
~~red_parsha~~	31	Не размечены
Механические повреждениея	135	Картинки с яблоком с механическим повреждением по центру, но на фоне могут быть как яблоки с повреждением, так и без

Датасет содержит:

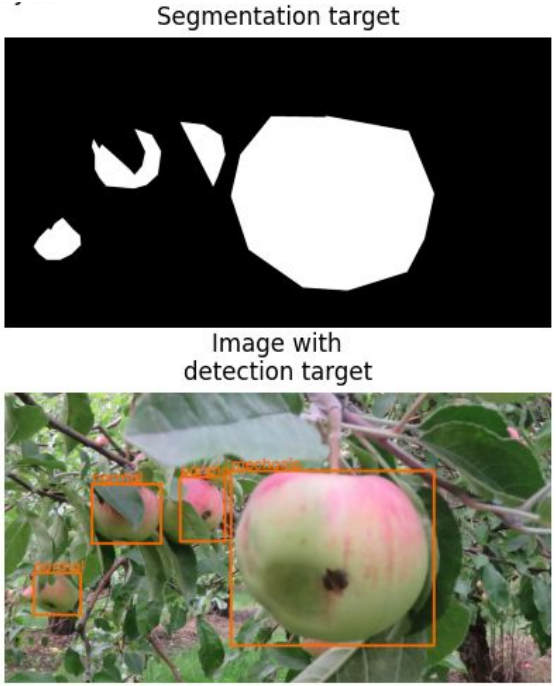
- подпапки с картинками
- json-файлы с метаданными к картинкам

Для каждой картинки задан список регионов.

- Один регион соответствует одному яблоку и задан точками, образующими многоугольник. Для визуализации были составлены двоичные маски на основе регионов.
- Каждый регион имеет метку класса:

  - red_normal
  - red_parsha
  - red_mechanical

  То есть имеется 3 класса (4 с учетом фона)



Работа с метаданными

Для удобства над метаданными были проведены следующие изменения:

- Json-файлы были пересобраны в один csv-файл (pandas dataframe). Так удобнее проводить аналитику
- Для каждого региона был получен bounding box и занесен в датафрейм
- Для каждого региона были посчитаны его площадь и площадь его bounding box’а и занесены в датафрейм
- Также в датафрейм были занесены размеры картинок и bounding box’ов
- В новый датасет не вошли картинки с отсутствующей аннотацией

Таким образом, из данных, размеченных для задачи instance segmentation (предсказывание маски, покрывающей каждый отдельный найденный объект), данные были дополнены, чтобы быть пригодны для задачи многоклассовой детекции.

Метрики качества

Основная метрика:
вариация mean Average Precision
для датасета COCO

$$mAP = \frac{1}{|classes|} \sum_{c \in classes} \frac{|TP_c|}{|FP_c| + |TP_c|}$$

Самые популярные модели имеют от ~0.4 до ~0.8 (Так как для задачи детекции в реальном времени бывает важна не только точность предсказаний, но и скорость работы модели)

Ориентиры: > 0.55

Дополнительная метрика:
вариация mean Average Recall
для датасета COCO

$$mAR = \frac{1}{|classes|} \sum_{c \in classes} \frac{|TP_c|}{|FN_c| + |TP_c|}$$

Самые популярные модели имеют от ~0.25 до ~0.35

Ориентиры: > 0.25

2 Предварительный анализ данных

Вывод картинки с таргетами из папки «red»

Image

Segmentation target

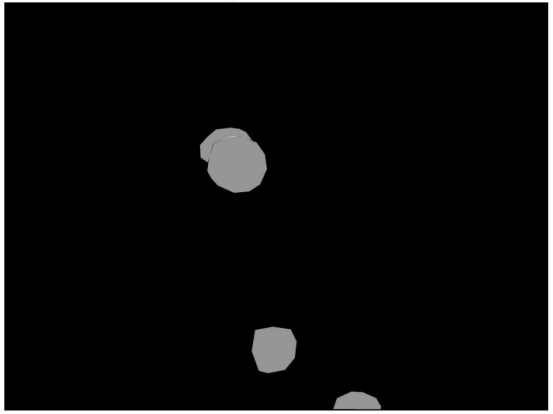


Image with
segmentation target

Image with
detection target

Вывод картинки с таргетами из папки «red_old»

Image

Segmentation target

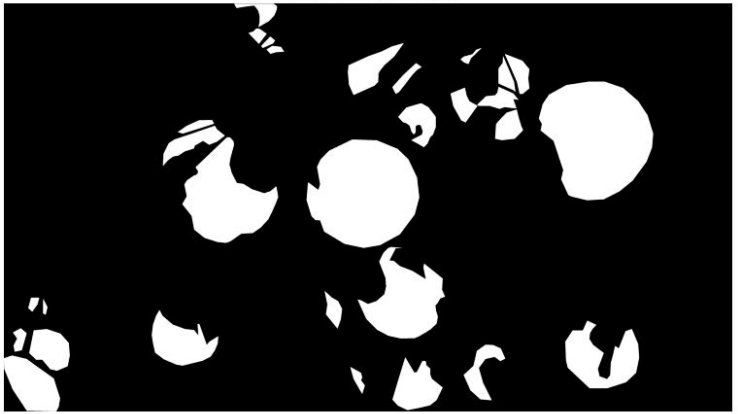
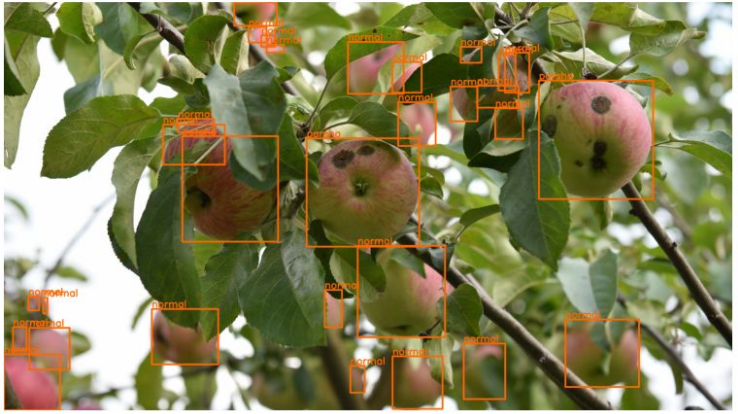


Image with
segmentation target

Image with
detection target



Вывод картинки с таргетами из папки «Механические повреждения»

Image

Segmentation target

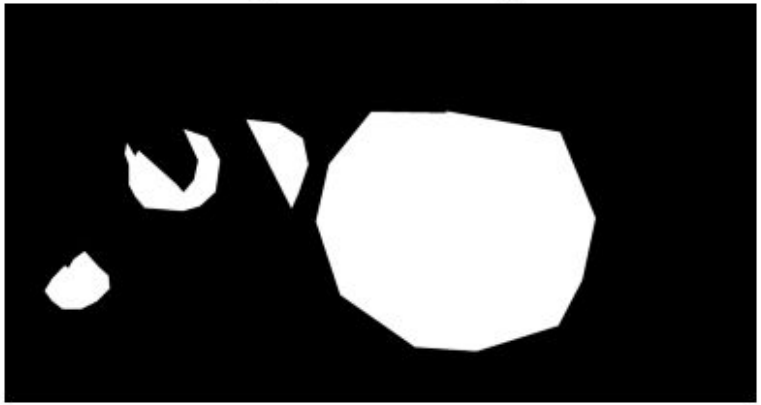


Image with
segmentation target

Image with
detection target

Наблюдения (недочеты):

- Размечены не все яблоки на снимке
- Может быть размечена одна видимая часть яблока, а другая видимая часть нет
- В одних случаях размечена только видимая часть яблока, в других – загороженное другими объектами яблоко достраивается

Всё это может негативно повлиять на обучение модели.

Наблюдения (нейтральные):

- Каждая область всегда соответствует одному яблоку, а не кластеру

Вывод датафрейма с метаданными

```
df
```

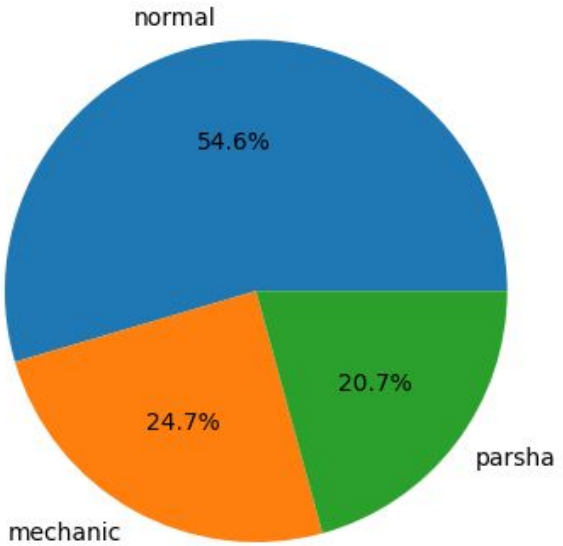
	Unnamed: 0	image_name	folder_name	label	box	image_width	image_height	box_width	box_height	box_area	polygon	polygon_area
0	0	DSC_7743.JPG	red_old	parsha	[(1875, 843), (2572, 1514)]	4512	2528	697	671	467687	[(1875, 1218), (1906, 1318), (1929, 1341), (19...	352374.5
1	1	DSC_7743.JPG	red_old	parsha	[(3314, 485), (4021, 1219)]	4512	2528	707	734	518938	[(3338, 623), (3377, 565), (3484, 506), (3620,...	379330.5
2	2	DSC_7743.JPG	red_old	normal	[(2767, 522), (2933, 737)]	4512	2528	166	215	35690	[(2767, 558), (2784, 636), (2833, 724), (2859,...	18625.0
3	3	DSC_7743.JPG	red_old	normal	[(2829, 276), (2958, 375)]	4512	2528	129	99	12771	[(2829, 347), (2851, 366), (2877, 375), (2892,...	3113.5
4	4	DSC_7743.JPG	red_old	normal	[(2931, 517), (3187, 653)]	4512	2528	256	136	34816	[(2931, 629), (2941, 634), (3008, 634), (3070,...	18441.0
...	...	...	...	...	...	...	...	...	...	...	...	...
675	675	IMG_2773.JPG	red	normal	[(1834, 2403), (2166, 2747)]	4032	3024	332	344	114208	[(1887, 2730), (1834, 2585), (1859, 2427), (19...	91872.5
676	676	IMG_2773.JPG	red	parsha	[(2438, 2883), (2793, 3015)]	4032	3024	355	132	46860	[(2438, 3013), (2466, 2932), (2574, 2883), (26...	35409.0
677	677	IMG_2770.JPG	red	normal	[(766, 4), (1637, 386)]	4032	3024	871	382	332722	[(766, 4), (792, 94), (928, 261), (1036, 336),...	255353.0
678	678	IMG_2770.JPG	red	parsha	[(1733, 1600), (2124, 2025)]	4032	3024	391	425	166175	[(1733, 1784), (1780, 1676), (1848, 1619), (20...	130824.0
679	679	IMG_2770.JPG	red	normal	[(1725, 1916), (2096, 2240)]	4032	3024	371	324	120204	[(1751, 1916), (1804, 1984), (1874, 2019), (19...	75624.5

680 rows × 12 columns

В датасете 680 отдельных яблок

Анализ данных с визуализацией

Круговая диаграмма представленных в датасете классов. Классы не сбалансированы, но не критичным образом. Если в будущем понадобится улучшение качества модели, можно попробовать достичь сбалансированности с помощью сэмплирования, апсемплинга или даунсемплинга.



Количество картинок в датасете

```
df['image_name'].nunique()
```

```
196
```

Отсутствующих полей в датасете нет

```
df.isnull().sum()
```

```
Unnamed: 0      0
image_name      0
folder_name     0
label           0
box             0
image_width     0
image_height    0
box_width       0
box_height      0
box_area        0
polygon         0
polygon_area    0
dtype: int64
```

Анализ данных с визуализацией

Типы данных

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 680 entries, 0 to 679
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Unnamed: 0    680 non-null    int64
 1   image_name    680 non-null    object
 2   folder_name   680 non-null    object
 3   label         680 non-null    object
 4   box           680 non-null    object
 5   image_width   680 non-null    int64
 6   image_height  680 non-null    int64
 7   box_width     680 non-null    int64
 8   box_height    680 non-null    int64
 9   box_area      680 non-null    int64
 10  polygon       680 non-null    object
 11  polygon_area  680 non-null    float64
dtypes: float64(1), int64(6), object(5)
memory usage: 63.9+ KB
```

Статистические показатели для каждого количественного признака

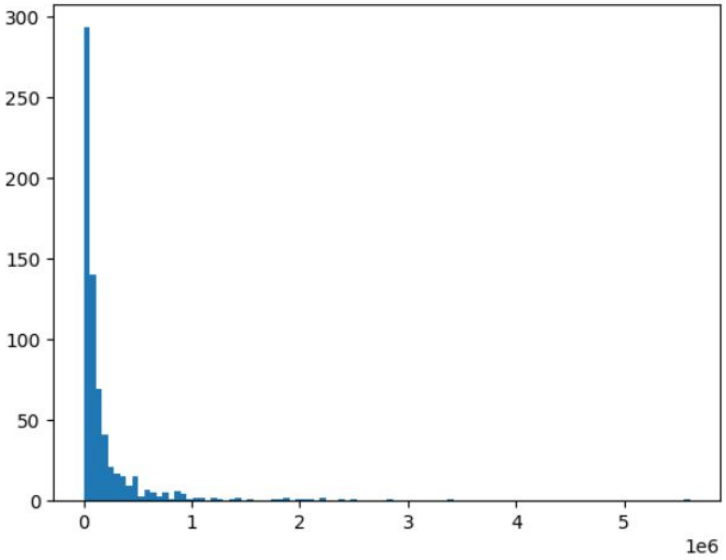
```
df.describe()
```

	Unnamed: 0	image_width	image_height	box_width	box_height	box_area	polygon_area
count	680.000000	680.000000	680.000000	680.000000	680.000000	6.800000e+02	6.800000e+02
mean	339.500000	3642.742647	2297.910294	419.204412	399.500000	2.711679e+05	2.015312e+05
std	196.443376	1240.134409	664.928749	336.863036	315.956486	5.438105e+05	4.226620e+05
min	0.000000	574.000000	430.000000	26.000000	34.000000	1.376000e+03	6.880000e+02
25%	169.750000	2602.500000	2115.000000	207.750000	205.000000	4.126225e+04	2.901462e+04
50%	339.500000	4512.000000	2528.000000	322.500000	313.000000	9.863500e+04	6.961850e+04
75%	509.250000	4512.000000	2528.000000	496.500000	488.000000	2.424698e+05	1.804599e+05
max	679.000000	5184.000000	4990.000000	2578.000000	2678.000000	6.903884e+06	5.607488e+06

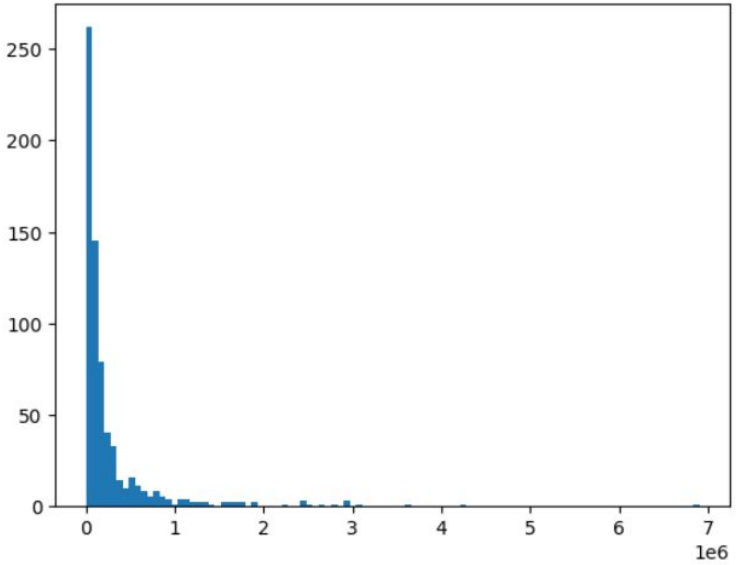
Анализ данных с визуализацией

Гистограммы площадей многоугольников, задающих регионы, и bounding box’ов

```
plt.hist(df['polygon_area'], 100);
```



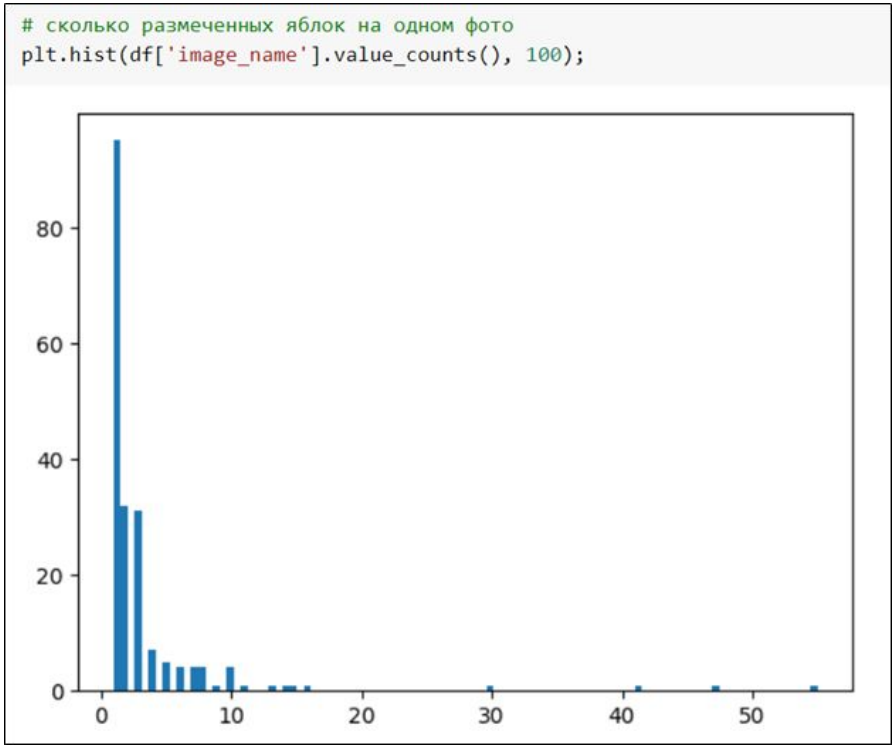

```
plt.hist(df['box_area'], 100);
```



Анализ данных с визуализацией

Гистограмма количества размеченных яблок на одном фото

```
# сколько размеченных яблок на одном фото
plt.hist(df['image_name'].value_counts(), 100);
```



Тот факт, что в датасете мало фотографий с количеством размеченных яблок > 10, может ограничить пригодность обученной на нем модели.

Возможно, модель будет лучше справляться с детекцией малого количества яблок.

Если в ходе обучения модель правильно предсказала яблоко, но оно не было размечено – то ошибка будет неправильно посчитана, и у модели хуже обновятся веса. Возможно, из-за того, что на фото не всегда размечены все яблоки, модель будет показывать немного пониженные результаты на любых картинках, в том числе с малым количеством яблок.

Выбросы и очистка

Картинки, к которым отсутствует аннотация, не попали в пересобранный датасет.

3 Преобразование данных

Картинки

- Все картинки требуют перевода в Pytorch-тензоры.
- К тренировочным картинкам также будет применяться рандомное отражение по горизонтали, с целью расширить датасет.
- Так как выбранные (в следующих главах) модели принимают картинки любого размера, resize не требуется.
- В будущем можно добавить нормализацию и исследовать ее влияние на качество предсказаний.
- Также можно попробовать добавить дополнительные аугментации.

Таргеты детекции и сегментации

Таргетами детекции и сегментации являются координаты точек полигона и координаты bounding box'ов. При отражении и других трансформациях картинки, эти координаты тоже должны измениться. Пакет torchvision.transforms.v2.transforms позволяет применять преобразования и к картинке, и к таргетам.

Лейблы классов

Классы были закодированы следующим образом:

0 – фон,
1 – normal,
2– parsha,
3 – mechanic

4 Datasats, Dataloaders и разделение данных

- Был написан Pytorch-датасет, наследующийся от torch.utils.data.Dataset
- Был установлен seed для воспроизводимости случайного разбиения.
- Данные были случайным образом разделены на тренировочные, валидационные и тестовые в пропорции 8:1:1 (Общепринятой практикой является разделение датасета в отношении 60/20/20, однако наш датасет мал, и лучше оставить больше экземпляров для обучения).
- На основе каждого под-датасета был создан Dataloader

Размеры датасетов

```
len(dataset_train), len(dataset_val), len(dataset_test)
```

```
(156, 20, 20)
```

5 Модели

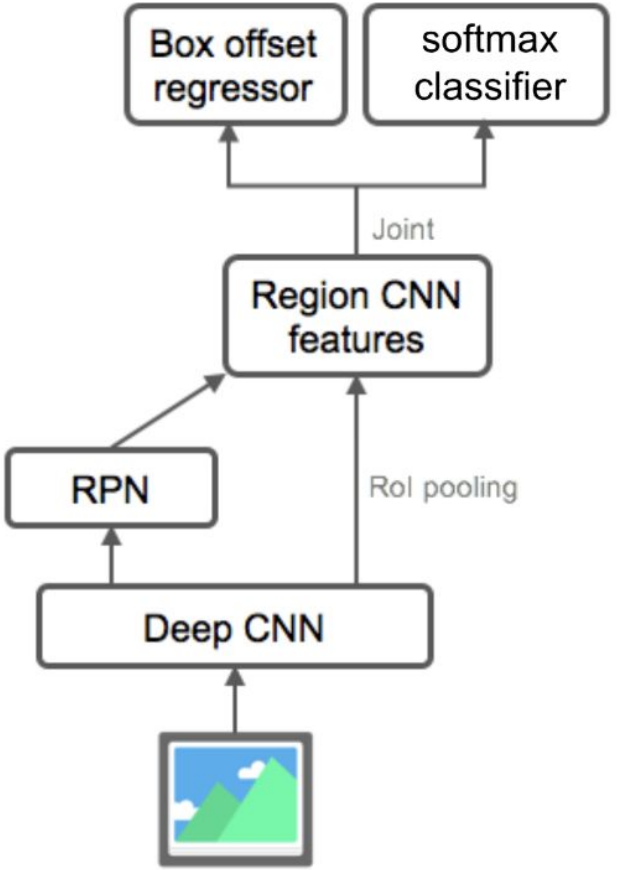
Faster R-CNN

Архитектура включает две основные части – Region Proposal Network (RPN) и Fast R-CNN. Сначала RPN генерирует несколько пропозалов регионов, которые могут содержать объекты, а затем Fast R-CNN классифицирует их и определяет их границы.

Функция ошибки: multi-task loss, который состоит из трех частей – классификации объектов, определения границ и совместного представления.

Методы регуляризации (weight decay, dropout, batch normalization) также используются для избежания переобучения.

При обучении модели Faster R-CNN используется алгоритм обратного распространения ошибки

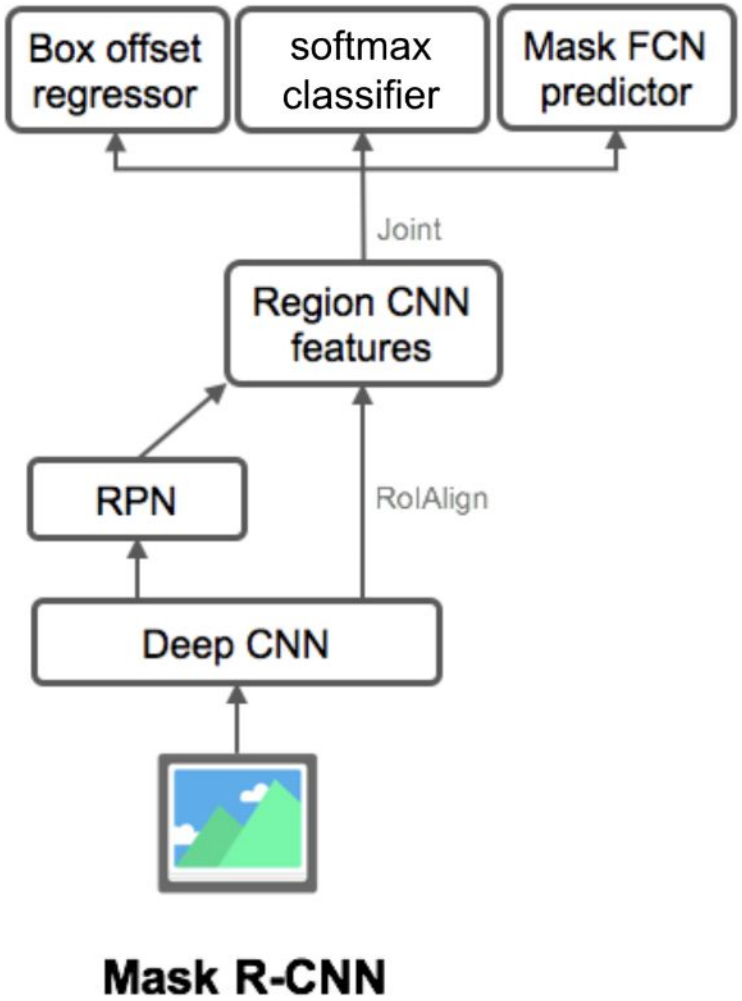


Faster R-CNN

Mask R-CNN

Mask R-CNN расширяет Faster R-CNN, добавляя третью часть - сегментацию (маску) объектов. Для этого в той же сети добавляется branch, который предсказывает маски объектов.

Функция ошибки также включает в себя компоненту, отвечающую за маску объектов.



6 Реализация модели на PyTorch + вариации архитектуры

Возьмем предобученную модель Faster R-CNN и поменяем последний слой, либо возьмем предобученную модель Mask R-CNN и заменим не только последний слой, но и часть, отвечающую за сегментацию масок.

При необходимости можно настроить количество размораживаемых последних слоев моделей (по умолчанию равное 3).

Загрузчик данных был реализован ранее.

Вывод промежуточных и финальных метрик качества и диагностических параметров процесса обучения модели реализован посредством логирования на W&B.

Для подсчета метрик импортируются дополнительные скрипты pytorch utils с классом COCO-метрик.

7 Обучение

Настройка гиперпараметров

Количество эпох: 10

Размер батча:

- 1 для модели Mask
- 2 для модели Faster

Optimizer: SGD

```
# optimizer
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)

# learning rate scheduler
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                               step_size=3,
                                               gamma=0.1)
```

Проведение обучения

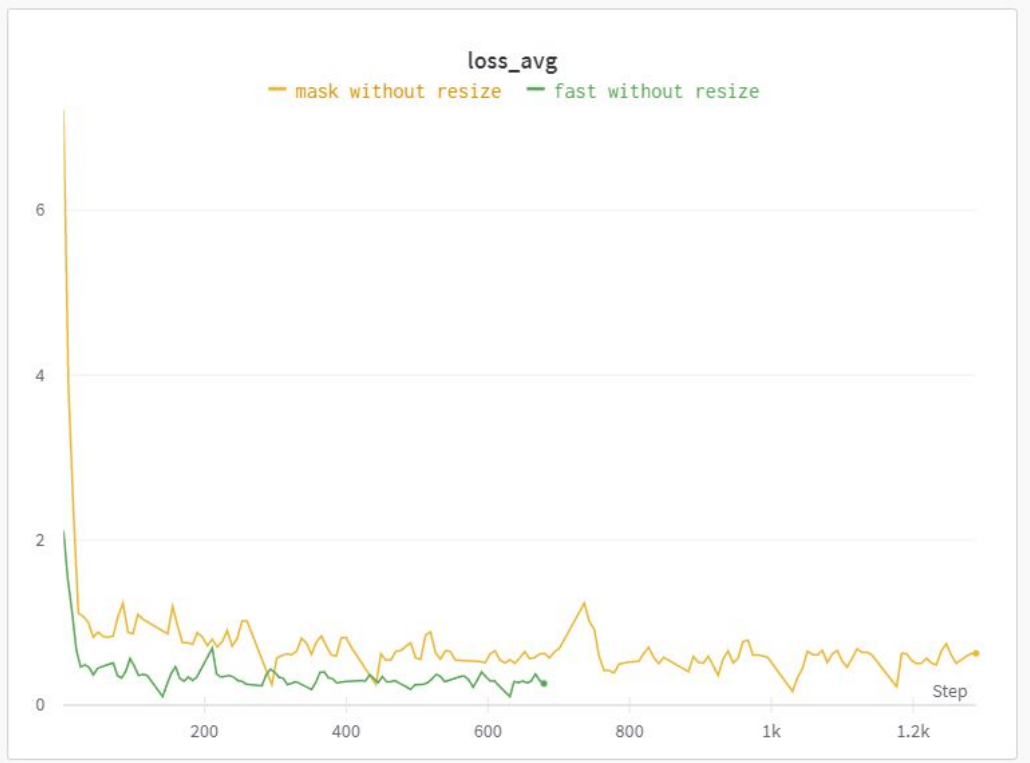
```
# train on train + evaluate on val
train(model_mask, dataloader_masks_train, dataloader_masks_val, 'mask1')
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.774
0.6853356408431873
Epoch: [7]  [  0/156]  eta: 0:02:52  lr: 0.000050  loss: 0.1663 (0.1663)  loss_class
Epoch: [7]  [ 10/156]  eta: 0:01:07  lr: 0.000050  loss: 0.2508 (0.3348)  loss_class
Epoch: [7]  [ 20/156]  eta: 0:00:59  lr: 0.000050  loss: 0.3166 (0.4316)  loss_class
Epoch: [7]  [ 30/156]  eta: 0:01:13  lr: 0.000050  loss: 0.6590 (0.5399)  loss_class
Epoch: [7]  [ 40/156]  eta: 0:01:06  lr: 0.000050  loss: 0.5033 (0.5286)  loss_class
Epoch: [7]  [ 50/156]  eta: 0:01:06  lr: 0.000050  loss: 0.4171 (0.5664)  loss_class
Epoch: [7]  [ 60/156]  eta: 0:00:57  lr: 0.000050  loss: 0.4759 (0.5675)  loss_class
Epoch: [7]  [ 70/156]  eta: 0:00:52  lr: 0.000050  loss: 0.4299 (0.5526)  loss_class
Epoch: [7]  [ 80/156]  eta: 0:00:49  lr: 0.000050  loss: 0.5583 (0.5803)  loss_class
Epoch: [7]  [ 90/156]  eta: 0:00:40  lr: 0.000050  loss: 0.5998 (0.5767)  loss_class
Epoch: [7]  [100/156]  eta: 0:00:34  lr: 0.000050  loss: 0.5390 (0.5710)  loss_class
Epoch: [7]  [110/156]  eta: 0:00:28  lr: 0.000050  loss: 0.4889 (0.5558)  loss_class
Epoch: [7]  [120/156]  eta: 0:00:22  lr: 0.000050  loss: 0.3402 (0.5706)  loss_class
Epoch: [7]  [130/156]  eta: 0:00:16  lr: 0.000050  loss: 0.3326 (0.5750)  loss_class
Epoch: [7]  [140/156]  eta: 0:00:10  lr: 0.000050  loss: 0.6104 (0.5804)  loss_class
Epoch: [7]  [150/156]  eta: 0:00:03  lr: 0.000050  loss: 0.5574 (0.5839)  loss_class
Epoch: [7]  [155/156]  eta: 0:00:00  lr: 0.000050  loss: 0.5574 (0.5817)  loss_class
Epoch: [7] Total time: 0:01:40 (0.6448 s / it)
creating index...
index created!
Test:  [ 0/20]  eta: 0:01:53  model_time: 0.1969 (0.1969)  evaluator_time: 2.7774 (2
Test:  [19/20]  eta: 0:00:01  model_time: 0.1689 (0.1729)  evaluator_time: 0.2882 (1
Test: Total time: 0:00:38 (1.9235 s / it)
Averaged stats: model_time: 0.1689 (0.1729)  evaluator_time: 0.2882 (1.4799)
Accumulating evaluation results...
DONE (t=0.02s).
Accumulating evaluation results...
DONE (t=0.02s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.689
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.850
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.761
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.689
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.447
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.810
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.813
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.813
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.640
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.850
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.761
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.640
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.423
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.765
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.772
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.772
0.6891986938841667
```

```
# train on train + evaluate on val
train(model_fast, dataloader_train, dataloader_val, 'fast1')

Epoch: [8]  [70/78]  eta: 0:00:06  lr: 0.000050  loss: 0.2155 (0.3122)  loss_classi
Epoch: [8]  [77/78]  eta: 0:00:00  lr: 0.000050  loss: 0.2212 (0.3038)  loss_classi
Epoch: [8] Total time: 0:01:04 (0.8255 s / it)
creating index...
index created!
Test:  [ 0/10]  eta: 0:00:07  model_time: 0.3140 (0.3140)  evaluator_time: 0.0087 (
Test:  [ 9/10]  eta: 0:00:00  model_time: 0.2788 (0.2894)  evaluator_time: 0.0087 (
Test: Total time: 0:00:06 (0.6010 s / it)
Averaged stats: model_time: 0.2788 (0.2894)  evaluator_time: 0.0087 (0.0087)
Accumulating evaluation results...
DONE (t=0.02s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.492
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.664
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.573
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.276
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.499
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.455
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.735
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.735
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.450
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.741
0.4924018061341396 4
Epoch: [9]  [ 0/78]  eta: 0:01:24  lr: 0.000005  loss: 0.1040 (0.1040)  loss_classi
Epoch: [9]  [10/78]  eta: 0:01:05  lr: 0.000005  loss: 0.1553 (0.2857)  loss_classi
Epoch: [9]  [20/78]  eta: 0:00:49  lr: 0.000005  loss: 0.1741 (0.2658)  loss_classi
Epoch: [9]  [30/78]  eta: 0:00:43  lr: 0.000005  loss: 0.2191 (0.2909)  loss_classi
Epoch: [9]  [40/78]  eta: 0:00:32  lr: 0.000005  loss: 0.1781 (0.2673)  loss_classi
Epoch: [9]  [50/78]  eta: 0:00:23  lr: 0.000005  loss: 0.2411 (0.2912)  loss_classi
Epoch: [9]  [60/78]  eta: 0:00:15  lr: 0.000005  loss: 0.3186 (0.3035)  loss_classi
Epoch: [9]  [70/78]  eta: 0:00:06  lr: 0.000005  loss: 0.1712 (0.2934)  loss_classi
Epoch: [9]  [77/78]  eta: 0:00:00  lr: 0.000005  loss: 0.1712 (0.2946)  loss_classi
Epoch: [9] Total time: 0:01:04 (0.8206 s / it)
creating index...
index created!
Test:  [ 0/10]  eta: 0:00:08  model_time: 0.3276 (0.3276)  evaluator_time: 0.0218 (
Test:  [ 9/10]  eta: 0:00:00  model_time: 0.2704 (0.2936)  evaluator_time: 0.0097 (
Test: Total time: 0:00:06 (0.6315 s / it)
Averaged stats: model_time: 0.2704 (0.2936)  evaluator_time: 0.0097 (0.0100)
Accumulating evaluation results...
DONE (t=0.02s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.495
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.665
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.576
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.287
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.500
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.442
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.727
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.739
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.500
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.743
0.4946561060979965
That's it!
```

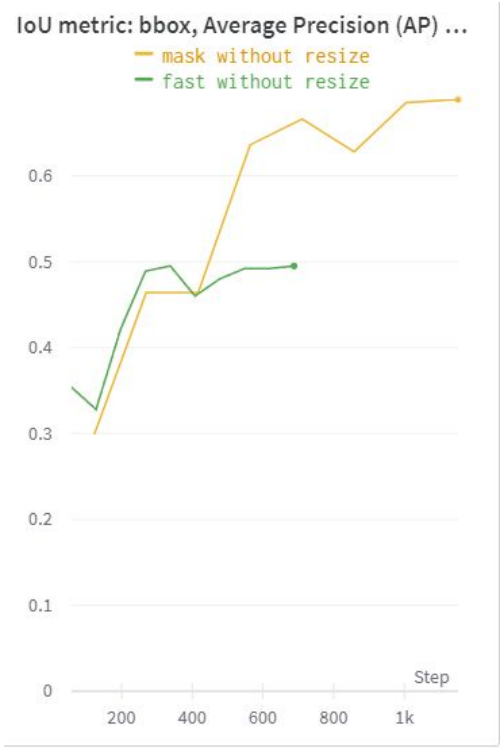
8 Итоги обучения

Loss на train-выборке

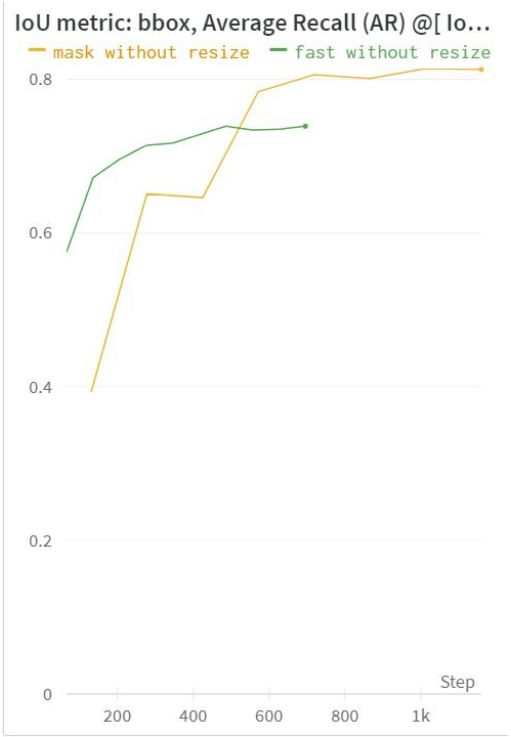


Оценки метрик на Val

Основная метрика COCO mAP на валидационной выборке



Доп. метрика COCO mAR на валидационной выборке



Модель Faster достигла показателя mAP=0.495 (и mAR=0.442) после 10 эпох.

Модели Mask удалось достичь mAP=0.689 (и mAR=0.447) на 8 эпохах.

Т.к. в случаях обеих моделей значение основной метрики не стало со временем понижаться, то пока что нельзя говорить о переобучении, и в будущем можно попробовать увеличить количество эпох.

Оценки метрик на test

Mask test

```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.490
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.646
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.570
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.087
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.500
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.329
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.613
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.669
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.082
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.687
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.453
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.638
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.503
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.058
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.463
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.314
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.582
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.627
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.055
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.644
0.4899266768821241
```

Faster test

```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.559
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.717
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.632
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.026
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.583
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.414
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.555
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.666
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.035
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.702
0.558512157023643
```

Модель Mask на тестовой выборке имеет mAP=0.490 (и mAR=0.329).

Модель Faster на тестовой выборке имеет mAP=0.559 (и mAR=0.414).

9 Оценки вариаций архитектур моделей

На предыдущих шагах были сравнены 2 модели с похожей архитектурой: Faster R-CNN и Mask R-CNN.

Ориентиры по метрикам были достигнуты. Модель Faster с наивысшим результатом по основной метрике на тесте и наименьшим временем работы (за счет отсутствия отдельной части для сегментации масок) выбирается для инференса.

Однако, хотя модель Faster была успешнее на тестовой выборке, модель Mask сильно превосходила ее на валидационной. Возможно, причина в том, что в тесте много плохо размеченных или не размеченных яблок. Модель Mask могла правильно найти все яблоки, но многие ее правильные предсказания оценились как ошибочные.

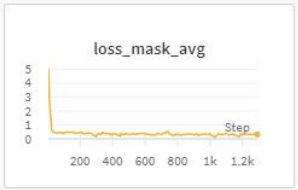
10 Тонкая настройка

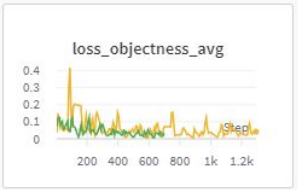
В настройке гиперпараметров нет необходимости, так как ориентиры по метрикам достигнуты.

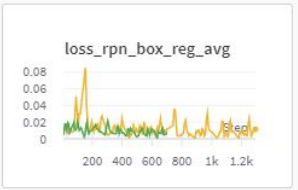
11 Результаты

Основные графики обучения были представлены в Итогах обучения. Дополнительные графики обучения (losses):

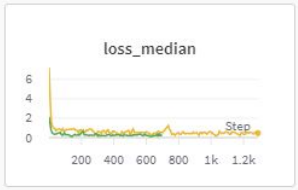
avg 6 Add panel

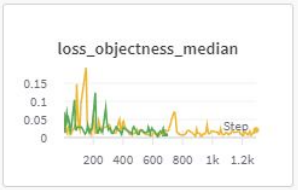


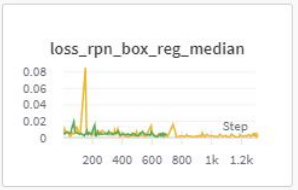




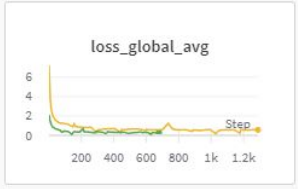
median 6 Add panel

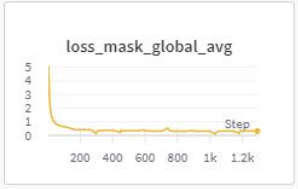


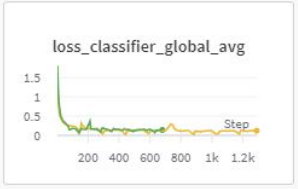


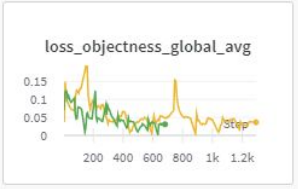


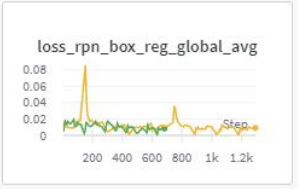
global_avg 6 Add panel

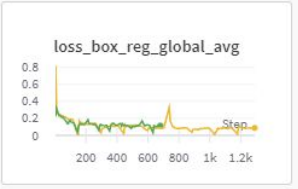




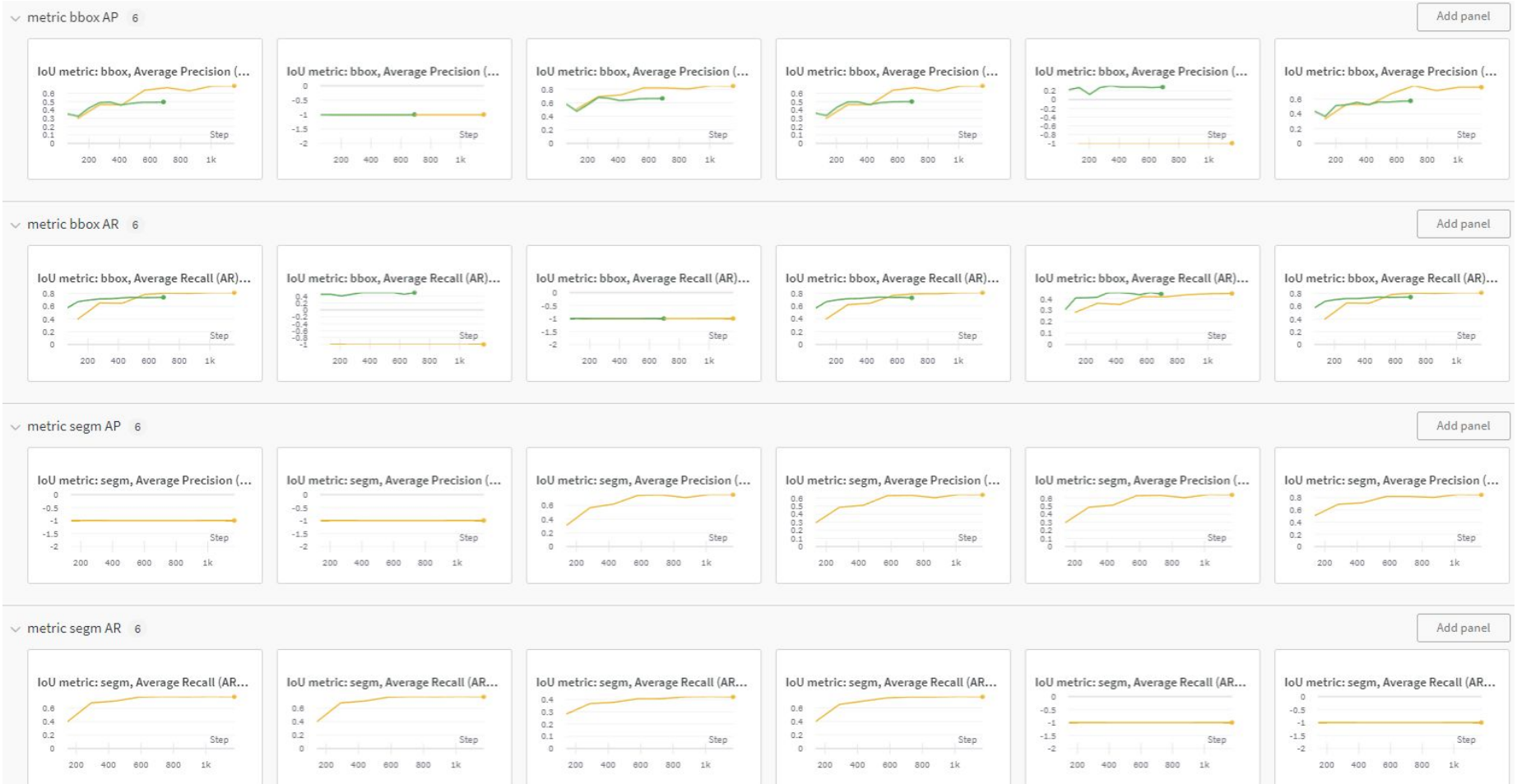








Дополнительные графики обучения (metrics)



Дополнительные графики обучения (other)

Таблица сравнения моделей

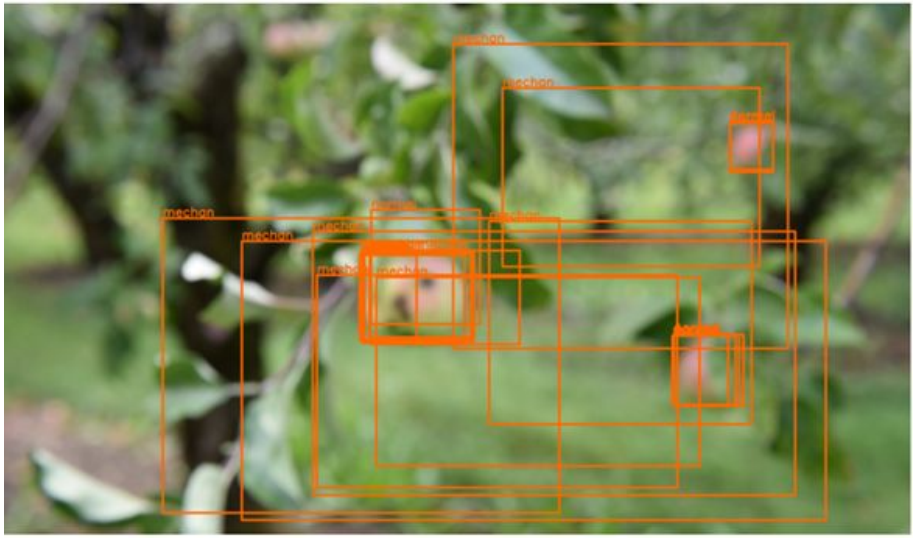
	COCO mAP (val)	COCO mAR (val)	COCO mAP (test)	COCO mAR (test)
Faster R-CNN	0.495	0.442	**0.559**	**0.414**
Mask R-CNN	**0.689**	**0.447**	0.490	0.329

Выводы по моделям для двух задач

- В целом, обе модели достигли удовлетворительных результатов и могут быть пригодны для использования в производстве.
- В ходе валидации и тестирования модели продемонстрировали нестабильное поведение. При анализе датасета были выявлены недочеты разметки, которые и могли обусловить различное поведение моделей на разных выборках. (Возможно, причина в том, что в тесте много плохо размеченных или не размеченных яблок. Модель Mask могла правильно найти все яблоки, но многие ее правильные предсказания оценились как ошибочные.)
- Чтобы улучшить результаты, стоит в первую очередь попробовать пересмотреть датасет и во вторую очередь исследовать различные гиперпараметры и подобрать более современные архитектуры.

Инференс: предсказание модели на элементе тестовой выборки

По умолчанию. Выходные значения модели нуждаются в обработке.



После отбрасывания элементов с оценкой вероятности <= 0.5 и установки IOU treshold=0.3 (исключение рамок, перекрывающих друг друга).

Спасибо за внимание!

Репозиторий доступен по ссылке:
https://github.com/c-nemo/Apple-Multiclass-Detection