

go through image segmentation first
to get a better idea of this session

companies have pre-trained models
and they're available as APIs.
In this session we will use Amazon's API

AWS API doesn't require much code
to try. Google does require it.

They have lots of features in the
API & a succinct documentation of
all of that. I don't care, so
I haven't taken. We can just look it up for my API.

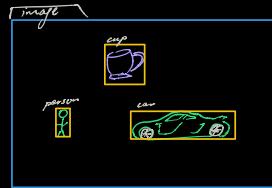


Limitations

It's important to note the limitations of object detection so you can avoid or mitigate the effects of false negatives (missed objects) and limited detail.

- Objects are generally not detected if they're small (less than 5% of the image).
- Objects are generally not detected if they're arranged closely together (a stack of plates, for example).
- Objects are not differentiated by brand or product names (different types of sodas on a store shelf, for example). However, you can get brand information from an image by using the Brand detection feature.

Well see you vs → there's another session for this
↓
it's written in C
so the code is slower
& less optimized.



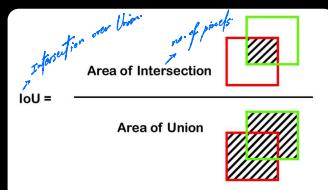
↑ also, AWS API has good docs

```

{
    "objects": [
        {
            "rectangle": {
                "x": 730,
                "y": 66,
                "w": 135,
                "h": 85
            },
            "object": "kitchen appliance",
            "confidence": 0.501
        },
        {
            "rectangle": {
                "x": 523,
                "y": 377,
                "w": 185,
                "h": 46
            },
            "object": "computer keyboard",
            "confidence": 0.51
        },
        {
            "rectangle": {
                "x": 471,
                "y": 218,
                "w": 289,
                "h": 226
            },
            "object": "Laptop",
            "confidence": 0.85,
            "parent": {
                "object": "computer",
                "confidence": 0.851
            }
        }
    ]
}
  
```

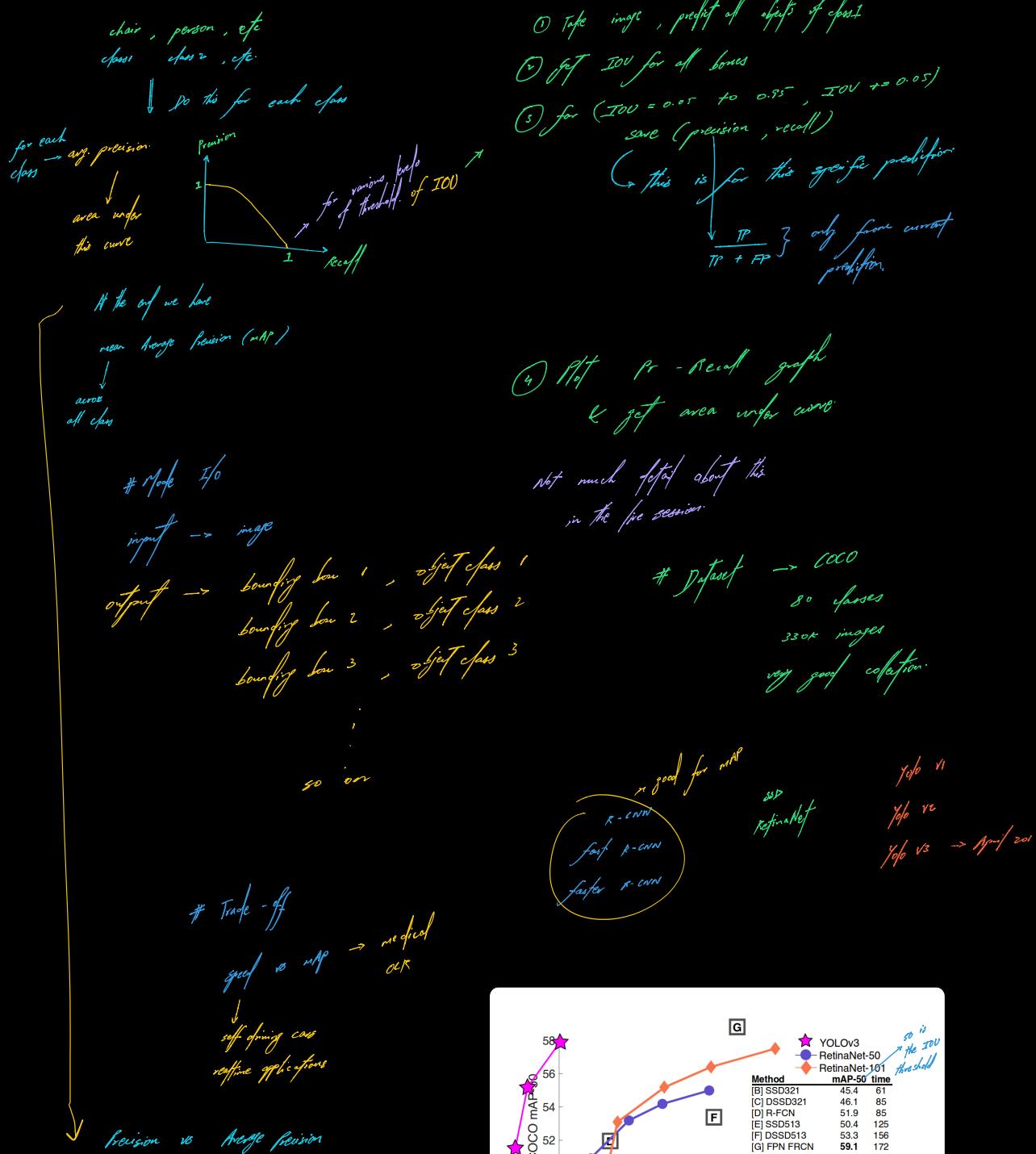
↑ also gives hierarchy info

Performance Metric

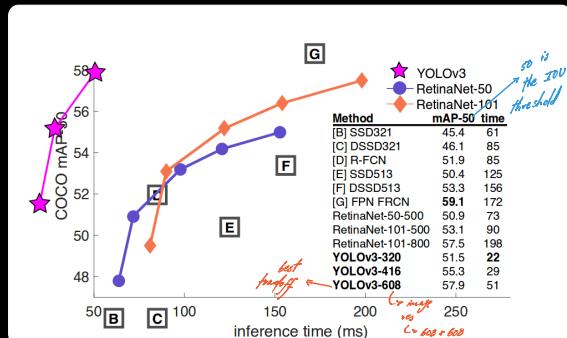


(this is just for one box)

if we print json simply,
it will be non-inflated
this function prints json clearly
print(json.dumps(json_file))



Precision refers to precision at a particular decision threshold. For example, if you count any model output less than 0.5 as negative, and greater than 0.5 as positive. But sometimes (especially if your classes are not balanced, or if you want to favor precision over recall or vice versa), you may want to vary this threshold. Average precision gives you average precision at all such possible thresholds, which is also similar to the area under the precision-recall curve. It is a useful metric to compare how well models are ordering the predictions, without considering any specific decision threshold.



YOLO vs - feature extractor (DarkNet-53)

fully convolutional network (FCN) → no maxpool

conv → catch whom → loss/ReLU

no maxpooling → they use stride = 2 for conv
kings works the same.

pre-trained on ImageNet.

skip connections allow deeper 53 layer architecture.

YOLO vs many

DarkNet-53 (+ skip have
skip connection)

Type	Filters	Size	Output
Convolutional	32	3 × 3	256 × 256
Convolutional	64	3 × 3 / 2	128 × 128
Convolutional	32	1 × 1	
Convolutional	64	3 × 3	
1x Residual			128 × 128
Convolutional	128	3 × 3 / 2	64 × 64
Convolutional	64	1 × 1	
2x Convolutional	128	3 × 3	
Residual			64 × 64
Convolutional	256	3 × 3 / 2	32 × 32
Convolutional	128	1 × 1	
Convolutional	256	3 × 3	
Residual			32 × 32
Convolutional	512	3 × 3 / 2	16 × 16
Convolutional	256	1 × 1	
Convolutional	512	3 × 3	
Residual			16 × 16
Convolutional	1024	3 × 3 / 2	8 × 8
Convolutional	512	1 × 1	
Convolutional	1024	3 × 3	
Residual			8 × 8
Avgpool		Global	
Connected		1000	
Softmax			

DarkNet-53 model

53 conv layers

why not use ResNet as
feature extractor? It's already has better performance.

It has same Top1, Top5 but half

the FLOPs. Performance sucks. That's why.

↓

YOLO v3 → 78 FPS

YOLO v2 → 171 FPS

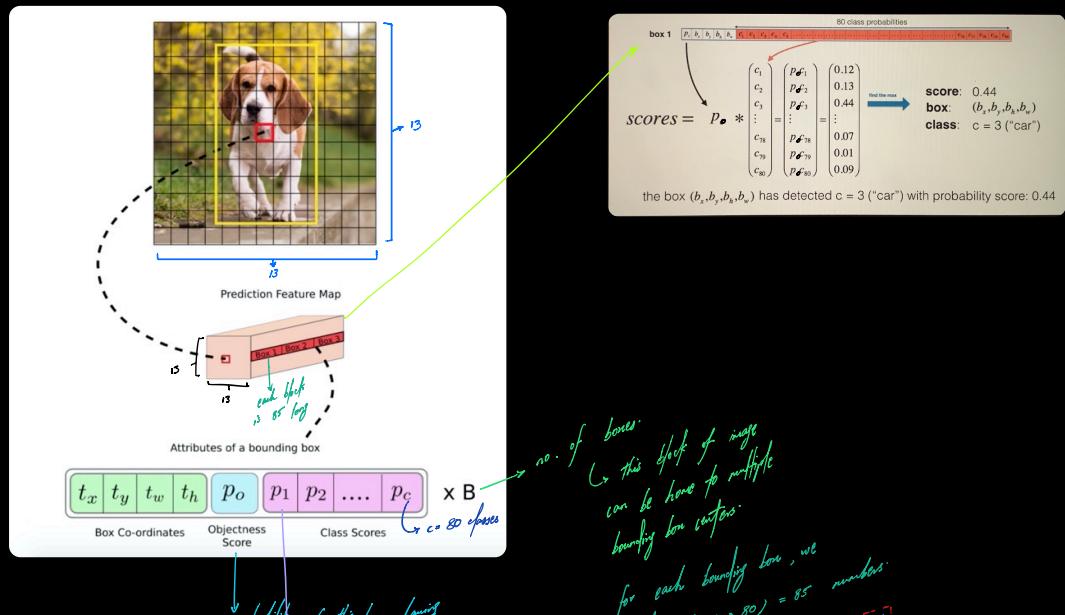
ResNet101 → 37 FPS → Meh

well, we can't do it
because, it's defined yield & accuracy.

input image → 416 × 416 × 3
feature extracted → 10 × 10 × 1024
↓ because of conv stride = 2 → 5 of them
↳ 1024 filters in the last layer

from DarkNet53
pre-trained on ImageNet

Bounding boxes & offset



t_x	t_y	t_w	t_h	p_o	p_1	p_2	\dots	p_c	$\times B$
Box Co-ordinates				Objectness Score	Class Scores				$\rightarrow c = 80$ classes

probability of this box having something of interest

prob. for each class

$$\text{probability(class 10 object)} = P_o * P_{10}$$

no. of boxes
↳ this block of image
can be have up to multiple
bounding box outputs.
for each bounding box, we
and $(1 + 80) = 85$ numbers.
We can change that $13 \times 13 \times 1024$
↳ depending
on how
many boxes
we want for each
block of image.

for eg. if I want 5 po
for each block to:

$$13 \times 13 \times (5 \times (5 + 80))$$

↓
no. of = 5 boxes

↓
classes

$$\therefore 13 \times 13 \times 425$$

in yolo they use 13×13
smaller $B =$ faster \rightarrow that much less off to predict.
larger $B =$ larger map

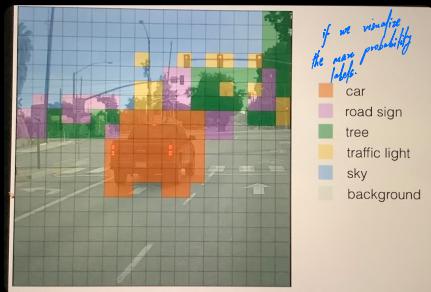
all this is differentiable, so it works

backward gives us $13 \times 13 \times 1024$ the...

$$\downarrow$$

425 pt conv

$\therefore 13 \times 13 \times 425 \rightarrow$ they used this thing to optimize backprop

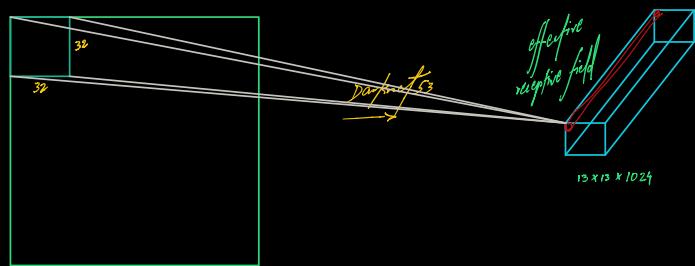


→ intermediate result
of $yolo v3$

* If there's only one object, that " P_o "

↓
probability of
having an object
of interest will be
zero or close to
zero

* $yolo$ is designed to work
on square images but
we can always resize
the input to be a square

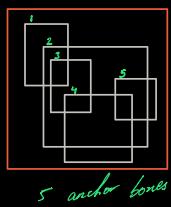


* what if there are multiple objects?
spans multiple receptive fields?

* first they tried to predict (bx, by, bw, bh)
for a bounding box → turned out to be too
hard to train.

→ if it's far away → it'll be almost all
blurred away. They want
it to be very close to
anchor boxes

* solution → have anchor boxes



5 anchor boxes

* later represent bb as a modification
of anchor boxes

$$\hookrightarrow (cx, gy, pw, ph)$$

sigmoid function

$$\begin{array}{c} \uparrow \\ s(tw) + cx \\ \downarrow \\ s(tg) + gy \\ \downarrow \\ (bx, by, bw, bh) \\ \uparrow \\ b_w e^{tw} \\ \uparrow \\ b_h e^{tg} \end{array}$$

→ means clamping on train effects
bounding boxes

$k = 5$ anchor boxes

↓
5 boxes in which there's a high
likelihood of finding an object

↓
this can act as "prior" info
to find objects more successfully.

left work with objectives not P_o
prob. of finding an object in the batch

learning of deep learning &
loss function
→ made teams all this implicitly since
we penalize the model accordingly

tn | tg | tw | th | P_o | P_1 | P_2 | ... | P_n

(if class = woman) → if we do find
a woman, if we
put her in person
or woman class.

* Default weights
are used for it only
loss are propagated to
all the layers

loss function

if we keep $P_o = 3$

$12 \times 12 \times 255$

... $12 \times 12 \times 3$ possible boxes

they found ✓ is a hyperparameter
forward = 5
Anomaly = 0.5
works well → values for weight

range will be
0 to 1
if imbalance

$\left[\begin{array}{l} \textcircled{1} \text{ some weight} \\ \vdots \textcircled{2} \text{ } \lambda_{\text{card}} * \text{squared loss on } \text{tn}, \text{tg}, \text{tw} \text{ & } \text{th} \\ \text{for each } \textcircled{3} \text{ } + \text{log. loss for } P_o \\ \text{bounding box containing an object in ground truth} \end{array} \right]$

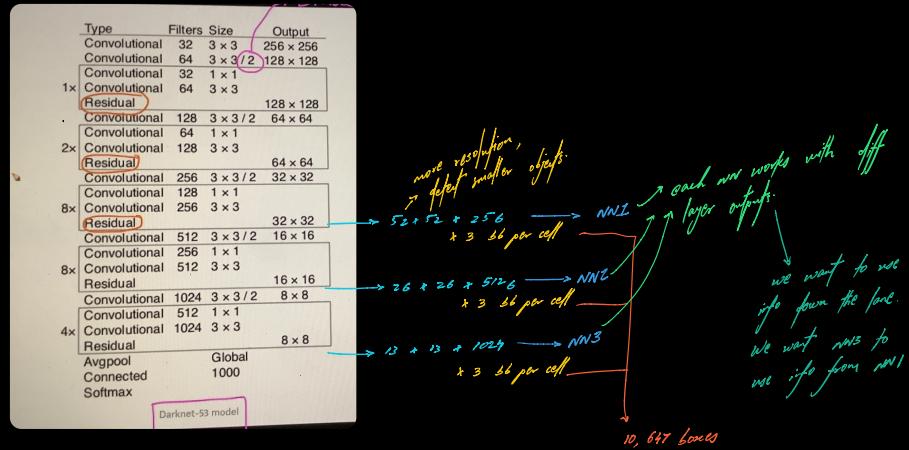
+ $\sum \text{Anomaly} * \text{log. loss for } P_o$

no object boxes in ground truth

→ now have we
done something
else so we just want
 P_o to be either
zero or one.

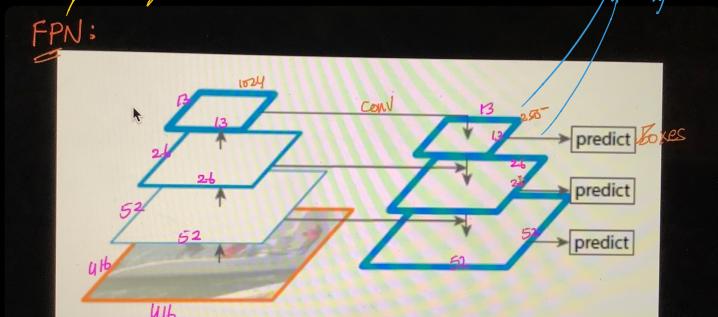
We use pretrained Darknet
but we're not freezing the weights.

* Multi-Scale Prediction



Feature Pyramid Networks:

FPN:



DARKNET

512 filters

3×3 filters

not info from surrounding cells