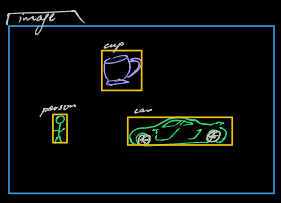
go through image segmentation first to get a better idea of this session.

companies have pretrained models and they're available as APIs. In this session we will use Azure's API

Well see YOLO v3 → there's another session for this
it's written in C
↳ # speed
python code is slower & less optimized.



Azure API doesn't require credit card to try. Google does require it.

They have lots of features in the API & excellent documentation for all of that. I don't care, so I haven't tried them. We can just look at one → for any API.

GCP, AWS, etc all have good docs

JSON →


```
{
  "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
```


} image center coordinates
} bounding box dimension
} also gives hierarchy info

Limitations

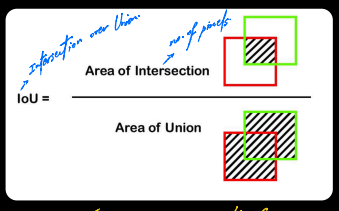
It's important to note the limitations of object detection so you can avoid or mitigate the effects of false negatives (missed objects) and limited detail.

- 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.

performance metric

if we print json simply, it's ugly & non-indented
this function prints json cleanly

```
print(json.dumps(json_file))
```



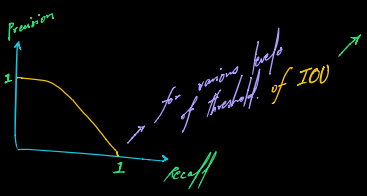
↳ this is just for one box.

chair, person, etc
class 1, class 2, etc.

↓ Do this for each class

for each class → avg. precision.

↓ area under this curve



① Take image, predict all objects of class 1

② get IOU for all boxes

③ for (IOU = 0.05 to 0.95, IOU += 0.05)

save (precision, recall)

↳ this is for this specific prediction.

$\frac{TP}{TP + FP}$ } only from current prediction.

At the end we have

mean Average Precision (mAP)

↓ across all class

④ Plot Pr - Recall graph & get area under curve.

Not much detail about this in the live session.

Model I/O

input → image

output → bounding box 1 - object class 1
bounding box 2 - object class 2
bounding box 3 - object class 3
⋮
so on

Dataset → COCO
80 classes
330K images
very good collation.

not good for mAP

R-CNN
fast R-CNN
faster R-CNN

SSD
RetinaNet

YOLO V1
YOLO V2
YOLO V3 → April 2018

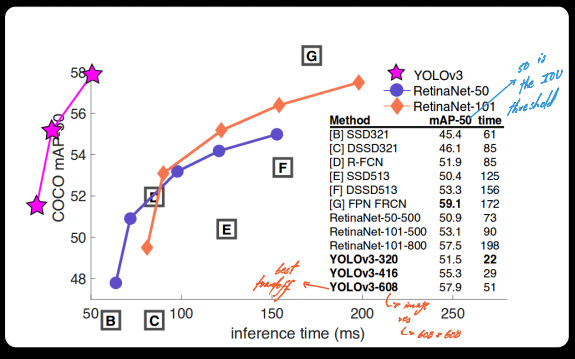
Trade-off

speed vs mAP → medical use case

↓ self driving cars
real time applications

Precision vs Average Precision

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.



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

YOLO V3 - feature extractor (DarkNet-53)

Fully convolutional Network (FCN) → no maxpools

conv -> Batch Norm → Leaky ReLu

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

pretrained on ImageNet.

skip connections allow deeper 53 layer architecture.

↓

YOLO v2 used Darknet-19 (it didn't have skip connections)

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

skip connection from prev. block

Darknet-53 model

53 CNN layers

why not use ResNet as feature extractor? It already has better performance. It has same Top1, Top5 but half the FPS. Performance sucks. That's why.

↓

YOLO V3 → 78 FPS

YOLO V2 → 171 FPS

ResNet152 → 37 FPS → MEH

we'll use YOLOv3-416 as an example, it's balanced speed & accuracy.

input image → 416 × 416 × 3

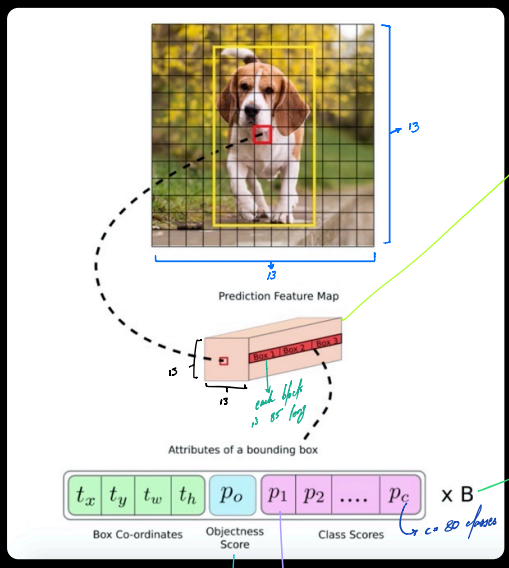
feature extracted → 13 × 13 × 1024 (because of conv stride = 2 → 5 of them in between)

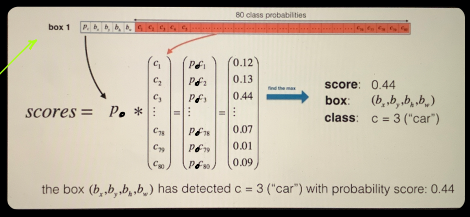
↳ 1024 filters in the last layer.

↓

from Darknet-53 pre-trained on ImageNet

Bounding boxes & output





→ no. of boxes.
↳ this block of image can be have to multiple bounding box centers.

for each bounding box, we need $(4 + 1 + 80) = 85$ numbers.

We can change that $13 \times 13 \times 1024$
↳ depending on how many boxes we want for each block of image.

probability of this box having something of interest

prob. for each class

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

for eg. if I want 5 BB for each block #.

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

no. of boxes $= 5$

classes

$\therefore 13 \times 13 \times 425$

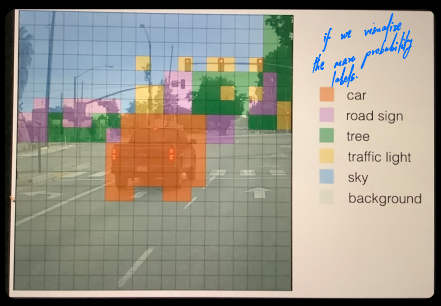
in yolo they use $13 = 5$
smaller 13 = faster. → that much less stuff to predict.
larger 13 = larger map

all this is differentiable, so it works.

Darknet gives us $13 \times 13 \times 1024$ tho...

425
1x1 conv

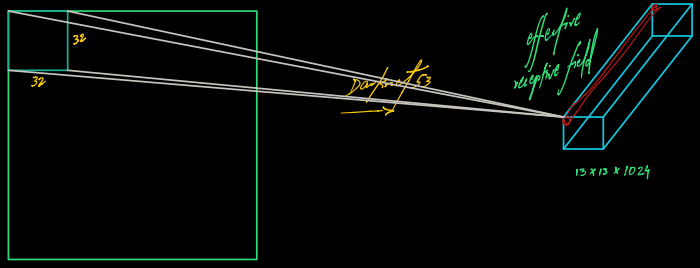
$\therefore 13 \times 13 \times 425$ → they used this thing to optimize function



→ intermediate result of YOLO v3

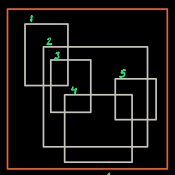
If there's only one object, then "p_c" ↓ probability of having an object of interest will be zero or close to zero

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



what if the object spans multiple receptive fields?

first they tried to predict (b_x, b_y, b_w, b_h) for a bounding box → Turned out to be too hard to train.

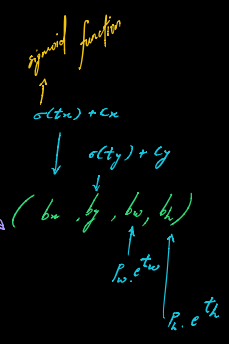


5 anchor boxes

solution → have anchor boxes.

let's represent bb as a modification of anchor boxes

↳ (c_x, c_y, p_w, p_h)



→ kmeans clustering on train data's 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.

Darknet weights are used for init only. loss are propagated to all the layers.

let's work with objectness score (p_0)

prob. of finding an object in this block.

t_x	t_y	t_w	t_h	p_0	p_1	p_2	...	p_c

- if class = woman
- if class = person.

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

loss function.

if we keep $B = 3$

$13 \times 13 \times 255$

∴ $13 \times 13 \times 3$ possible boxes

∴ Σ → for each bounding box containing an object in ground truth.

1. λ_{coord} * squared loss on t_w, t_h, t_x & t_y (λ → some constant) → range will be more. # imbalance in loss.
2. log. loss for p_0 (→ bcoz its binary)
3. log. loss for each p_i ↳ bcoz its logistic regression.

$+ \sum \lambda_{no.obj} \cdot$ log loss for p_0 → 0 to 1

(Σ → no object boxes in ground truth.) → over here we don't care about anything else. So we just want p_0 to be close to zero or zero.

they found ↙ $\lambda_{coord} = 5$, $\lambda_{noobj} = 0.5$ works well

it's a hyp. parameter. ← that's why the λ values for weight