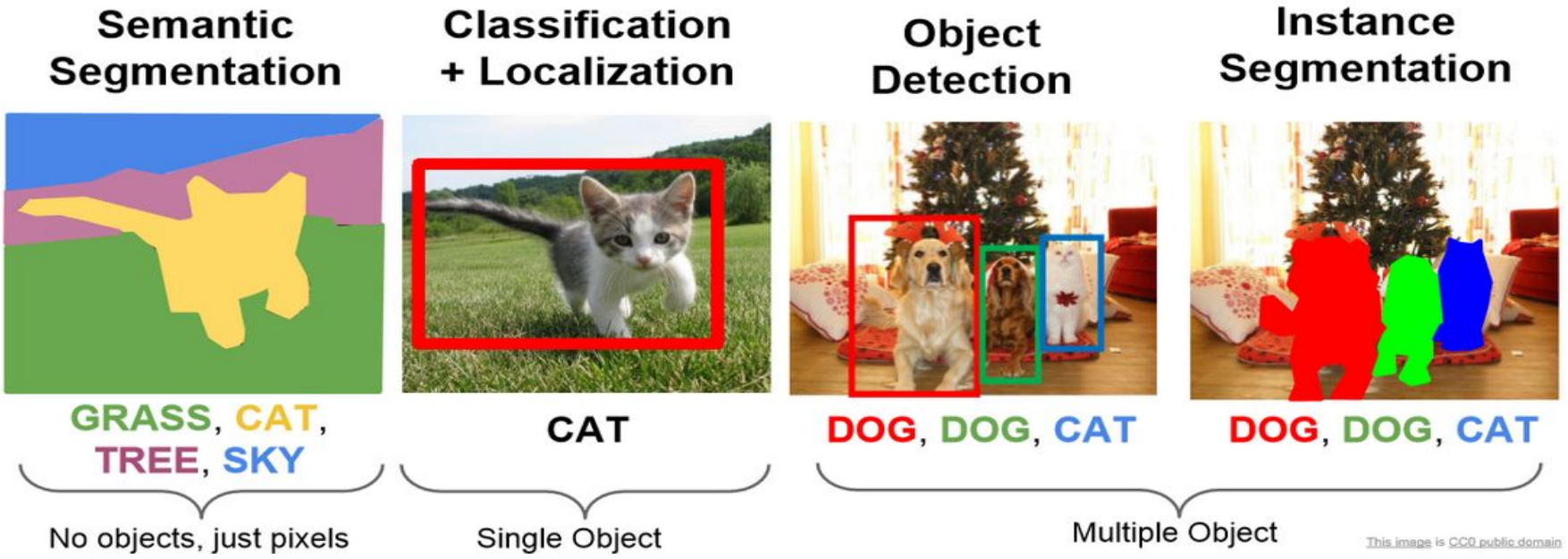
Object Detection

Cats and dogs

Задачи компьютерного зрения

Нас интересует эта задача



Задача: object detection

Задача состоит в определении местонахождения объектов на картинке и их класс. Местонахождение объекта в задачах object detection определяется через координаты четырехугольной рамки (bounding box) внутри которой располагается наш объект. Соответственно разметка каждого объекта на картинке имеет вид

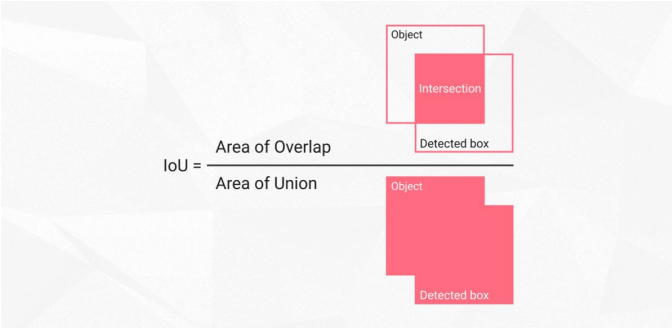
("class_label" , "xmin", "ymin", "xmax", "ymax")

("class_label" ,"x", "y", "width", "height")   где x, y центр рамки

Популярные датасеты на чем обычно обучают модели

PASCAL VOC

Метрики: как сравнивают модели?



Нужные данные до метрик

IoU - Intersection over Union

Precision: TP(correct class and IoU >= Threshold) FP – one of two fails

Метрики:

AP (Average Precision)

mAP(mean Average Precision)

AP не просто средний как обычно считают в задачах классификации, а считается немного сложнее

Метрики: AP и mAP

Чтобы понять AP нам нужно построить график precision vs recall. Точность модели считается хорошей если мы держим на высоком уровне precision при увеличении recall.

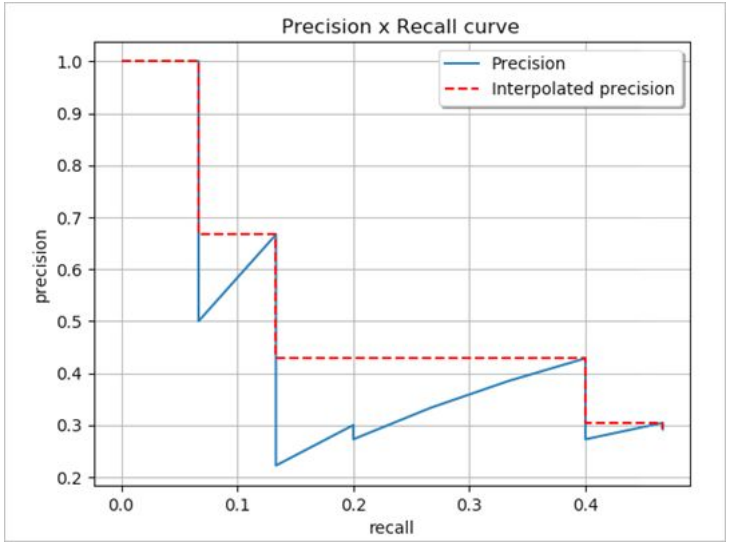
A
$$AP = \int_0^1 p(r)dr$$
площадь под графиком:

Чтобы облегчить калькуляцию мы сглаживаем precision до максимума precision с следующего справа от recall:

$$AP = \Sigma \left(r_{n+1} - r_n \right) p_{interp}(r_{n+1})$$

$$p_{interp}(r_{n+1}) = \max_{\tilde{r} \geq r_{n+1}} p(\tilde{r})$$

mAP - средний AP всех классов. Нужно не забывать про IoU >= Threshold который определяет как будет считаться precision. Поэтому результатах алгоритмов детекции указывают какой **threshold** был использован для расчета: mAP@.5 mAP@.95 соответственно threshold 0.50 и 0.95

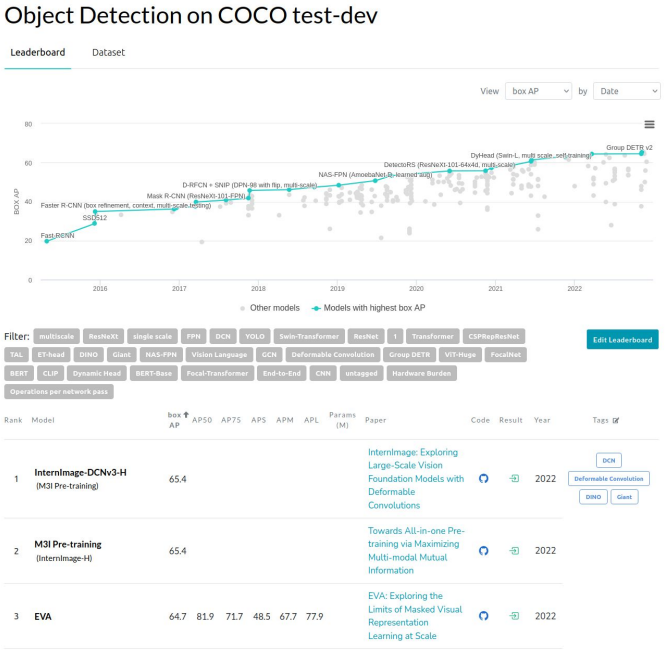


Есть и другая калькуляция mAP (COCO mAP) где варьируют threshold . mAP@[0.5:0.05:0.95] где шагом 0,05 считаем AP и усредняют результаты по threshold и по всем классам. Но мы не используем это метрику.

Выбор алгоритма:

Если зайти на сайт paper_with_code можно увидеть очень много реализаций, у которых точность высокая.

Но мой выбор пал на два алгоритма: Yolov5s и EfficientDet D0. Причина в том что оба алгоритма заявляют о быстрой скорости детекции при этом имея хорошую точность.

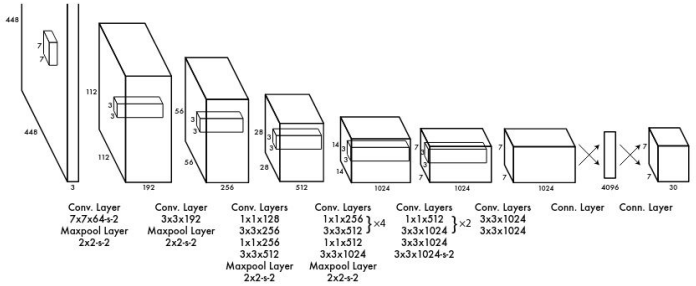


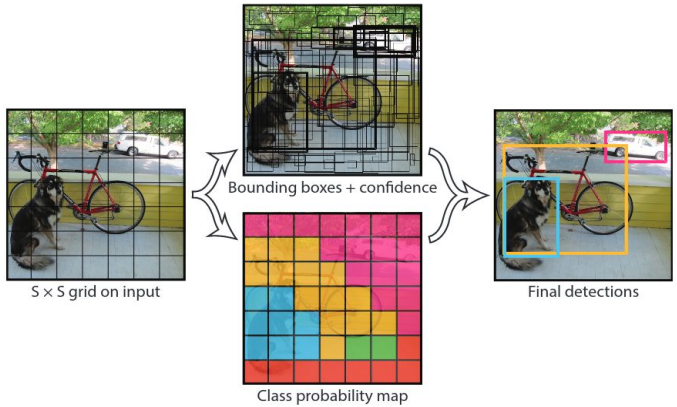
Семейства архитектур Yolo

Основная новизна Yolo на тот момент:

Single Stage Detection - в отлиции от Faster-RCNN где алгоритм состоял из 2 частей как ROI Polling и Classifier, смотрел на картинку один раз.

Есть алгоритм SSD который тоже один раз использует картинку. В SSD модель выбирает для клетки ancor box наибольшей IoU старается приблизиться максимально groundtruth bbox. На том же месте Yolo выдает несколько вариантов bbox-а, из них выбирает максимум IOU и на нем обучается, исключая через non max suppresion.





YOLOv*

YOLOv2:

- Batch normalization
- Anchor boxes (Надо понимать что ancorbox не sliding window который идет по картинке и детектит объект. А наоборот мы сперва прогнозируем центр объекта и от него отталкиваясь подбираем ancorbox).
- Darknet-19 architecture

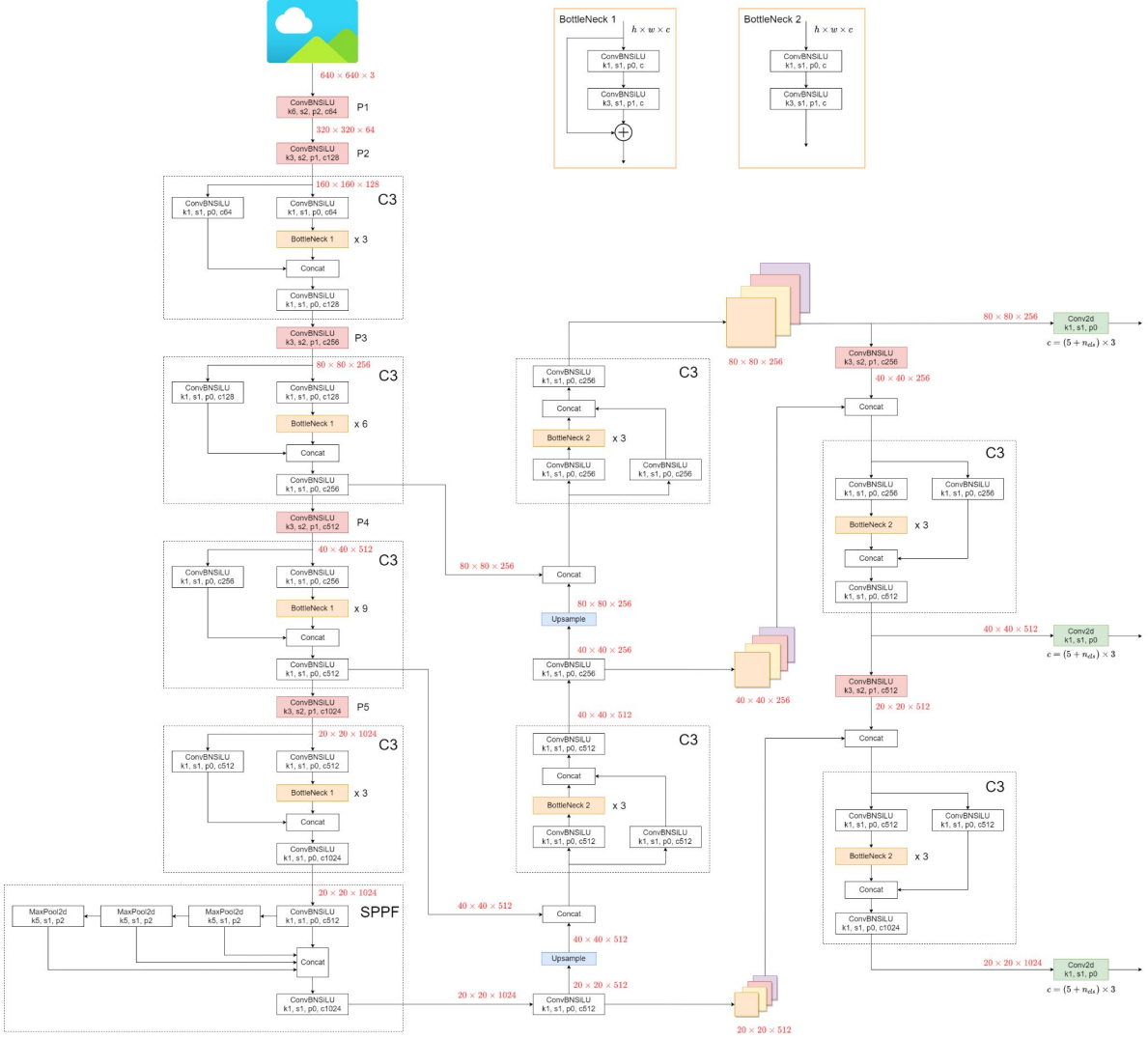
YOLOv3:

- **FPN** (**Feature Pyramid Network**) используя 3 прогноза на 3 слоях сети дал возможность детектировать очень маленькие объекты
- Перевели прогноз класса на задачу классификации

YOLOv4:

- Использовали Path aggregation network (PAN) и spatial pyramid pooling (SPP) layer как часть Neck соединяющий backbone с head
- Bag of freebies - увеличивает точность не увеличивая скорость детекции
- Bag of specials - значительно увеличивает точность при небольшой потере в скорости

YOLOv5


BottleNeck 1
BottleNeck 2
640 × 640 × 3
ConvBNSiLU k6, s2, p2, c64
P1
320 × 320 × 64
ConvBNSiLU k3, s2, p1, c128
P2
160 × 160 × 128
C3
ConvBNSiLU k1, s1, p0, c64
BottleNeck 1
x 3
Concat
ConvBNSiLU k1, s1, p0, c128
ConvBNSiLU k3, s2, p1, c256
P3
80 × 80 × 256
ConvBNSiLU k1, s1, p0, c128
BottleNeck 1
x 6
ConvBNSiLU k1, s1, p0, c256
ConvBNSiLU k3, s2, p1, c512
P4
40 × 40 × 512
ConvBNSiLU k1, s1, p0, c256
BottleNeck 1
x 9
ConvBNSiLU k1, s1, p0, c512
ConvBNSiLU k3, s2, p1, c1024
P5
20 × 20 × 1024
ConvBNSiLU k1, s1, p0, c512
BottleNeck 1
x 3
ConvBNSiLU k1, s1, p0, c1024
SPPF
MaxPool2d k5, s1, p2
ConvBNSiLU k1, s1, p0, c512
Concat
Upsample
BottleNeck 2
x 3
Conv2d k1, s1, p0
$c = (5 + n_{cls}) \times 3$

YOLOv5

Основная новизна:

- PyTorch вместо Darknet(C)
- мозаичная аугментация
- автообучение anchor-ов

Efficient Det D0

- Новая методика feature fusion- a BiFPN (a bidirectional feature pyramid network)
- Compound scaling (width, depth, or resolution)
- Model Scaling (не только backbone но и части FPN и head)

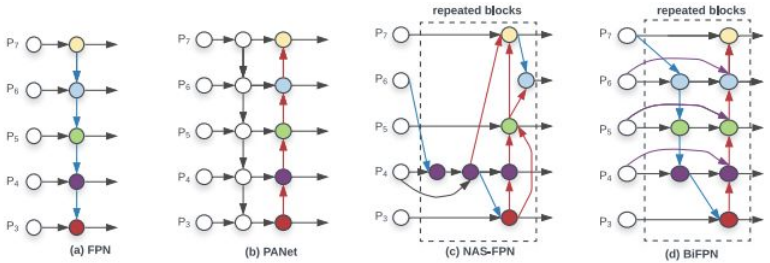


Figure 2: **Feature network design –** (a) FPN [20] introduces a top-down pathway to fuse multi-scale features from level 3 to 7 (P_3 - P_7); (b) PANet [23] adds an additional bottom-up pathway on top of FPN; (c) NAS-FPN [8] use neural architecture search to find an irregular feature network topology and then repeatedly apply the same block; (d) is our BiFPN with better accuracy and efficiency trade-offs.

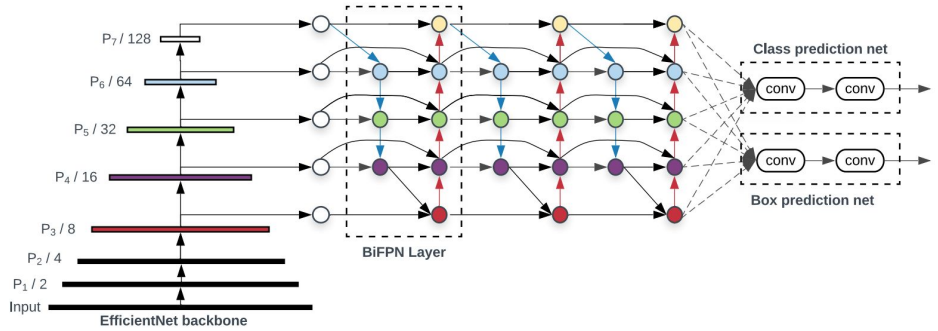
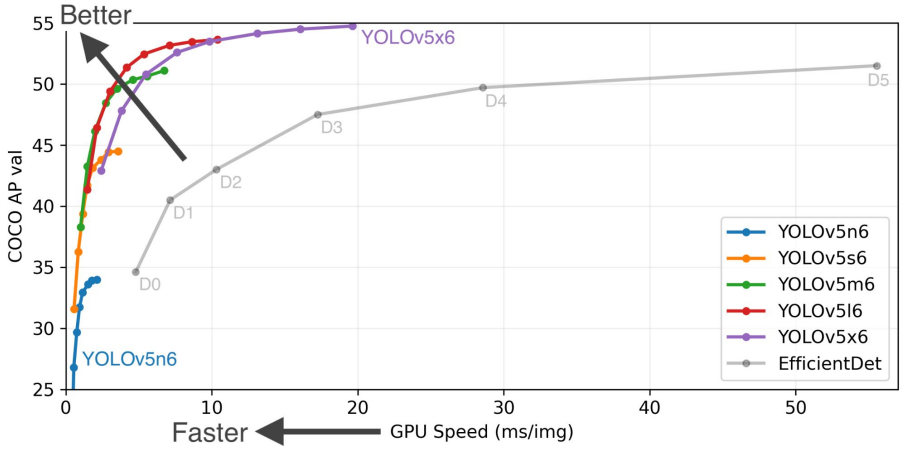


Figure 3: **EfficientDet architecture –** It employs EfficientNet [36] as the backbone network, BiFPN as the feature network.

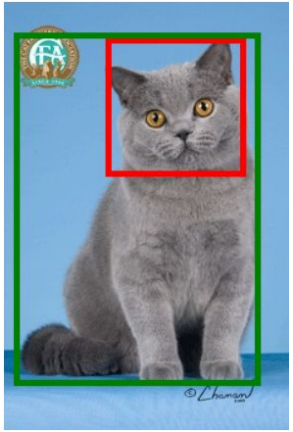
YOLOv5s vs EfficientDet D0

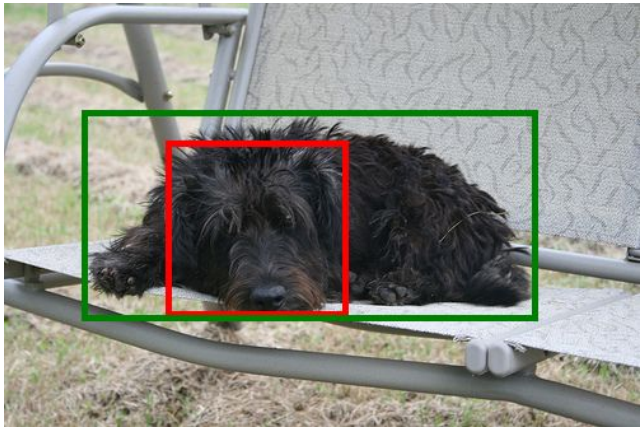
Как видно на графике YOLOv5s более точный чем EfficientDetD0 и быстрее по скорости inference-а.



Inference YOLOv5 (not trained on our dataset)

Ниже результаты детекции. Как видно разметка наших картинок отличается и берет только лицевую часть животных(красная) когда с YOLOv5 объект целиком детектится - зеленая рамка), . Если посчитаем точность такой модели то уверенно могу сказать что она будет низкой. Придется до-обучать модель на наших картинках и разметках.





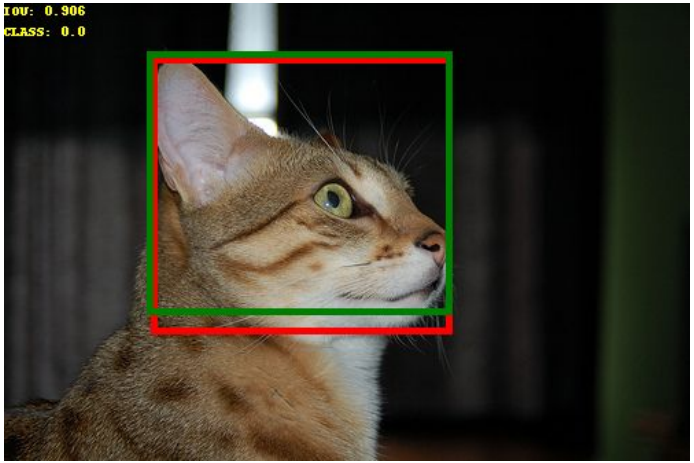
Results YOLOv5s (trained on cats and dogs)

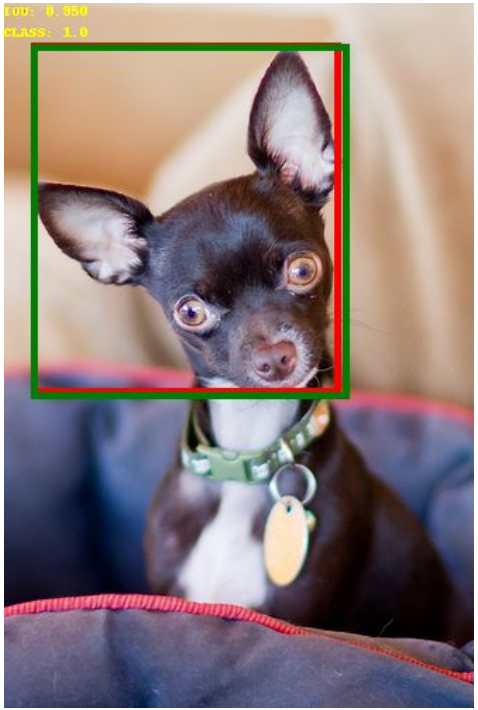
Я разделил датасет на 72% 8% 20% для train, val, test. Для обучения использовал pre-trained weights от MS COCO Dataset. Я выбрал YOLOv5s т.к. у меня возникли проблемы с CUDA и cudnn, пришлось обучаться на CPU, и YOLOv5s не такая большая. Ниже результаты до-обученной модели на 2 классах:

```
########## VAL ######################
Validating runs/train/siemence_cat_dogs2/weights/best.pt...
Fusing layers...
YOLOv5s summary: 157 layers, 7015519 parameters, 0 gradients, 15.8 GFLOPs
               Class     Images  Instances          P          R      mAP50   mAP.5:.95
                 all        295        295      0.974      0.982      0.991      0.813
                 cat        295         99      0.985       0.99      0.994       0.86
                 dog        295        196      0.964      0.974      0.988      0.765
Results saved to runs/train/siemence_cat_dogs2


########## TEST ######################
Fusing layers...
YOLOv5s summary: 157 layers, 7015519 parameters, 0 gradients, 15.8 GFLOPs
test: Scanning /home/temir/Documents/Kaggle/Siemense/data/test.cache... 738 imag
               Class     Images  Instances          P          R          mAP50   mAP.5:.95
                 all        738        738      0.988      0.983       0.99      0.842
                 cat        738        237      0.996      0.987      0.993      0.883
                 dog        738        501       0.98      0.978      0.987      0.801
Speed: 0.7ms pre-process, 76.8ms inference, 0.4ms NMS per image at shape (32, 3, 640, 640)
Results saved to runs/val/exp  738 labels saved to runs/val/exp/labels
```

Results YOLOv5s (trained on cats and dogs)





Results YOLOv5s (inference img size 256x256) x5 on speed

test: Scanning /home/temir/Documents/Kaggle/Siemense/data/test.cache... 738 imag

Class	Images	Instances	P	R	mAP50	mAP0.5:0.95
all	738	738	0.967	0.957	0.975	0.773
cat	738	237	0.977	0.975	0.985	0.815
dog	738	501	0.957	0.939	0.966	0.732

Speed: 0.1ms pre-process, 14.3ms inference, 0.3ms NMS per image at shape (32, 3, 256, 256)

Results saved to runs/val/exp3

738 labels saved to runs/val/exp3/labels

Доп. задача: Найти все изображения с данной породой

Для этой задачи использовал библиотеку DeepImageSearch

Главное идея алгоритма:

- Берем наши картинки прогоняем через VGG16(weights='imagenet') и берем результаты последнего fully_connected_layer. Это будет как info векторизация.
- Все это сохраняем в meta pkl файле в нормализованном виде.
- Для поиска нам нужно сперва выбрать картинки кота чью породу ищем. И задаем сколько близких картинок нам нужно.
- Далее когда запускаем поиск мы заново делаем векторизацию нужной картинки кота.
- Для поиска схожих векторов используем библиотеку: Annoy (Approximate Nearest Neighbors Oh Yeah) is a C++ library with Python bindings to search for points in space that are close to a given query point.
- Далее выдаем результаты.

Результаты DeepImageSearch