Фаза 2 • Неделя 2 • Понедельник

Локализация объектов • Object localization

Сегодня

Эльбрус буткемп

- что делать, если нужно не просто классифицировать изображение, но и найти участок на изображении, содержащий объект?
- какую архитектуру модели можно для этого использовать?
- а лоссы?

Классификация

Эльбрус буткемп

```
label: cat
```

```
label: dog
```

Все как всегда: берем ResNet (Inception, ViT, ConvNeXt и тд), файнтюним его на свою задачу и получаем ответ.

Классификация vs. Локализация

Эльбрус буткемп

```
label: cat
```

```
label: cat, bbox coordinates: x1, y1, x2, y2
```

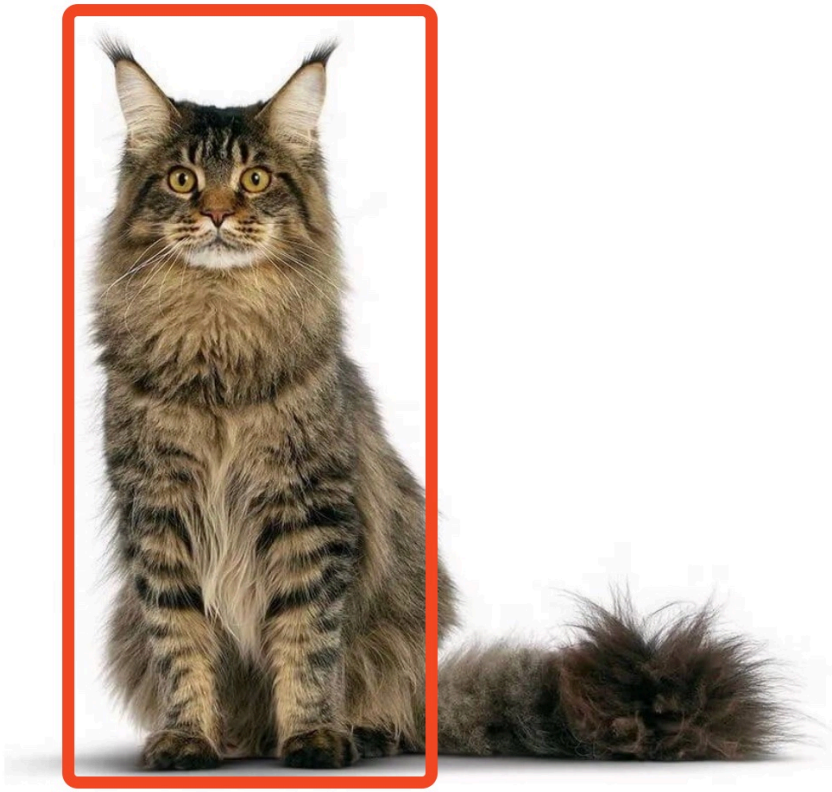
Bounding box, bbox, ограничивающая рамка – регион, в котором находится объект. Как правило прямоугольный.

Локализация

ResNet (ViT, Inception, ...) остается, только теперь будем выход этой модели отправлять в:

- маленькую полносвязную сеть, которую будем обучать на классификацию
- маленькую полносвязную сеть, которую будем обучать на регрессию 4 координат рамки – bounding box'a

```
label: cat, bbox coordinates: x1, y1, x2, y2
```



Выборка

Эльбрус буткемп

Классификация

```
train:
- cat
  - cat_1.jpg
  - cat_2.jpg
  ...
- dog
  - dog_1.jpg
  - dog_2.jpg
  ...

valid:
  ...
```

Указываем пути в `ImageFolder` и все прекрасно

Выборка

Классификация

```
train:
- cat
  - cat_1.jpg
  - cat_2.jpg
  ...
- dog
  - dog_1.jpg
  - dog_2.jpg
  ...

valid:
  ...
```

Указываем пути в `ImageFolder` и все прекрасно

Локализация / детекция

```
train:
- cat
  - cat_1.jpg
  - cat_1.txt (xml/json)
  - cat_2.jpg
  - cat_2.txt (xml/json)
  ...
- dog
  - dog_1.jpg
  - dog_1.txt (xml/json)
  - dog_2.jpg
  - dog_2.txt (xml/json)
  ...

valid:
...
```

Теперь каждой картинке соответствует файл с координатами прямоугольника, в котором расположен искомый объект

Файл с координатами?!


```
{
    "description": "",
    "tags": [
        {
            "id": 12345,
            "name": "train",
            "value": null,
            "labelerLogin": "roboflow",
            "createdAt": "2000-01-01T00:00:00.000Z",
            "updatedAt": "2000-01-01T00:00:00.000Z"
        }
    ],
    "size": {
        "height": 191,
        "width": 264
    },
    "objects": [
        {
            "id": 12345,
            "classId": 67890,
            "description": "",
            "geometryType": "polygon",

            "tags": [],
            "classTitle": "helmet",
            "points": {
                "exterior": [
                    [
                      ...
```



```
1 0.617 0.3594420600858369 0.114 0.17381974248927037
1 0.094 0.38626609442060084 0.156 0.2360515021459274
1 0.295 0.3959227467811159 0.13 0.19527896995708155
1 0.785 0.398068669527897 0.07 0.14377682403433475
1 0.886 0.40879828326180256 0.124 0.18240343347639484
1 0.723 0.398068669527897 0.102 0.1609442060085837
1 0.541 0.35085836909871243 0.094 0.16952789699570817
1 0.428 0.4334763948497854 0.068 0.1072961373390558
1 0.375 0.40236051502145925 0.054 0.1351931330472103
1 0.976 0.3927038626609442 0.044 0.17167381974248927
...
```


Может выглядеть буквально как угодно, но ряд стандартов есть

Файл с координатами?!

```
<annotation>
        <folder>single mushroom</folder>
        <filename>filename.jpg</filename>
  ...
        <size>
                <width>227</width>
                <height>227</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>mushroom</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>53</xmin>
                        <ymin>28</ymin>
                        <xmax>187</xmax>
                        <ymax>209</ymax>
                </bndbox>
        </object>
</annotation>
```

Часто в реализациях готовых моделей эти файлы парсятся автоматически, но сегодня мы будем разбирать файл слева вручную в классе `Dataset`.

Обычная иерархическая структура, когда встретитесь с парсингом сайтов – вспомните и это.

Метрики для bounding box'ов и функции потерь для модели

Метрики и лоссы для классификации

Эльбрус буткемп

- Для классификации любые метрики: accuracy, precision, recall, f1, ...
- Для классификации обычный лосс: кросс энтропия

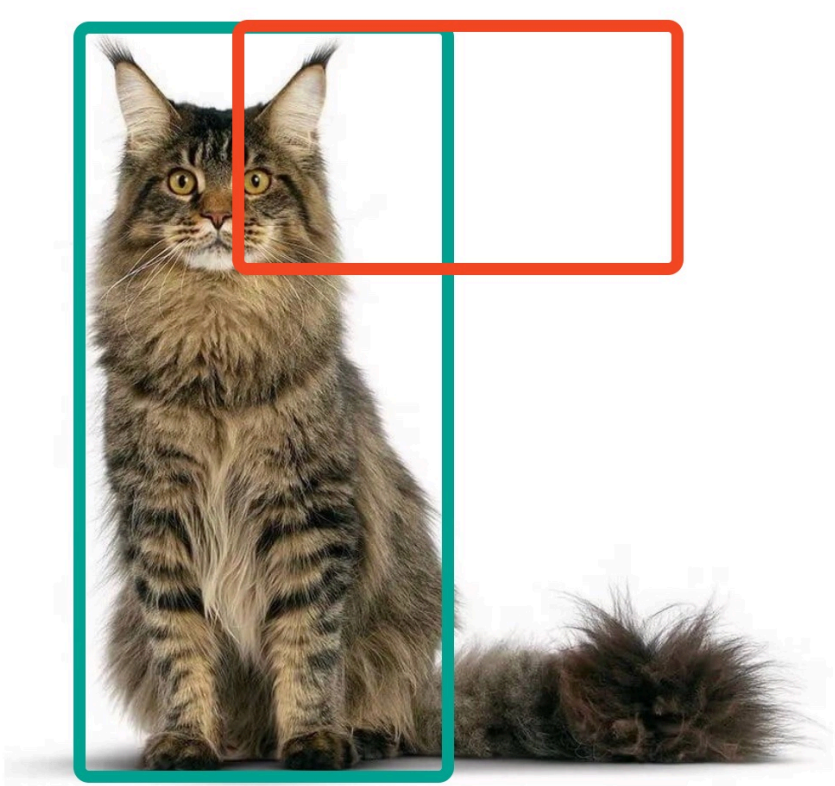
IoU • Intersection over union • Jaccard similiarity

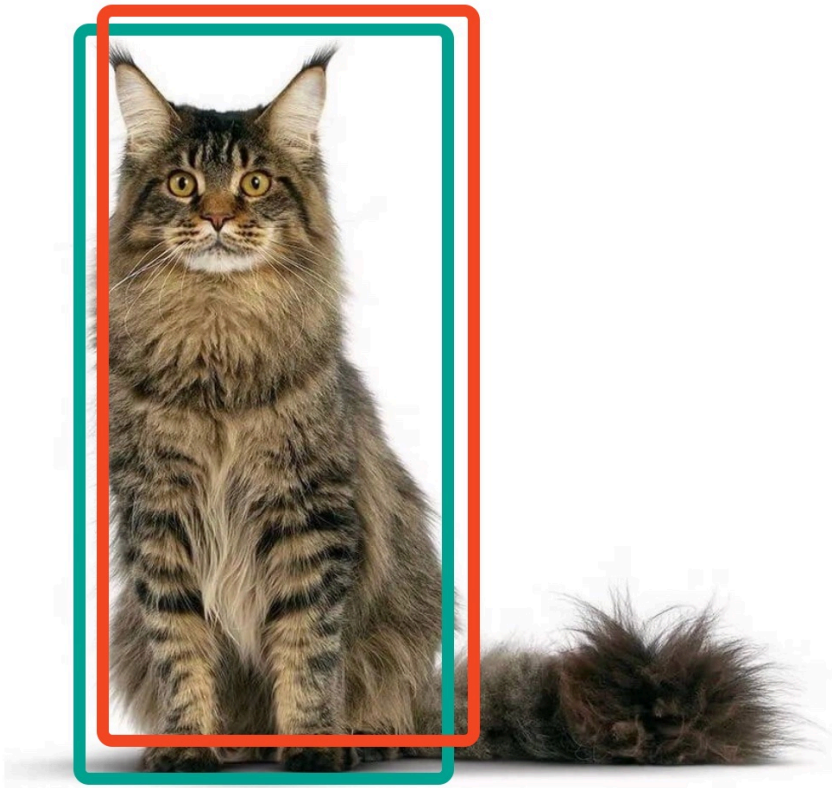
Эльбрус буткемп

Возвращаемся к коту. Его настоящая рамка выглядит так:

Что лучше?

● – предсказанная рамка

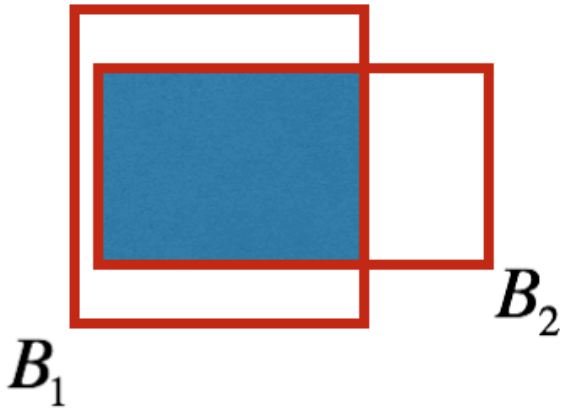




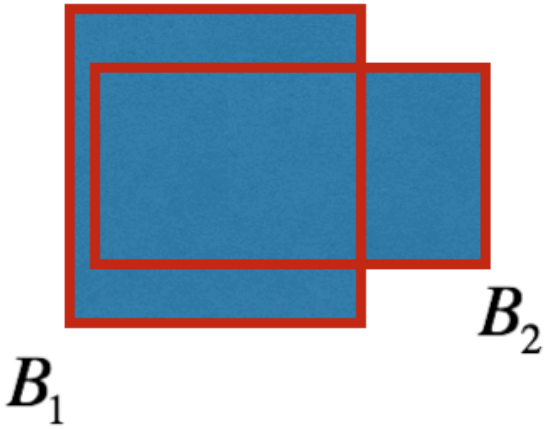
IoU • Intersection over union • Jaccard similiarity

Эльбрус буткемп

Intersection

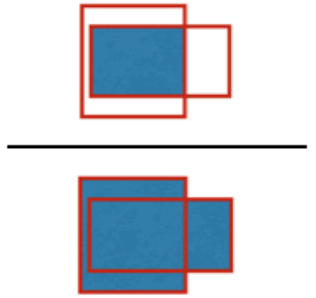


Union



Intersection over Union

$$IoU = \frac{B_1 \cap B_2}{B_1 \cup B_2} =$$



- Задача локализации – **максимизировать** эту метрику
- Будем использовать её из библиотеки `torchmetrics`
- На вход будем передавать предсказанные координаты, в качестве таргета – истинные из выборки
- А завтра рассмотрим её «настоящий» вариант

Лосс для координат bbox'ов

Эльбрус буткемп

Формально – любой регрессионный:

- MSE (L2Loss)
- MAE (L1Loss)

Координаты – обычные числа, нам нужно минимизировать между ними разницу.

Буквально

Эльбрус буткемп

```
print(true_box[:2])

>> tensor([
  [0.1674, 0.1189, 0.9471, 0.6828],
  [0.4449, 0.2555, 0.6828, 0.5551]
  ])

print(pred_box[:2])

>> tensor([
  [0.1700, 0.2217, 0.8329, 0.7919],
  [0.2043, 0.2542, 0.7981, 0.7587]])

print(true_box[:2] - coords[:2])

>> tensor([
  [-0.0026, -0.1028,  0.1143, -0.1091],
  [ 0.2406,  0.0014, -0.1153, -0.2036]])

# На самом деле вместо этого можно просто использовать F.mse_loss(true_box, pred_box)
print((true_box[:2] - coords[:2]).pow(2).sum()/true_box[:2].size(0))

>> tensor(0.0741)
```

Объединение лоссов

Эльбрус буткемп

Теперь есть два лосса:

- кросс энтропия для классификации
- регрессионный лосс для координат

Итоговый лосс – обычная сумма:

```
loss_clf = F.cross_entropy(logits, torch.Tensor(true_label))
loss_coords = F.mse(coords, true_box)
loss = loss_clf + loss_coords
loss.backward()
optimizer.step()
optimizer.zero_grad()
```

Объединение лоссов

Эльбрус буткемп

Теперь есть два лосса:

- кросс энтропия для классификации
- регрессионный лосс для координат

Итоговый лосс – обычная сумма:

```
loss_clf = F.cross_entropy(logits, torch.Tensor(true_label))
loss_coords = F.mse(coords, true_box)
loss = loss_clf + loss_coords
loss.backward()
optimizer.step()
optimizer.zero_grad()
```

Итоговый лосс формулой выглядит веселее

Эльбрус буткемп

$$L(c^t, c^p, x^{1t}, y^{1t}, x^{2t}, y^{2t}, x^{1p}, y^{1p}, x^{2p}, y^{2p}) =$$

$$\frac{1}{N}\sum_{i=1}^{0}[(x_i^{1p} - x_i^{1t})^2 + (x_i^{2p} - x_i^{2t})^2 + (y_i^{1p} - y_i^{1t})^2 + (y_i^{2p} - y_i^{2t})^2] -$$

$$- \frac{1}{N}\sum_{i}^{N}\sum_{j}^{C-1} c_{ij}^t \log c_{ij}^p$$

Логирование

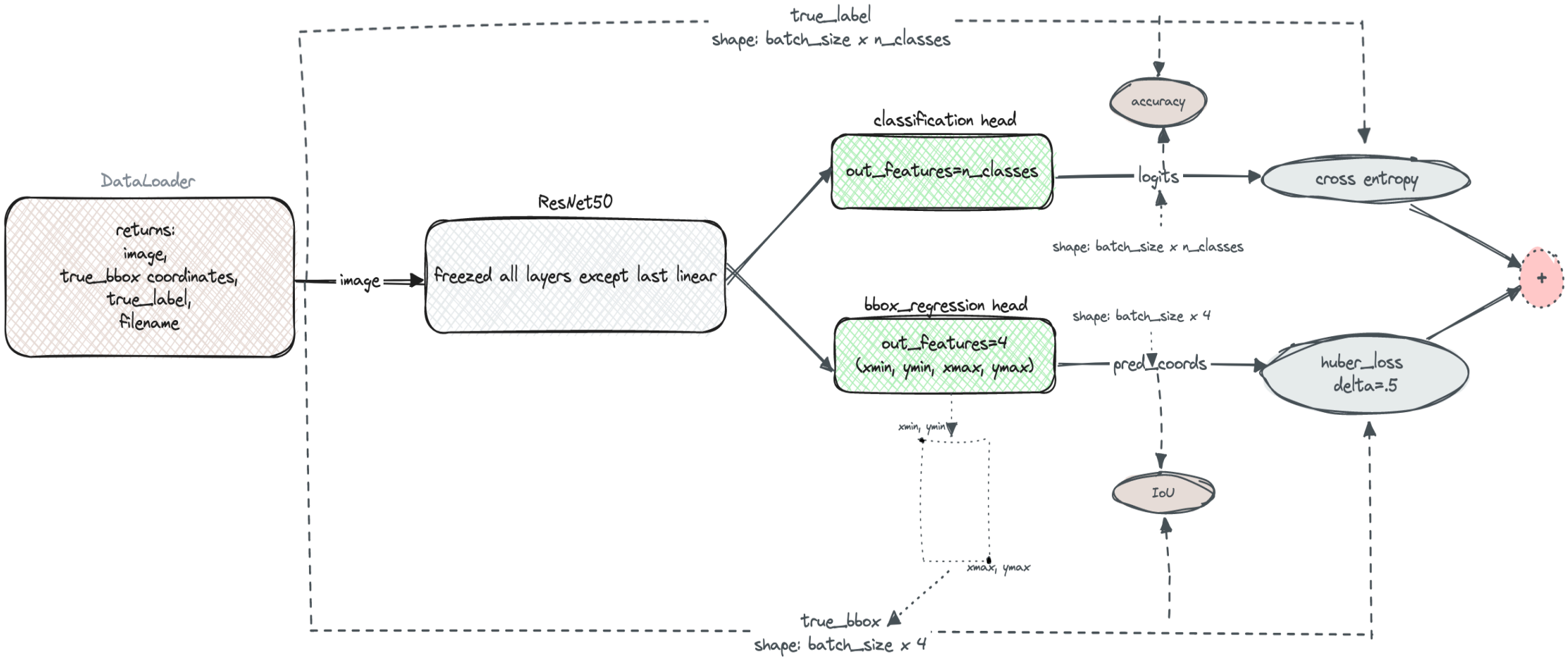
Эльбрус буткемп

Во время обучения и валидации придется собирать намного больше данных:

1. Лосс классификации
2. Лосс регрессии
3. Суммарный лосс
4. Метрику классификации
5. Метрику IoU

> Итого 10 величин для отслеживания успешности обучения

Архитектура



Итоги

Эльбрус буткемп

- решаем сразу две задачи одной моделью: классификация и регрессия на координаты bbox'ов
- две задачи – две характеристики у каждого элемента выборки: класс объекта на картинке и координаты региона
- две задачи – две у «головы» модели: одна прогнозирует класс, вторая – координаты рамки
- лоссы привычные, но теперь мы их суммируем
- новое: IoU для измерения точности локализации рамки по отношению к истинной