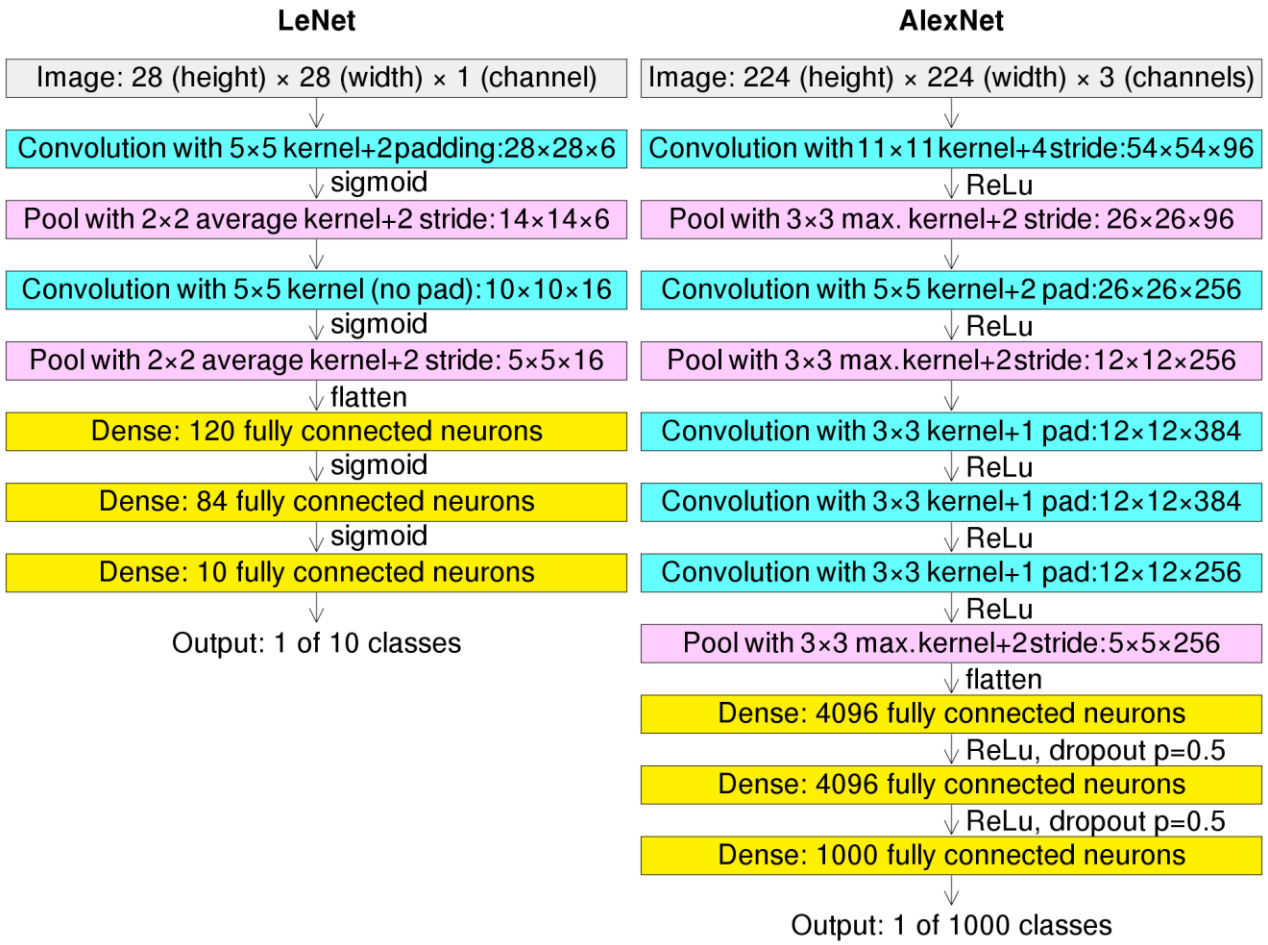
Convolutional Neural Networks 2

Deep CNNs

Topics

- Convolutional Neural Network architectures
- Residual blocks
- Dense connections
- Object detection
- Semantic segmentation

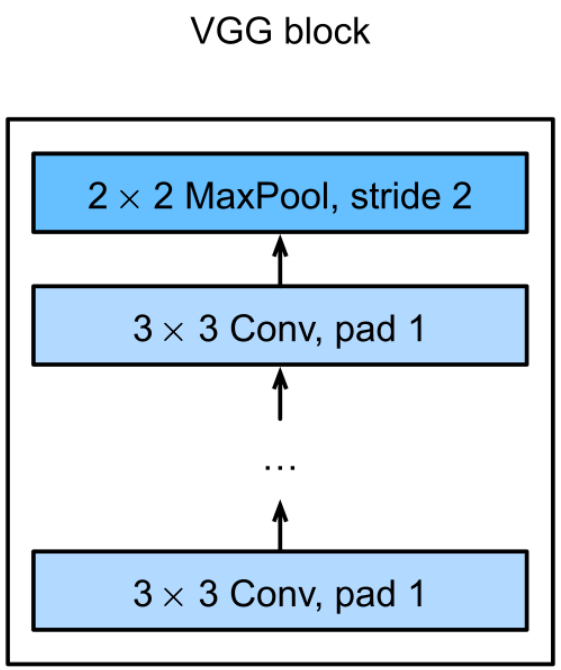
LeNet and Alexnet



Alexnet in PyTorch: https://pytorch.org/hub/pytorch_vision_alexnet/
LeNet in PyTorch: https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html

VGG

- Introduced by Oxford Visual Geometry Group
- General idea:
  - Use many small conv. layers rather than few large ones
- Network consists of five blocks with several conv and activation layers.
  - 3×3 conv with padding
  - ReLU
- Each block is followed by a 2×2 pooling layer



VGG variants

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

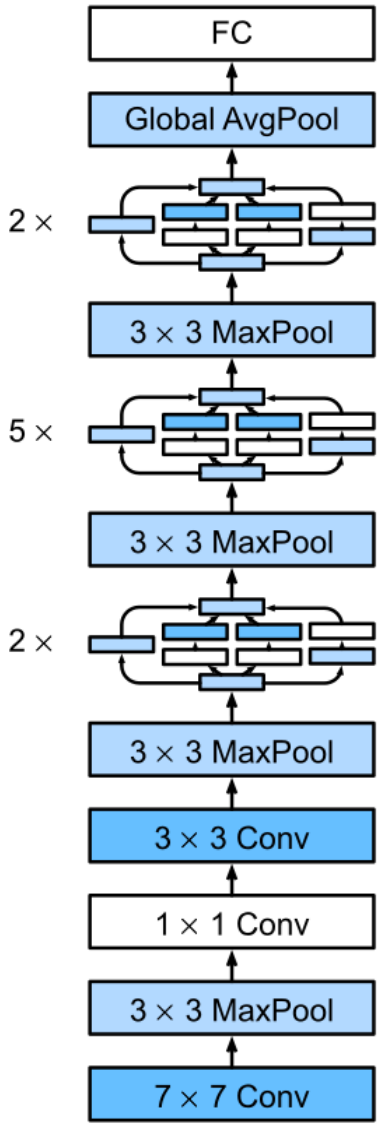
- The authors, Simonyan and Zisserman found that several layers of deep and narrow convolutions (i.e., 3 × 3) were more effective than fewer layers of wider convolutions.
- VGG is computationally-heavy but produces a nice hierarchical feature representation
- VGG is hard to train – If you want to train it, start with few layers and add more, when you see progression.

VGG in PyTorch: https://pytorch.org/hub/pytorch_vision_vgg/

LRN: Local response normalization

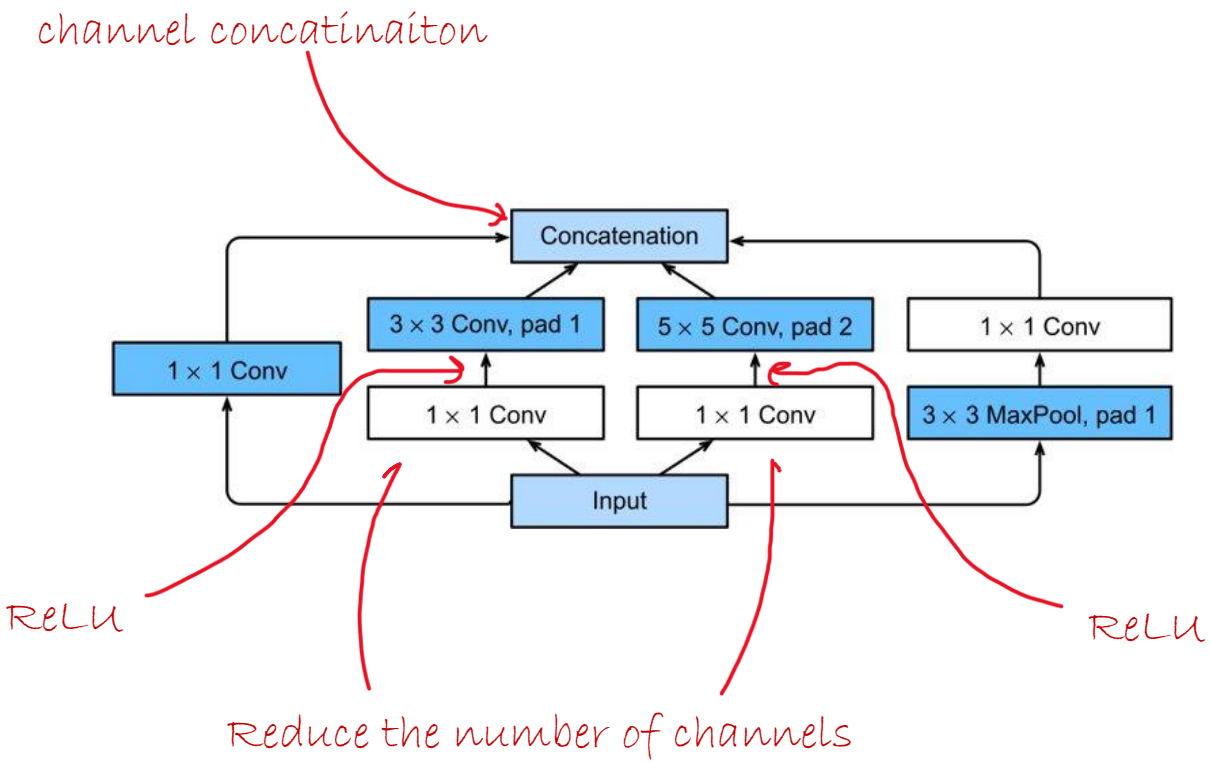
GoogLeNet

- Researchers from Google won the ImageNet Challenge in 2014 with a deep but lightweight architecture called GoogLeNet



https://arxiv.org/pdf/1409.4842v1.pdf

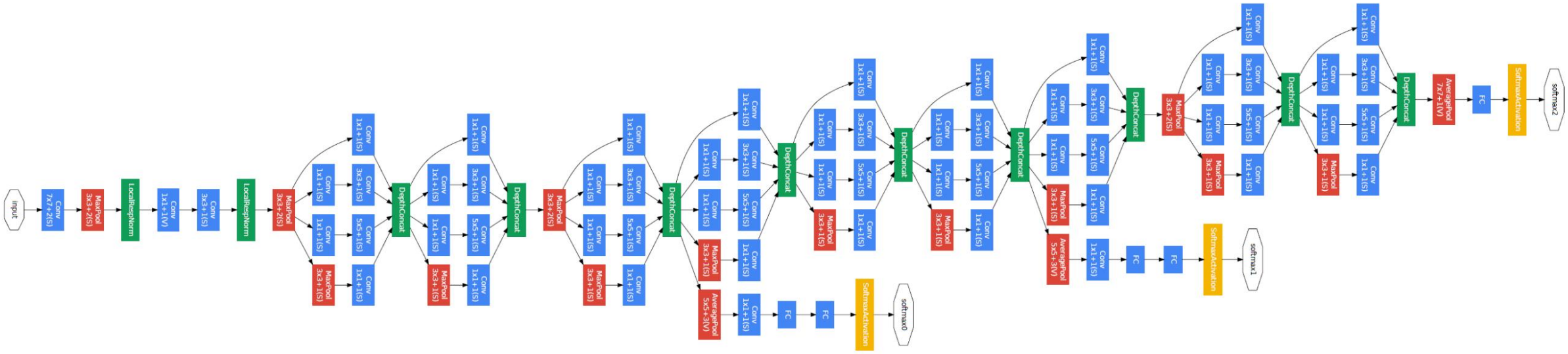
GoogLeNet inception block



- Four parallel paths with different window sizes to extract information of different spatial sizes
- All operations use padding to ensure that all outputs have the same size
- GoogLeNet uses nine inception bloks

GoogLeNet in Pytorch: https://pytorch.org/hub/pytorch_vision_googlenet/

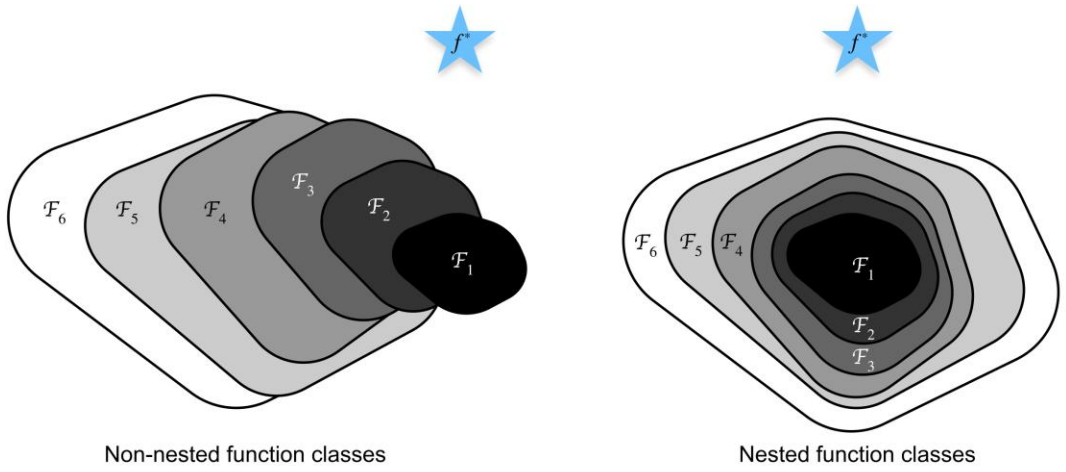
GoogLeNet


softmax2
SoftmaxActivation
FC
AveragePool
7x7+1(V)
DepthConcat
Conv
1x1+1(S)
Conv
3x3+1(S)
Conv
5x5+1(S)
MaxPool
3x3+1(S)
softmax1
SoftmaxActivation
FC
FC
MaxPool
3x3+2(S)
DepthConcat
Conv
1x1+1(S)
Conv
3x3+1(S)
Conv
5x5+1(S)
Conv
1x1+1(S)
MaxPool
3x3+1(S)
AveragePool
5x5+3(V)
softmax0
SoftmaxActivation
FC
FC
AveragePool
5x5+3(V)
MaxPool
3x3+1(S)
Conv
1x1+1(S)
Conv
3x3+1(S)
Conv
5x5+1(S)
Conv
1x1+1(S)
MaxPool
3x3+2(S)
LocalRespNorm
Conv
1x1+1(V)
Conv
3x3+1(S)
LocalRespNorm
MaxPool
3x3+2(S)
MaxPool
3x3+2(S)
input
Conv
7x7+2(S)

ResNet

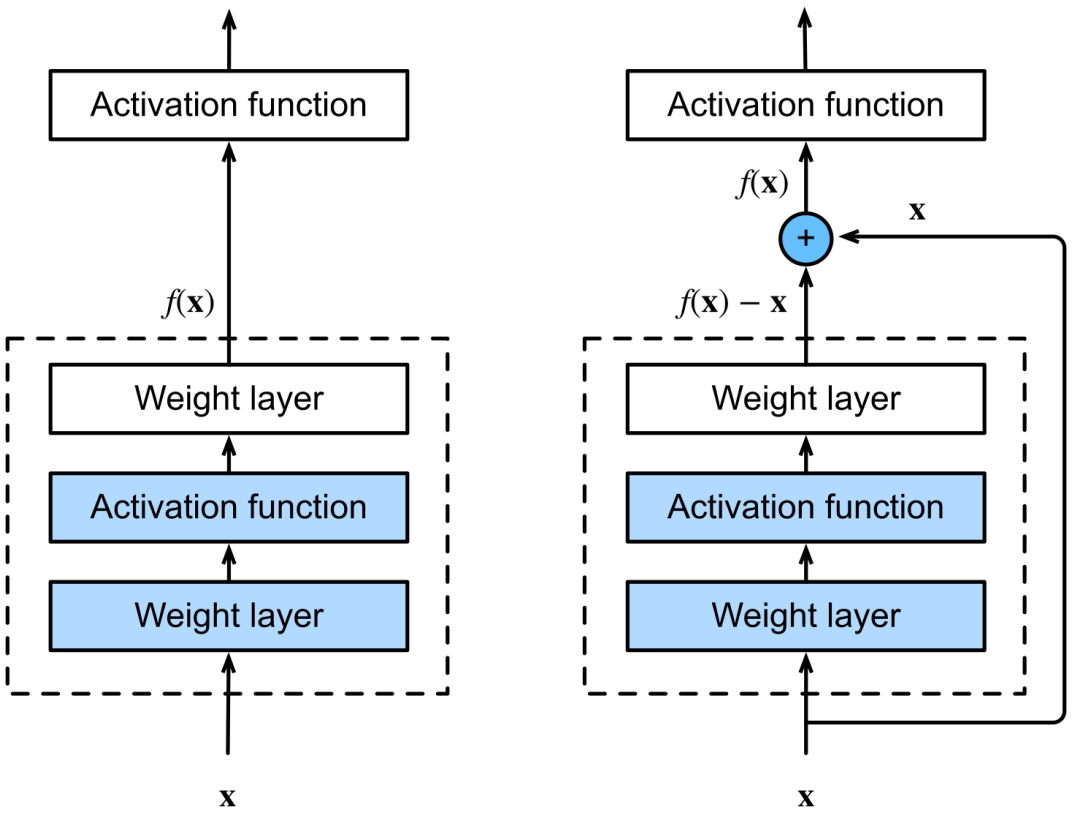
- Problem: When networks gets deeper, they can learn more abstract features. But training becomes harder, since gradients are smaller.
- Solution: Residual blocks with shortcuts

Residual network



- f^* is the "truth" function for solving a task, and F_x is all functions that a given network architecture can reach.
- If we create a new network architecture by adding layers, we are not guaranteed to approach f^* (left image)
- If we create a new architecture, by adding layers, where the previous architecture is in the subset of the new architecture (right image), we will not get further away from f^* and hopefully closer.
  I.e., $F_1 \subseteq \ldots \subseteq F_6$
  - If the new layers can be trained into an identity function, we are guaranteed to create nested functions (right image).
- He et al. proposed a residual block to solve this problem:

Residual block



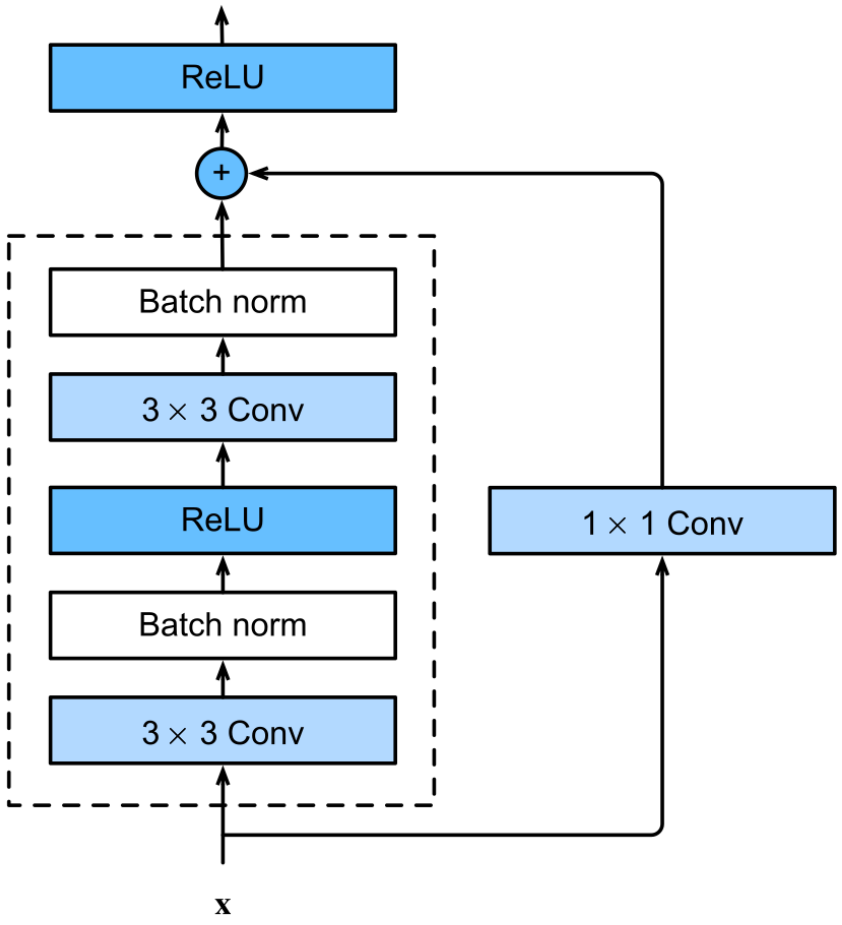
regular block

residual block

- If the identity mapping f(x) = x is the desired underlying mapping, weigh will decay to zero.
- With residual blocks, inputs can forward propagate faster through the residual connections across layers

shortcut connection (identity)

Residual block with 1x1 convolution

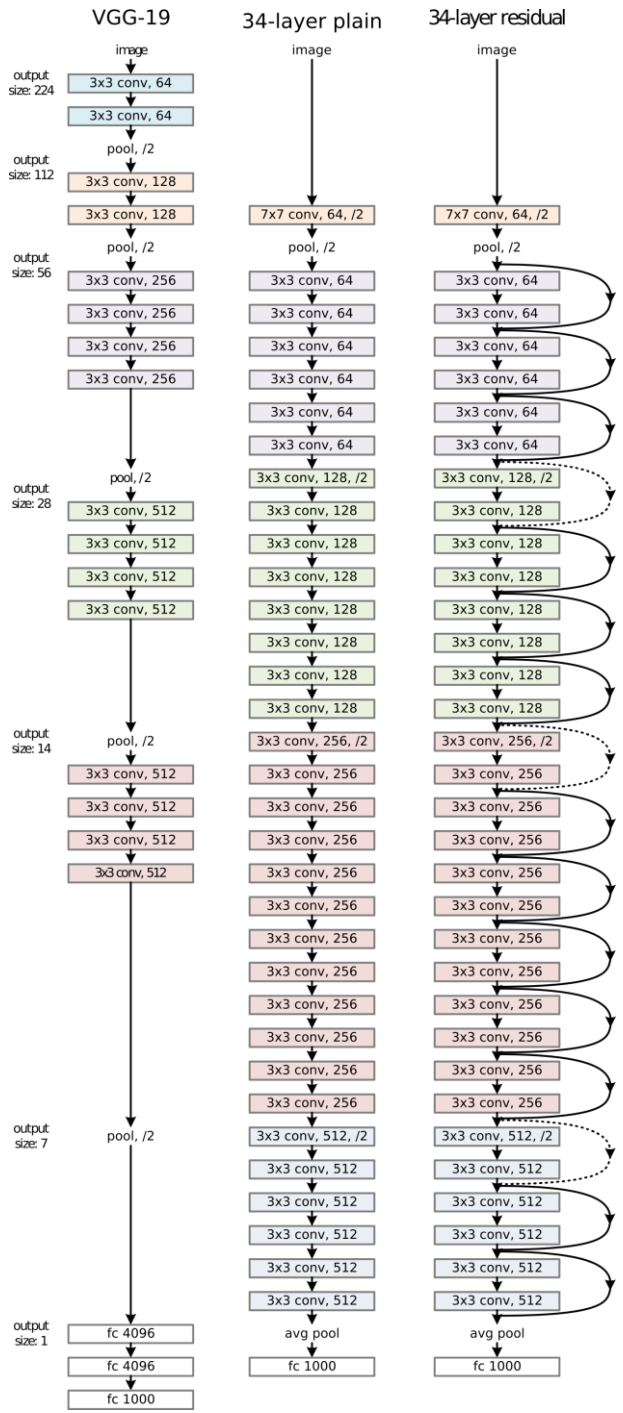


- A 1x1 convolution can be added to the shortcut to transform the input into the desired shape for the addition operation

Resnet

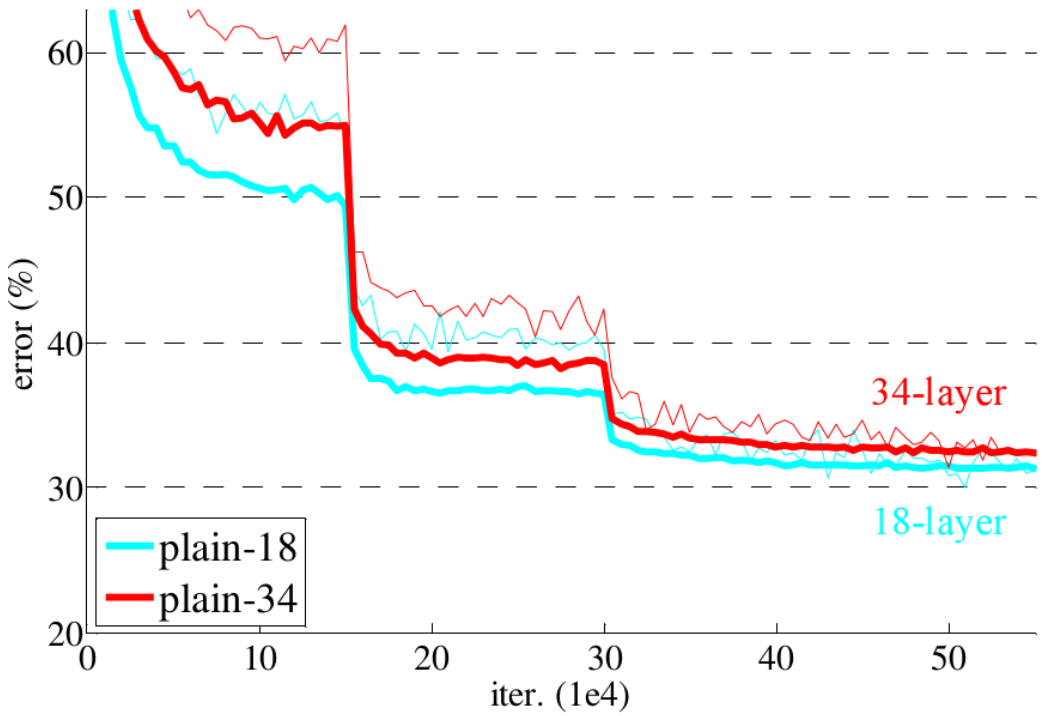
- A series of residual blocks are stacked, and finally a global average pooling layer aggregates all features

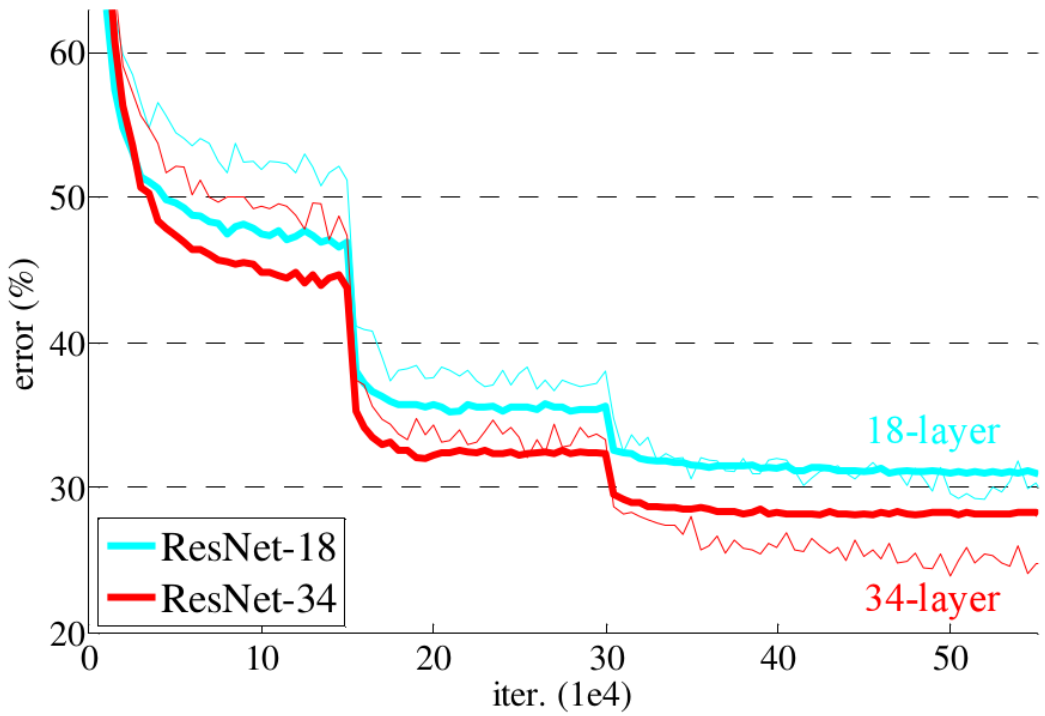
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9



Resnet in PyTorch: https://pytorch.org/hub/pytorch_vision_resnet/

Resnet training error on ImageNet





Thin curves denote training error, bold curves denote validation error

Note how ResNets trains better (not faster) than plan networks

Resnet in PyTorch: https://pytorch.org/hub/pytorch_vision_resnet/

HOW TO MAKE NETWORKS DEEP?

Solving the problem of vanishing gradient or convergence issues (to win the ImageNet competition)

2012: Using ReLU and dropout (to avoid overfitting)

- **Network**: AlexNet

2014: Using Auxiliary classifiers and a “train … add layers … train … add layers”-method

- **Network**: GoogleNet (Going deep with conv.) and VGG (Very Deep CNN …)

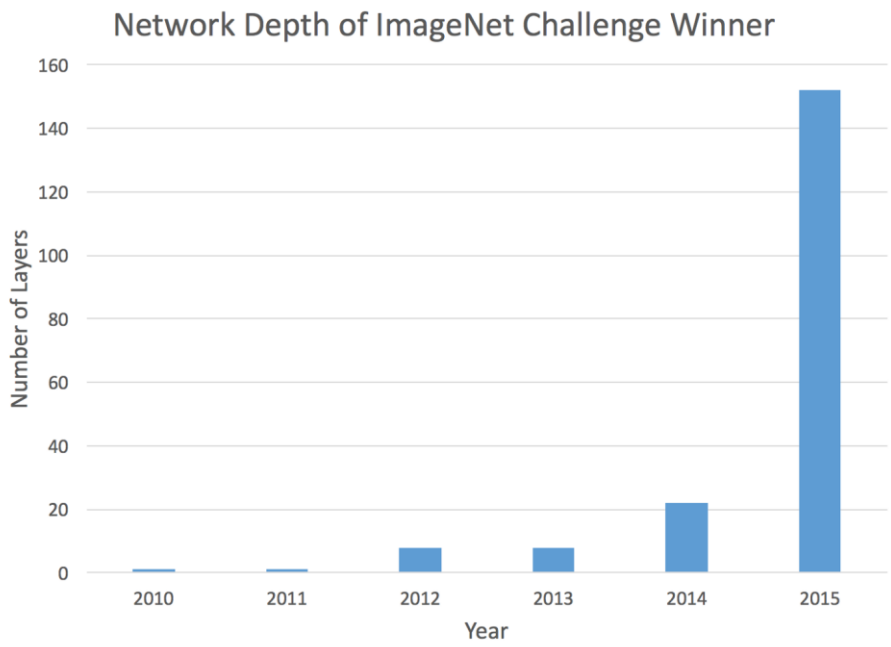
2015: Using normalization

- **Network:** GoogleNet “v2” – Batch normalization
- (**Network:** All you need is a good init)

2016: Using residual connections

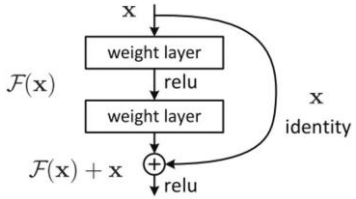
- **Network**: ResNet

2017: DenseNet???



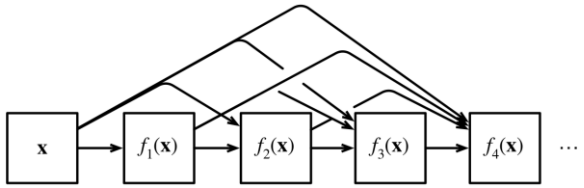
TRANSITION/SHORTCUT NETWORKS

- Improve flow of information and gradient through the network
  - **ResNet**: Residual connections (identity mappings)



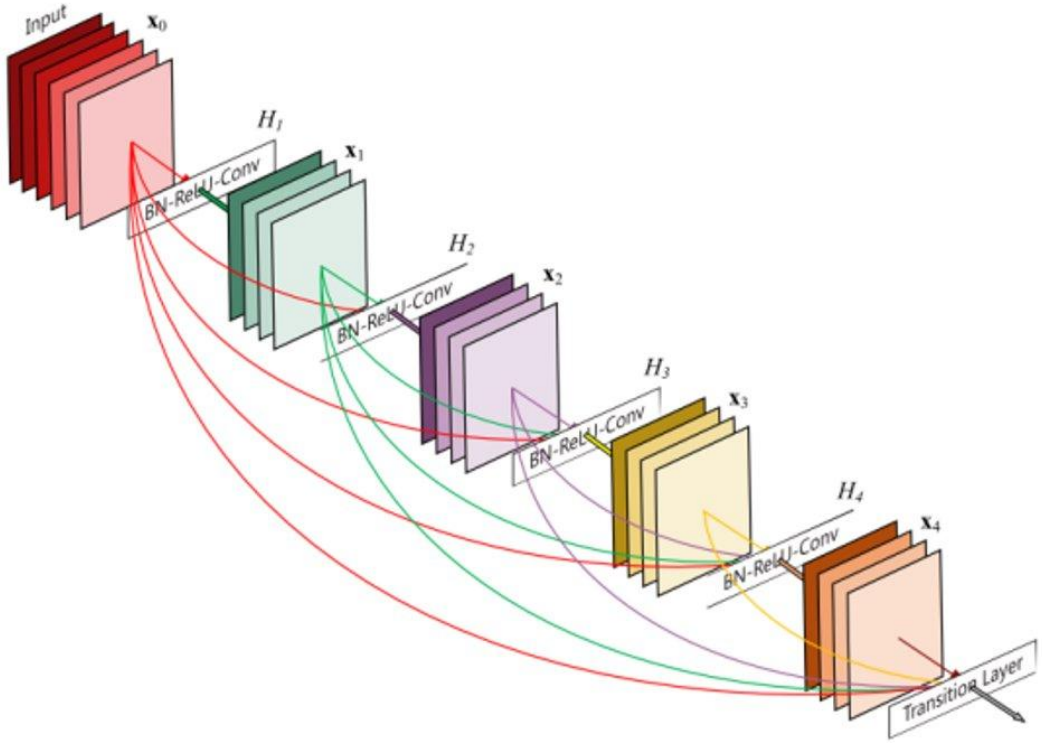
  - **DenseNet**: Densely connected.

$$\mathbf{x} \rightarrow [\mathbf{x}, f_1(\mathbf{x}), f_2([\mathbf{x}, f_1(\mathbf{x})]), f_3([\mathbf{x}, f_1(\mathbf{x}), f_2([\mathbf{x}, f_1(\mathbf{x})])]), \ldots]$$



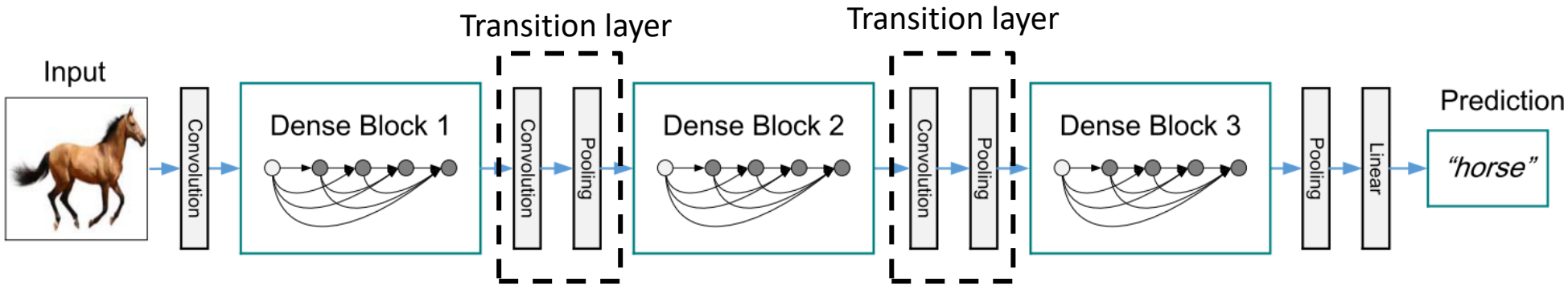
DenseNet

- Densely connected. All layers are connected
- Claim to reduce overfitting for small data sets
- DenseNet vs ResNet
  - DenseNet Improves flow of information and gradients in the network
  - Features are combined using concatenation (and not summation)



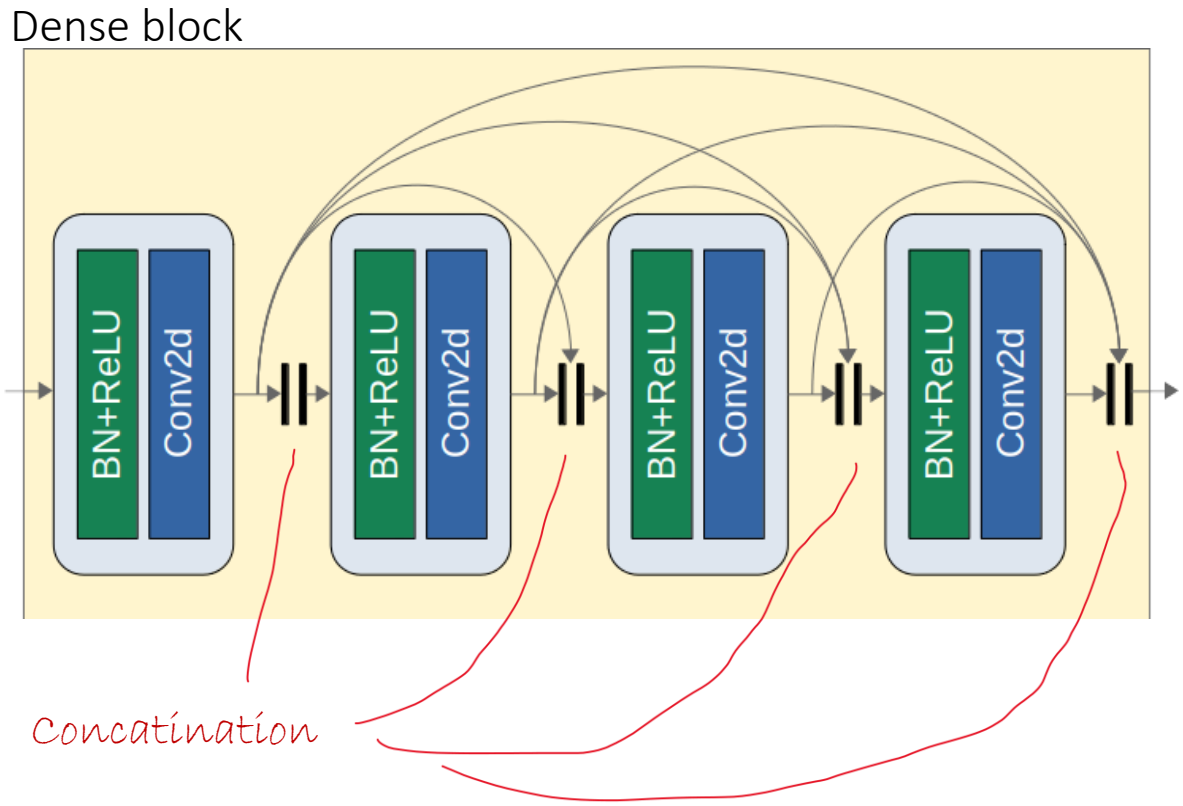
Densenet in PyTorch: https://pytorch.org/hub/pytorch_vision_densenet/

DenseNet


Input
Convolution
Dense Block 1
Transition layer
Convolution
Pooling
Dense Block 2
Transition layer
Convolution
Pooling
Dense Block 3
Pooling
Linear
Prediction
"horse"

Dense block

- A dense block consists of multiple convolution blocks, each using the same number of output channels.
- The convolution blocks can have several conv and Batch-Norm layers. Here only showed with one conv and BN layer
- During forward propagation, we concatenate the input and output of each convolution block on the channel dimension.



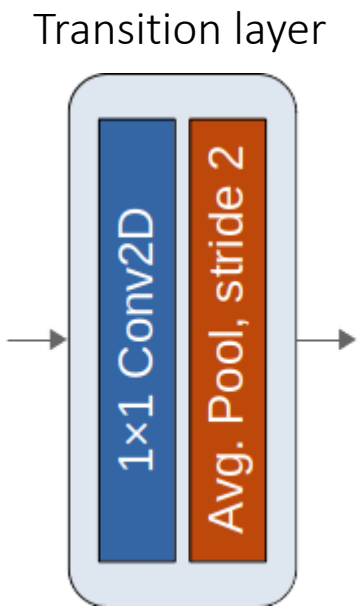
Transition Layers

Each dense block will increase the number of channels

--> adding too many of them will lead to an excessively complex model.

Transition layers are used to control the complexity of the model.

- 1 × 1 Conv. layer reduces number of layers
- Avg pool reduces spatial dimension



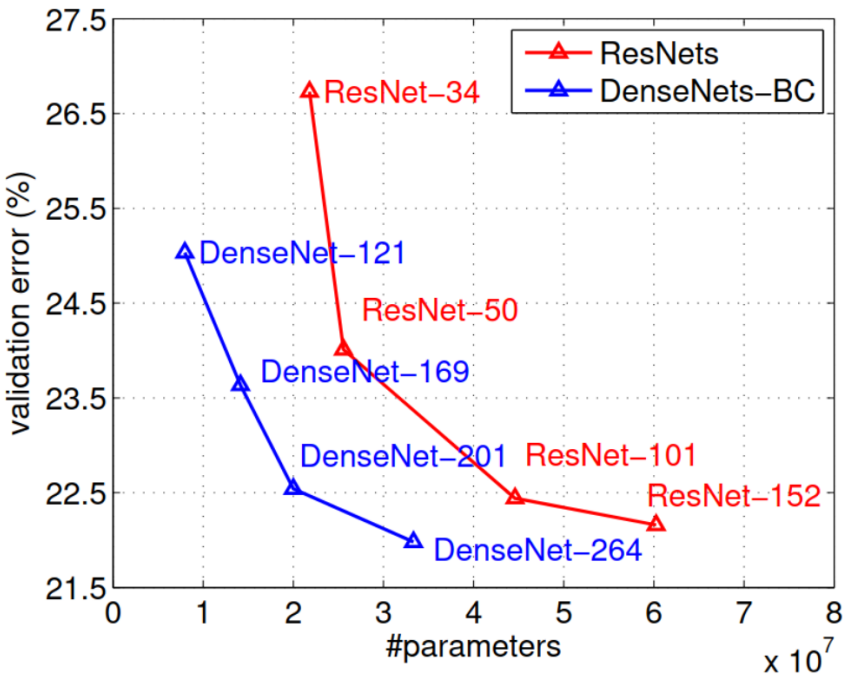
DenseNet variants

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

DenseNet vs. Resnet

Figure: DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (left) and FLOPs during test-time (right)

Although the concatenation operations reuse features to achieve computational efficiency, unfortunately they lead to heavy GPU memory consumption. As a result, applying DenseNet may require more memory-efficient implementations that may increase training time



Do I have to implement these architectures myself?

Classification

The following classification models are available, with or without pre-trained weights:

- AlexNet
- ConvNeXt
- DenseNet
- EfficientNet
- EfficientNetV2
- GoogLeNet
- Inception V3
- MaxVit
- MNASNet
- MobileNet V2
- MobileNet V3
- RegNet
- ResNet
- ResNeXt
- ShuffleNet V2
- SqueezeNet
- SwinTransformer
- VGG
- VisionTransformer
- Wide ResNet

```
import torchvision.models as models
resnet18 = models.resnet18()
alexnet = models.alexnet()
vgg16 = models.vgg16()
squeezenet = models.squeezenet1_0()
densenet = models.densenet161()
inception = models.inception_v3()
googlenet = models.googlenet()
shufflenet = models.shufflenet_v2_x1_0()
mobilenet_v2 = models.mobilenet_v2()
mobilenet_v3_large = models.mobilenet_v3_large()
mobilenet_v3_small = models.mobilenet_v3_small()
resnext50_32x4d = models.resnext50_32x4d()
wide_resnet50_2 = models.wide_resnet50_2()
mnasnet = models.mnasnet1_0()
```

PyTorch

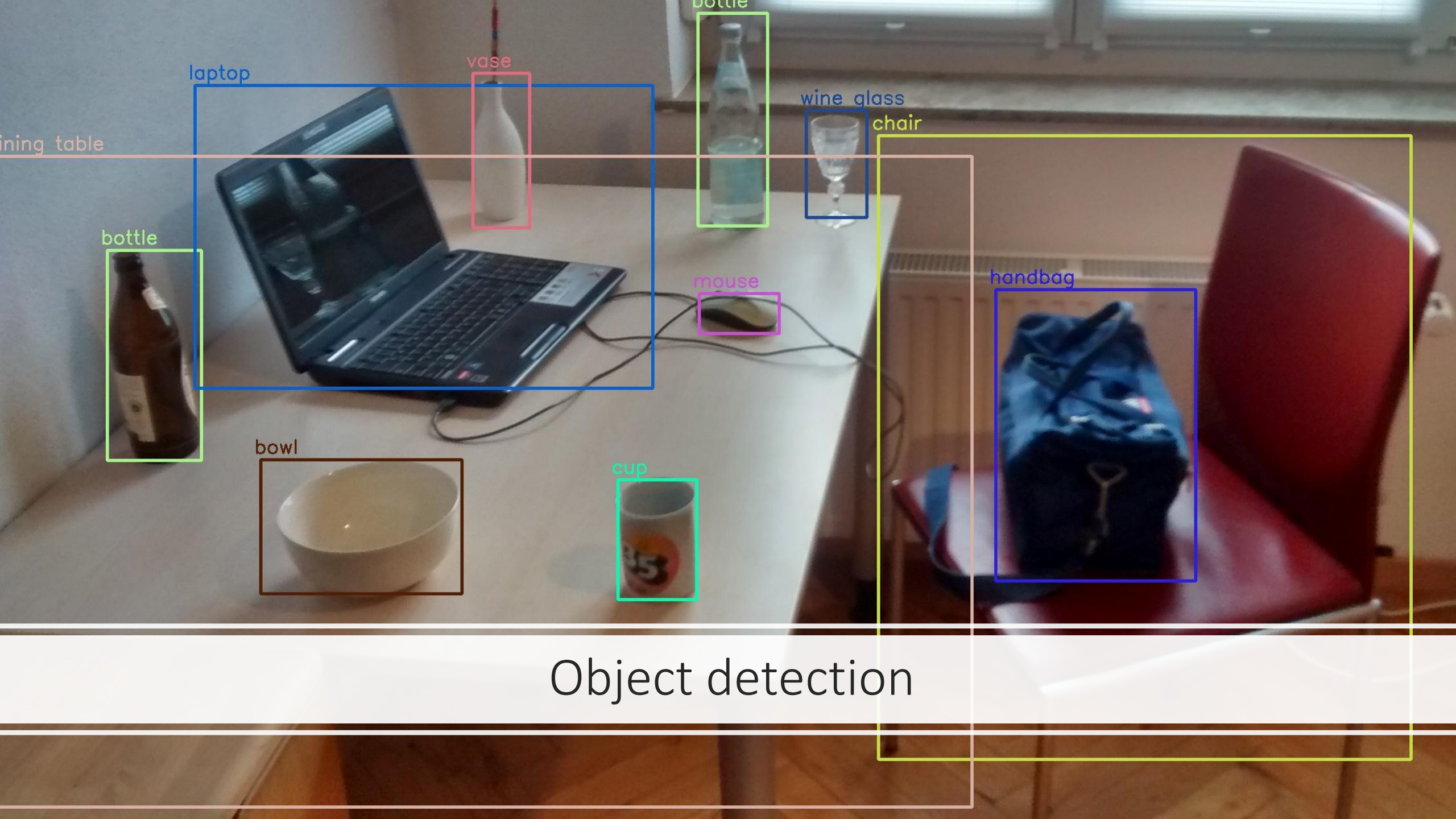
https://pytorch.org/vision/0.19/models.html

Example from MS-COCO

Image Classification **Object Detection** **Instance Segmentation**

Instance Segmentation vs. Semantic Segmentation

Input Image Semantic Segmentation Instance Segmentation



Object detection

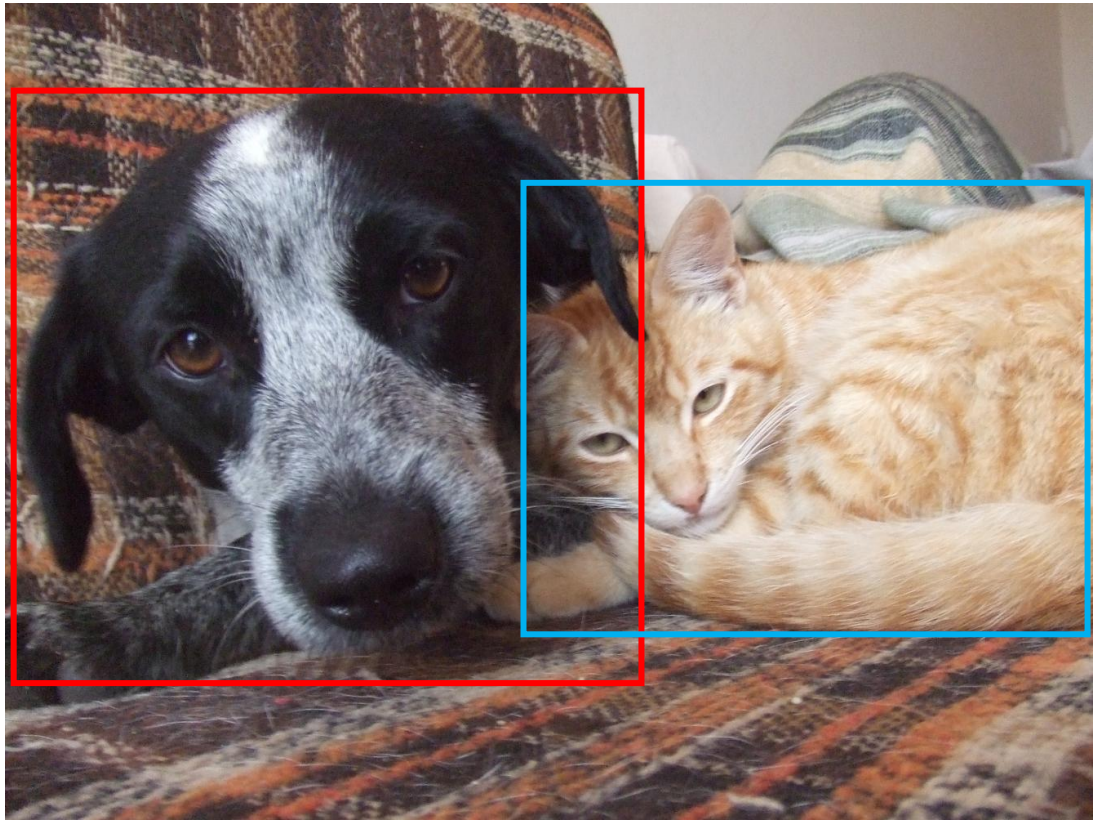
Object detection

- What and where
- Locations are often indicated by a bounding-box
- There are many approaches, but the most distinct are *two-step* detectors and *single-net* detectors

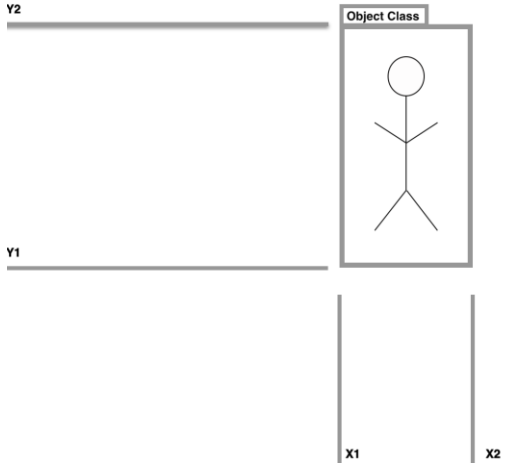
- Two step methods:
  - R-CNN, Fast R-CNN, Faster R-CNN
- Single-step methods:
  - YOLO
  - RetinaNet
  - DetectNet
  - SDD
  - …

Let's go through some of the approaches

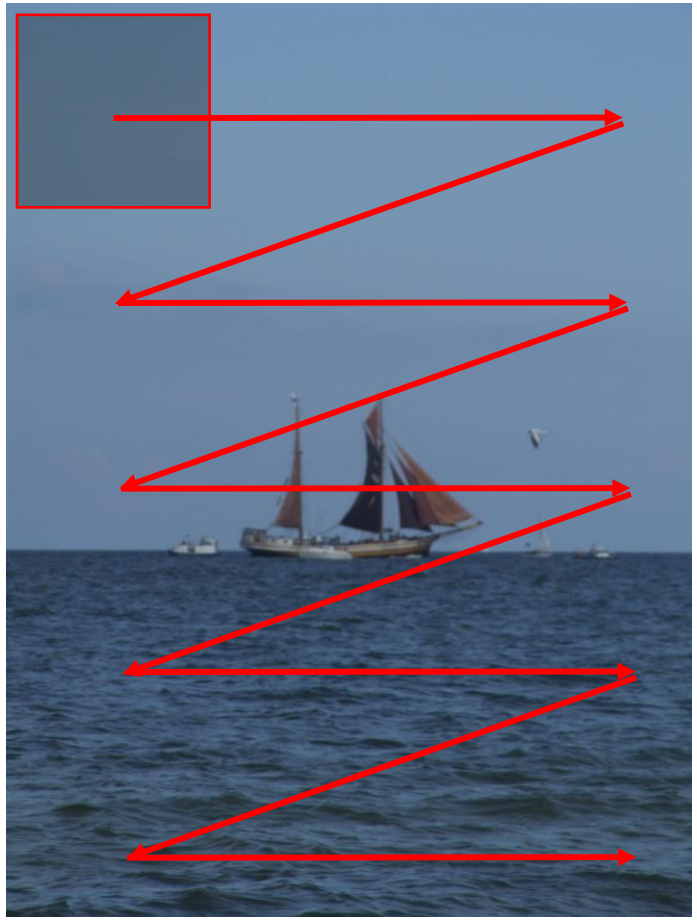
Bounding Boxes as location indicators



- Bounding boxes are often stored as either:
  - (x,y) coordinate of upper left corner + height+width
  - (x,y) coordinates of two opposite corners
  - (x,y) coordinate of center + height+width
  - Normalized (x,y) coordinate of center+height+width



Naïve sliding window approach



Pro:

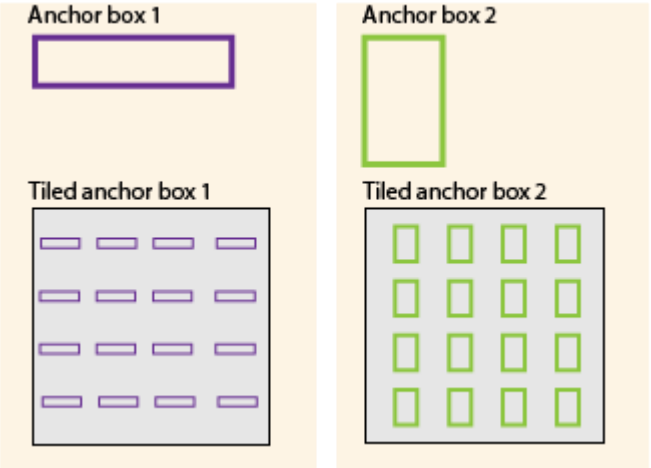
- easy to implement
- Can be trained on classification datasets like ImageNet

Con:

- Fixed box size
- Slow

Anchor boxes

- Anchor boxes eliminate the need to scan an image with a sliding window.
  - CNNs can produce image features for an entire image at once
- Anchor box prototypes are tiled across the image, at each location of the output.
- Each anchor box represents a specific prediction of a class.
  - I.e., It requires two anchor boxes to make two predictions per location. Three anchor boxes to make three predictions etc.



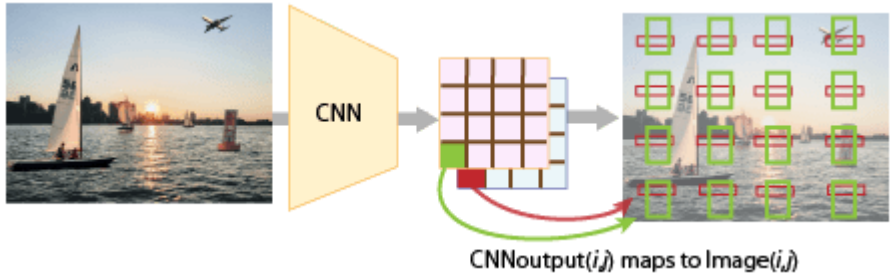


Image from https://www.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html

Anchor box location and size refinement

- Stride and pooling leads to smaller output feature maps, which increases the bbox location error.
- BBox offsets can be learned to refine the anchor box position and size.

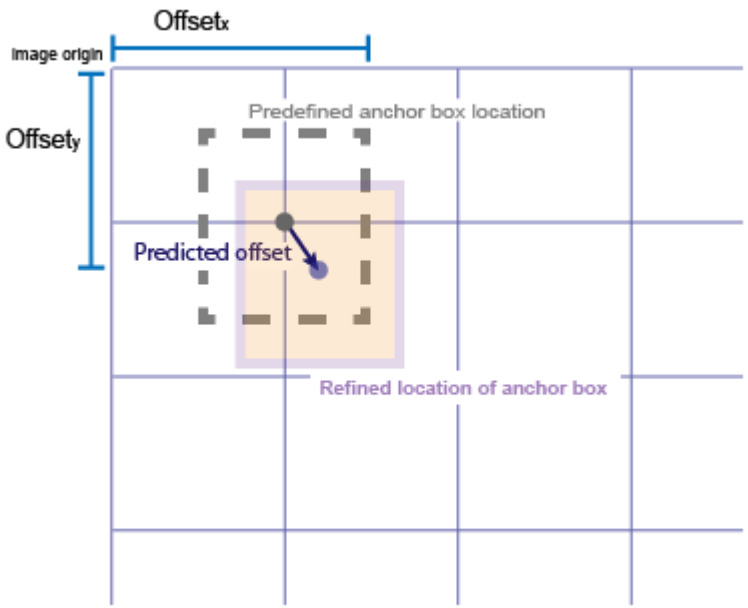


Image from https://www.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html

Training using anchor boxes

Many models start with anchor boxes as a prior and adjust from there.

Anchor boxes are a set of predefined bounding boxes of a certain height and width. These boxes are defined to capture the scale and aspect ratio of specific object classes you want to detect and are typically chosen based on object sizes in your training datasets.

1. Form thousands of candidate anchor boxes around the image
2. For each anchor box predict some offset from that box as a candidate box
3. Calculate a loss function based on the ground truth example
4. Calculate a probability that a given offset box overlaps with a real object
5. If that probability is greater than 0.5, factor the prediction into the loss function
6. By rewarding and penalizing predicted boxes slowly pull the model towards only localizing true objects

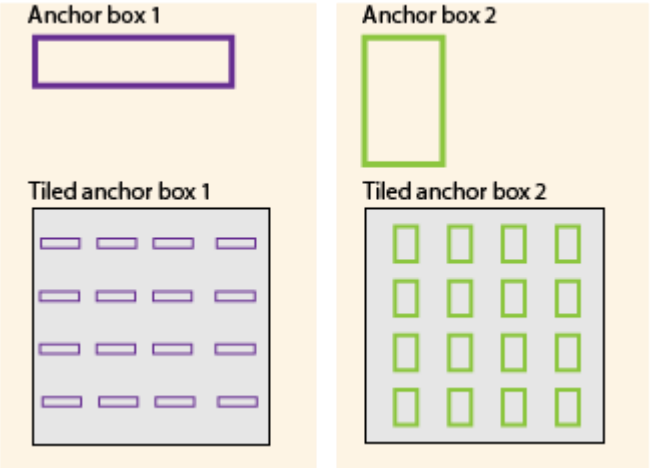
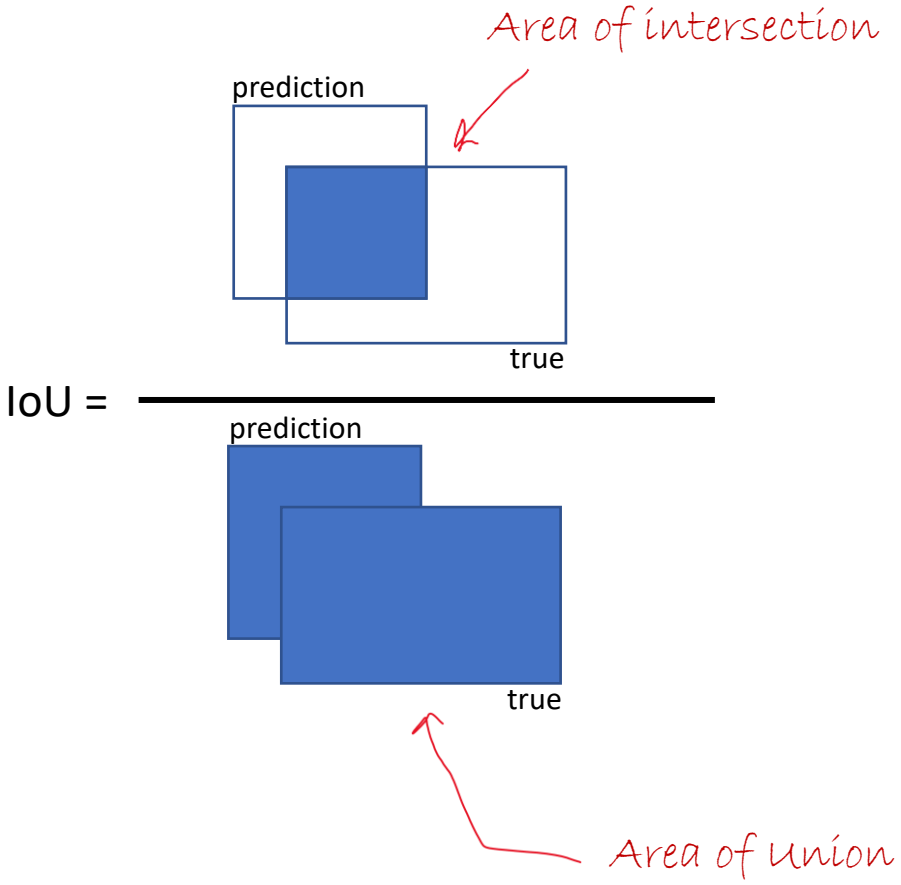


Image from https://www.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html

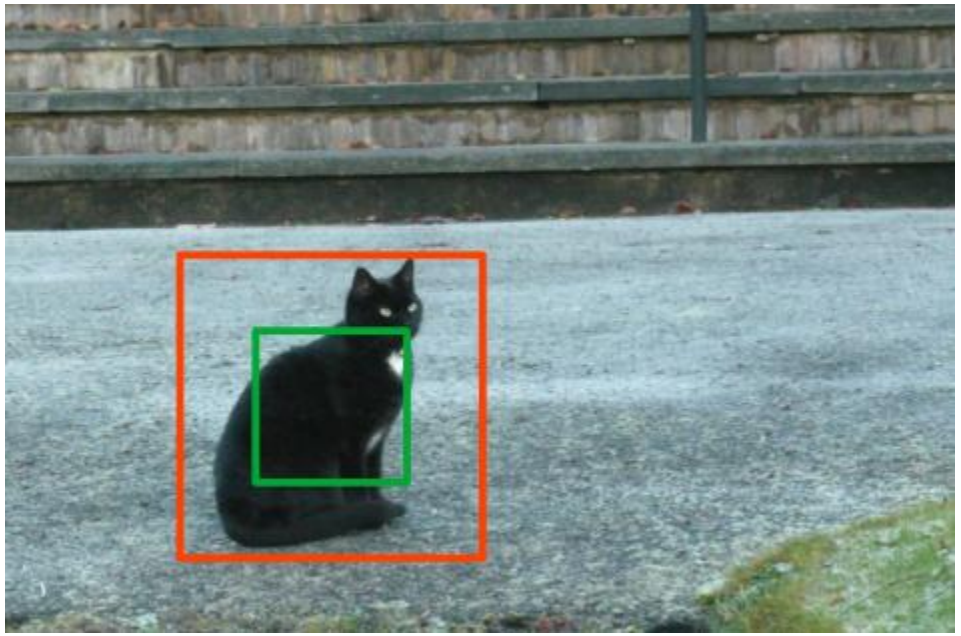
Intersection over union (IoU)

- A measure of how good a bounding box fits an object

$$IOU = \frac{Area\ of\ Intersection\ of\ two\ boxes}{Area\ of\ Union\ of\ two\ boxes}$$



IoU samples



True, predict

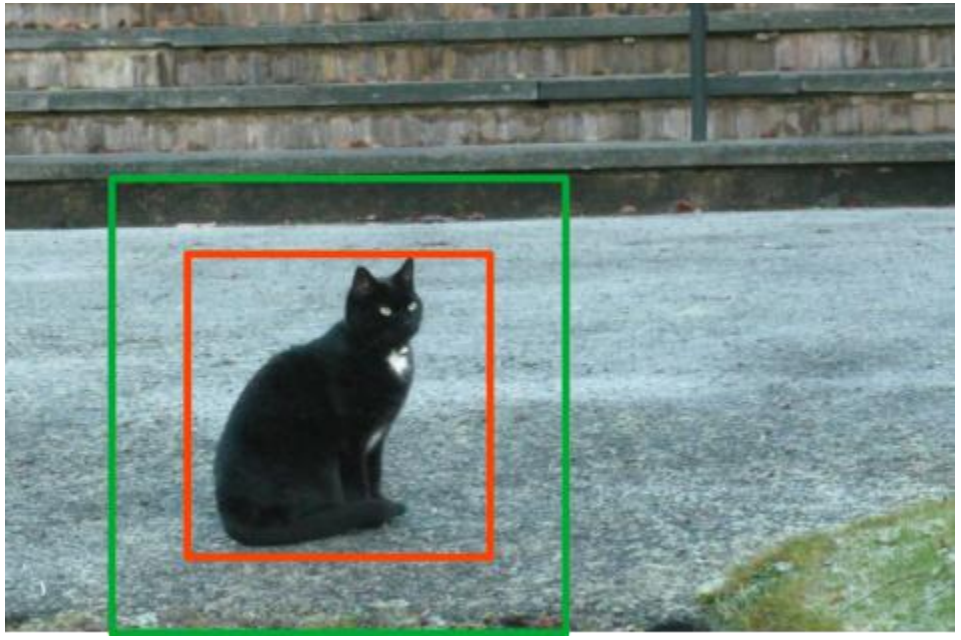
IoU = 0,25

IoU samples

IoU = 0,5

True, predict

IoU samples

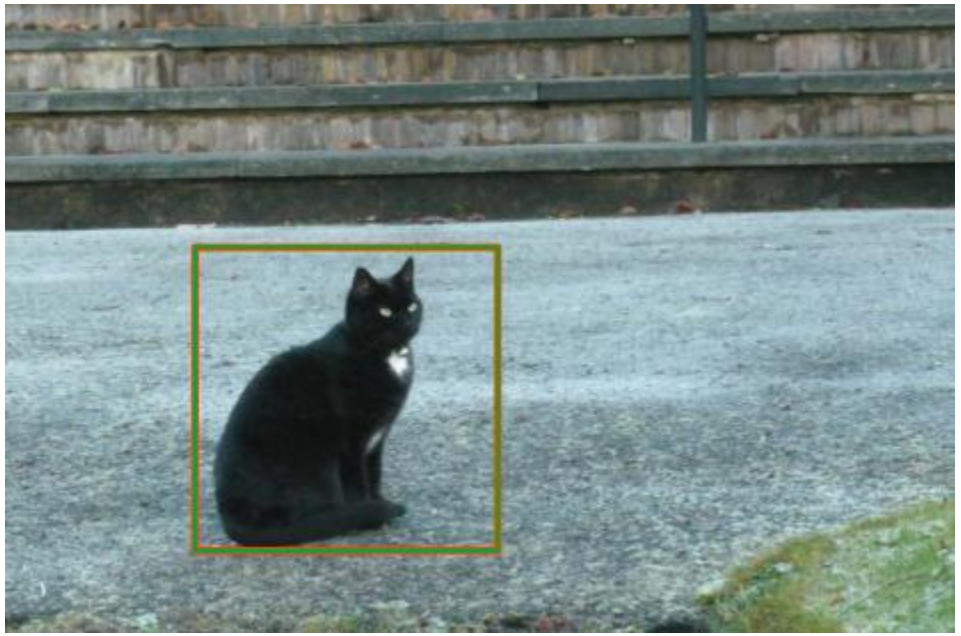


True, predict

IoU = 0,64

IoU samples

Is IoU a good measure?
Discuss for two minutes with the person next to you, why or why not?



True, predict

IoU = 1

Observations

When people annotate training data, the sizes of bounding boxes are often less precise for small objects.

For small bounding boxes, few pixels offset have huge impact on the IoU.

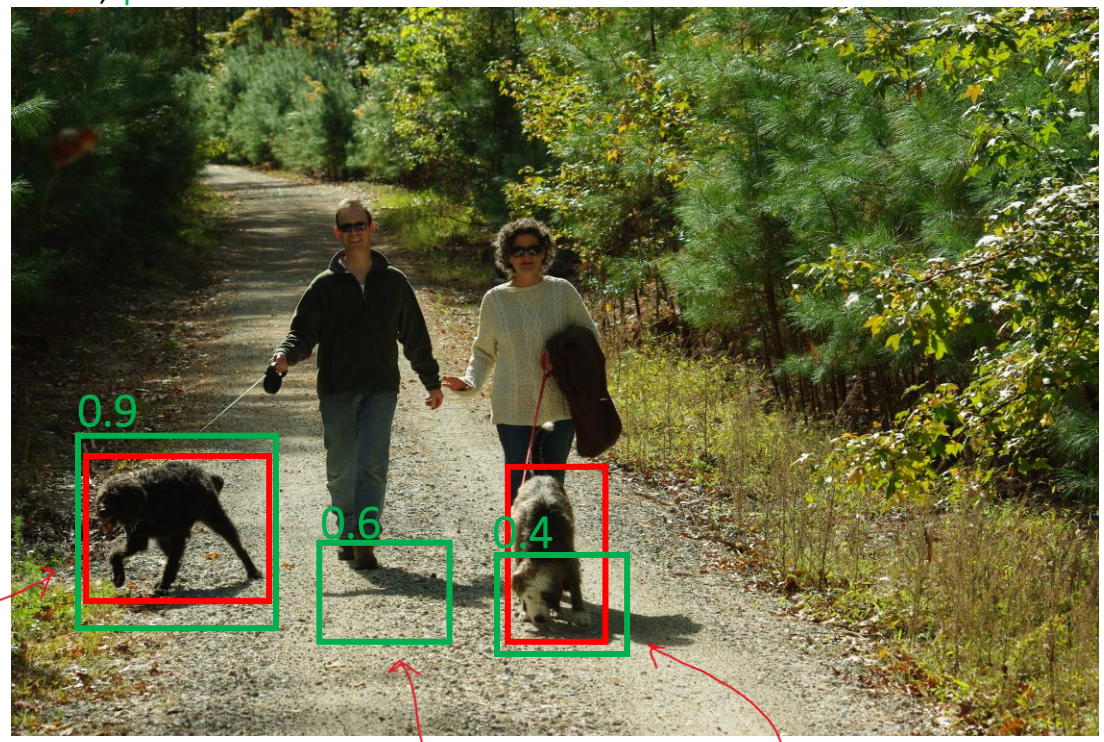
In this image, an IoU>0.5 is fine for the small boxes, based on the annotations

Do we always need a high IoU??

Mean Average Precision (mAP)

- Let's assume we have two object detectors that are trained on the same dataset. How do we determine which one is better?
  - We need some metric to evaluate object detection performance on the test set so that we can compare algorithms,
- We evaluate the IoU of annotations and predictions.
- We select a fixed IOU score to determine whether we should accept or reject a detection. This is typically 0.5 (often denoted mAP@0.50)

True, predict



IoU **is** greater than 0.5 (TP)

IoU **not** greater than 0.5 (FP)

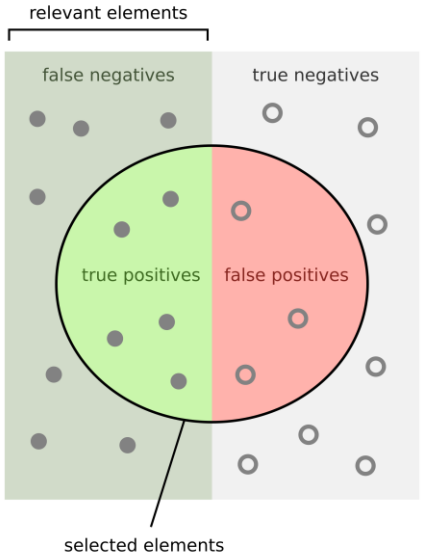
IoU **not** greater than 0.5 (FP)

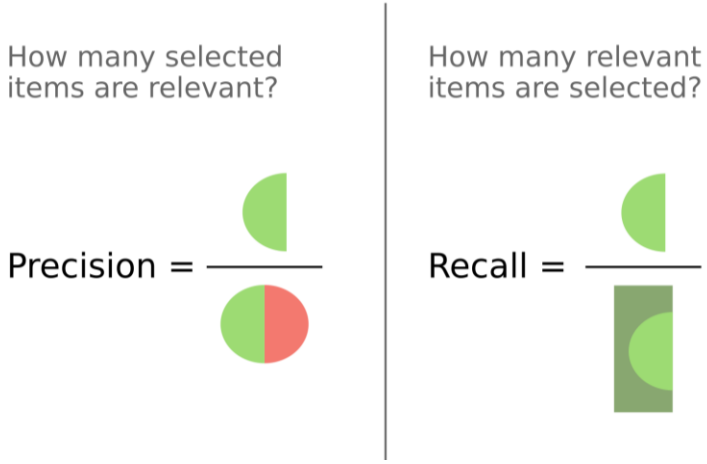
Precision and Recall

Precision = off all predictions, what fraction was correct?

Recall = Of all objects, what fraction did we detect?

We need to find a threshold for accepting detections, which balances precision and recall.

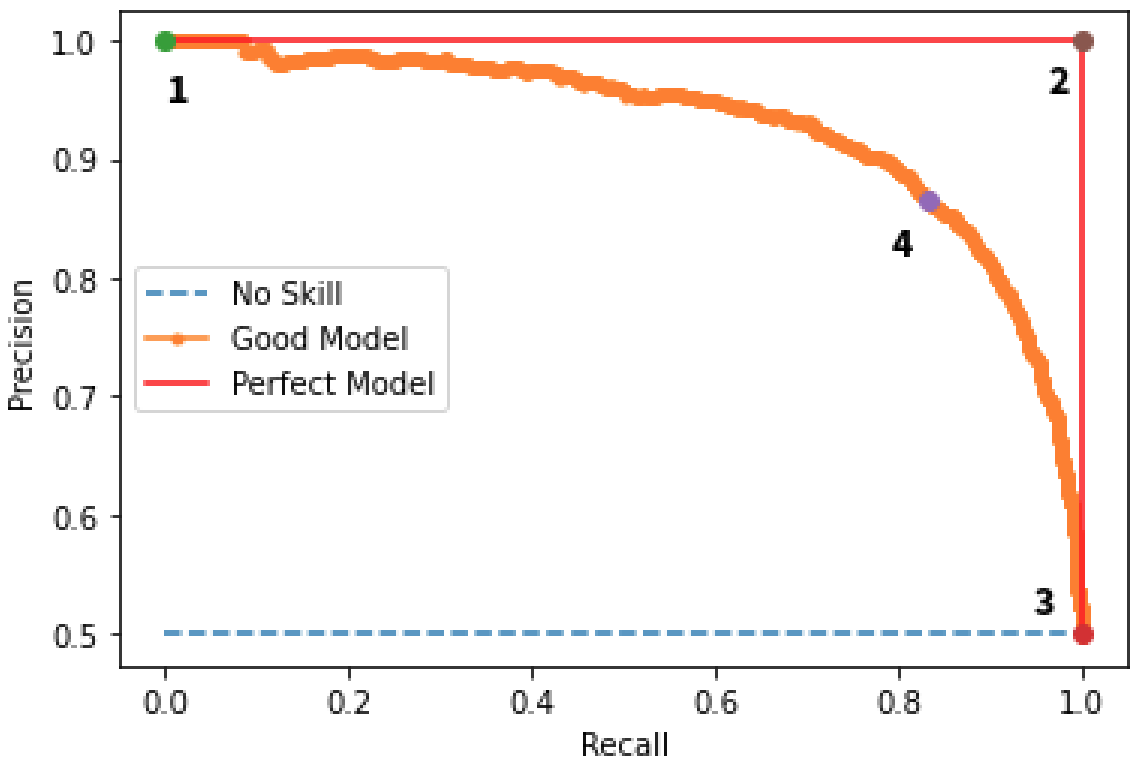




$$\text{precision} = \frac{TP}{TP + FP}$$

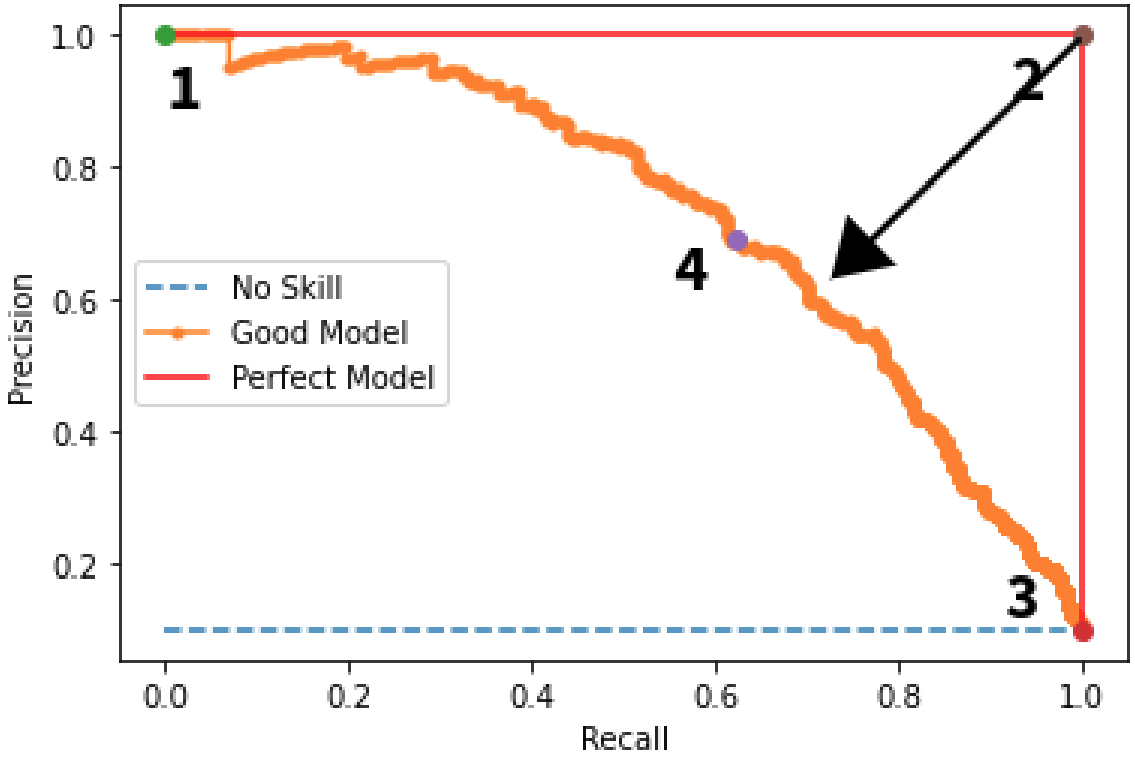
$$\text{recall} = \frac{TP}{TP + FN}$$

Precision-Recall curve



- On the curves, each point corresponds to a different confidence threshold, and its location corresponds to the resulting Precision and Recall when we choose that threshold.
  - Point 1 corresponds to the threshold of 1
  - Point 3 corresponds to the threshold of 0
  - Point 4 corresponds to the threshold somewhere in the range (0, 1)
  - Point 2 corresponds to a Perfect model (along with Point 3)
- The Area under the PR Curve is a metric that helps us compare 2 similar looking curves. The higher the area under curve, the better the performance.

Finding the optimal confidence threshold on Precision-Recall curve



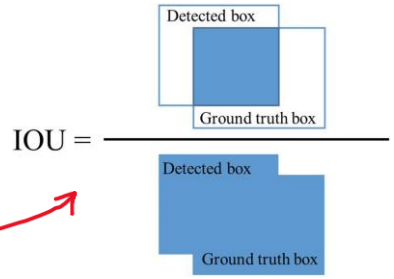
Two ways of selecting threshold:

- Find the euclidean distance of every point on the curve, which is denoted by (recall, precision) for a corresponding threshold, from (1,1).
  - Pick the point and the corresponding threshold, for which the distance is minimum.
- Find F1 score for each point (recall, precision) and the point with the maximum F1 score is the desired optimal point.

In some cases, you may prefer a higher recall or a higher precision. Then selecting the threshold is up to you :-)

https://analyticsindiamag.com/complete-guide-to-understanding-precision-and-recall-curves/

Mean Average Precision (mAP)



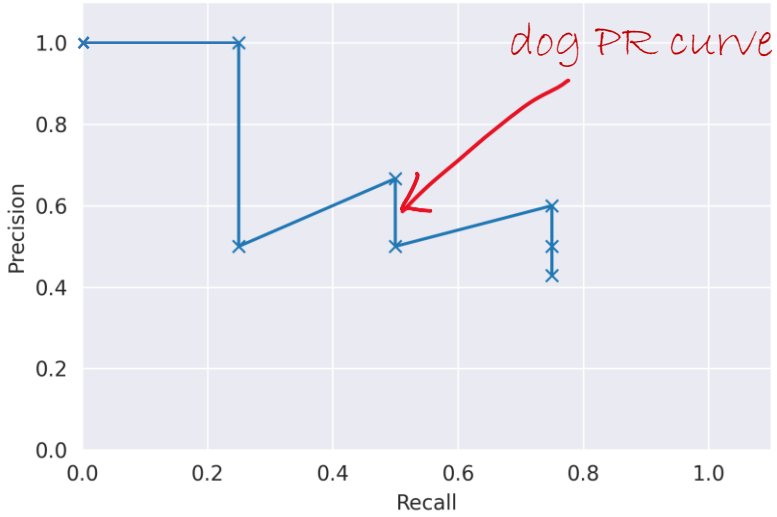
Box#	Confidence	TP/FP	Precision	Recall
0	1.0	--		
1	0.9	TP		
2	0.8	FP		
3	0.7	TP		
4	0.6	FP		
4	0.5	--		
5	0.4	TP		
6	0.3	FP		
6	0.2	--		
7	0.1	FP		
7	0.0	--		

- We use IoU to determine if box is TP or FP
- Let's say all our images have 4 annotated bounding boxes in total
- All bounding boxes in all images are sorted by the prediction confidence and we evaluate from the top while lowering the confidence threshold

Precision = off all predictions, what fraction was correct?
Recall = Of all objects (4), what fraction did we detect?

Mean Average Precision (mAP)

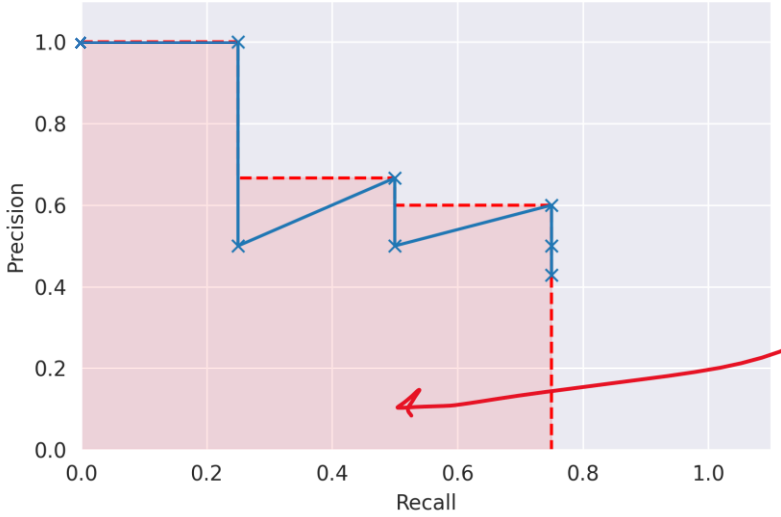
Box#	Confidence	TP/FP	Precision	Recall
0	1.0	--	1	0
1	0.9	TP	1/1	1/4
2	0.8	FP	1/2	1/4
3	0.7	TP	2/3	2/4
4	0.6	FP	2/4	2/4
4	0.5	--	2/4	2/4
5	0.4	TP	3/5	3/4
6	0.3	FP	3/6	3/4
6	0.2	--	3/6	3/4
7	0.1	FP	3/7	3/4
7	0.0	--	3/7	3/4



Now we have a PR-curve, but we are not finished yet.

Mean Average Precision (mAP)

Box#	Confidence	TP/FP	Precision	Recall
0	1.0	--	1	0
1	0.9	TP	1/1	1/4
2	0.8	FP	1/2	1/4
3	0.7	TP	2/3	2/4
4	0.6	FP	2/4	2/4
4	0.5	--	2/4	2/4
5	0.4	TP	3/5	3/4
6	0.3	FP	3/6	3/4
6	0.2	--	3/6	3/4
7	0.1	FP	3/7	3/4
7	0.0	--	3/7	3/4



$AP_{dog} = 0.57$

Now we have a PR-curve, but we are not finished yet. Next step is to interpolate this curve using the following interpolation:

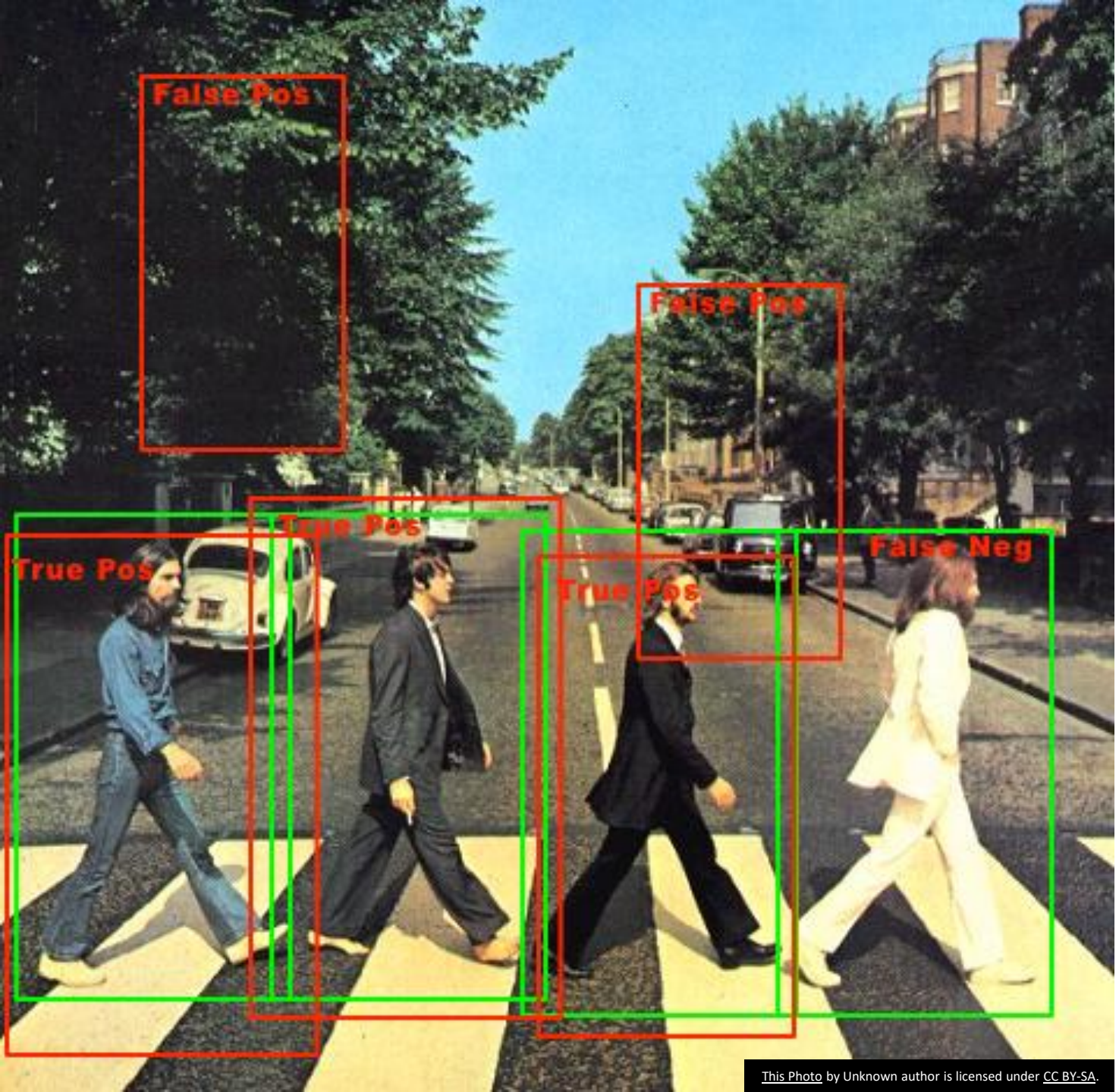
$$p_{interp}(R) = \max_{\widetilde{R}:\widetilde{R}\geq R} P(R)$$

A way to visualize this is: walk from right to left and take the highest value observed so far.
The average prevision is the area under that interpolated curve:

$$AP = \sum_n (R_{n+1} - R_n) P(R_{n+1})$$

This is just for one class. The Mean Average Precision (mAP) is the mean of the AP of all classes.

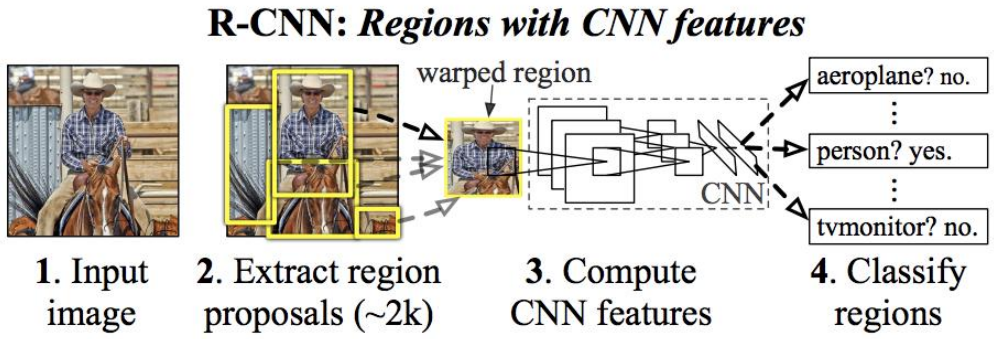
$$mAP = (AP_{dog} + AP_{cat})/2$$



This Photo by Unknown author is licensed under CC BY-SA.

Object detection methods

R-CNN



- 2000 regions are extracted from an image using selective search (fixed, non-trainable algorithm)
- Each region is fed into a CNN. If a region gets a high score, we keep it

Pro:

- It is pretty accurate

Con:

- Too slow for real-time operation

https://arxiv.org/abs/1311.2524

Accurate object detection is slow!

	Pascal 2007 mAP	**Speed**	
R-CNN	66.0	.05 FPS	20 s/img

Accurate object detection is slow!

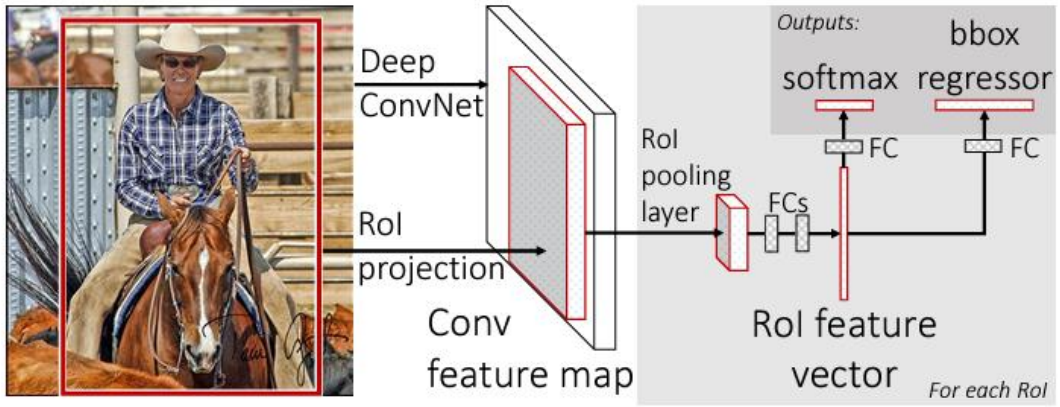
	Pascal 2007 mAP	**Speed**	
R-CNN	66.0	.05 FPS	20 s/img

Accurate object detection is slow!

	Pascal 2007 mAP	**Speed**	
R-CNN	66.0	.05 FPS	20 s/img

444m @ 80km/h

Fast R-CNN



- The entire image is passed through a CNN once to generate a feature map.
- The region proposals (from selective search) are applied this feature map using **RoI Pooling**.
- **RoI Pooling** extracts a fixed-size feature map for each proposal for classification.
- **Pro**:
- The CNN is only run once for the whole image

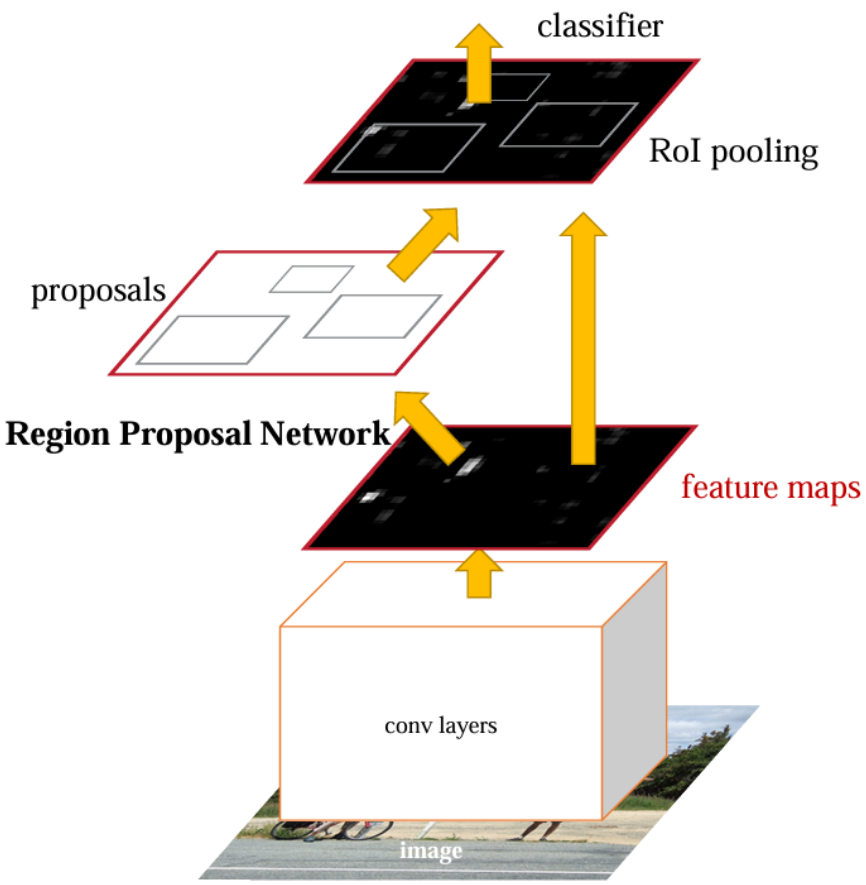
https://arxiv.org/abs/1504.08083

Accurate object detection is slow!

	Pascal 2007 mAP	**Speed**	
R-CNN	66.0	.05 FPS	20 s/img
Fast R-CNN	70.0	.5 FPS	2 s/img

44m @ 80km/h

Faster R-CNN



- The entire image is passed through a CNN once to generate a feature map.
- Region Proposal Network (RPN), which is a small network that learns to propose regions directly from the feature map
- These region proposals are then passed through RoI Pooling for classification.
- **Pro**:
- RPN learns to generate region proposals.

https://arxiv.org/abs/1506.01497

Accurate object detection is slow!

	Pascal 2007 mAP	**Speed**	
R-CNN	66.0	.05 FPS	20 s/img
Fast R-CNN	70.0	.5 FPS	2 s/img
Faster R-CNN	73.2	7 FPS	140 ms/img

3,1m @ 80km/h

YOLO

- Object detector that can detect all object in a single forward pass
- Fast!
- Several versions exist
  - Four officially (YOLOv1 – YOLOv4)
  - + some unofficial, pretty good and available in PyTorch (YOLOv5-v8)

- We will go through the original version.

You Only Look Once: Unified, Real-Time Object Detection

Joseph Redmon
University of Washington
pjreddie@cs.washington.edu

Santosh Divvala
Allen Institute for Artificial Intelligence
santoshd@allenai.org

Ross Girshick
Facebook AI Research
rbg@fb.com

Ali Farhadi
University of Washington
ali@cs.washington.edu

Abstract

We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second while still achieving double the mAP of other real-time detectors. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is far less likely to predict false detections where nothing exists. Finally, YOLO learns very general representations of objects. It outperforms all other detection methods, including DPM and R-CNN, by a wide margin when generalizing from natural images to artwork on both the Picasso Dataset and the People-Art Dataset.

1. Introduction

Humans glance at an image and instantly know what objects are in the image, where they are, and how they interact. The human visual system is fast and accurate, allowing us to perform complex tasks like driving with little conscious thought. Fast, accurate, algorithms for object de-

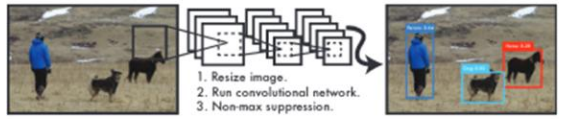


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

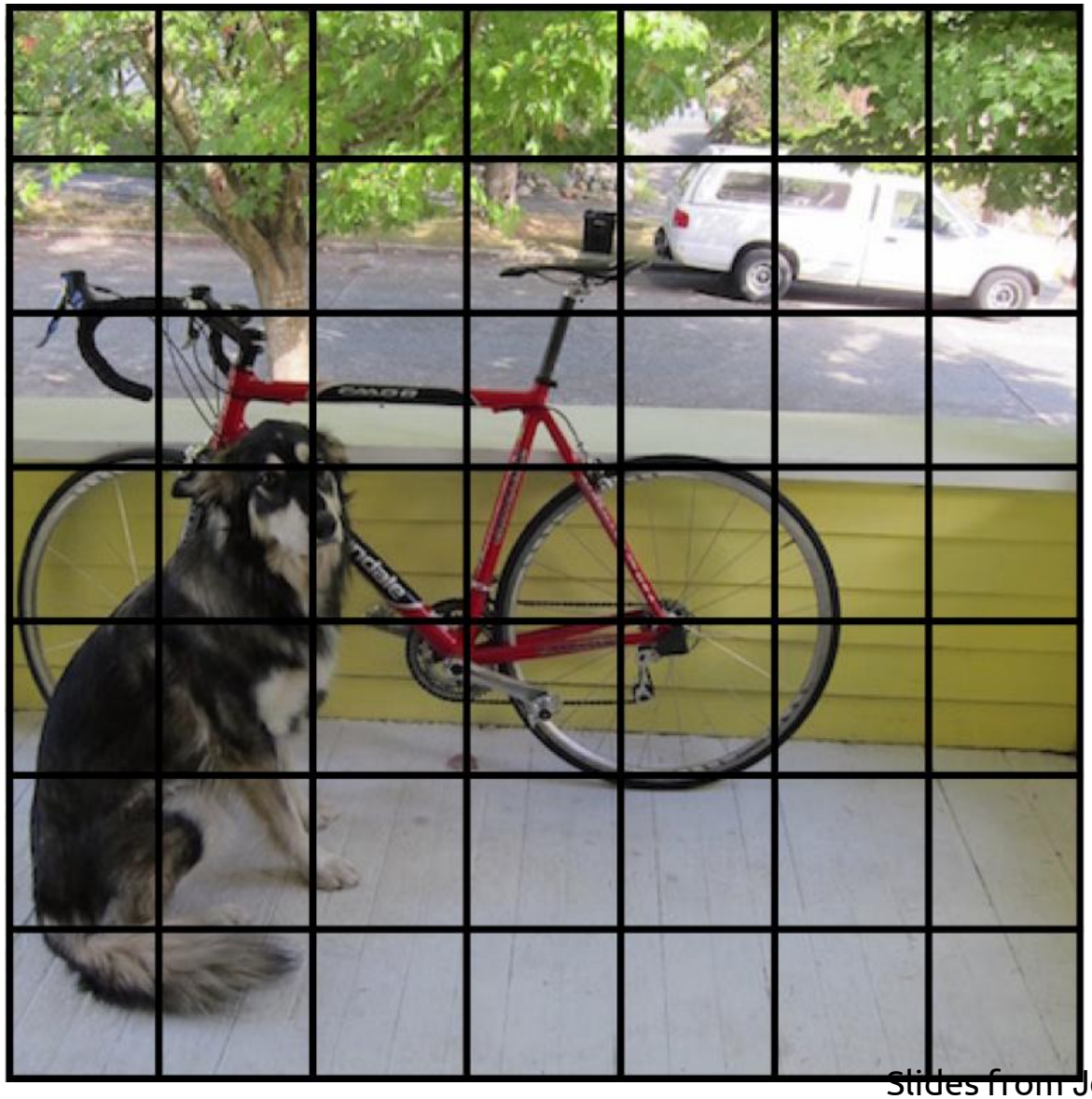
classifier for that object and evaluate it at various locations and scales in a test image. Systems like deformable parts models (DPM) use a sliding window approach where the classifier is run at evenly spaced locations over the entire image [10].

More recent approaches like R-CNN use region proposal methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes. After classification, post-processing is used to refine the bounding box, eliminate duplicate detections, and rescore the box based on other objects in the scene [13]. These complex pipelines are slow and hard to optimize because each individual component must be trained separately.

We reframe object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Using our system, you only look once (YOLO) at an image to predict what objects are present and where they are.

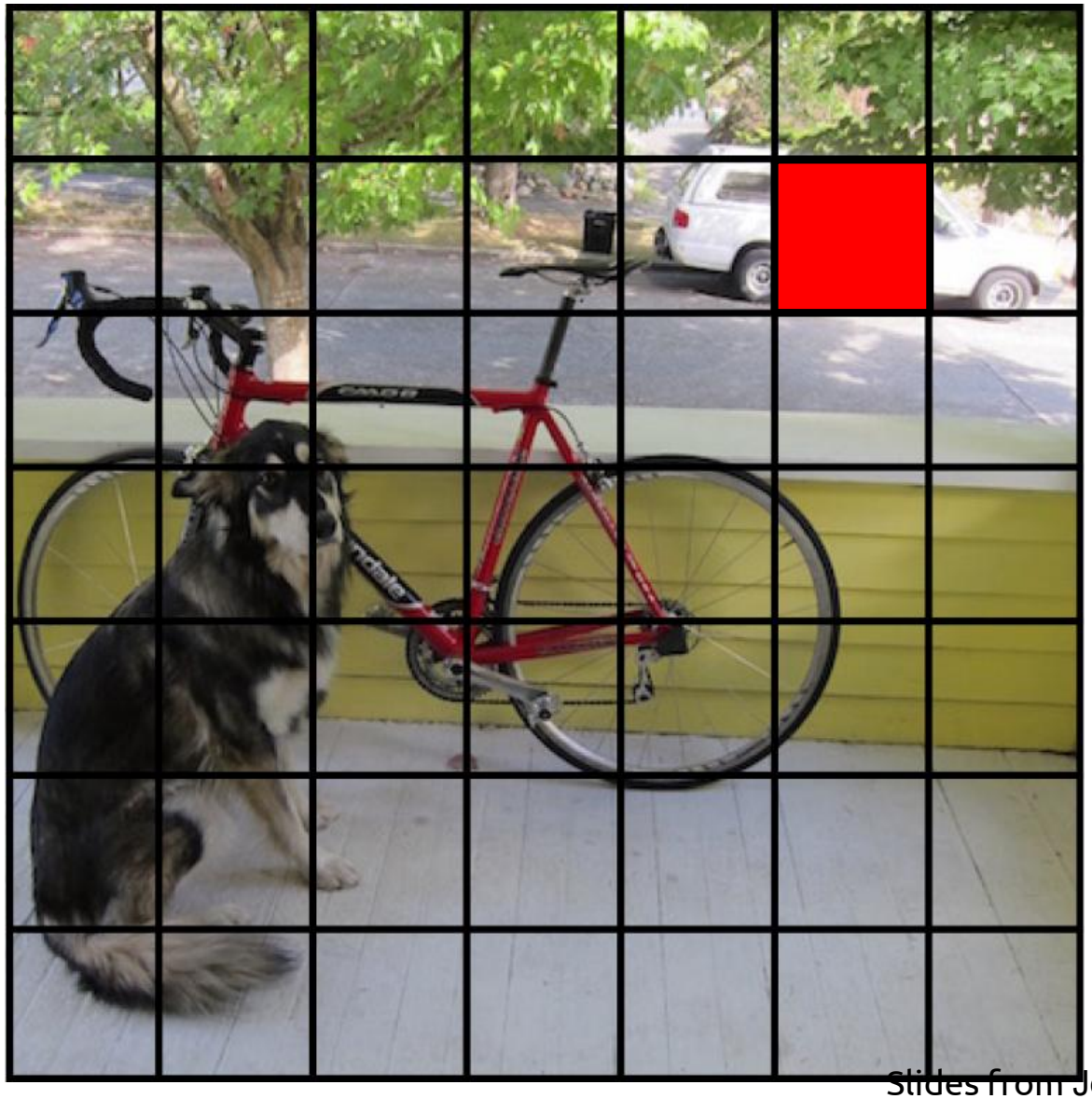
YOLO is refreshingly simple: see Figure 1. A single convolutional network simultaneously predicts multi-

YOLO: We split the image into a grid



Slides from Joseph Redmon's CVPR2016 presentation

Each cell predicts boxes and confidences for each of the boxes: P(Object)

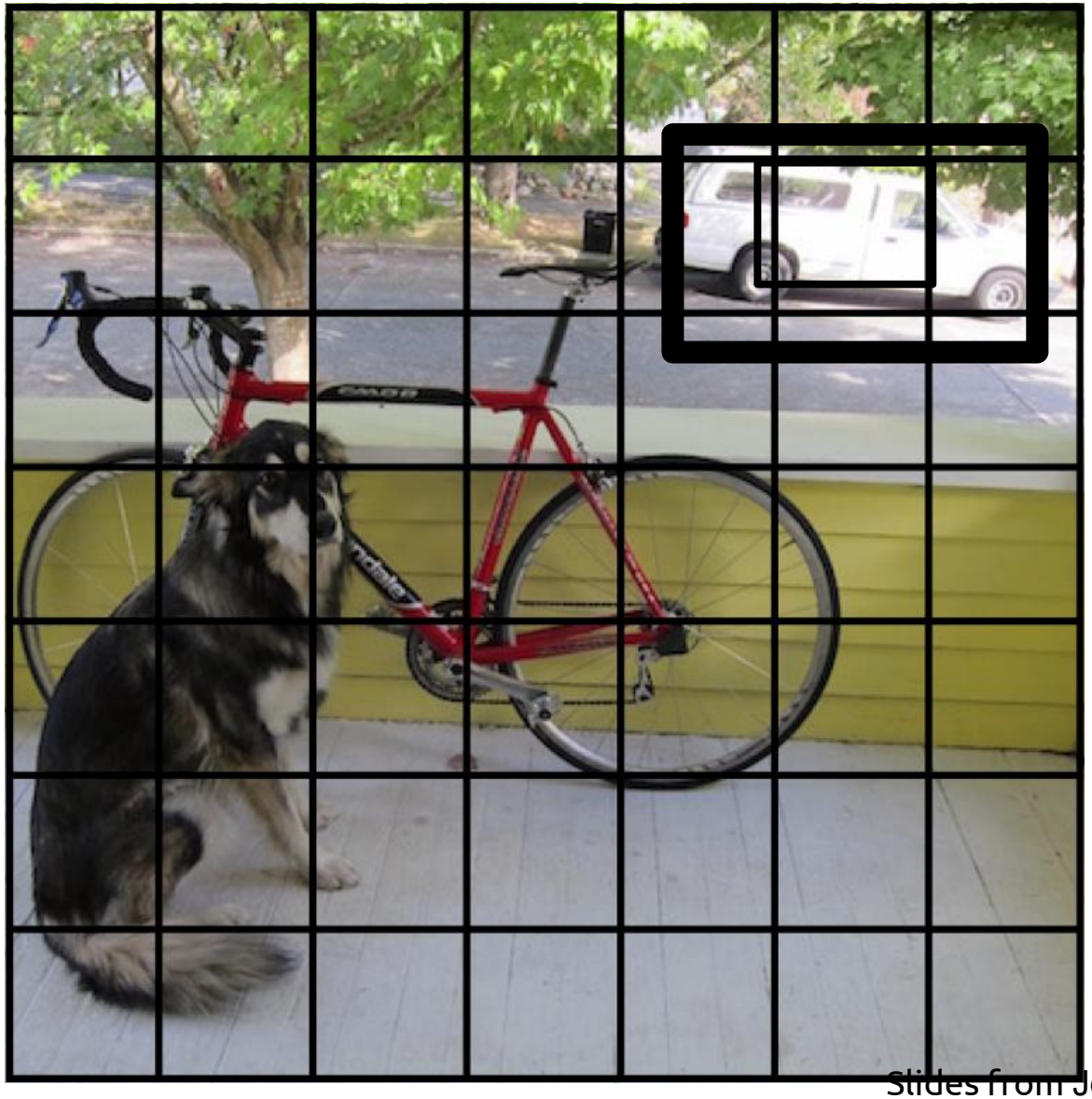


Slides from Joseph Redmon's CVPR2016 presentation

Each cell predicts boxes and confidences for each of the boxes: P(Object)

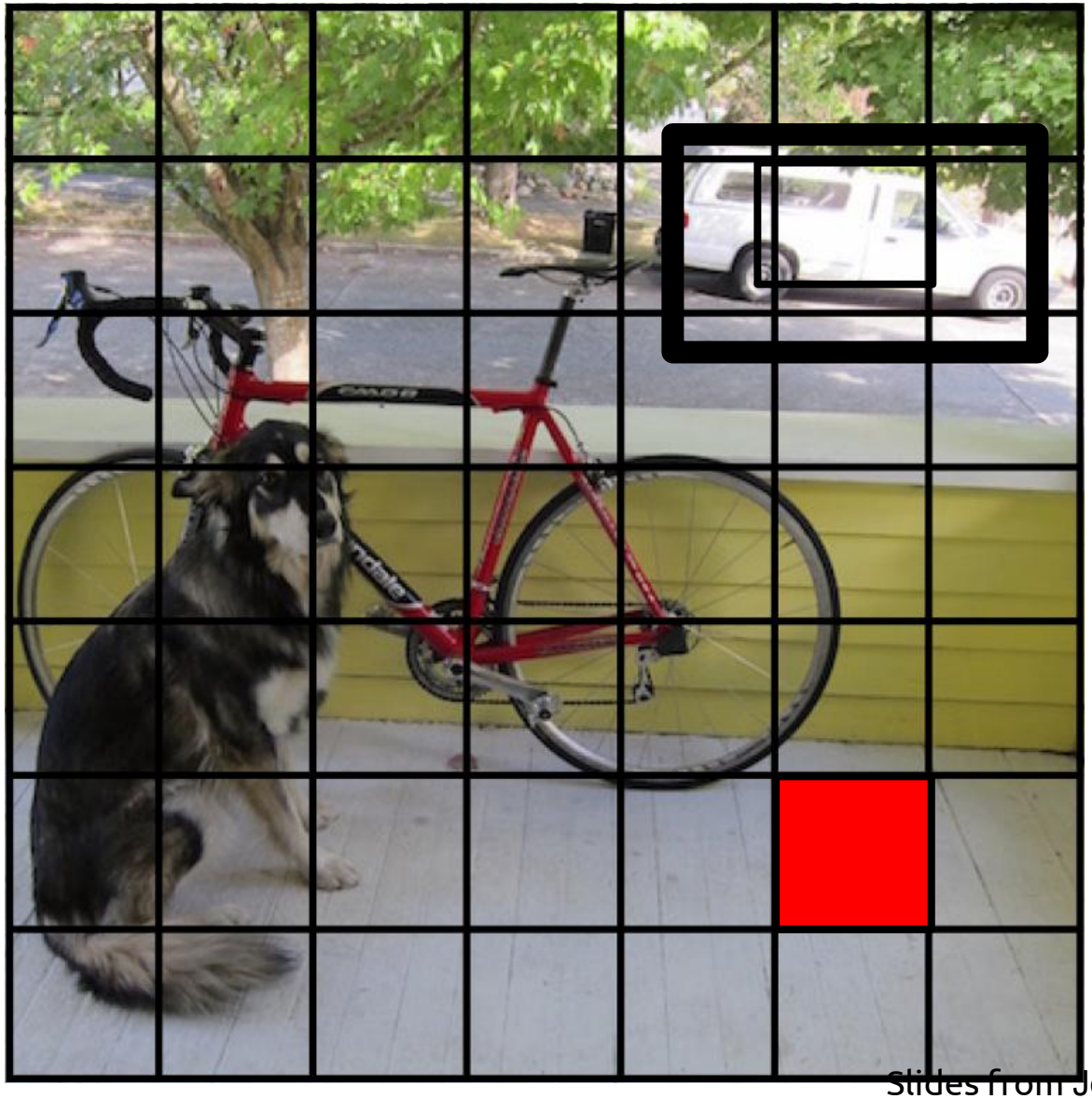
Slides from Joseph Redmon's CVPR2016 presentation

Each cell predicts boxes and confidences for each of the boxes: P(Object)



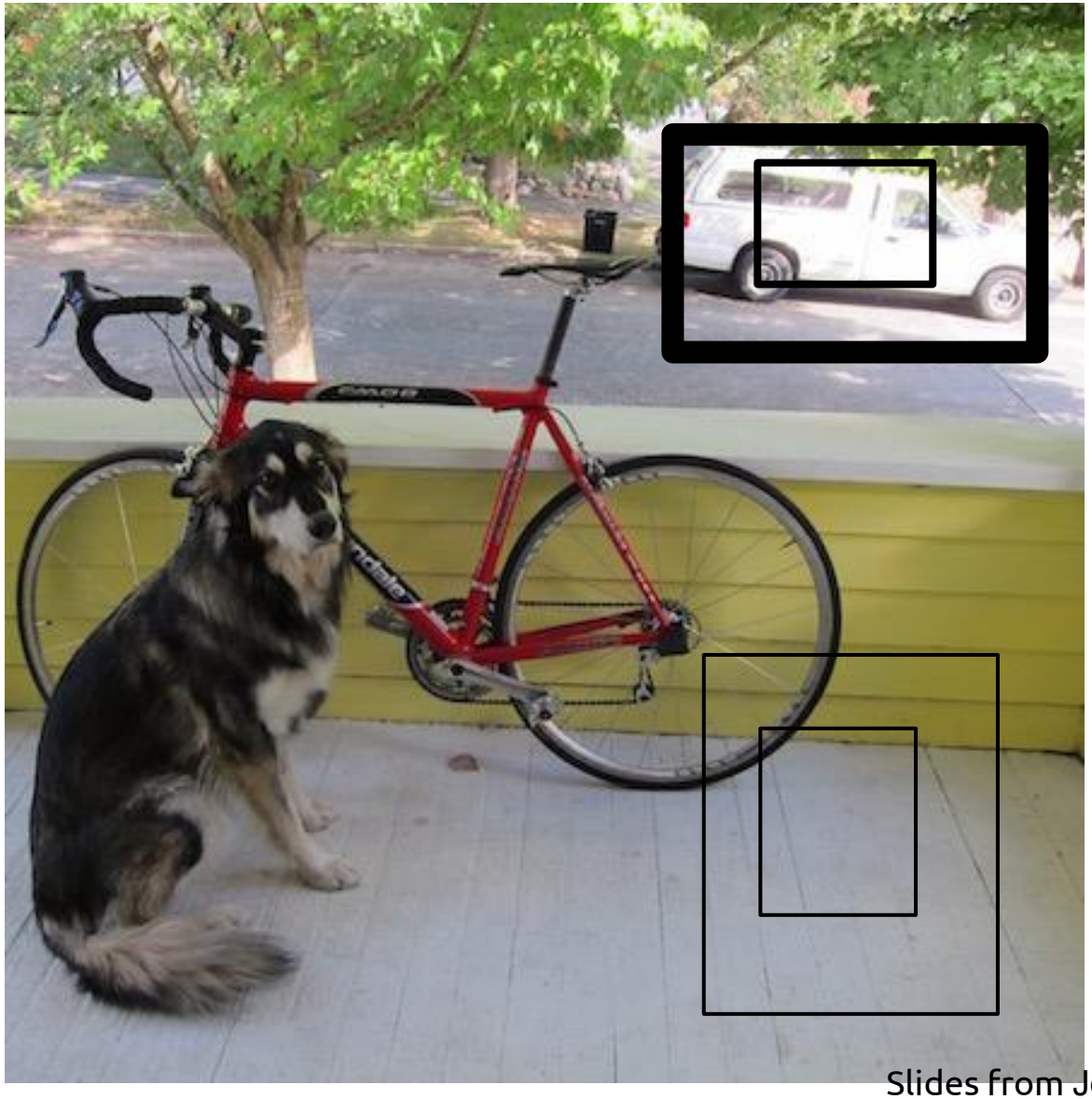
Slides from Joseph Redmon's CVPR2016 presentation

Some cell don't have any objects. We still predict a bounding box, but want the conficende to be low



Slides from Joseph Redmon's CVPR2016 presentation

Some cell don't have any objects. We still predict a bounding box, but want the conficende to be low

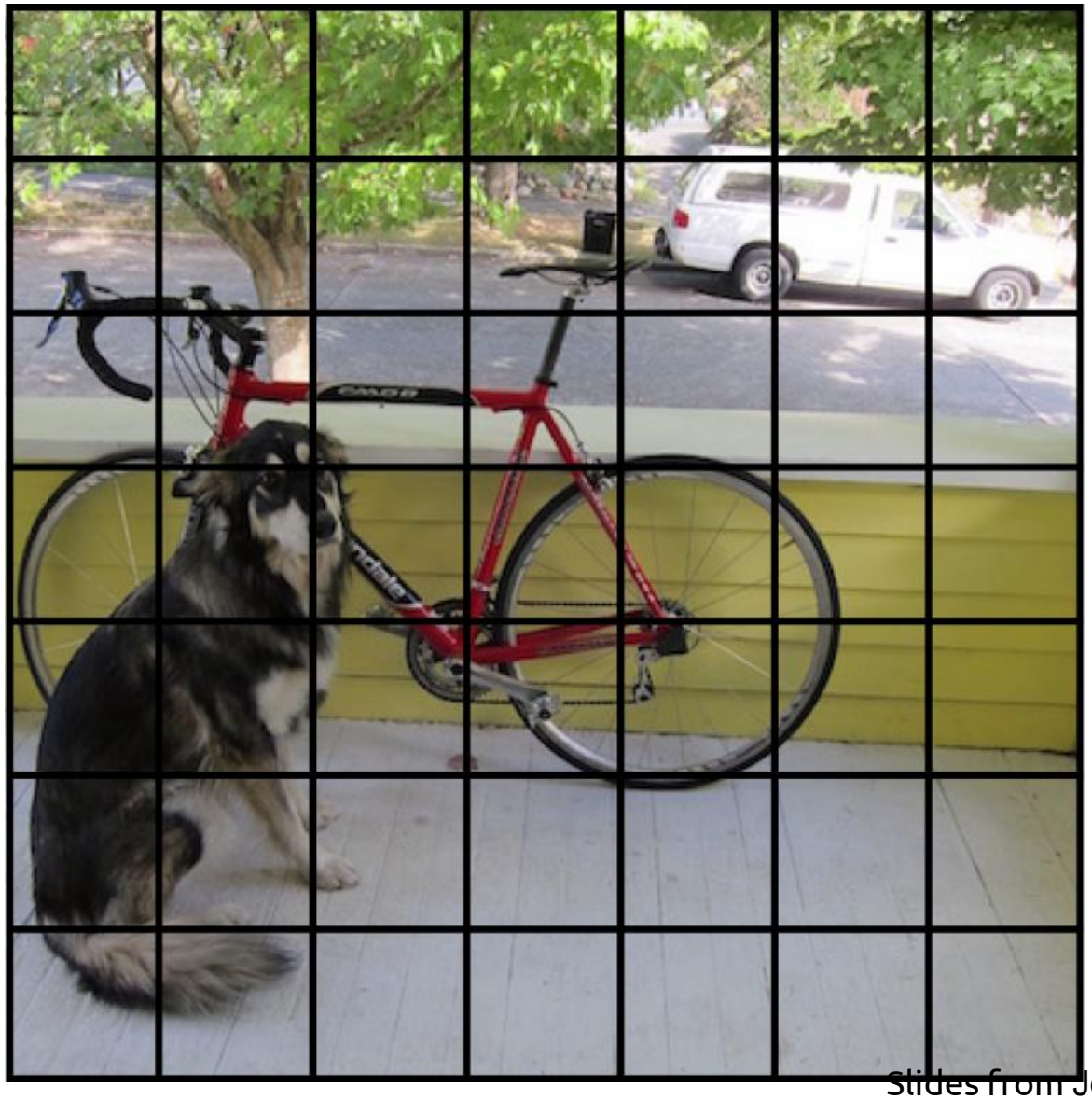


Slides from Joseph Redmon's CVPR2016 presentation

Visualization of all predicted boxes, line thickness corresponds to confidence

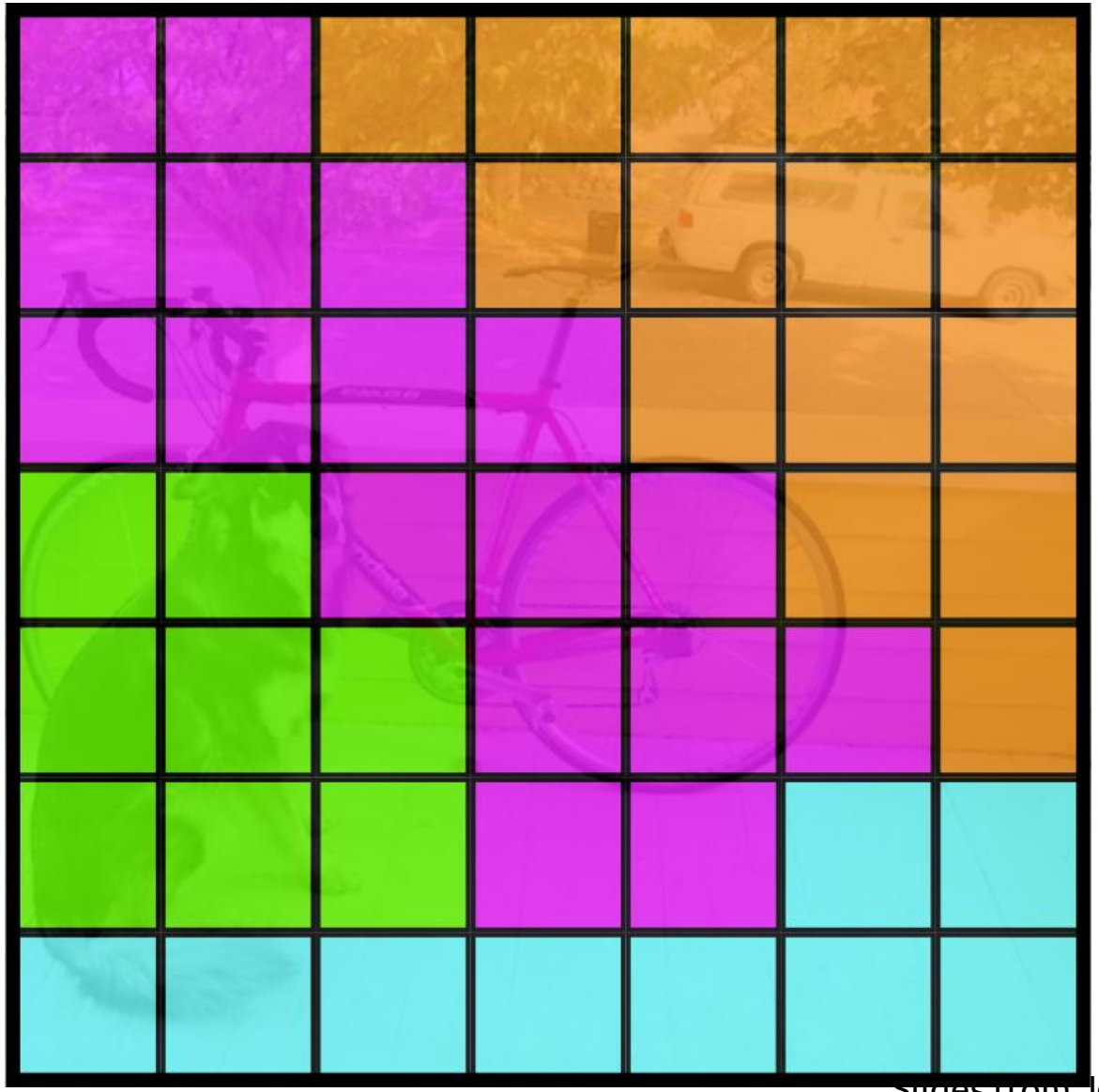
Slides from Joseph Redmon's CVPR2016 presentation

Each cell also predicts a class probability. Which looks like a corse segmentation map



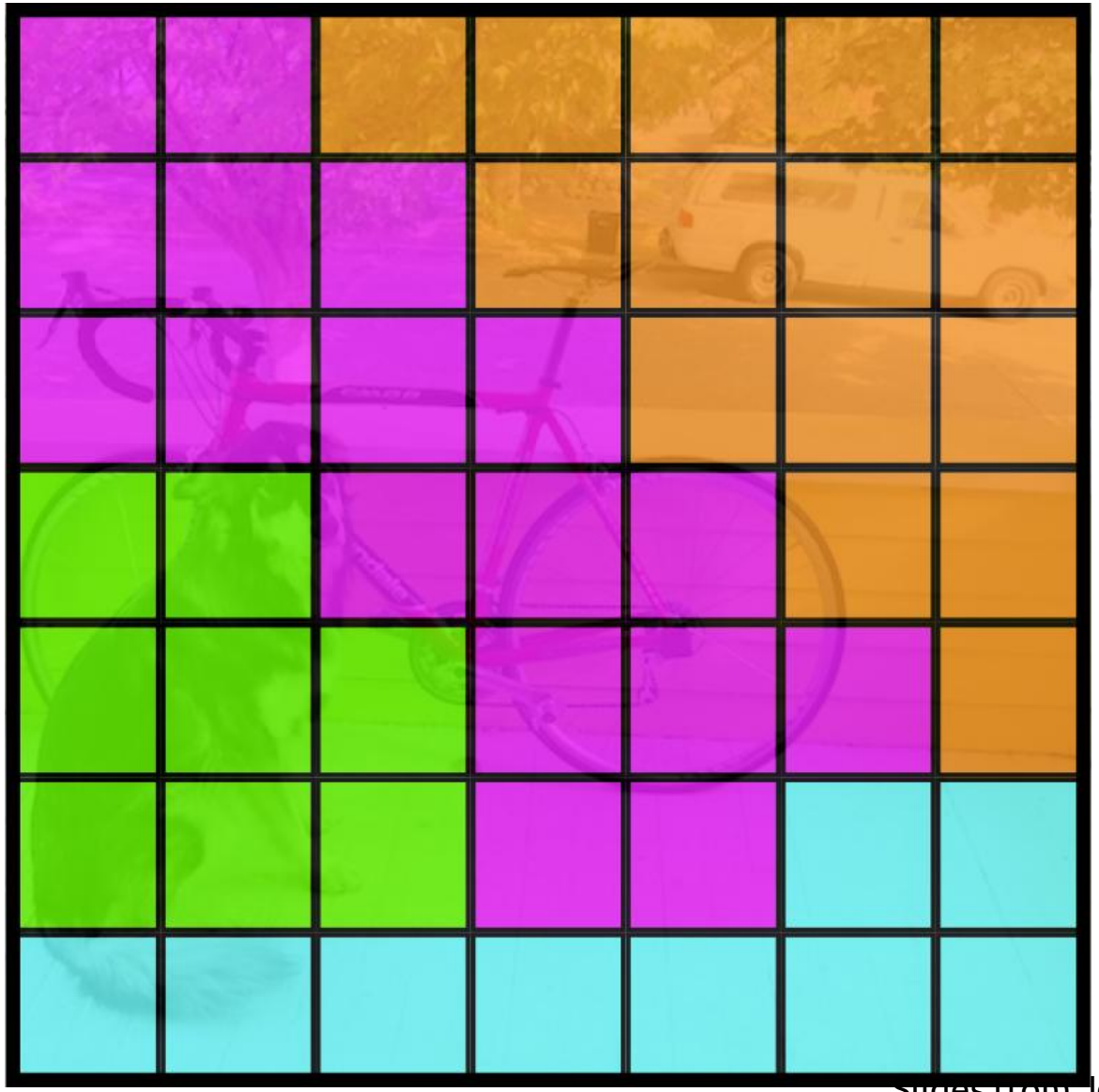
Slides from Joseph Redmon's CVPR2016 presentation

Each cell also predicts a class probability. Which looks like a corse segmentation map



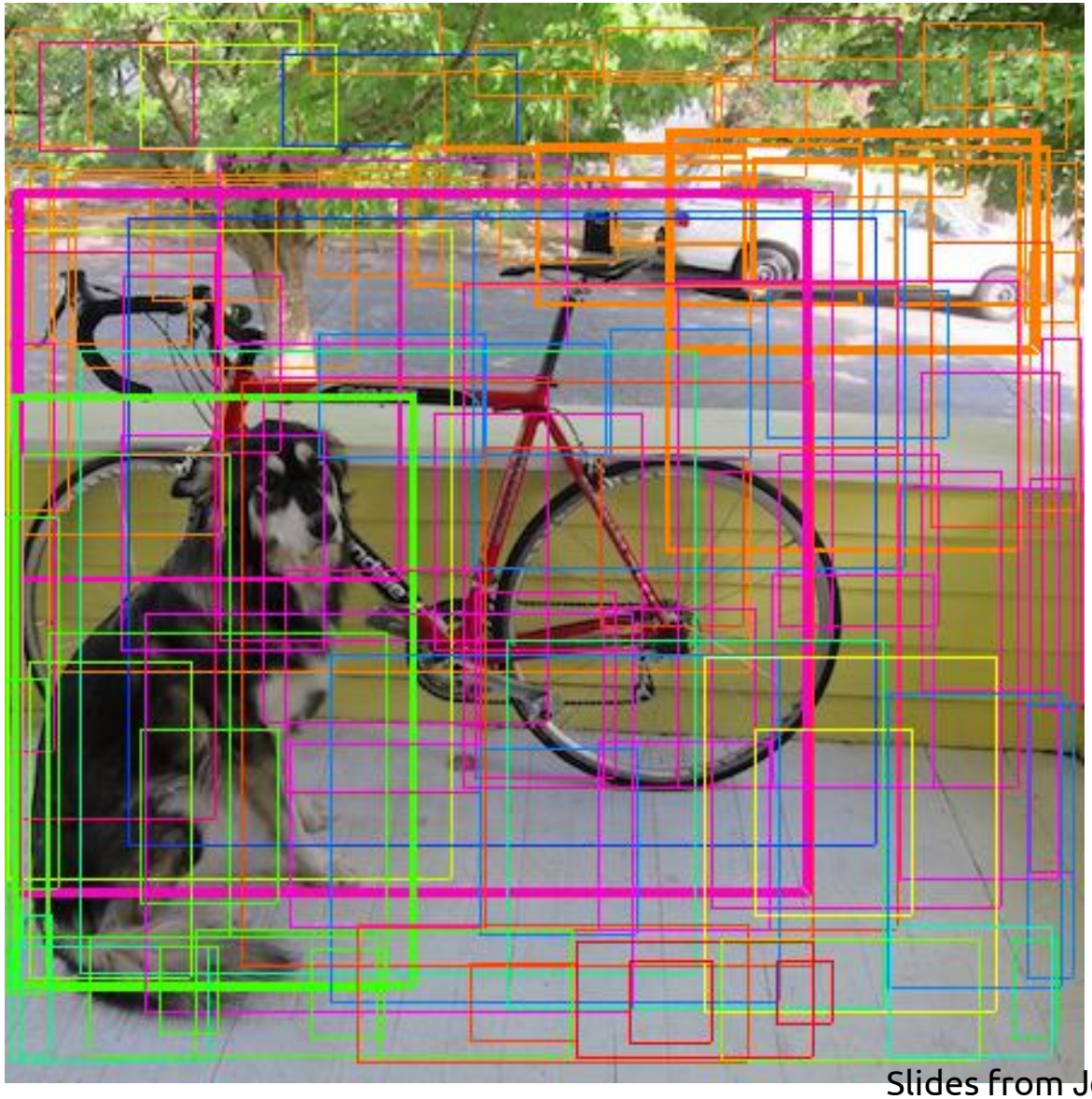
Slides from Joseph Redmon's CVPR2016 presentation

Class confidence conditioned on object: P(Car | Object)



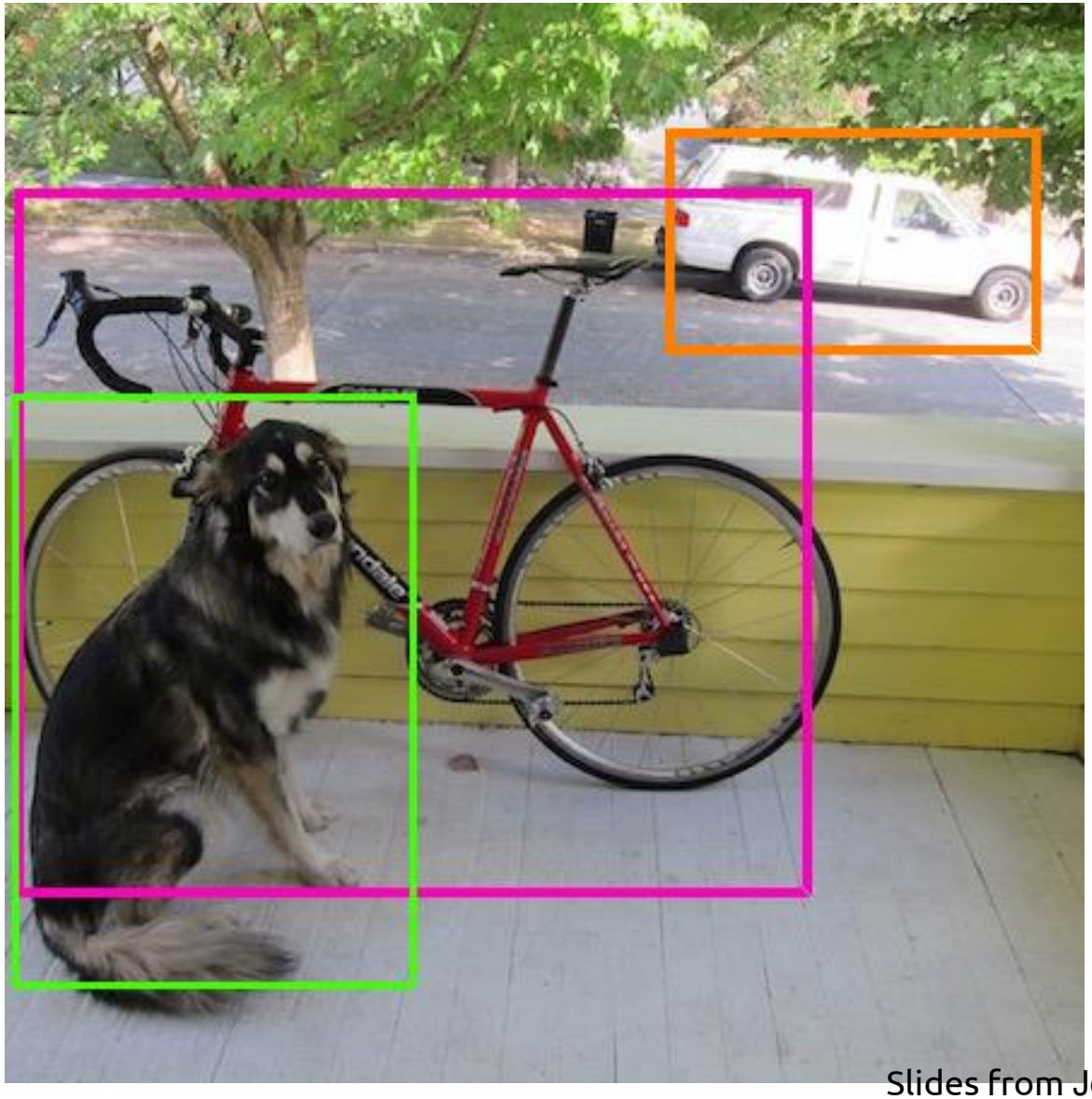
Slides from Joseph Redmon's CVPR2016 presentation

Then we combine the box confidence values and multiply them with the class confidences.



Slides from Joseph Redmon's CVPR2016 presentation

Finally, we do Non-max suppression and threshold detections

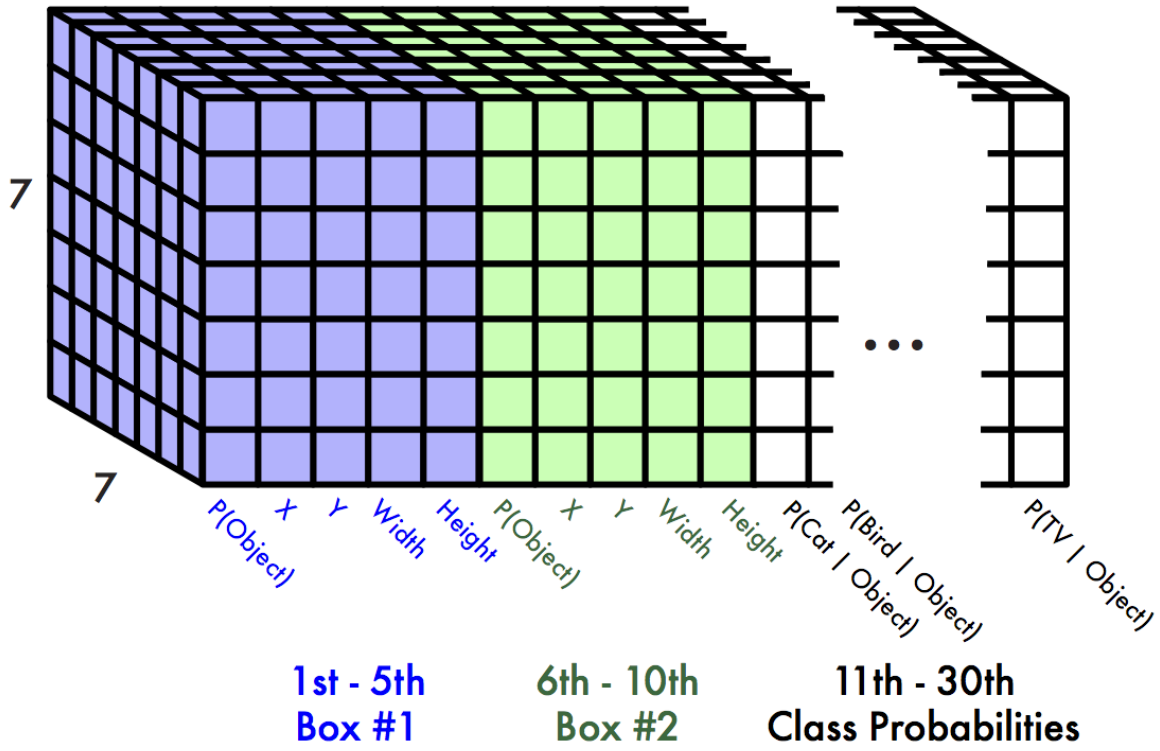


Slides from Joseph Redmon's CVPR2016 presentation

We are basically just trying to predict a tensor

Each cell predicts:

- For each bounding box:
  - 4 coordinates (x, y, w, h)
  - 1 confidence value
- Some number of class probabilities

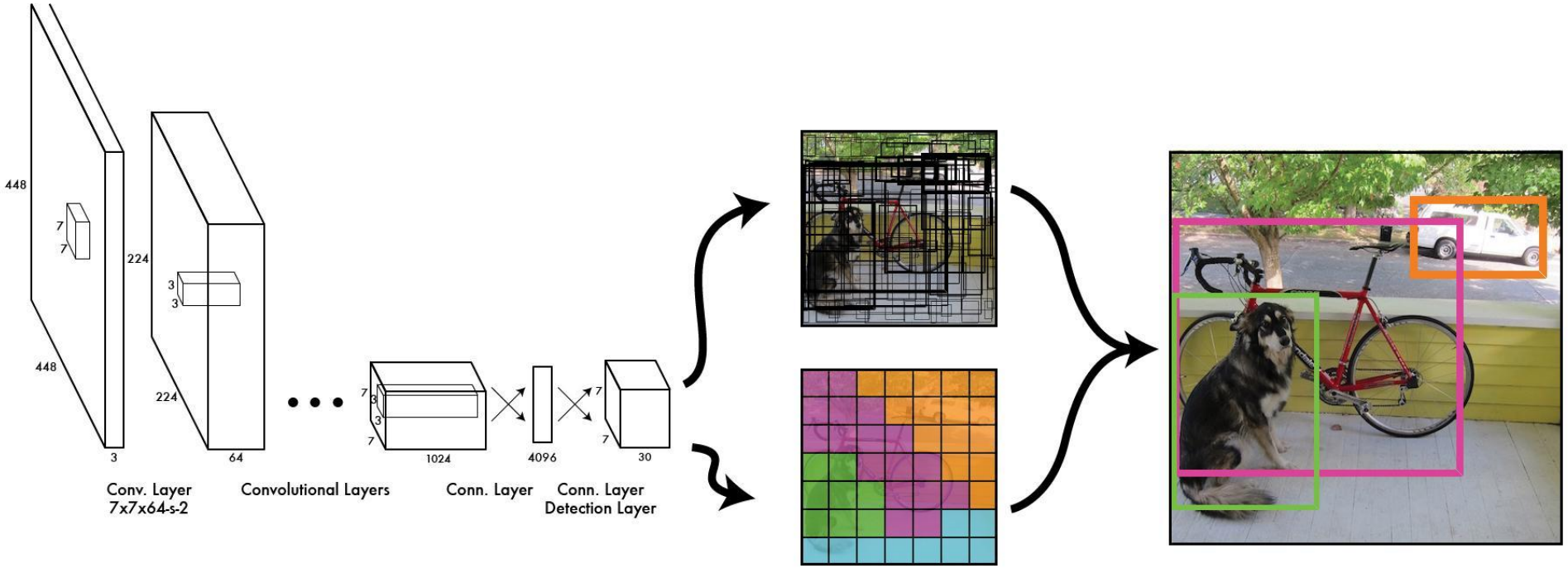


For Pascal VOC:

- 7x7 grid
- 2 bounding boxes / cell
- 20 classes

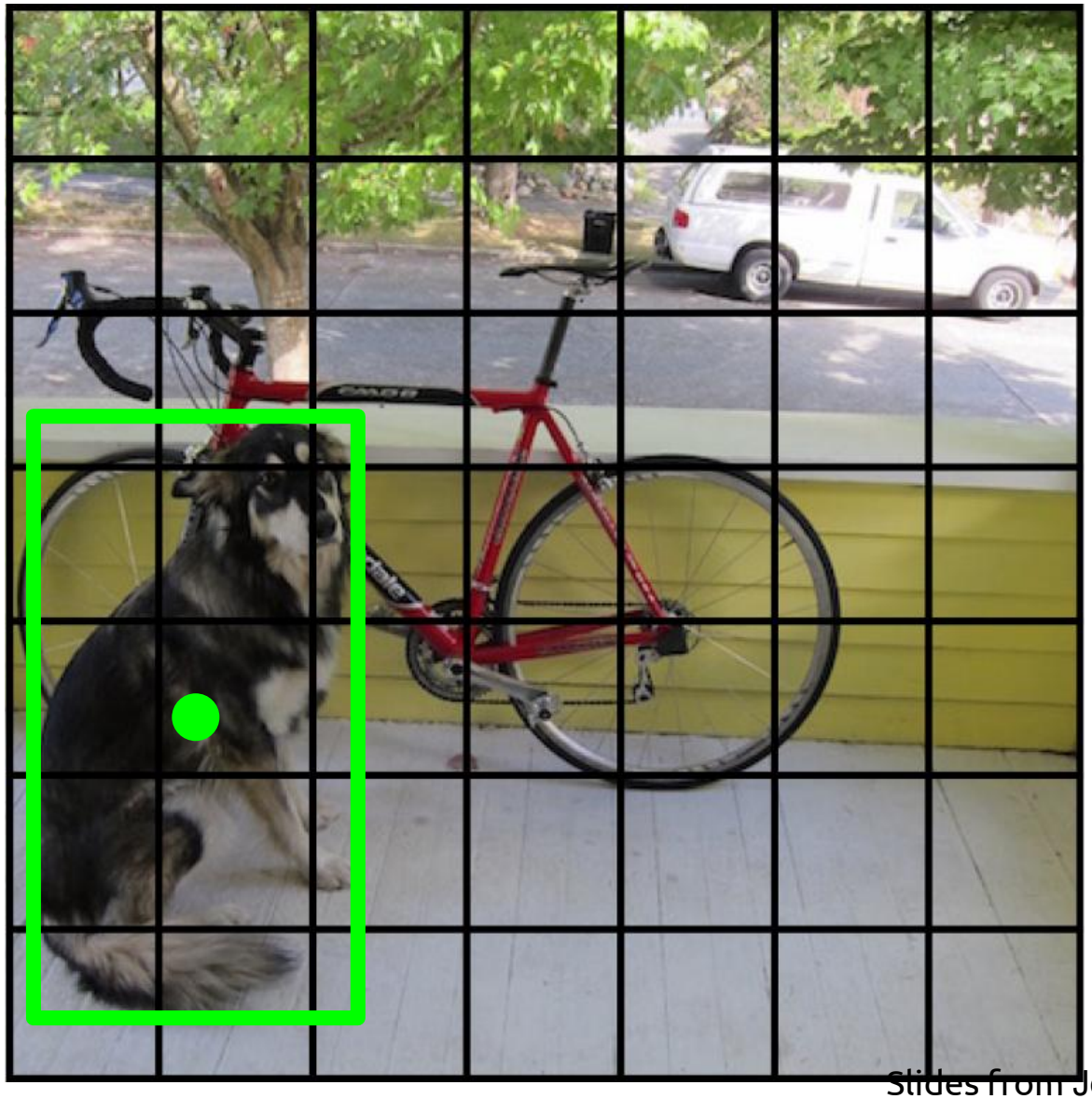
7 x 7 x (2 x 5 + 20) = 7 x 7 x 30 tensor = **1470 outputs**

Thus we can train one neural network to be a whole detection pipeline



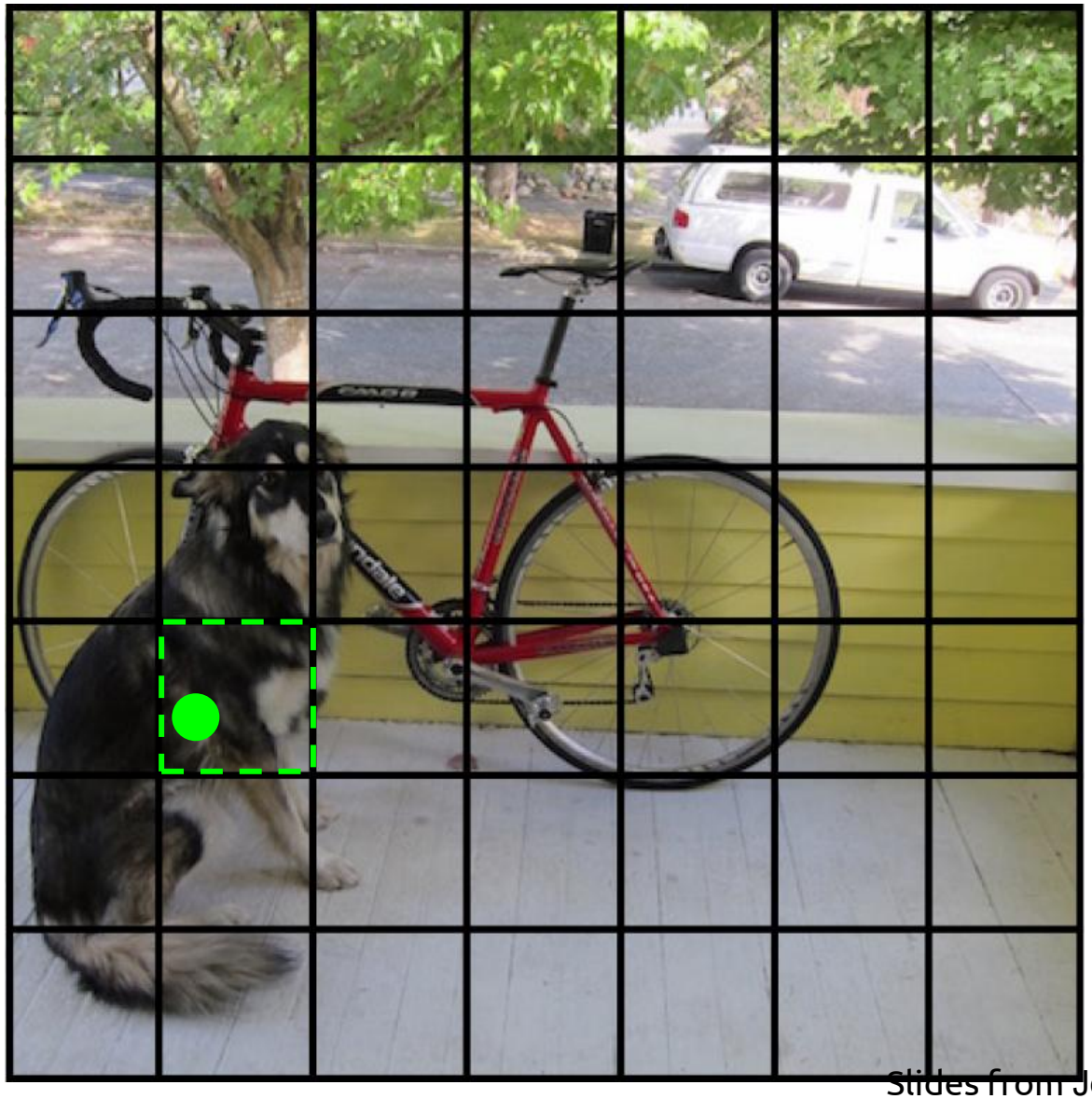
Slides from Joseph Redmon's CVPR2016 presentation

During training, match example to the right cell



Slides from Joseph Redmon's CVPR2016 presentation

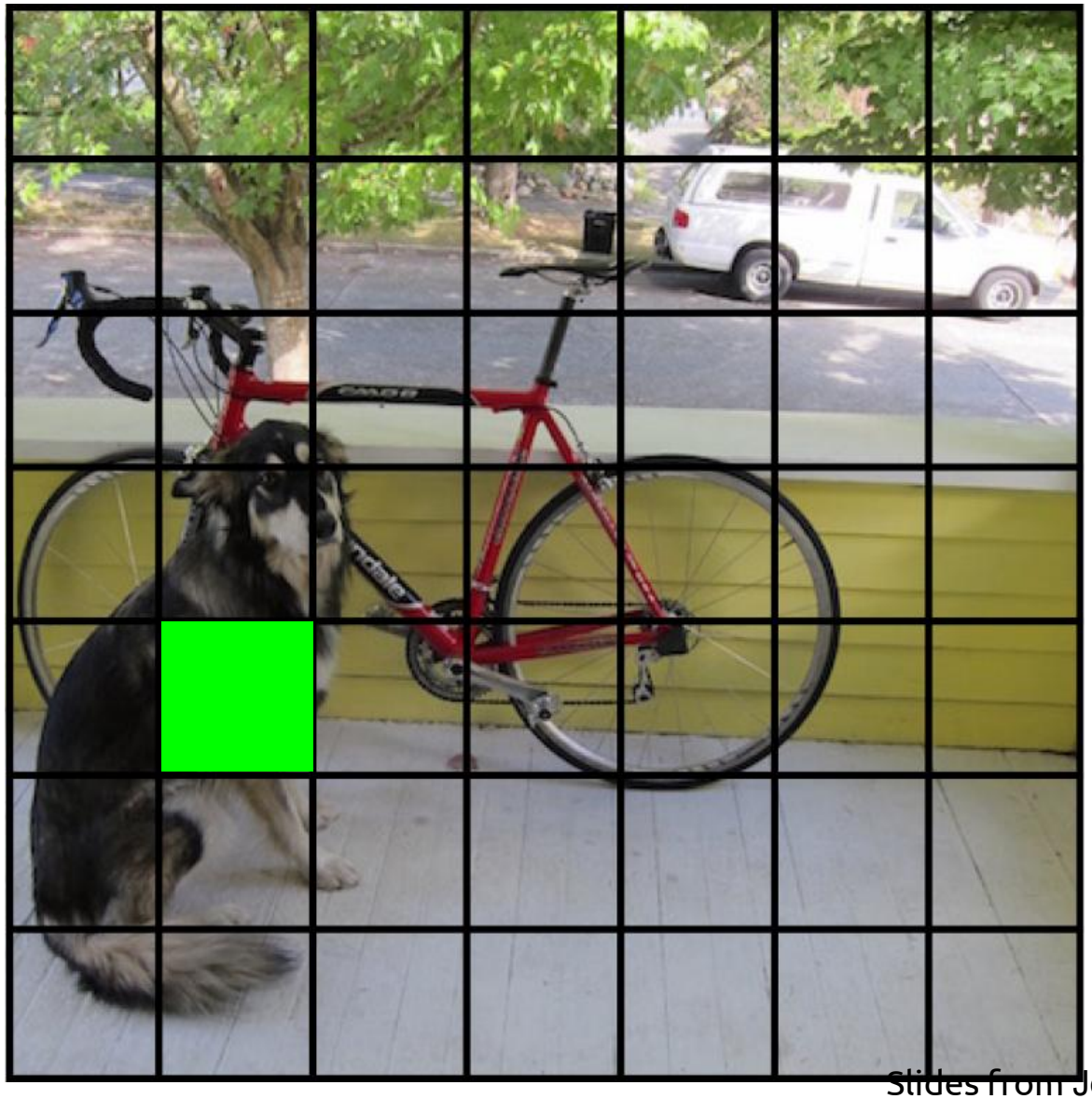
During training, match example to the right cell



Slides from Joseph Redmon's CVPR2016 presentation

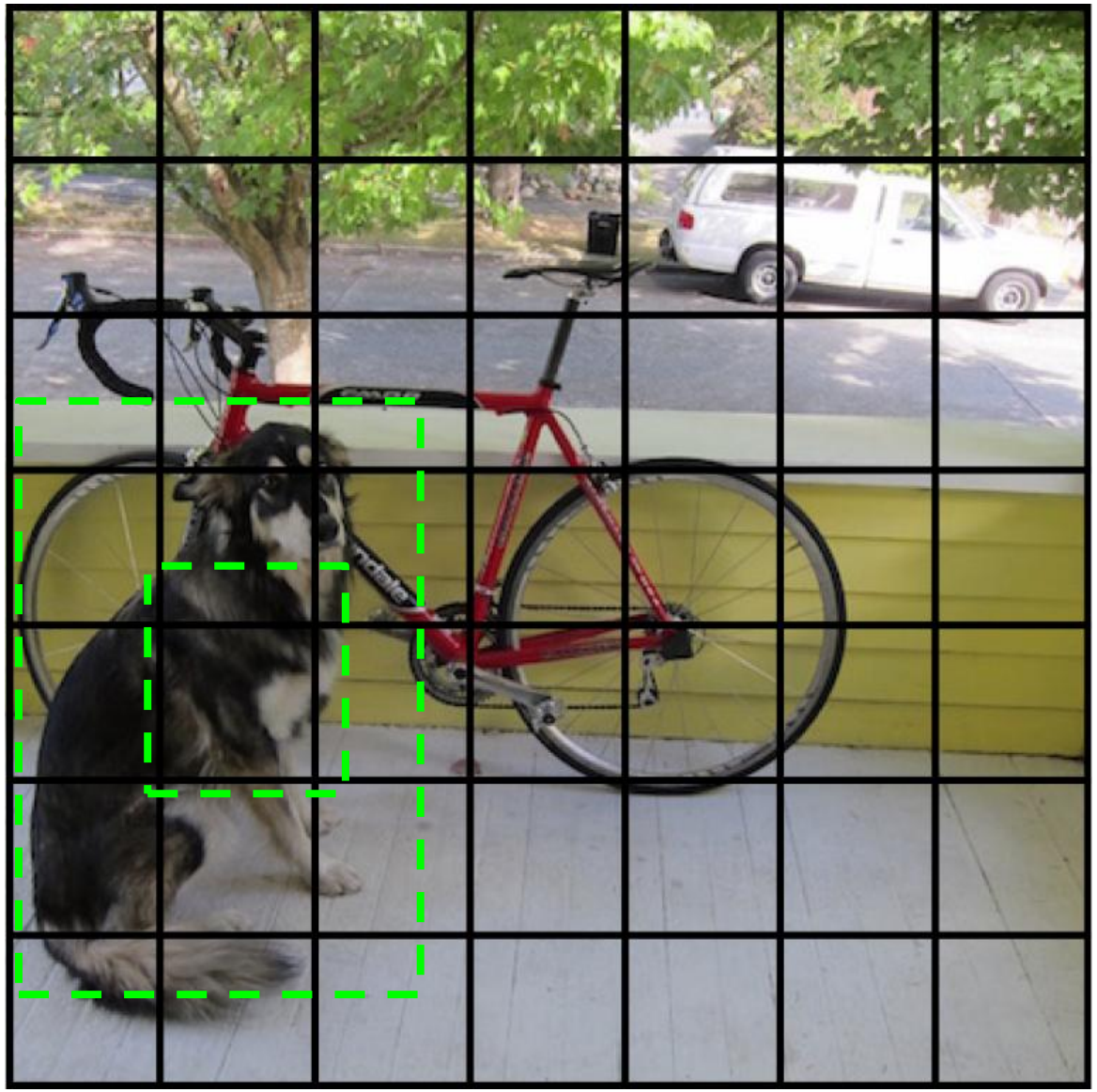
Adjust that cell’s class prediction

Dog = 1
Cat = 0
Bike = 0
...

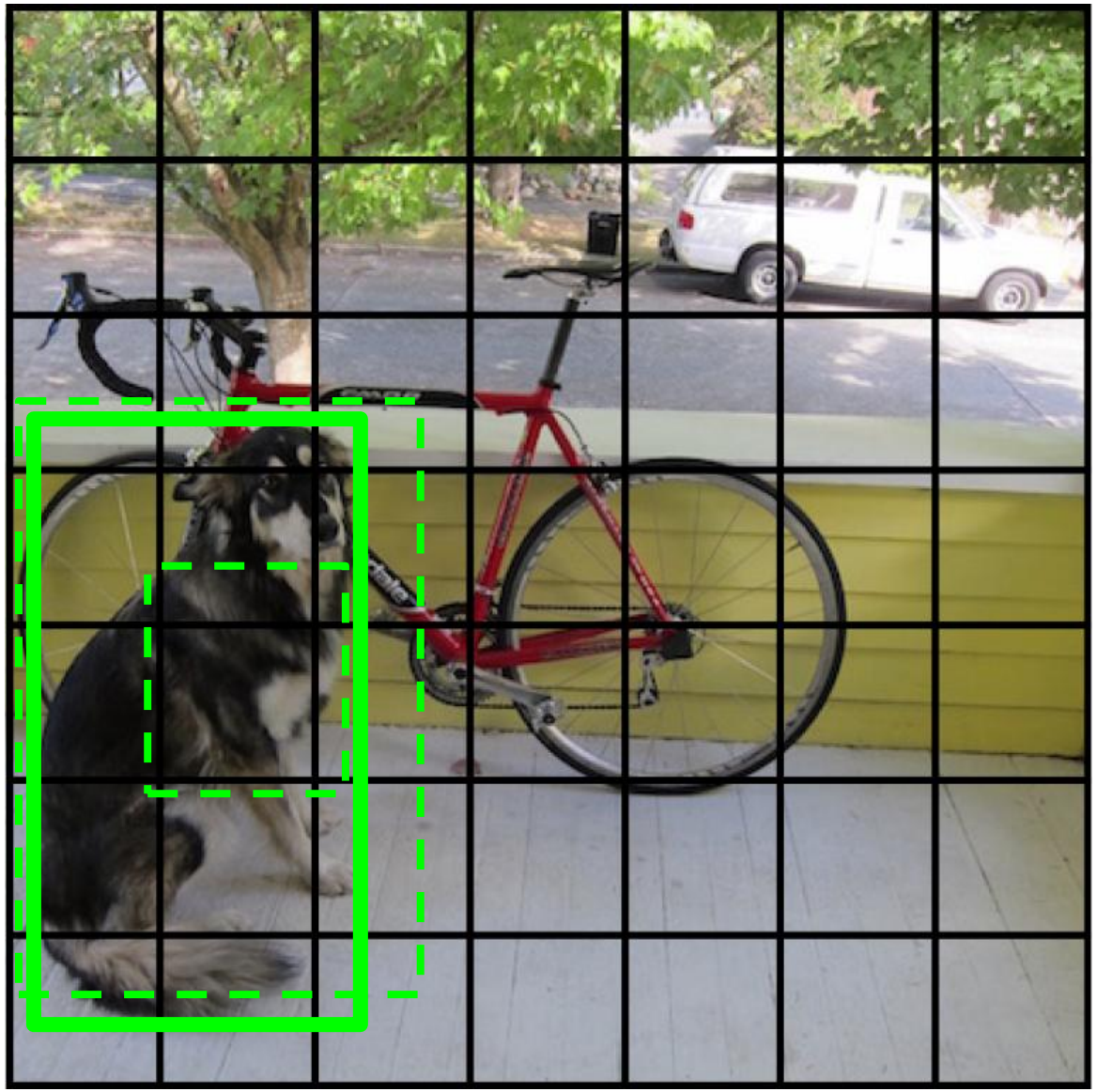


Slides from Joseph Redmon's CVPR2016 presentation

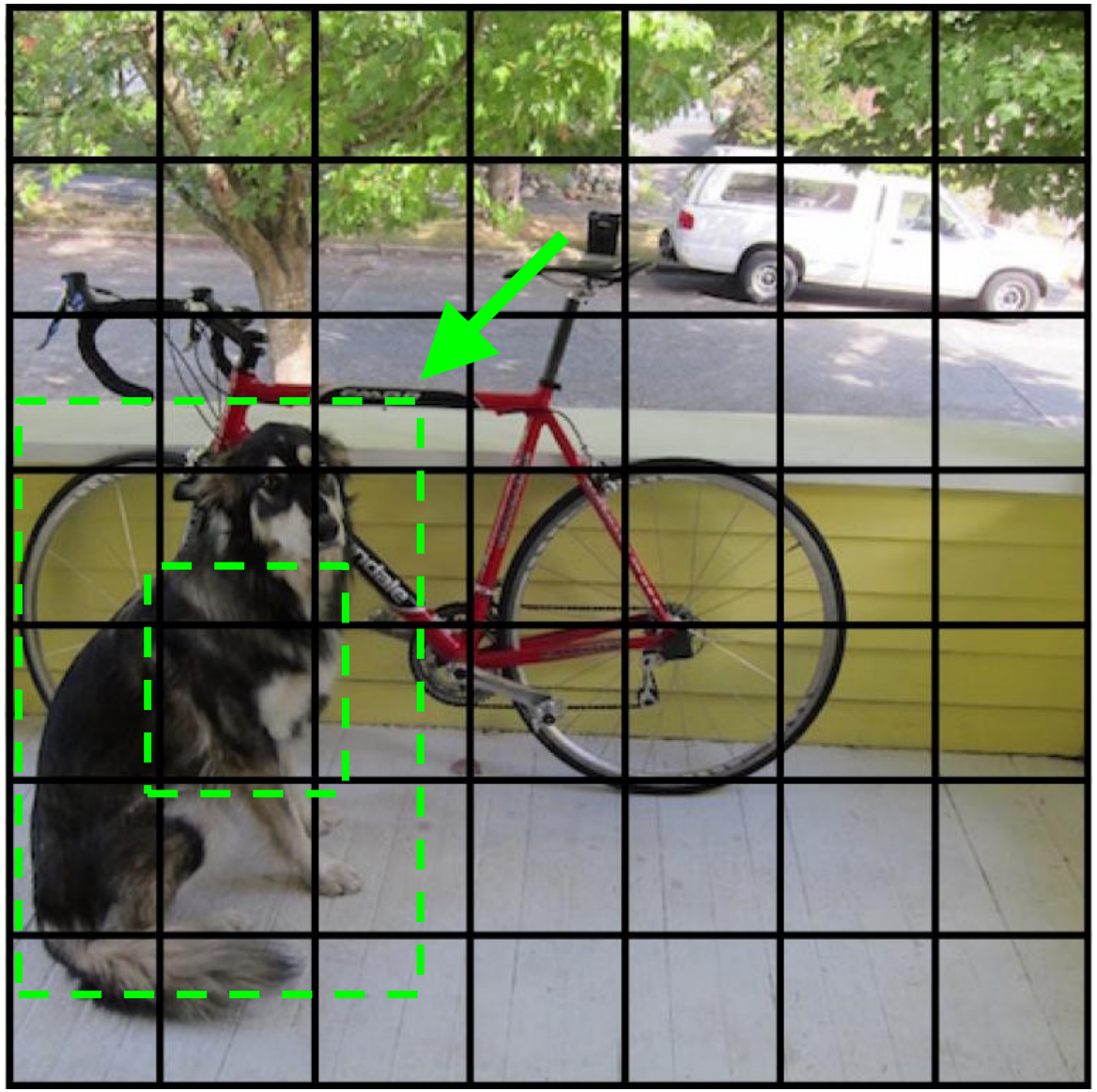
Look at that cell’s predicted boxes



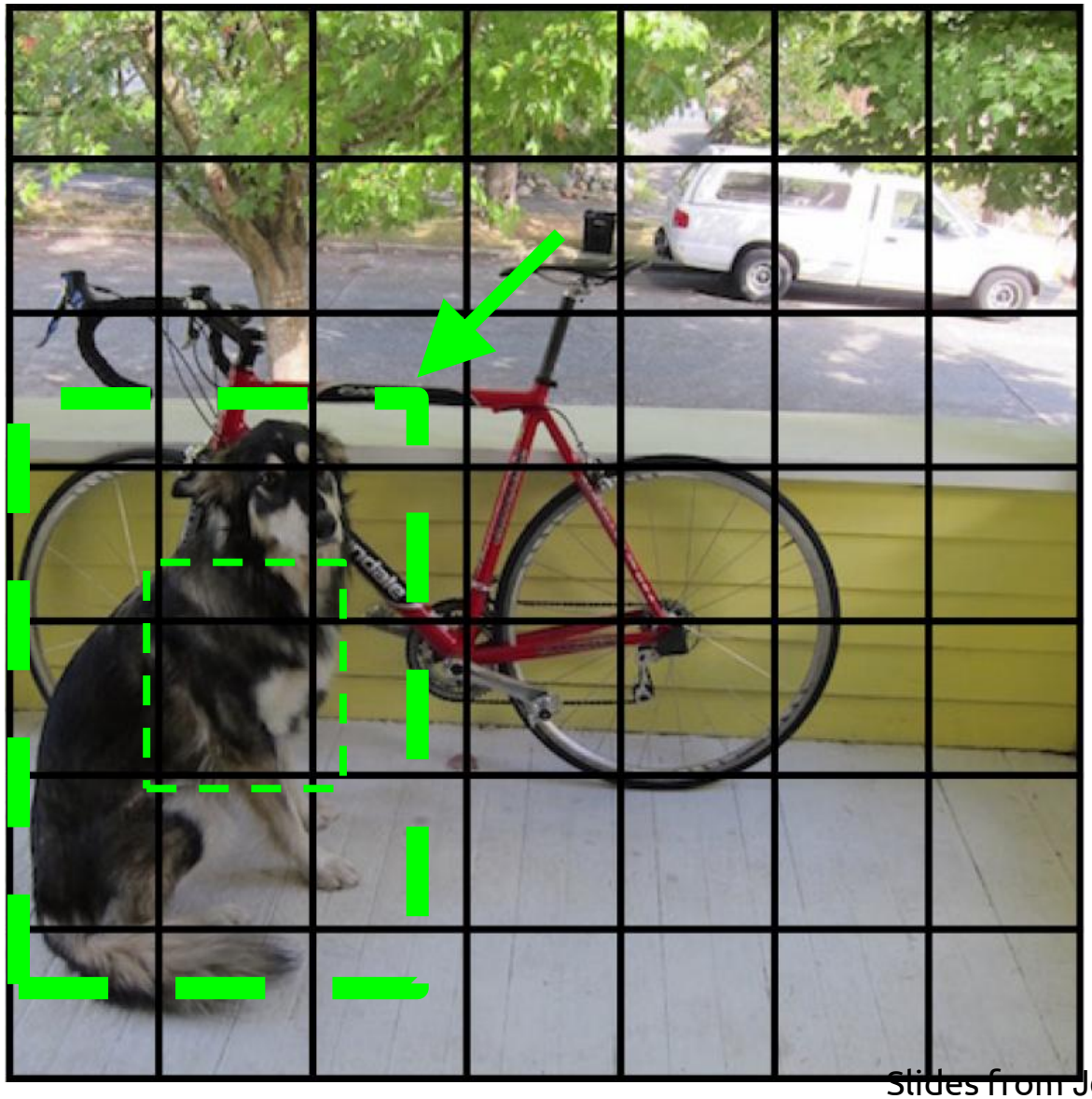
Find the best one using IoU, adjust it, increase the confidence



Find the best one using IoU, adjust it, increase the confidence

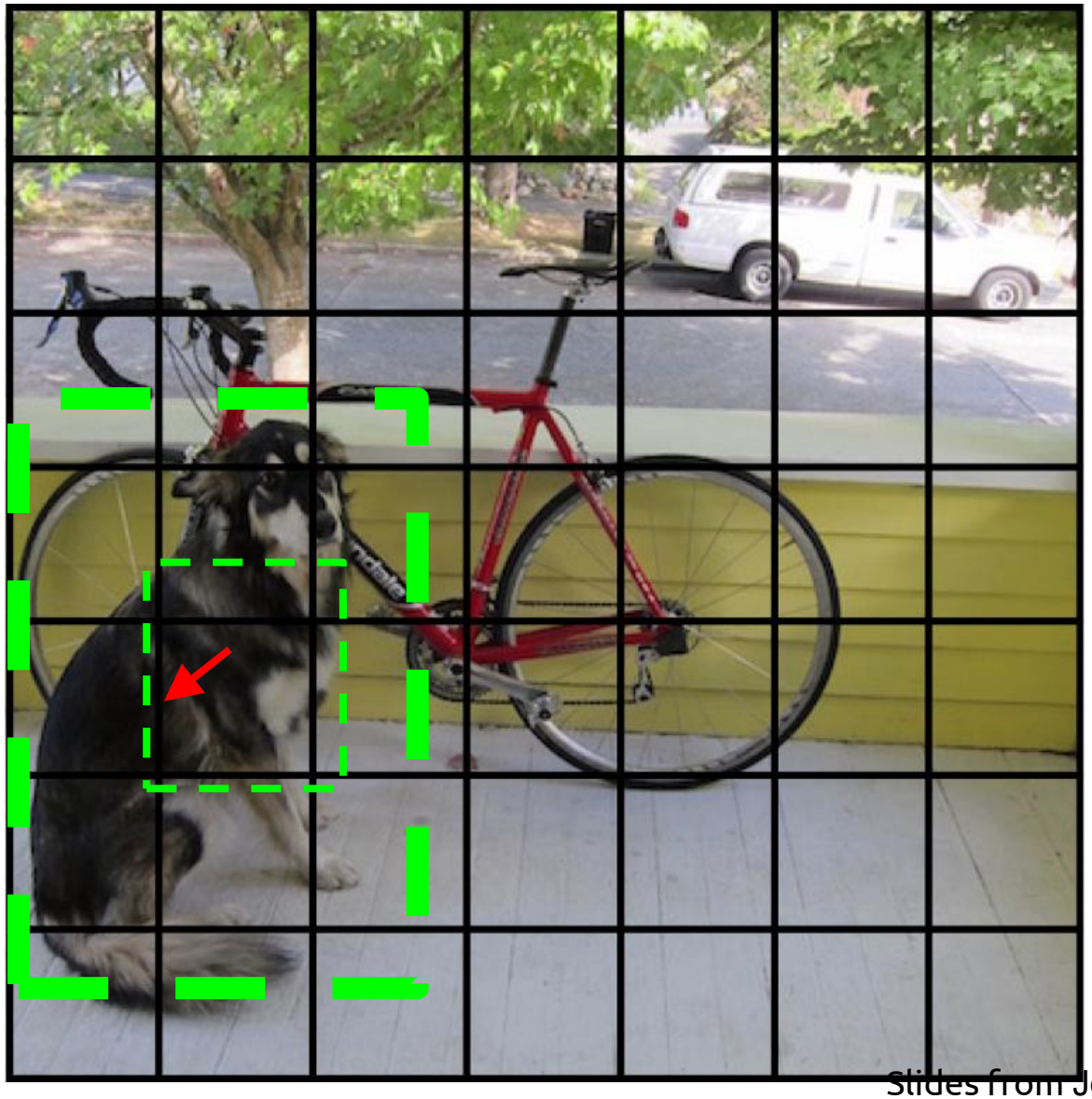


Find the best one using IoU, adjust it, increase the confidence



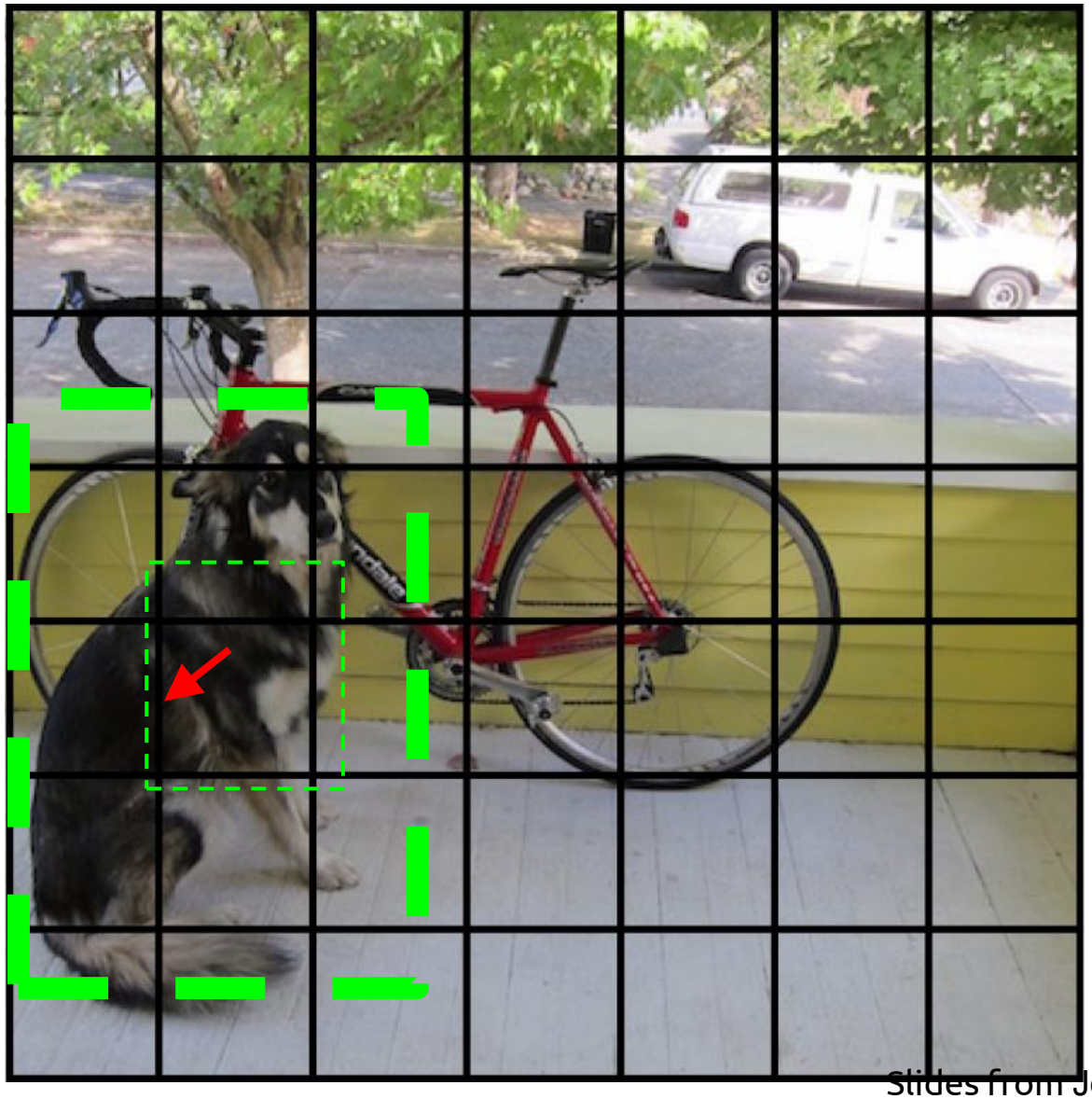
Slides from Joseph Redmon's CVPR2016 presentation

Decrease the confidence of other boxes



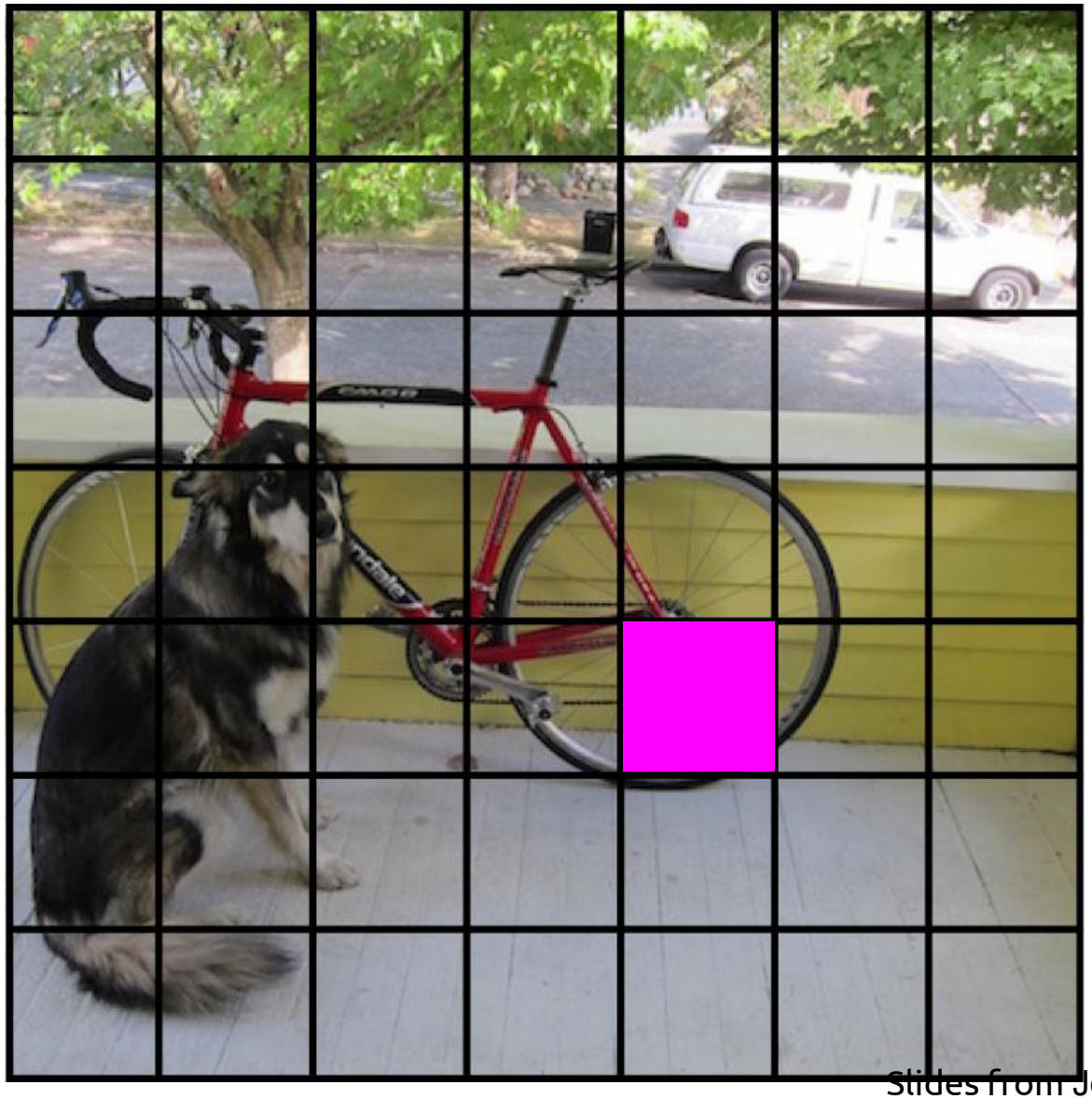
Slides from Joseph Redmon's CVPR2016 presentation

Decrease the confidence of other boxes



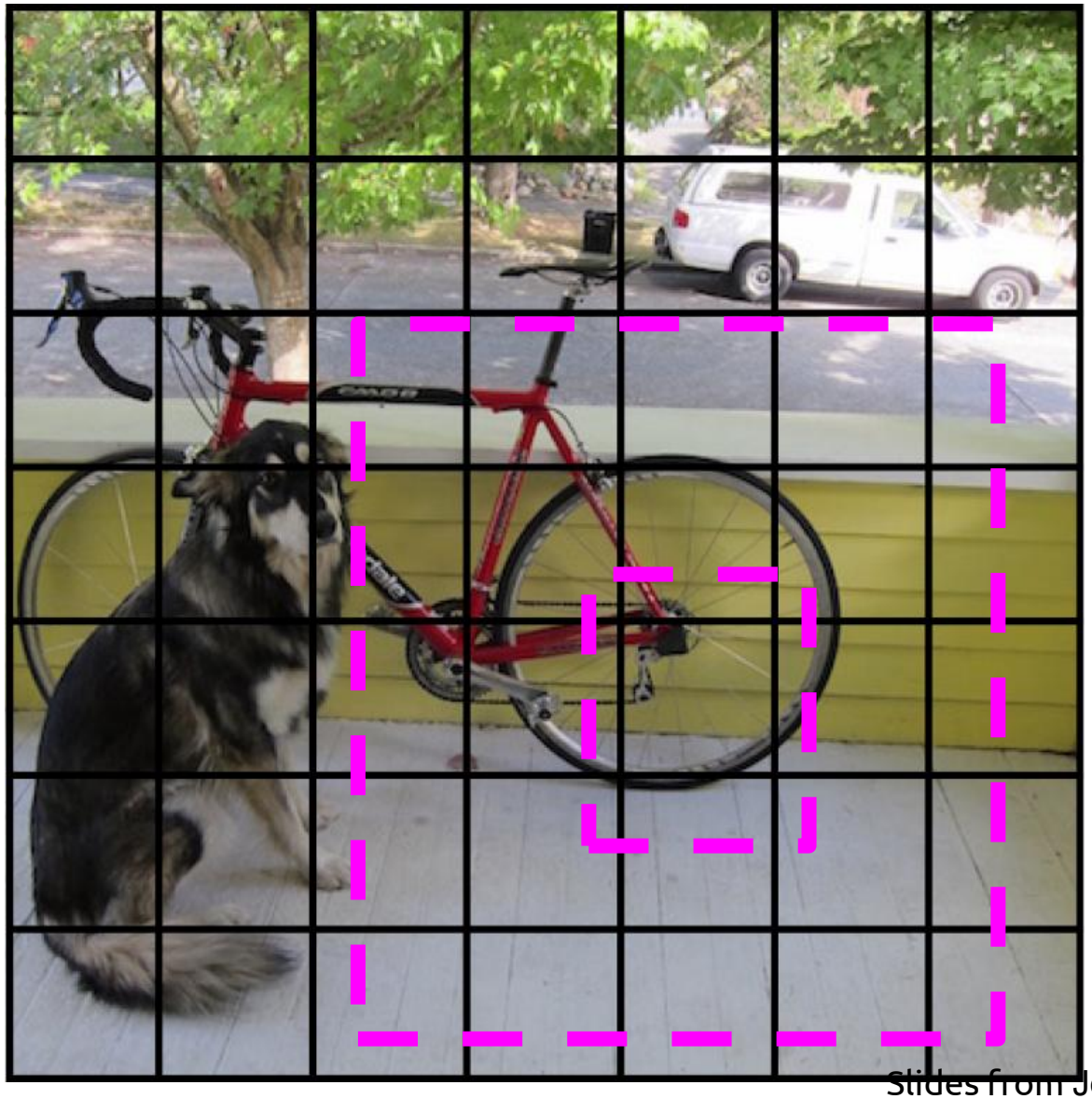
Slides from Joseph Redmon's CVPR2016 presentation

Some cells don’t have any ground truth detections!



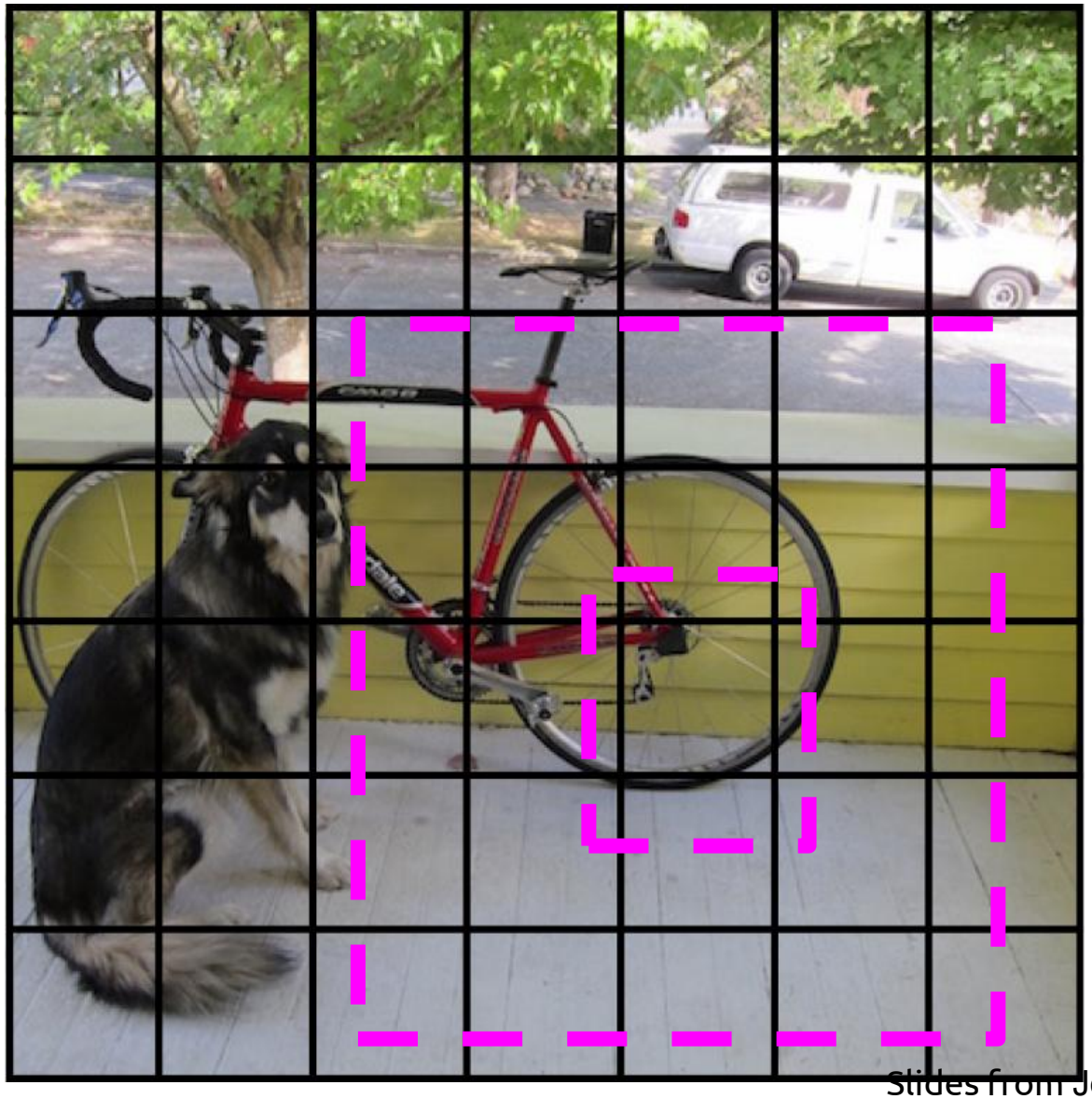
Slides from Joseph Redmon's CVPR2016 presentation

Some cells don’t have any ground truth detections!



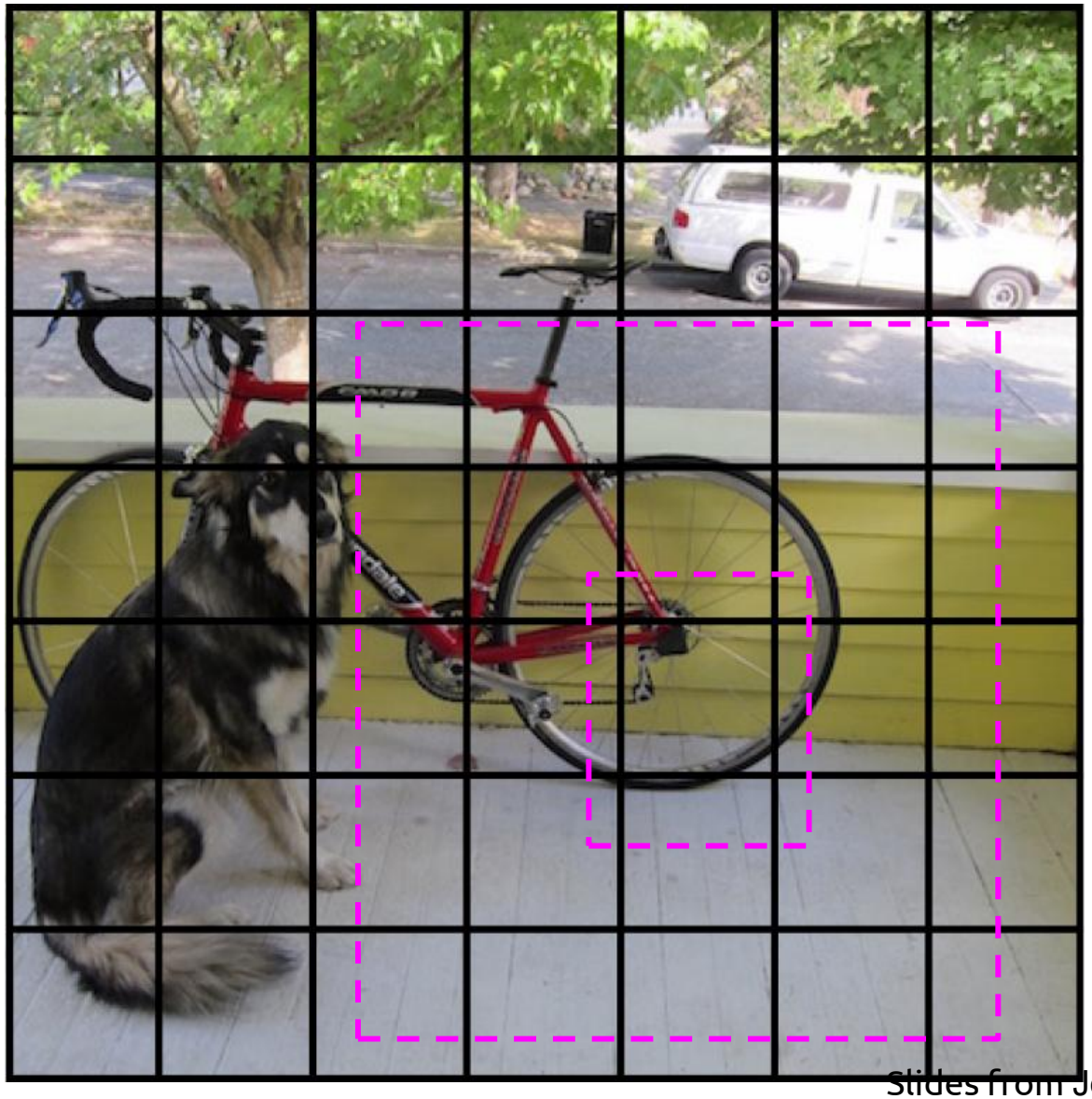
Slides from Joseph Redmon's CVPR2016 presentation

Decrease the confidence of these boxes



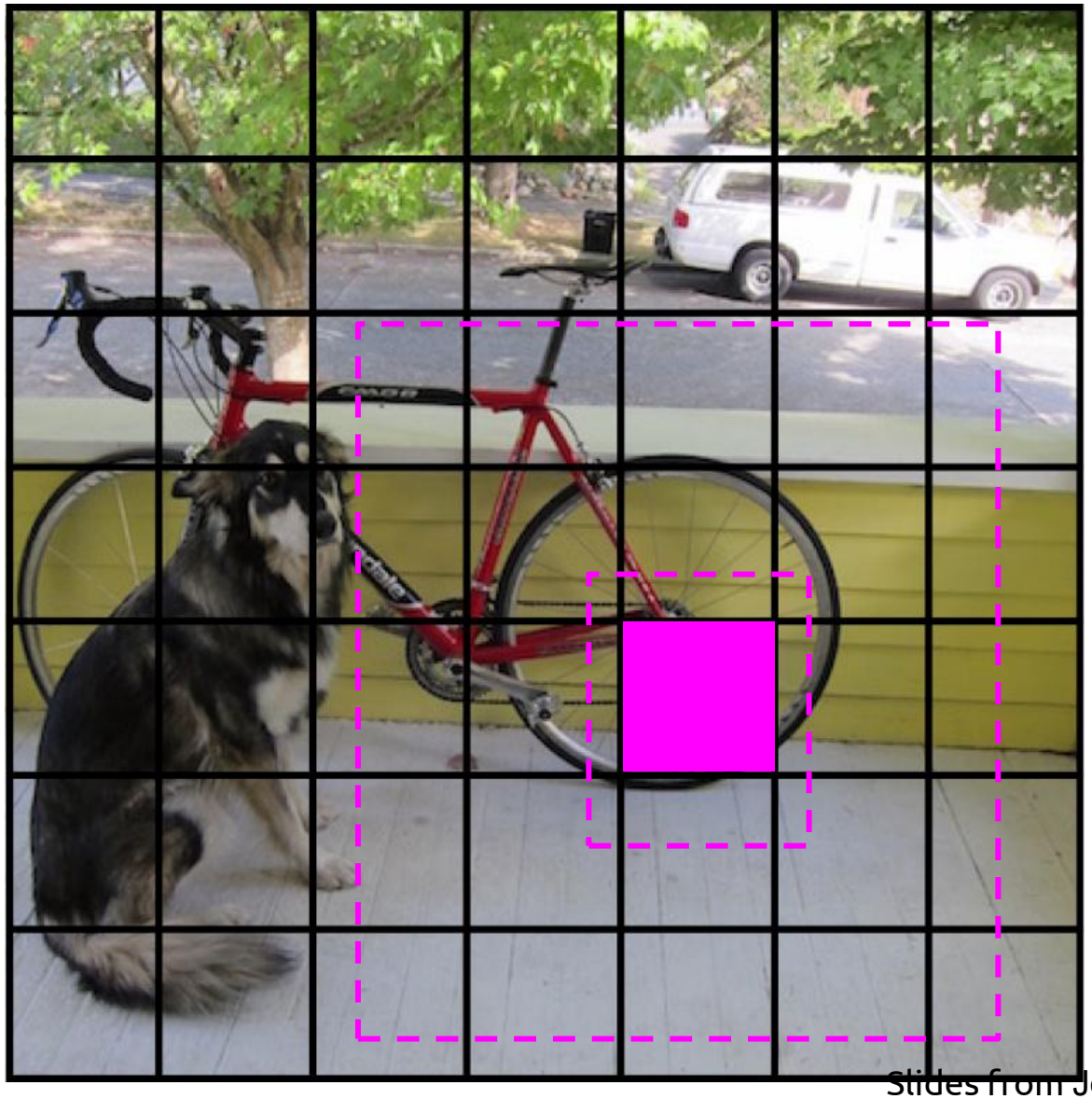
Slides from Joseph Redmon's CVPR2016 presentation

Decrease the confidence of these boxes



Slides from Joseph Redmon's CVPR2016 presentation

Don’t adjust the class probabilities or coordinates, since there is no ground truth in this region



Slides from Joseph Redmon's CVPR2016 presentation

YOLO Loss overview

1 if object in cell, 0 otherwise

Box coordinate midpoint

Box width and height. Sqrt helps prioritize small boxes

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(x_i - \hat{x}_i\right)^2 + \left(y_i - \hat{y}_i\right)^2$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i\right)^2$$

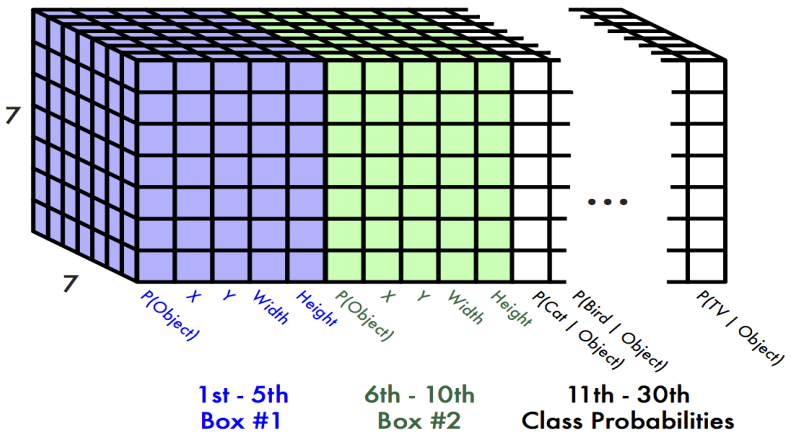
Confidence that there is a box in cell

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i\right)^2$$

Intersection over union of the predicted bounding box with the one from the label.

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c)\right)^2$$

confidence score of whether there is an object or not



Optimize the classification loss
Note: They use regression, where you would normally use cross entropy

Accurate object detection is slow!

	Pascal 2007 mAP	**Speed**	
R-CNN	66.0	.05 FPS	20 s/img
Fast R-CNN	70.0	.5 FPS	2 s/img
Faster R-CNN	73.2	7 FPS	140 ms/img
YOLO	69.0	45 FPS	22 ms/img

0,49m @ 80km/h

Accurate object detection is slow!

	Pascal 2007 mAP	**Speed**	
R-CNN	66.0	.05 FPS	20 s/img
Fast R-CNN	70.0	.5 FPS	2 s/img
Faster R-CNN	73.2	7 FPS	140 ms/img
YOLO	69.0	45 FPS	22 ms/img
YOLOv2	***76.8***	***67 FPS***	***15 ms/img***

0,33m @ 80km/h

Do I have to implement these architectures myself?

Classification

The following classification models are available, with or without pre-trained weights:

- AlexNet
- ConvNeXt
- DenseNet
- EfficientNet
- EfficientNetV2
- GoogLeNet
- Inception V3
- MaxVit
- MNASNet
- MobileNet V2
- MobileNet V3
- RegNet
- ResNet
- ResNeXt
- ShuffleNet V2
- SqueezeNet
- SwinTransformer
- VGG
- VisionTransformer
- Wide ResNet

Object Detection

The following object detection models are available, with or without pre-trained weights:

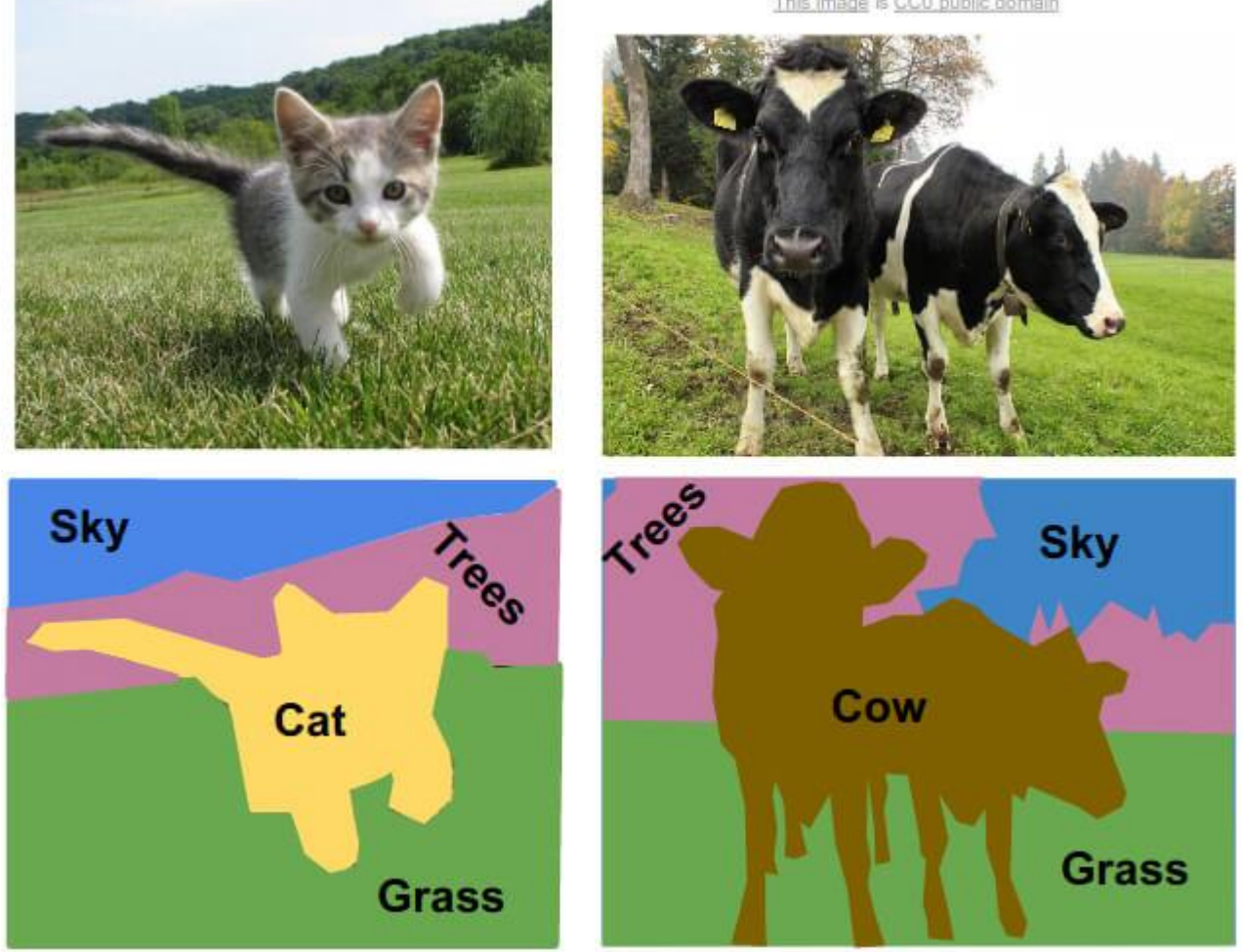
- Faster R-CNN
- FCOS
- RetinaNet
- SSD
- SSDlite

For Yolov5:
https://pytorch.org/hub/ultralytics_yolov5/

PyTorch

https://pytorch.org/vision/0.19/models.html

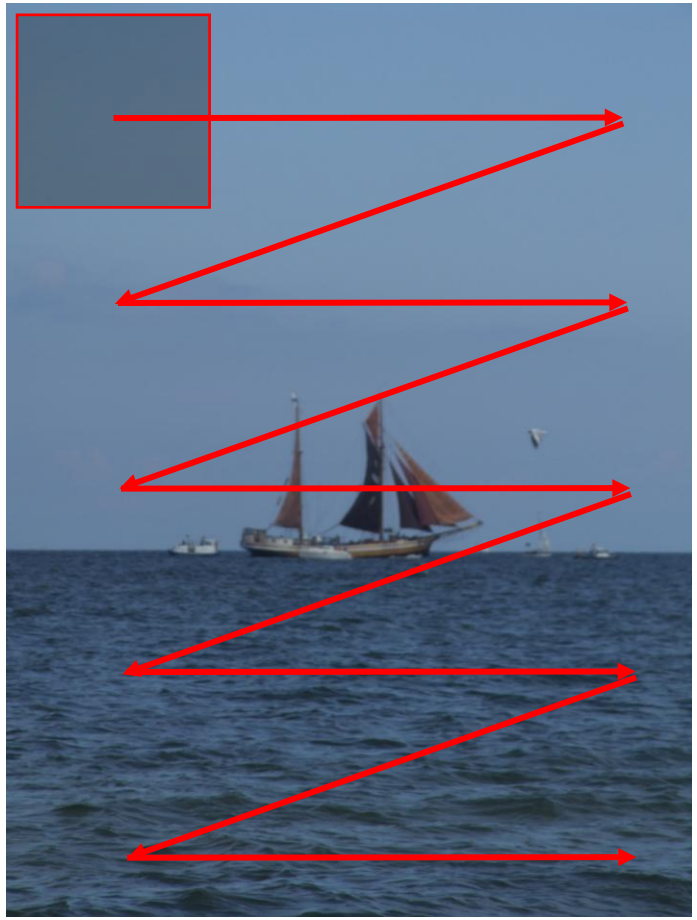
Semantic segmentation



Semantic segmentation

- Semantic segmentation focuses on how to divide an image into regions belonging to different semantic classes in pixel level

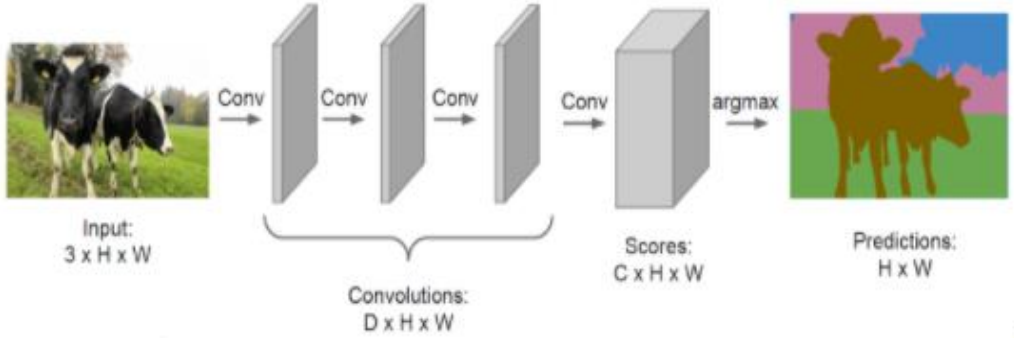
Naïve sliding window approach



Class of window determines class of pixel in center

Fully **C**onvolutional **N**etwork

A network that outputs a feature map.
Often the number of output channels is equal to the number of classes

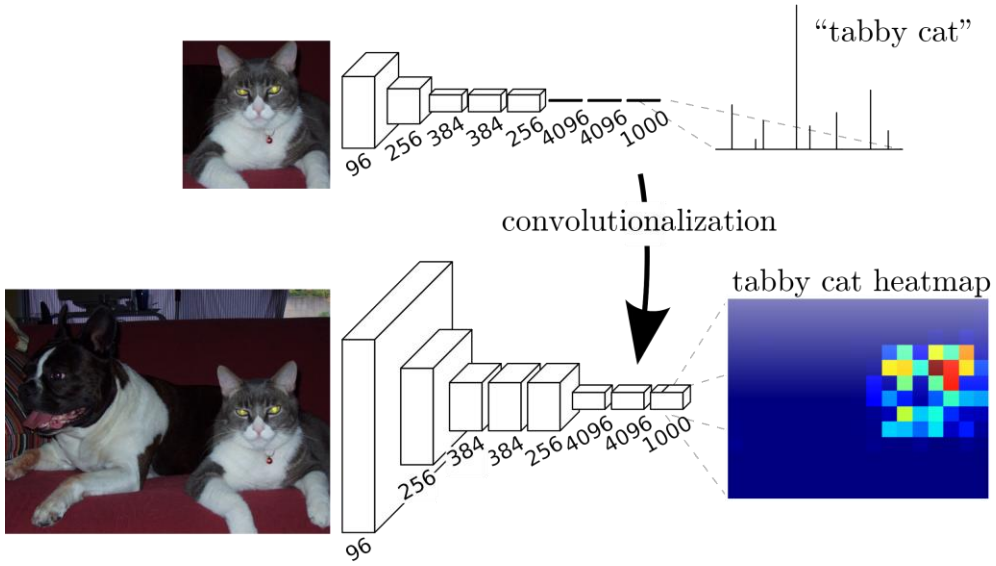


Sample

Architecture of a FCN (Source – Ref No 3)

Fully Convolutional Networks for Semantic Segmentation

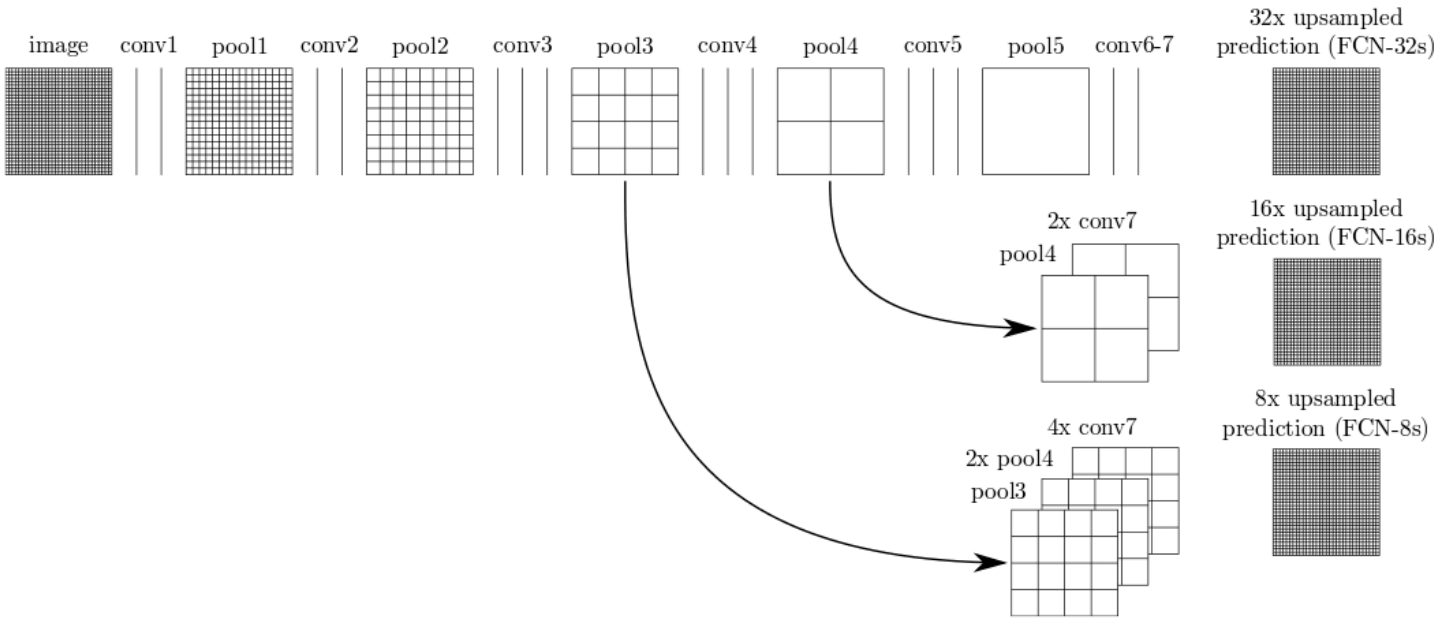
- A paper by Long, Shelhamer, and Darrell, 2014
- A modified version of the VGG16 architecture by Simonyan and Zisserman
- FC layers are changed to convolutions
- Output is a feature map that gives pixel-wise scores.



Shelhamer et al, 2015, Fully Convolutional Networks for Semantic Segmentation

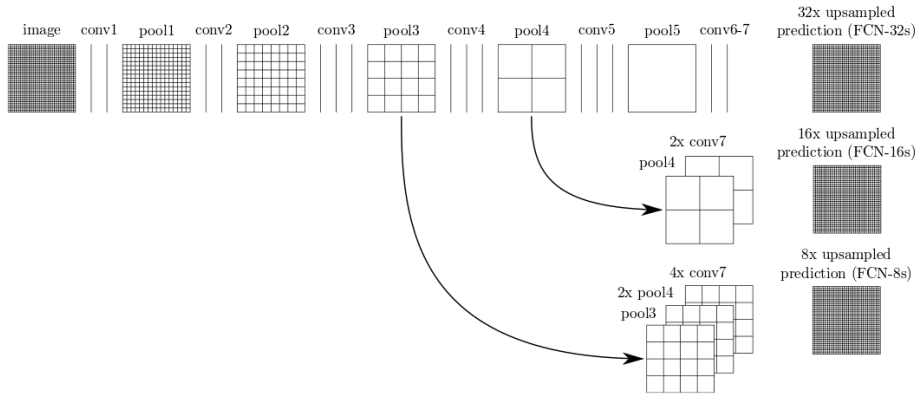
Fully Convolutional Networks for Semantic Segmentation

Three variants exists: FCN-32s, FCN-16s, and FCN-8s
The numbers refers to the needed upsampling factor.
FCN-32s only uses the last pooling layer
FCN-16s pulls features from the last two pooling layers
FCN-8s pulls features from the last three pooling layers

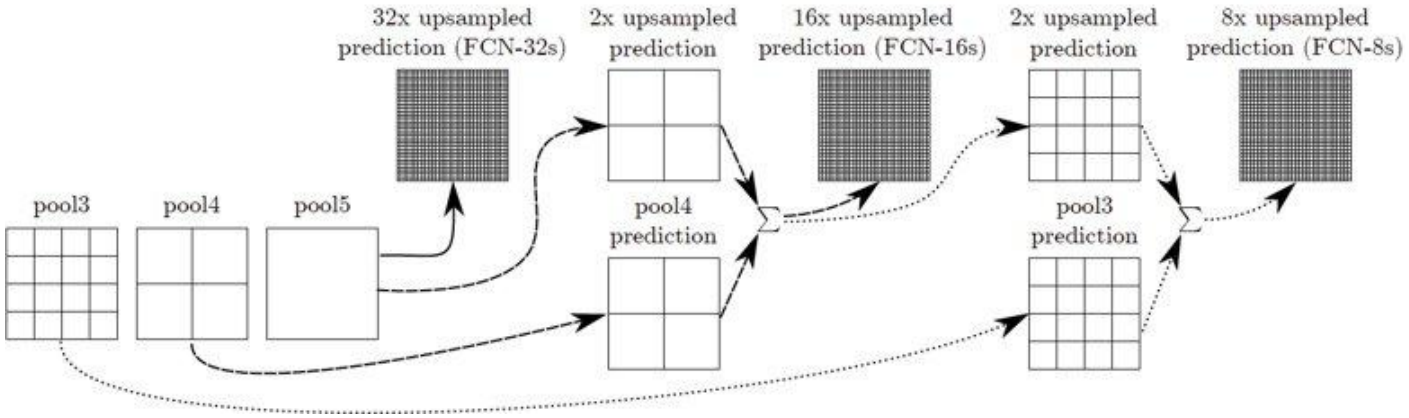


Shelhamer et al, 2015, Fully Convolutional Networks for Semantic Segmentation

Fully Convolutional Networks for Semantic Segmentation

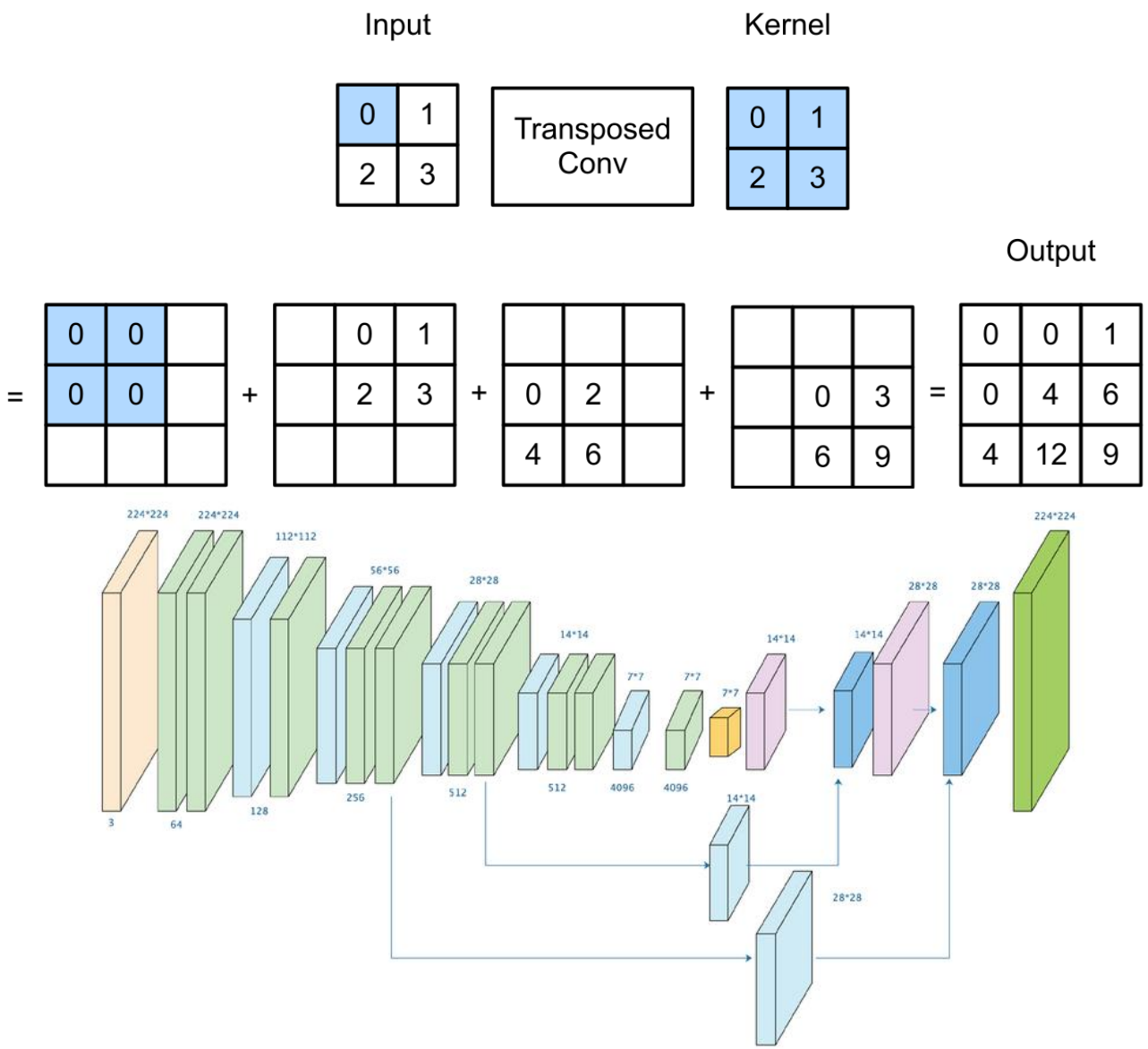


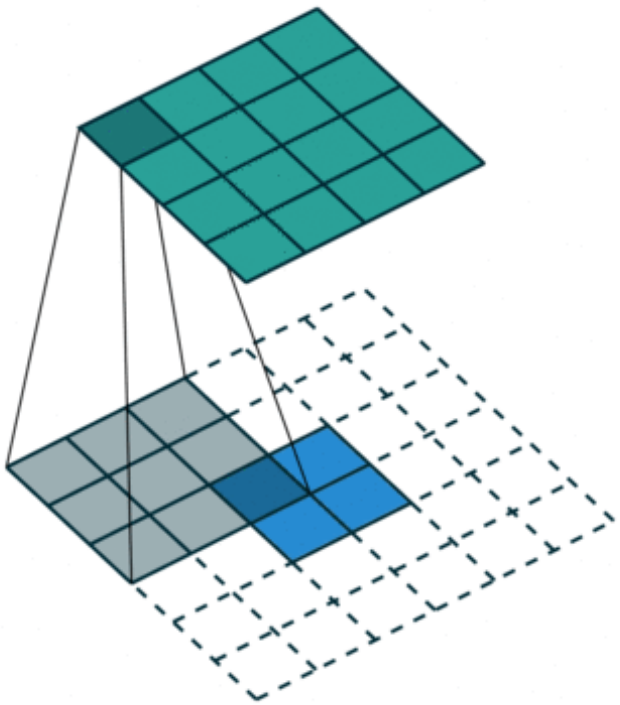
Pooling layers enables the network to look wider and learn semantics, but the price is that fine details are scarified.
Therefore, we need to upsample the output to match in size of the input



Often called deconvolution, but it is not the opposite of convolution!

Upsampling via Transposed Convolution ("deconvolution")

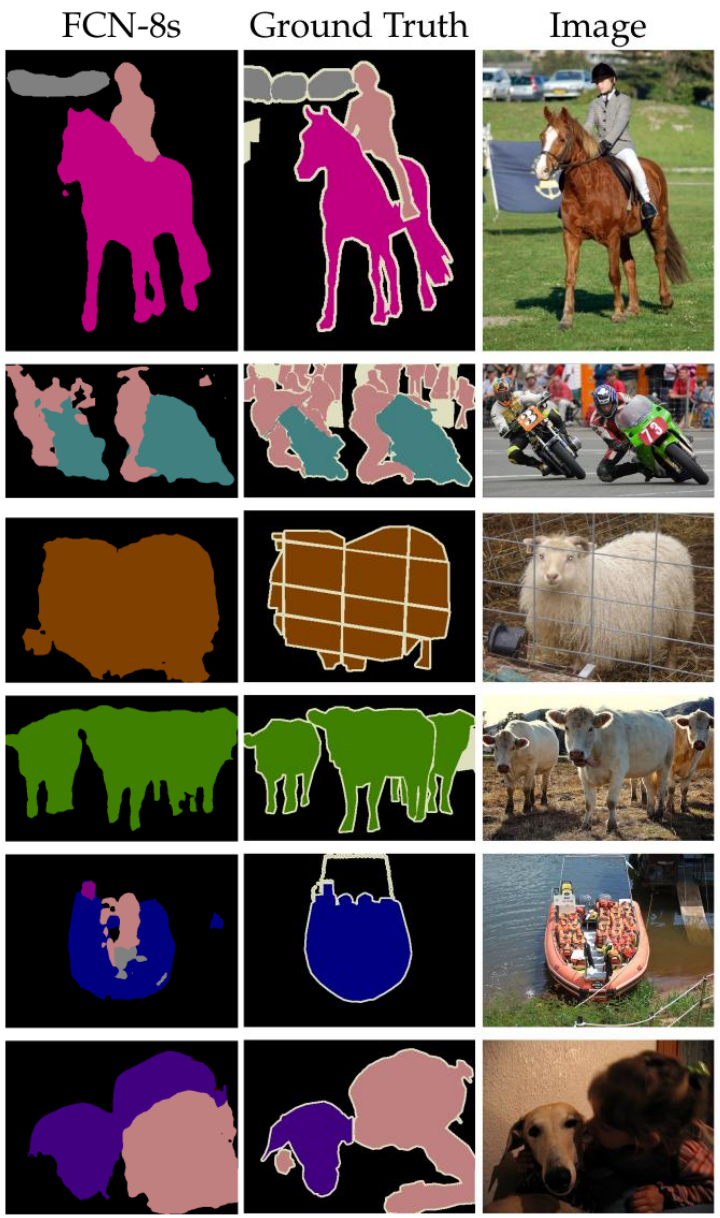




FCN results

Trained and evaluated on 59 classes from Pascal context

59 classes	Pixel acc.	Mean acc.	Mean IoU	f.w. IoU
FCN-32s	65.5	49.1	36.7	50.9
FCN-16s	66.9	51.3	38.4	52.3
FCN-8s	67.5	52.3	39.1	53.0

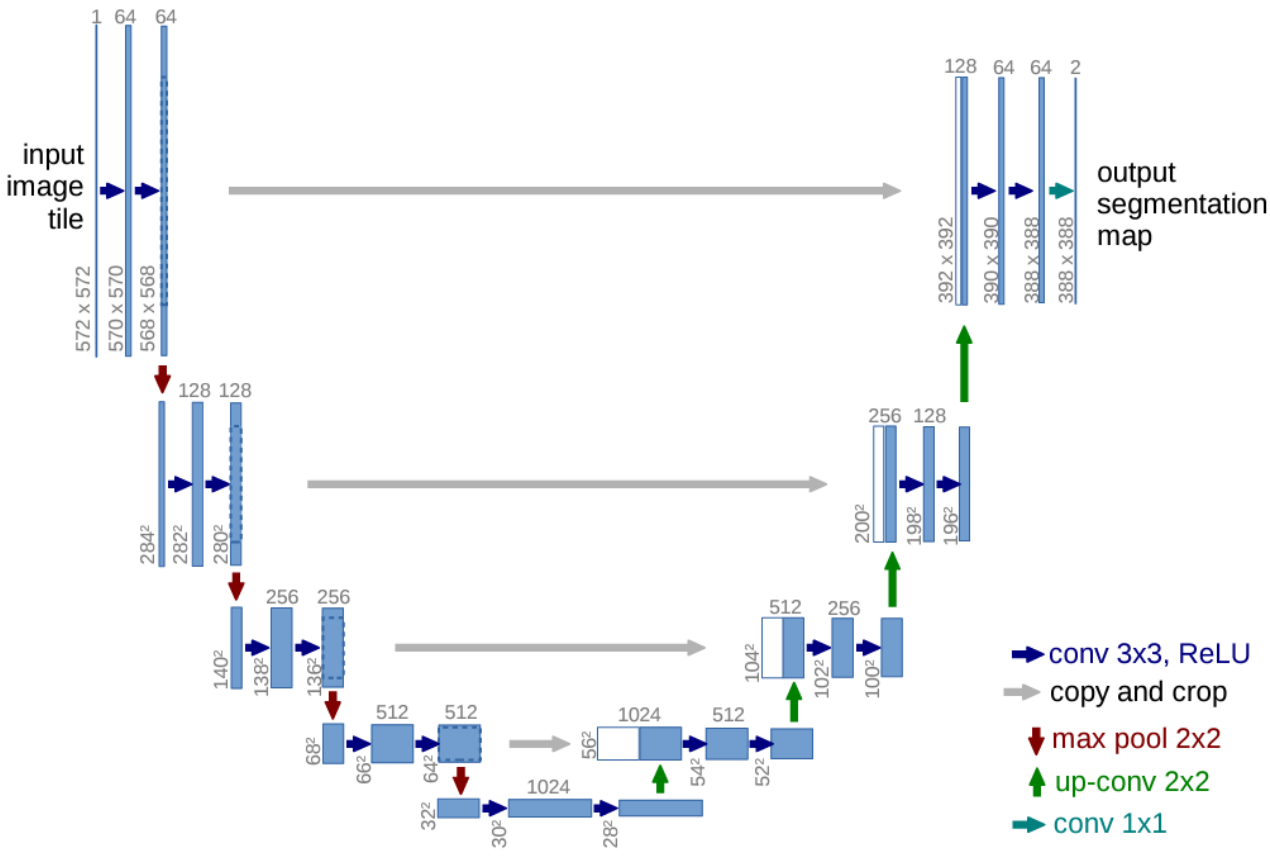


Shelhamer et al, 2015, Fully Convolutional Networks for Semantic Segmentation

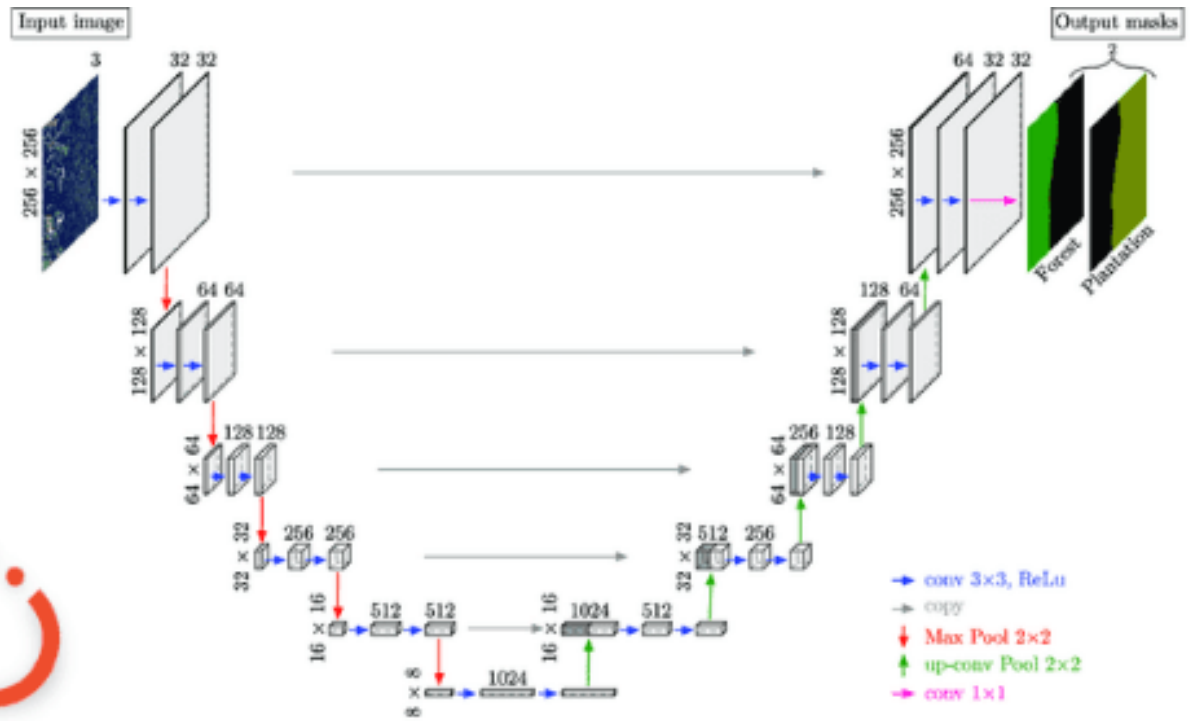
U-Net for semantic segmentation

- In the upsampling part, a large number of feature channels allow the network to propagate context information to higher resolution layers.

- The expansive path is more or less symmetric to the contracting path, and yields a u-shaped architecture.



https://arxiv.org/abs/1505.04597

Exercise

- Find a semantic segmentation dataset or use your own.
- You can find datasets on www.Kaggle.com
- If the image size are large, resize them to 256 x 256 and implement the shown U-Net architecture.
- Train the model.
- See the results
- If the image size is small, e.g., 224 x 224, you can make your own U-Net architecture.
- [Optional] Load any object detection model from pytorch and train it on an object detection task.