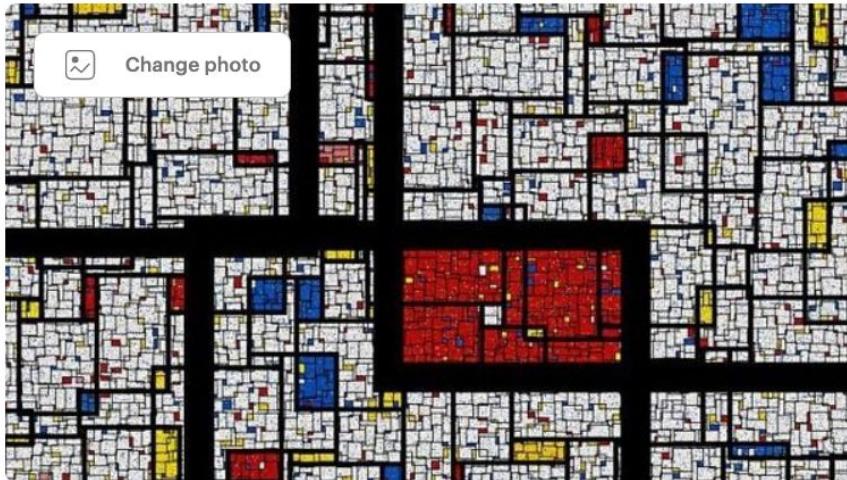


# Object Detection and RetinaNet

**Antonio Rueda-Toicen**  
Berlin Computer Vision Group  
**February 2022**

# About me

- Senior Data Scientist at Vinted
  - Background in academia (computer science and bioengineering)
  - Organizer of the [Berlin Computer Vision Group](#)



## Berlin Computer Vision Group

📍 Berlin, Germany

👤 384 members · Public group ?

👤 Organized by Antonio Rueda Toicen

Share: [Facebook](#) [Twitter](#) [LinkedIn](#)

[About](#)[Events](#)[Members](#)[Photos](#)[Discussions](#)[More](#)[Manage group](#) ▼[Create event](#) ▼

<https://www.meetup.com/Berlin-Computer-Vision-Group/>

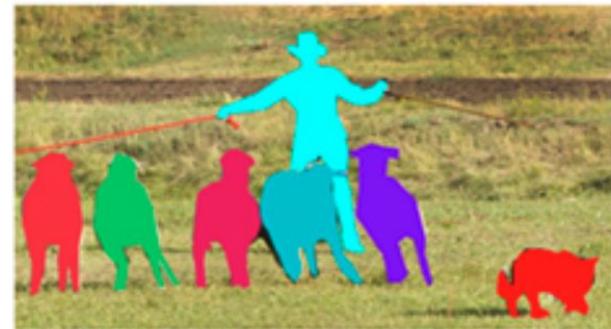
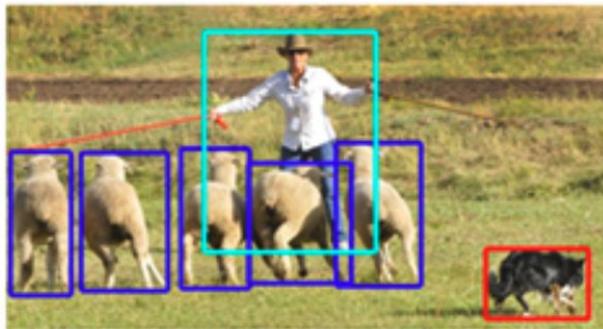
# About you

- Please let me know what is your expertise and what are you working on / what problems are you currently trying to solve (*bird's eye view*)

# Agenda

- An overview of the object detection problem
  - Two-stage vs single stage detectors
  - A review of quality metrics
    - i. Intersection over union (IoU)
    - ii. Mean average precision (MAP)
- Review of convolutional networks
  - Fully convolutional networks
  - Transfer learning
- Single-Shot Detectors (SSD)
  - SSD architecture
  - RetinaNet
    - i. Architecture
    - ii. Classification vs location loss
- Homework review

# Classification, detection, and segmentation



Classification refers to image-wide labels

Detection refers to localization of bounding boxes with labels

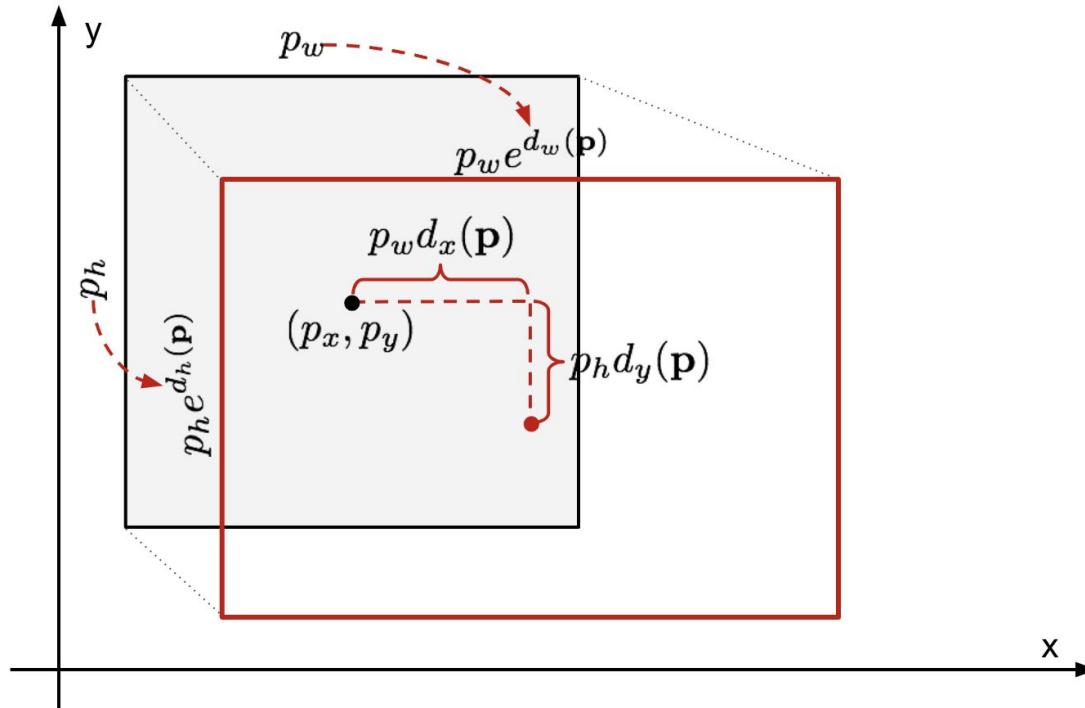
Segmentation refers to pixel-wise localization of the labels

# Goals of object detection

Given an input image we wish to obtain:

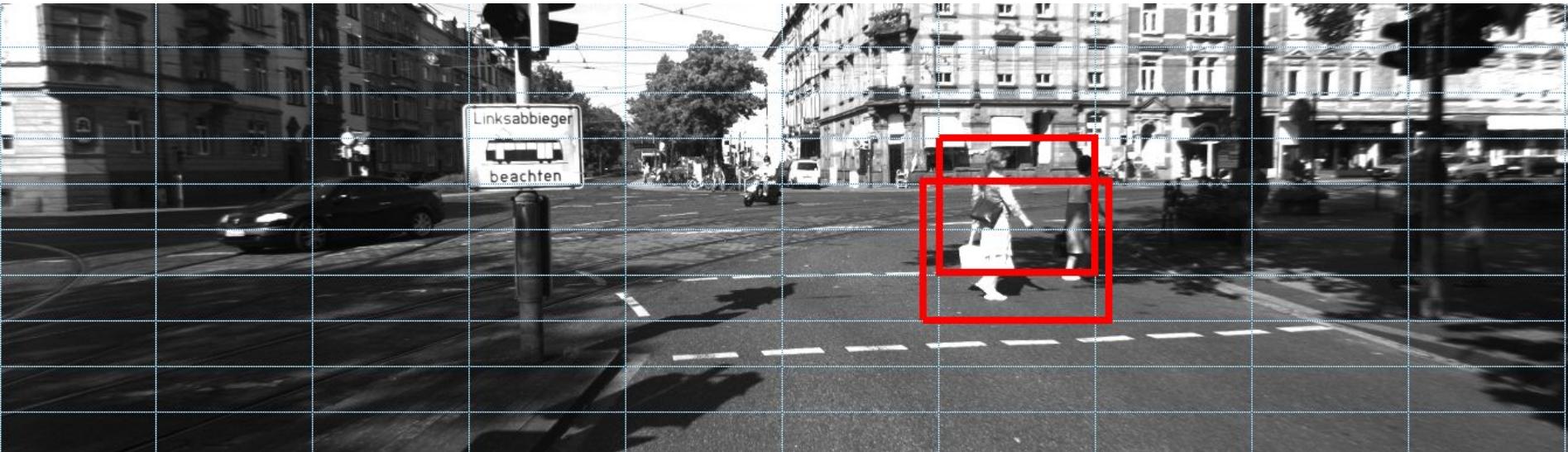
1. A **list of bounding boxes**, or the  $(x, y)$ -coordinates for each object in an image
2. A **class label** associated with each bounding box
3. The **probability** score associated with each bounding box and class label

# Object detection as bounding box regression (1st step)

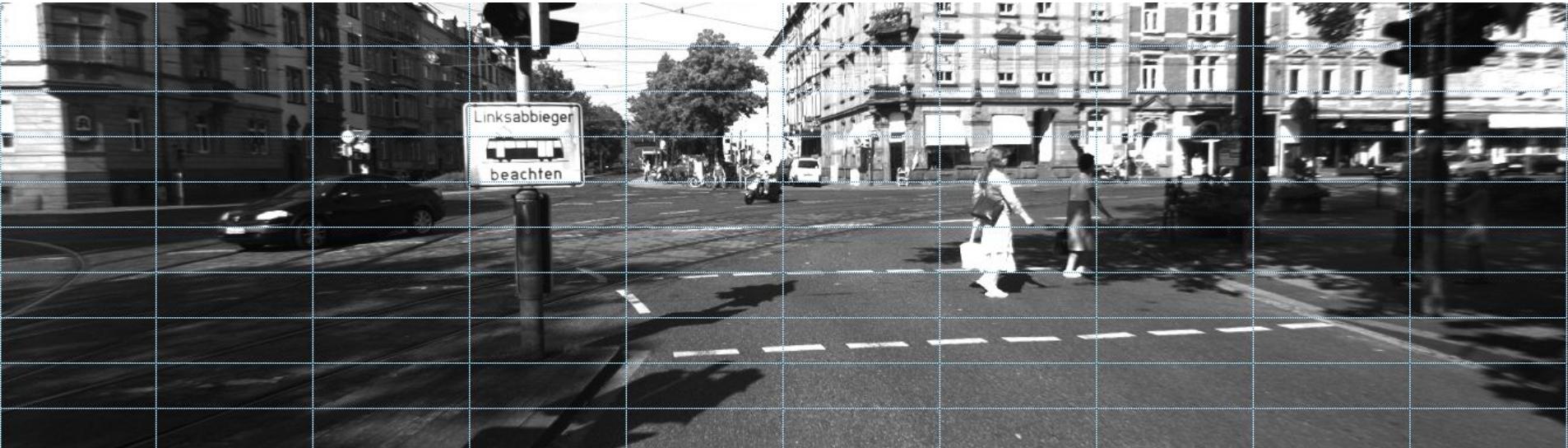


‘Regression’ means predicting a continuous value

# Breaking down an image into regions



# Breaking down an image into regions

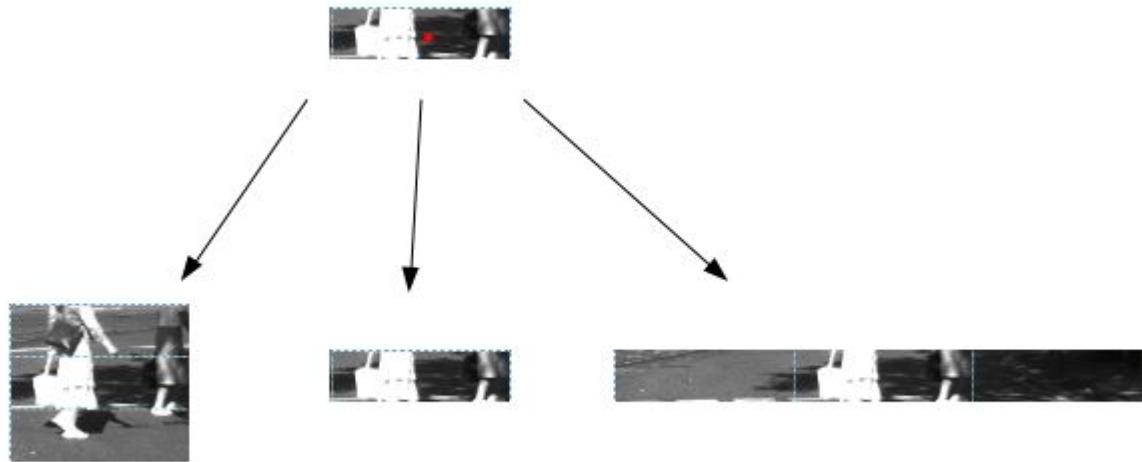


A sliding window approach partitions the image into discrete blocks

# Breaking down an image into regions

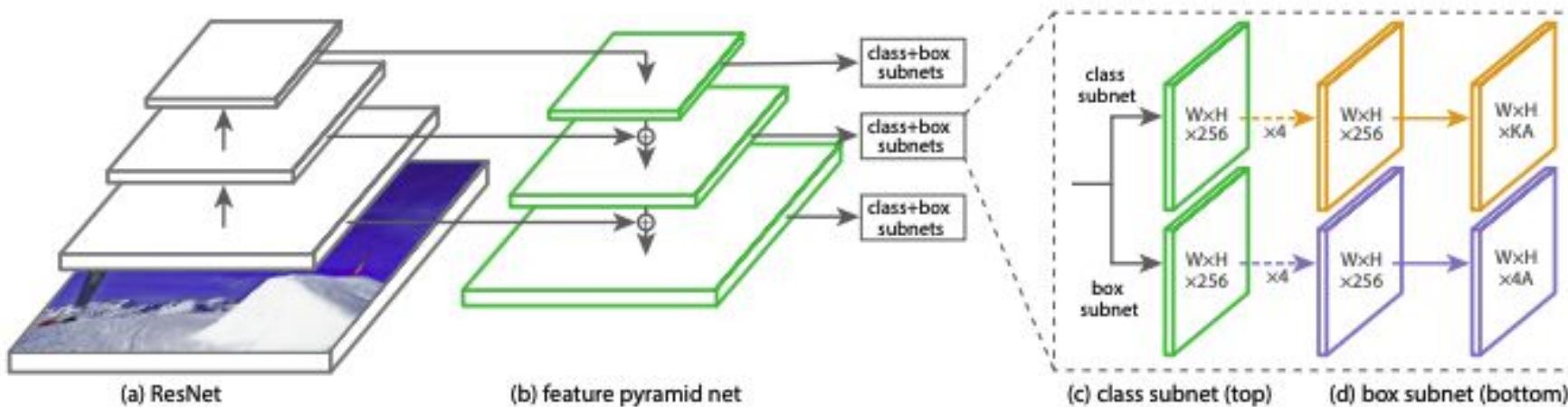


# Breaking down an image into regions

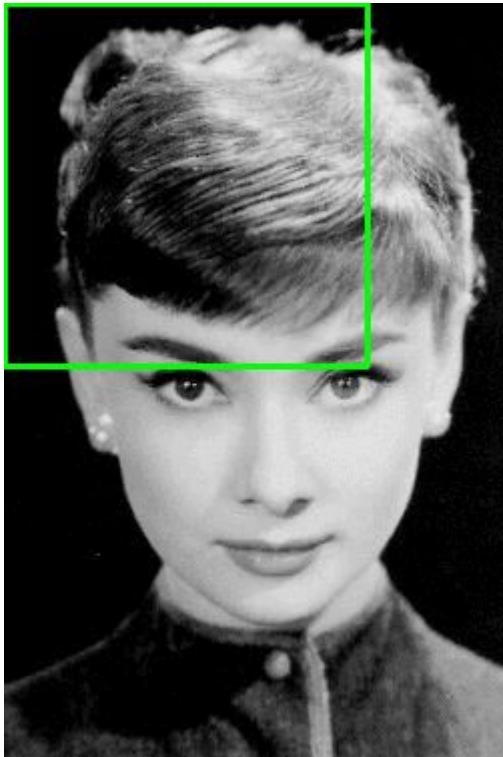


Finding the optimal image partitioning algorithm is not a trivial task

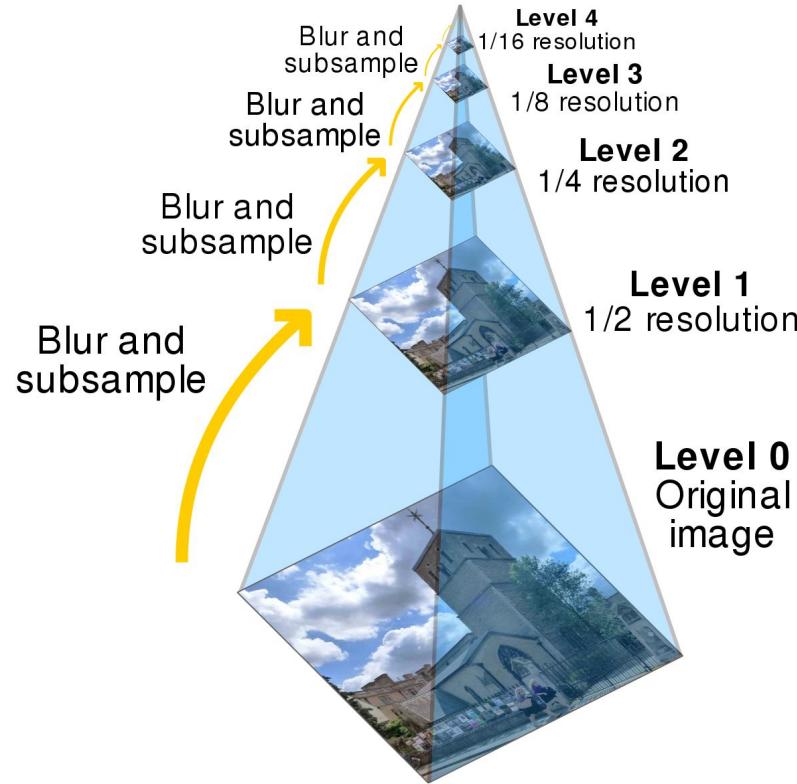
# RetinaNet's architecture



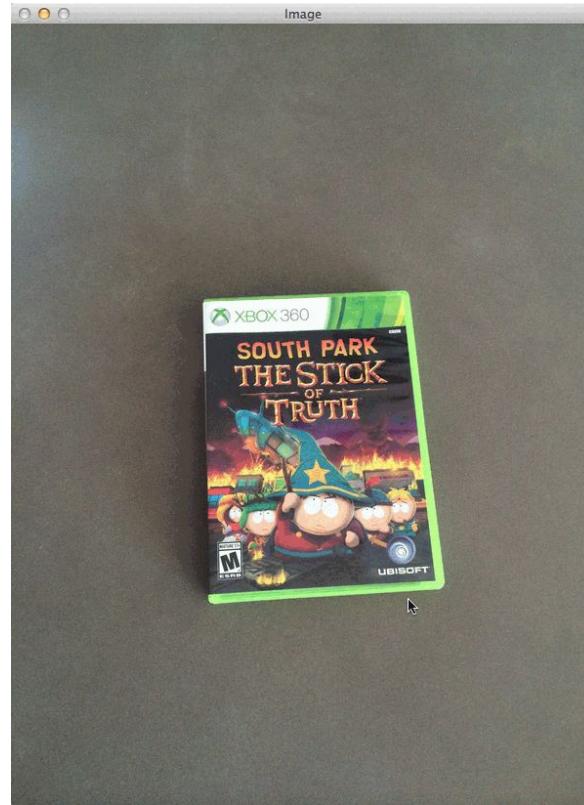
# Sliding windows



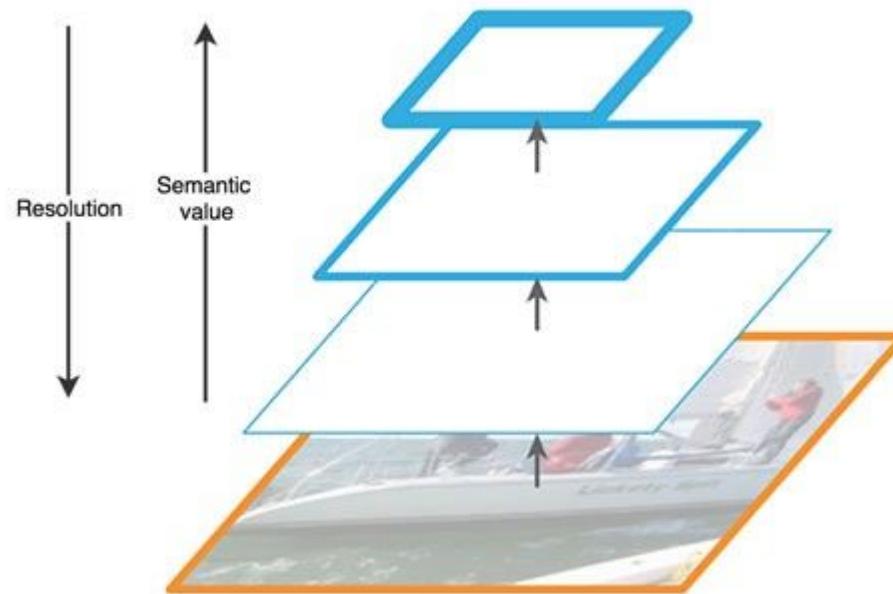
# Image pyramids



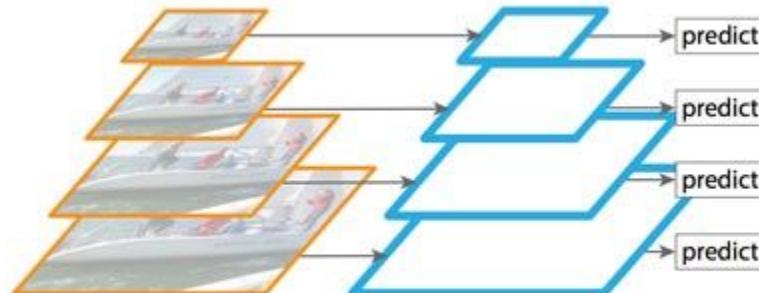
# Sliding windows on an image pyramid



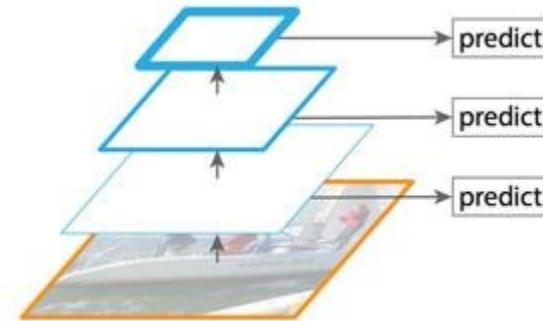
# Image Pyramids



# Image Pyramids

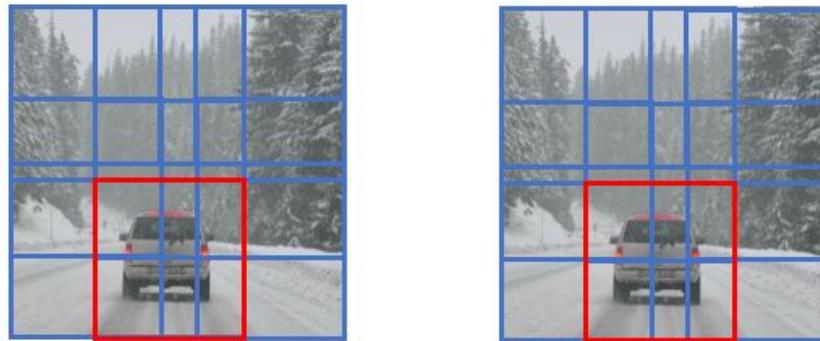
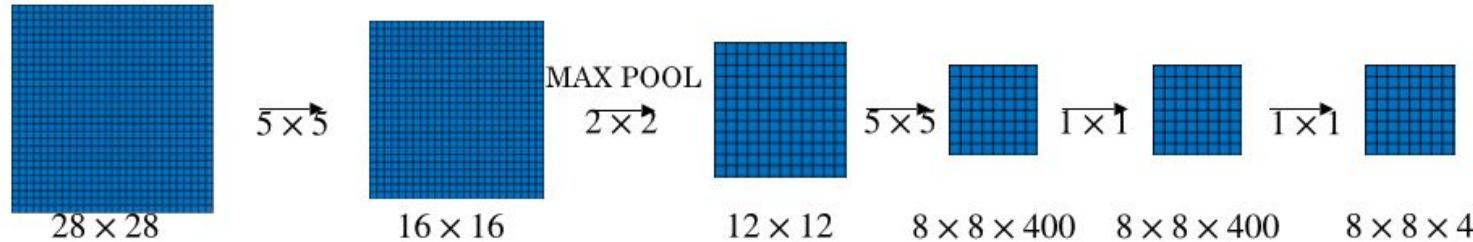


Pyramid of images



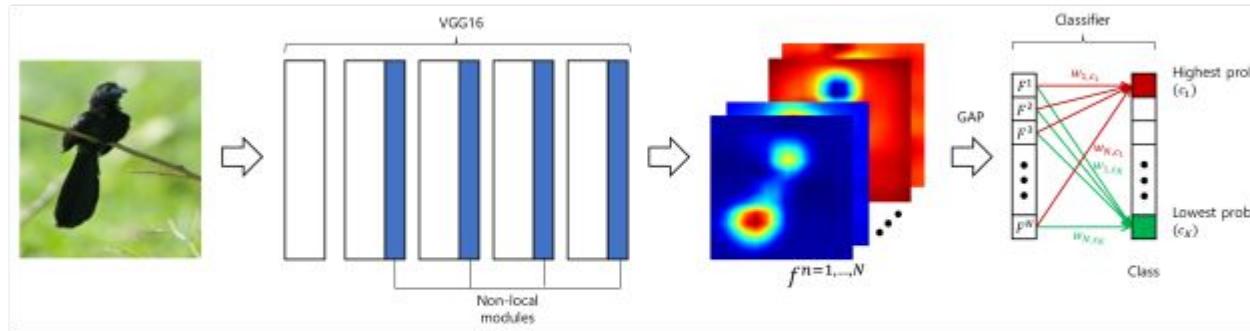
Pyramid of feature maps

# Pyramids and sliding windows in CNNs



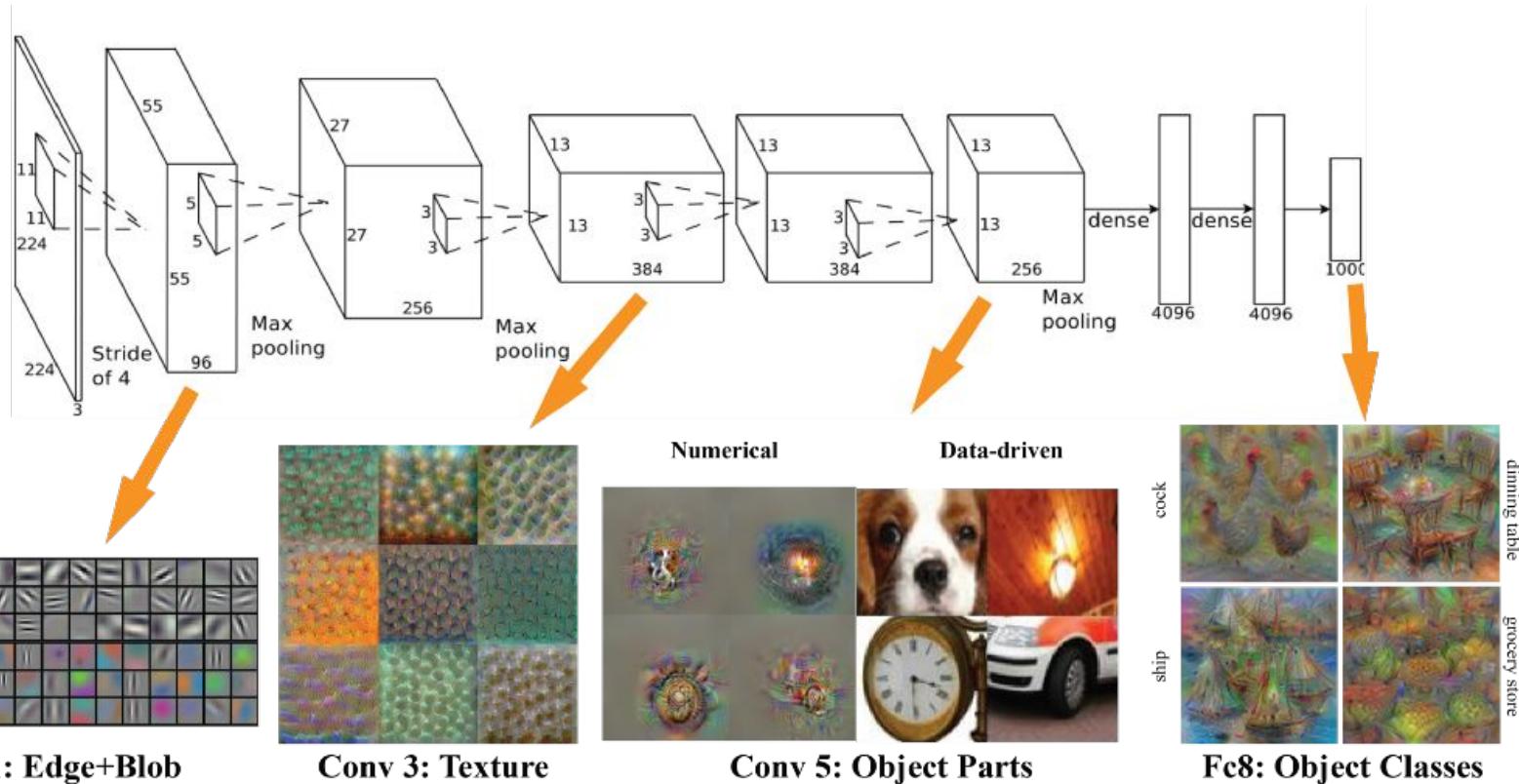
Andrew Ng

# Transfer learning of classification is currently used for feature extraction

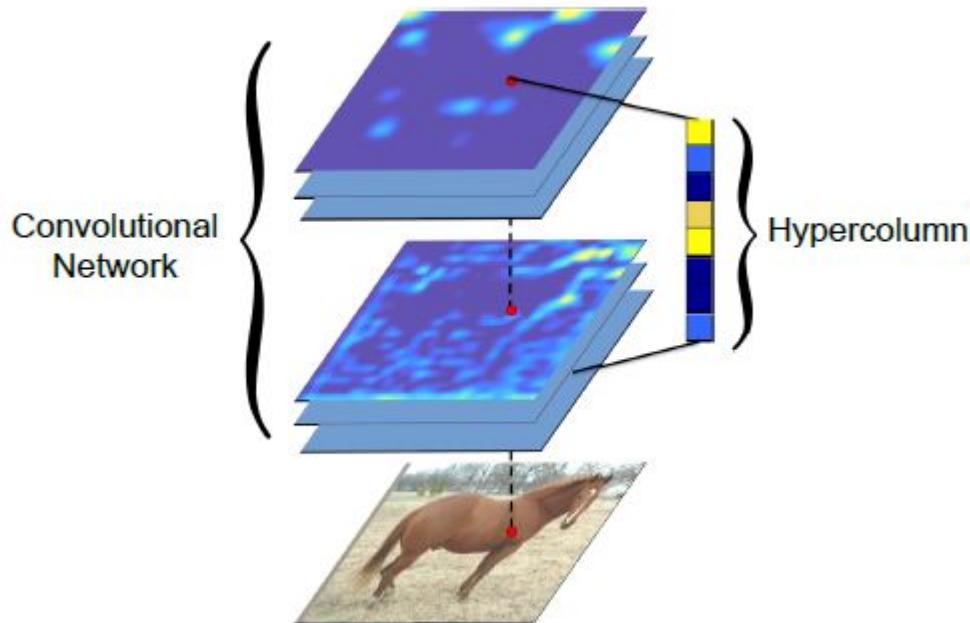


[Source](#)

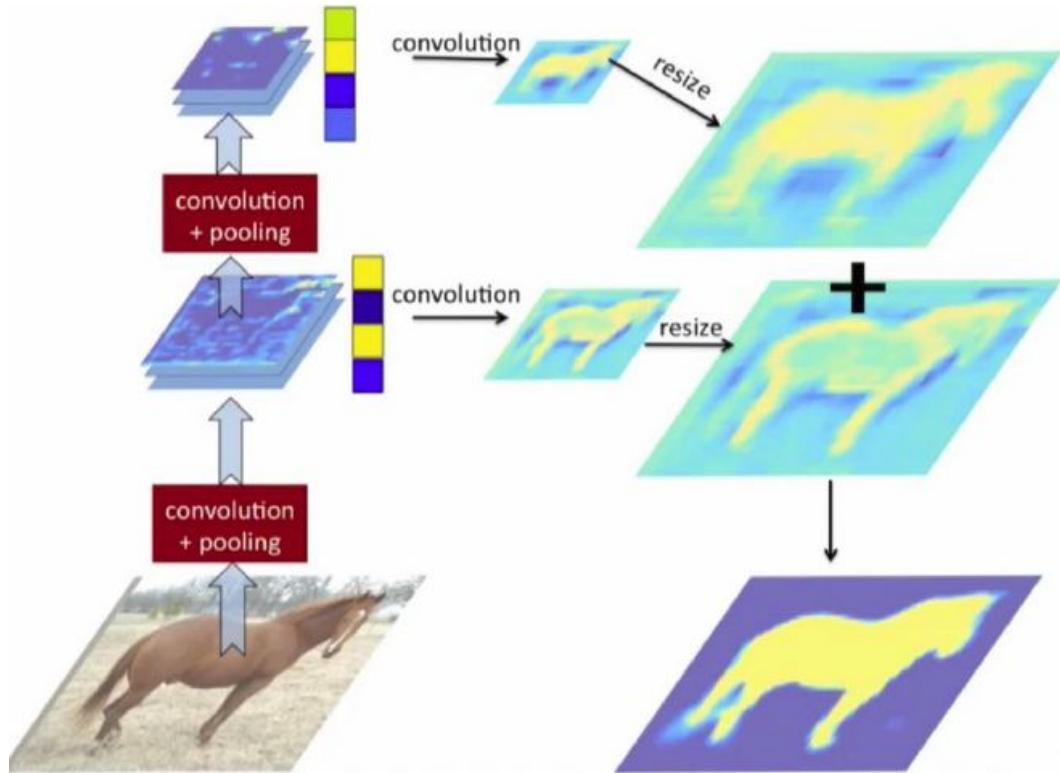
# What convolutional networks learn



# “Hypercolumns”



# “Hypercolumns”



# Anchor boxes

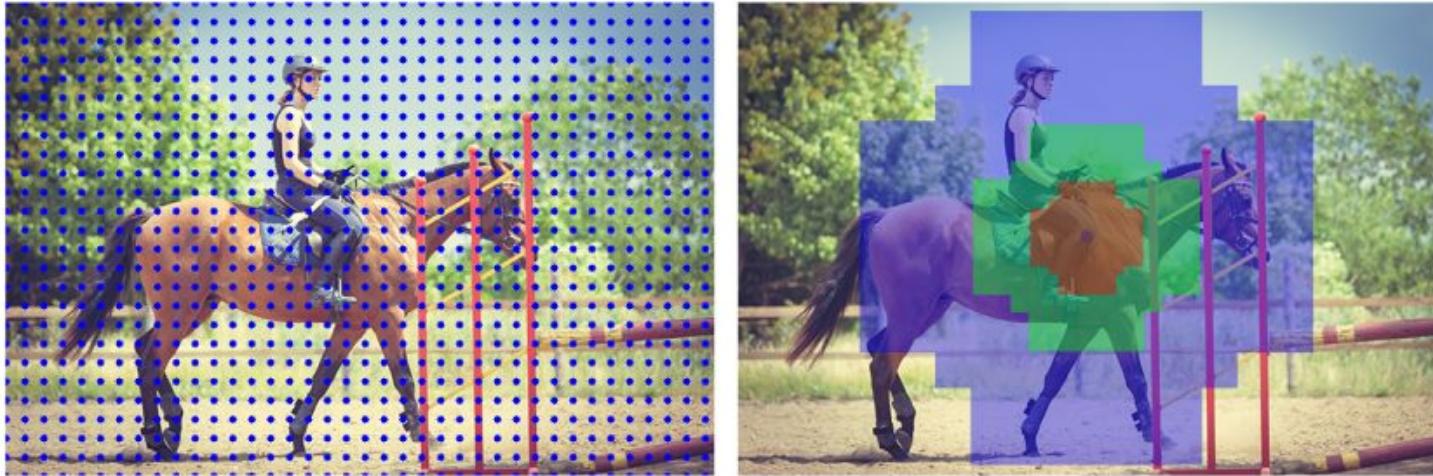
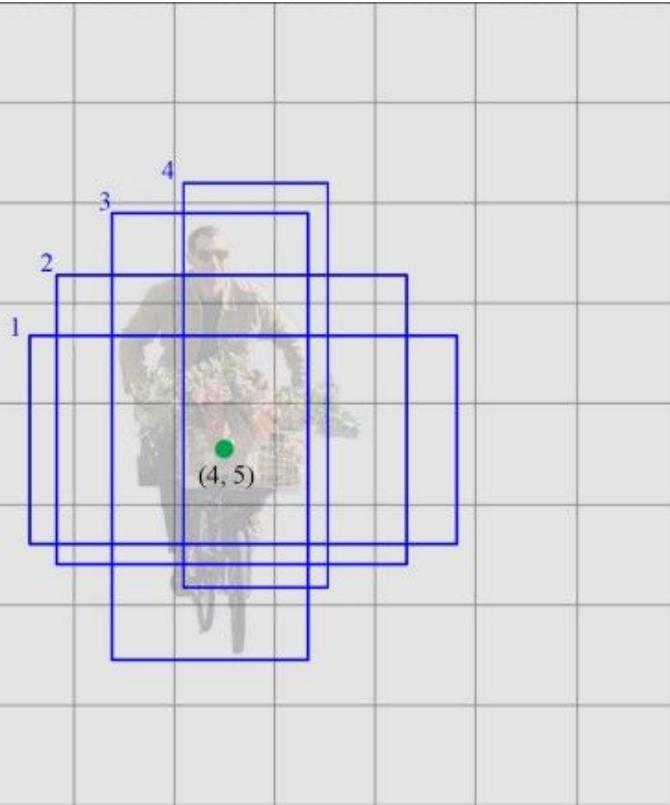


Figure 14.7: **Left:** Creating anchors starts with the process of sampling the coordinates of an image every  $r$  pixels ( $r = 16$  in the original Faster R-CNN implementation). **Right:** We create a total of nine anchors centered around *each* sampled  $(x, y)$ -coordinate. In this visualization,  $x = 300, y = 200$  (center blue coordinate). The nine total anchors come from every combination of scale:  $64 \times 64$  (red),  $128 \times 128$  (green),  $256 \times 256$  (blue); and aspect ratio:  $1 : 1, 2 : 1, 1 : 2$ .

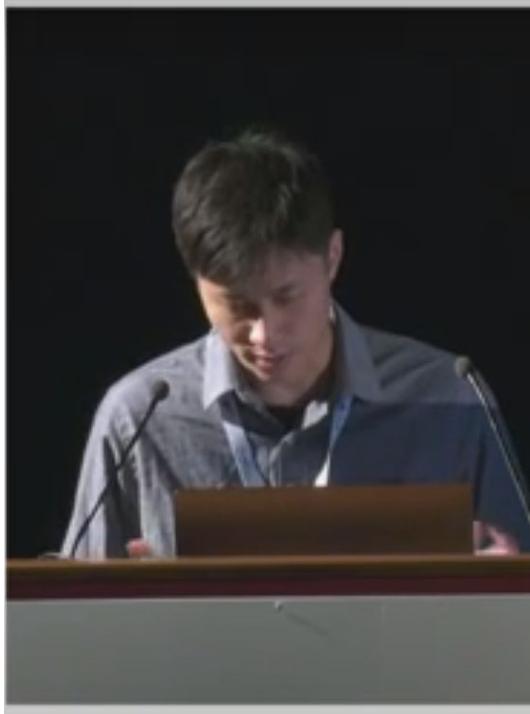
# Anchor boxes



# Anchor boxes



# RetinaNet's high number of anchor boxes



## Toward dense detection

- YOLOv1 – 98 boxes
- YOLOv2 – ~1k
- OverFeat – ~1-2k
- SSD – ~8-26k
- This work – **~100k**

# Retinanet's solution to the class imbalance problem

## Class Imbalance

- Few training examples from foreground
- Most examples from background
  - Easy and uninformative
  - Distracting

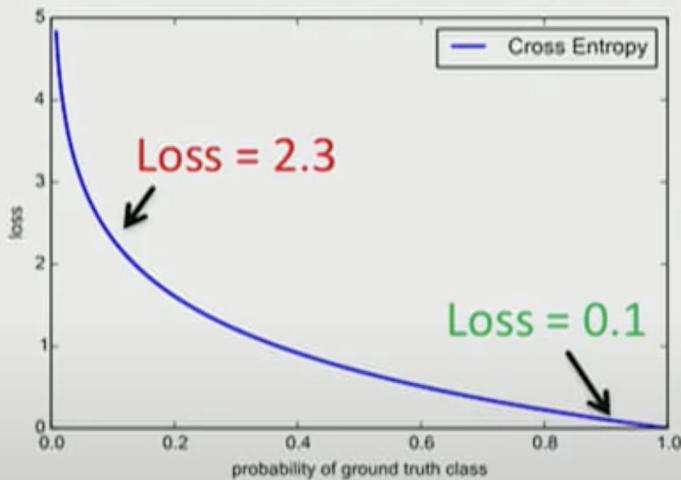
Many negative examples, no useful signal

Few positive examples, rich information

[RetinaNet: how Focal Loss fixes Single-Shot Detection](#)

# Retinanet's solution to the class imbalance problem

- 100000 easy : 100 hard examples
- 40x bigger loss from easy examples



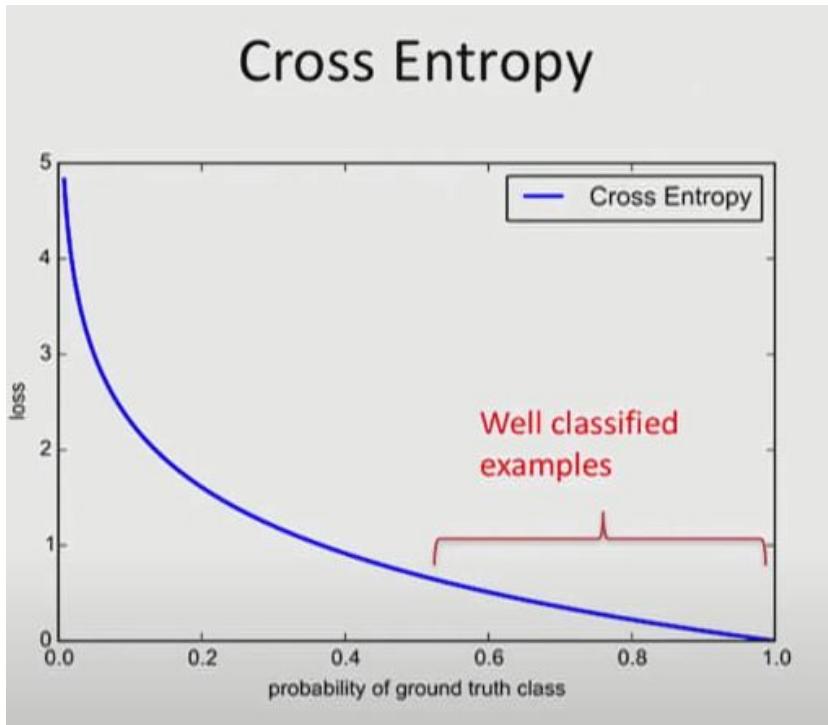
[RetinaNet: how Focal Loss fixes Single-Shot Detection](#)

# Cross entropy loss

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

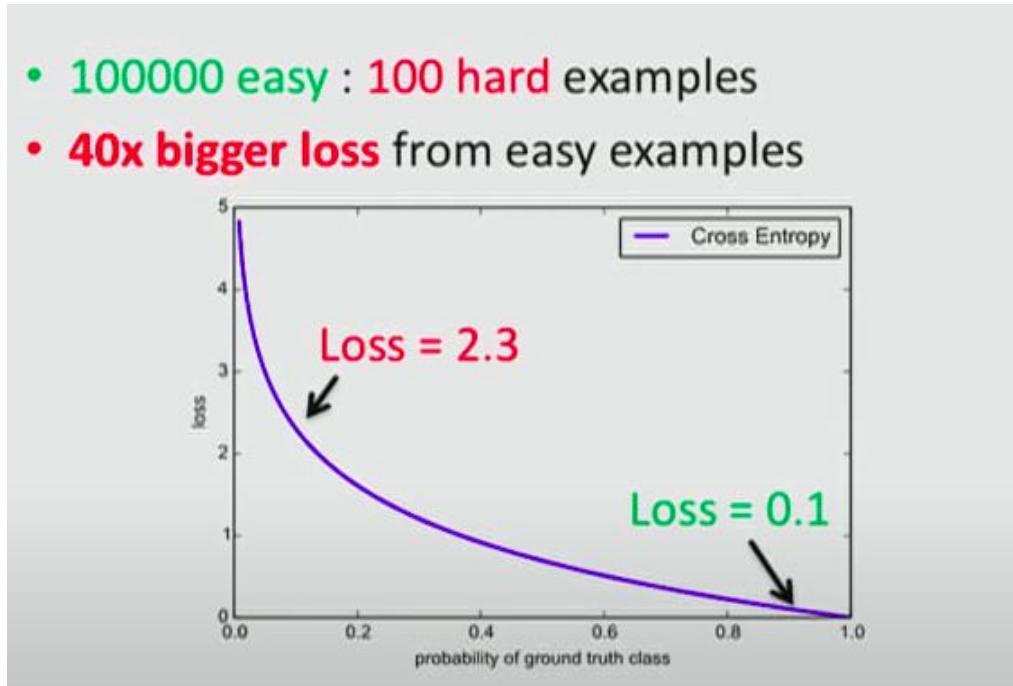
[Understanding cross entropy loss](#)

# Cross entropy at different confidence levels

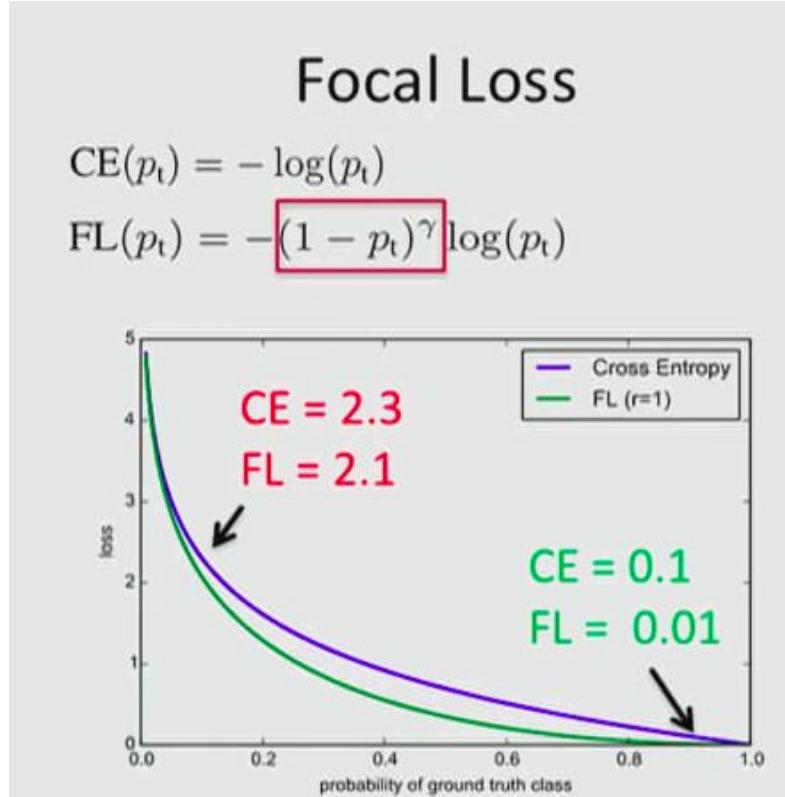


[Andrew Ng's explanation of cross entropy loss](#)

Regular cross entropy gives too much importance to easy cases



# Retinanet's solution to the class imbalance problem



## Focal loss as modified cross-entropy loss

$$\text{CE}(p, y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1 - p) & \text{otherwise.} \end{cases} \quad (1)$$

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{otherwise,} \end{cases} \quad (2)$$

$$\text{FL}(p_t) = -(1 - p_t)^\gamma \log(p_t). \quad (4)$$

[Colab notebook to play with these equations](#)

# Tweaking gamma $\gamma$

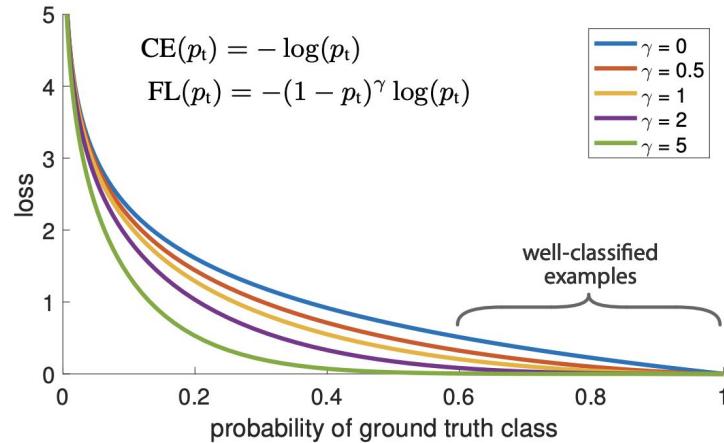
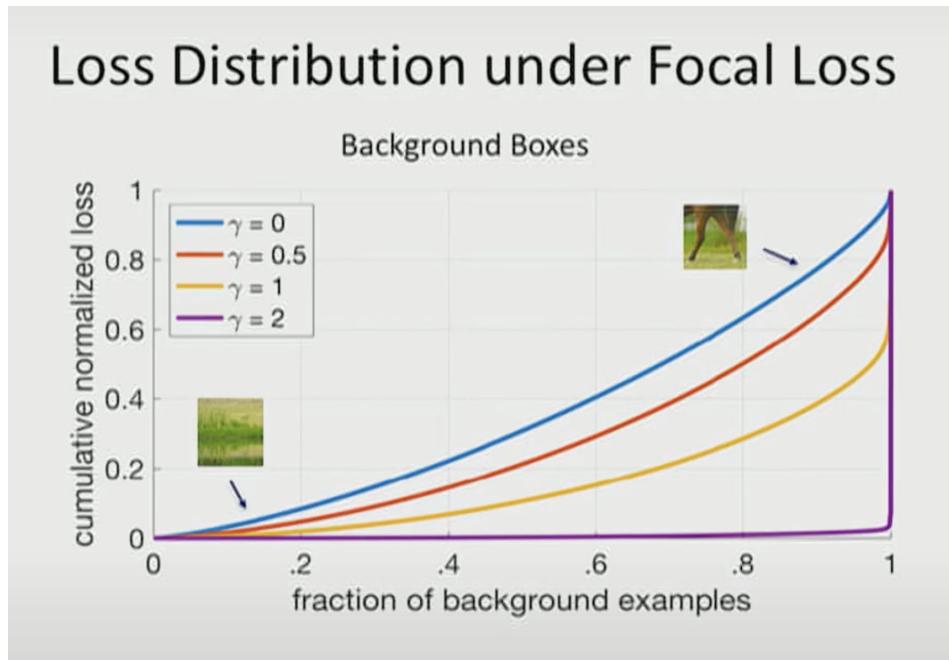


Figure 1. We propose a novel loss we term the *Focal Loss* that adds a factor  $(1 - p_t)^\gamma$  to the standard cross entropy criterion. Setting  $\gamma > 0$  reduces the relative loss for well-classified examples ( $p_t > .5$ ), putting more focus on hard, misclassified examples. As our experiments will demonstrate, the proposed focal loss enables training highly accurate dense object detectors in the presence of vast numbers of easy background examples.

$\gamma = 0$  regular cross entropy (no focus on infrequent samples)

$\gamma > 0.5$  reduces loss for samples **where the model has very high confidence**

# Tweaking gamma $\gamma$



$\gamma = 0$  regular cross entropy (no focus on infrequent samples)

$\gamma > 0.5$  reduces loss for samples where the model has very high confidence, usually has the effect of suppressing the background boxes contribution to the loss

# No free lunch: experiments must be done

## 4. Things We Tried That Didn't Work

We tried lots of stuff while we were working on YOLOv3. A lot of it didn't work. Here's the stuff we can remember.

**Anchor box  $x, y$  offset predictions.** We tried using the normal anchor box prediction mechanism where you predict the  $x, y$  offset as a multiple of the box width or height using a linear activation. We found this formulation decreased model stability and didn't work very well.

**Linear  $x, y$  predictions instead of logistic.** We tried using a linear activation to directly predict the  $x, y$  offset instead of the logistic activation. This led to a couple point drop in mAP.

**Focal loss.** We tried using focal loss. It dropped our mAP about 2 points. YOLOv3 may already be robust to the problem focal loss is trying to solve because it has separate objectness predictions and conditional class predictions. Thus for most examples there is no loss from the class predictions? Or something? We aren't totally sure.

[Extract from the YOLOv3 paper](#)

# Weight decay aka $\|x\|$ -regularization

$$\text{L1 ERROR FUNCTION} = -\frac{1}{m} \sum_{i=1}^m (1 - y_i) \ln(1 - \hat{y}_i) + y_i \ln(\hat{y}_i) + \boxed{\lambda(|w_1| + \dots + |w_n|)}$$

$$\text{L2 ERROR FUNCTION} = -\frac{1}{m} \sum_{i=1}^m (1 - y_i) \ln(1 - \hat{y}_i) + y_i \ln(\hat{y}_i) + \boxed{\lambda(w_1^2 + \dots + w_n^2)}$$

L1 and L2 regularization

# Smooth L1 Loss

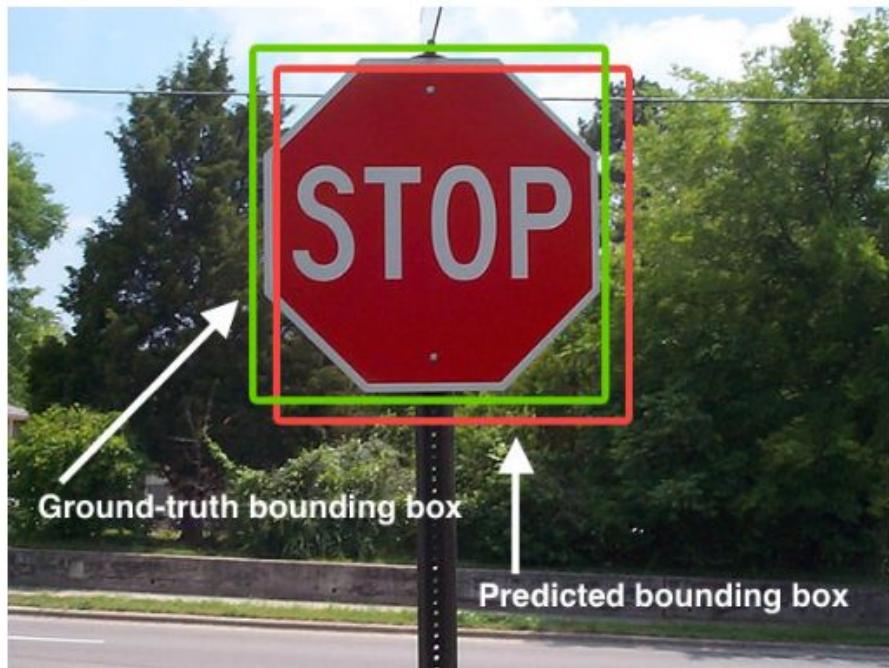
Smooth L1-loss can be interpreted as a combination of L1-loss and L2-loss. It behaves as L1-loss when the absolute value of the argument is high, and it behaves like L2-loss when the absolute value of the argument is close to zero. The equation is:

$$L_{1;smooth} = \begin{cases} |x| & \text{if } |x| > \alpha; \\ \frac{1}{|\alpha|}x^2 & \text{if } |x| \leq \alpha \end{cases}$$

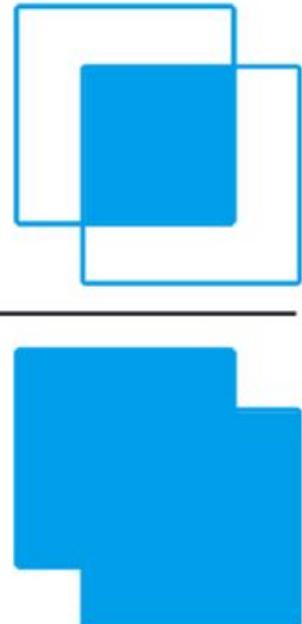
$\alpha$  is a hyper-parameter here and is usually taken as 1.  $\frac{1}{\alpha}$  appears near  $x^2$  term to make it continuous.

Smooth L1-loss combines the advantages of L1-loss (steady gradients for large values of  $x$ ) and L2-loss (less oscillations during updates when  $x$  is small).

# Measuring performance with IoU

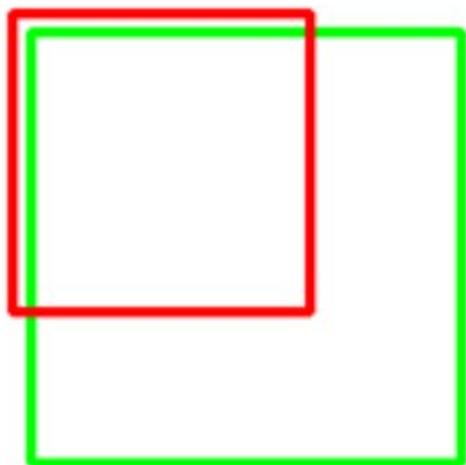


$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



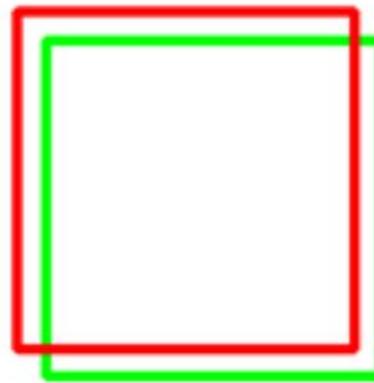
# Measuring performance with IoU

IoU: 0.4034



Poor

IoU: 0.7330



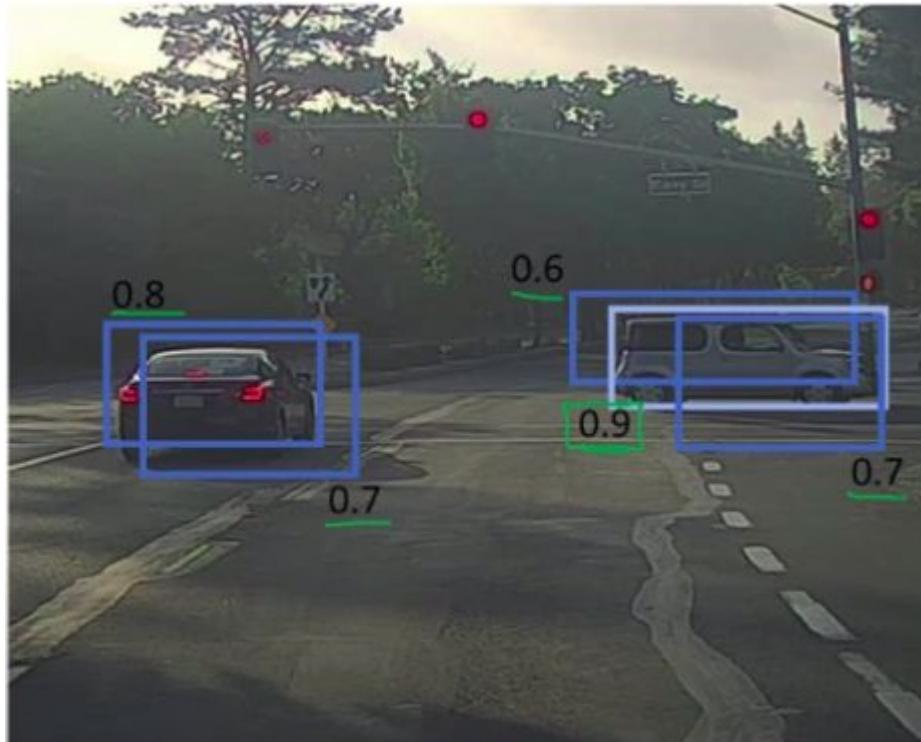
Good

IoU: 0.9264

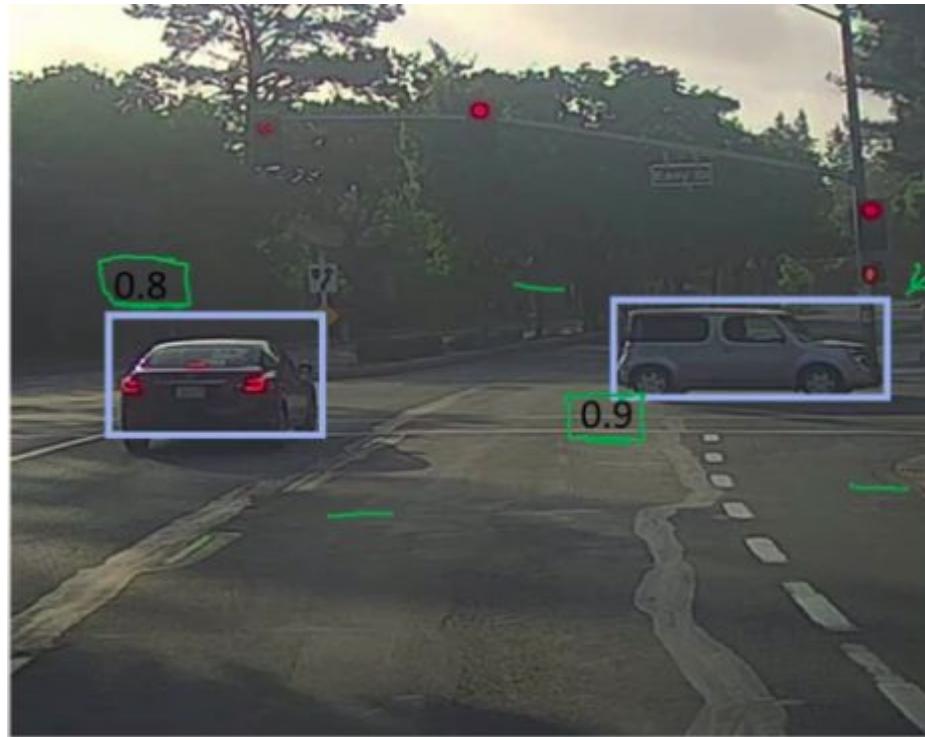


Excellent

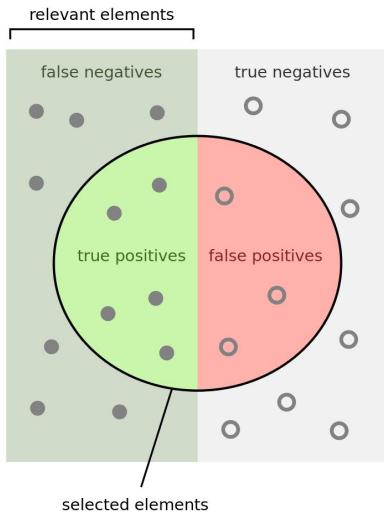
# Non-max suppression



# Non-max suppression

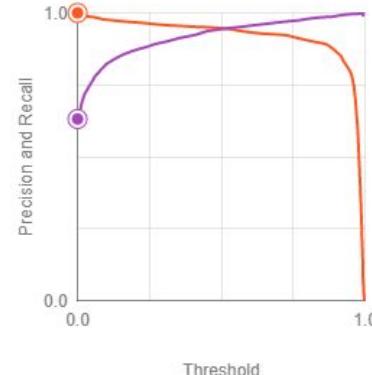


# Precision, recall, and f1 metrics



$$\text{Precision} = \frac{\text{How many selected items are relevant?}}{\text{How many relevant items are selected?}}$$
$$\text{Recall} = \frac{\text{How many relevant items are selected?}}{\text{How many relevant elements?}}$$

Precision and Recall vs Threshold



$$F_1 = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

- |                |  |
|----------------|--|
| True Positive  | = dog (pixel) labeled as dog (pixel)         |
| True Negative  | = non dog (pixel) labeled as non dog (pixel) |
| False Positive | = non dog (pixel) labeled as dog (pixel)     |
| False Negative | = dog (pixel) labeled as non dog (pixel)     |

# Understanding mean Average Precision (mAP)

$$AP = \sum_{k=0}^{k=n-1} [Recalls(k) - Recalls(k + 1)] * Precisions(k)$$

$Recalls(n) = 0, Precisions(n) = 1$

$n = Number\ of\ thresholds.$

[Mean Average Precision \(mAP\) Explained](#)

# Understanding mean Average Precision (mAP)

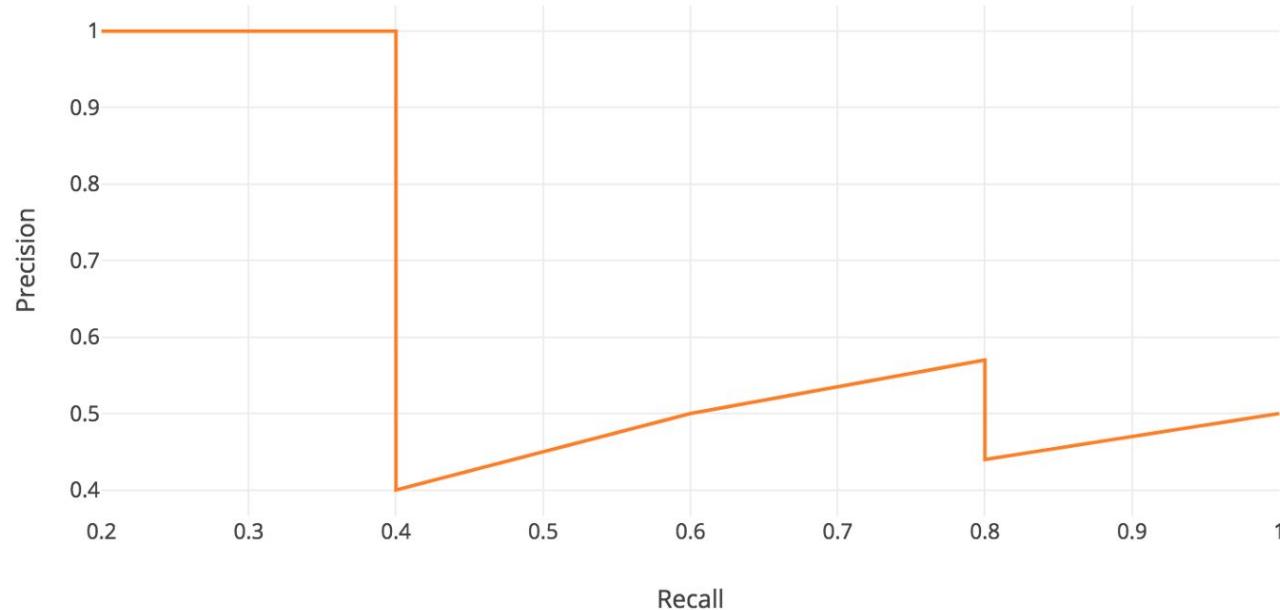
$$mAP = \frac{1}{n} \sum_{k=1}^{k=n} AP_k$$

$AP_k$  = the AP of class  $k$

$n$  = the number of classes

[Mean Average Precision \(mAP\) Explained](#)

# Understanding mean Average Precision (mAP)



[Mean Average Precision \(mAP\) Explained](#)



COCO

Common Objects in Context

info@cocodataset.org

Home People Dataset Tasks Evaluate

## COCO Explorer

COCO 2017 train/val browser (123,287 images, 886,284 instances). Crowd labels not shown.

<https://cocodataset.org/#explore>

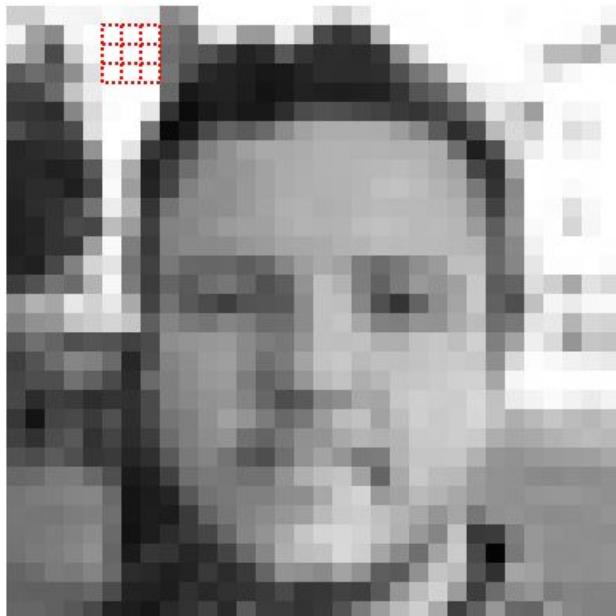
# Convolution kernels (aka filters)

206	205	247	245	244	233	247	245	136	151	258	255	258	255	258	255	234	307	311	251	254	254	255	255	252	255	255	254	255	247			
244	281	137	244	255	254	255	254	118	102	228	155	232	193	172	52	66	253	255	254	255	255	255	255	254	255	254	255	244				
182	159	70	205	249	255	255	255	110	96	84	35	44	99	53	44	43	54	121	235	255	255	255	255	245	357	186	176	232				
90	108	96	143	223	255	255	255	252	175	73	41	31	24	25	36	45	44	44	46	111	148	234	252	254	255	248	231	248	255	254		
67	69	107	196	235	255	255	255	104	29	35	29	20	25	34	32	30	32	34	83	85	102	141	232	247	248	255	255	255	255			
55	51	34	218	255	255	255	255	111	12	23	24	26	25	25	75	82	71	66	60	139	228	255	255	255	254	246	255	255	254	255		
79	55	56	75	224	255	255	255	118	11	27	74	91	91	206	340	182	273	372	373	273	272	158	137	92	46	70	187	217	208	254	232	235
38	43	47	52	147	255	255	255	91	41	81	145	180	180	169	172	178	178	176	177	177	177	176	172	110	31	208	238	254	244	245	245	
40	43	33	36	90	245	271	312	32	60	110	139	145	151	162	171	174	178	173	182	184	185	183	173	173	71	45	165	254	255	254	254	
37	44	44	31	69	250	258	36	70	129	143	142	150	162	173	175	177	178	182	192	194	188	180	170	120	51	137	255	254	250	254		
34	45	51	64	110	237	253	253	56	118	130	143	144	154	164	176	178	177	177	183	185	185	183	180	61	141	254	253	245	245			
36	32	50	74	71	138	256	255	133	134	134	144	160	163	173	176	178	180	180	185	187	182	183	186	148	250	254	234	247	245			
32	38	52	54	159	250	256	57	129	138	138	140	151	156	186	186	171	178	180	187	186	185	185	183	183	260	326	126	246	255	254	254	
36	32	72	129	212	228	115	69	131	104	103	104	104	103	134	156	170	182	125	108	121	143	156	159	180	181	104	134	230	253	253	251	
61	62	118	107	207	247	244	130	60	100	110	119	203	81	94	947	392	178	126	85	123	147	181	200	92	100	222	207	187	227	225		
144	176	233	230	220	239	375	67	105	88	70	82	83	85	139	193	192	180	83	93	94	145	181	205	197	170	245	245	245	246	245		
127	145	249	185	204	213	187	95	133	132	117	133	136	158	110	130	181	187	187	127	148	147	171	188	110	121	228	233	240	215	213		
87	112	300	79	82	65	72	145	142	153	153	138	120	146	140	180	180	175	174	193	186	198	206	125	128	255	149	186	185	185	185		
63	103	104	134	129	139	70	73	152	144	155	159	131	114	184	184	186	185	186	185	191	190	190	190	200	200	200	243	251	249	240	234	
69	78	78	113	97	74	43	206	127	140	152	155	125	97	112	150	185	184	174	183	186	186	202	208	209	208	247	254	254	254	254		
72	44	63	50	46	52	49	42	177	137	137	149	132	103	78	90	134	168	166	189	207	204	203	203	236	236	234	241	242	243	243		
50	26	69	73	59	50	48	46	71	127	144	143	144	124	120	120	126	187	182	180	182	206	205	204	234	234	187	185	187	183	183		
65	49	77	89	50	65	43	61	106	127	141	147	113	100	121	145	148	189	181	178	181	201	201	202	202	174	188	186	178	183	184	184	
82	70	72	70	54	58	37	47	98	123	132	116	89	71	114	146	145	122	124	150	187	187	176	187	149	146	152	155	157	158	158		
104	257	122	122	128	70	27	33	66	111	122	120	114	124	124	175	175	290	186	182	203	202	170	200	187	188	246	145	229	137	142	140	145
117	128	123	123	125	21	26	37	88	115	122	128	128	143	142	160	202	212	233	234	186	186	185	185	244	146	244	131	251	144	144		
119	114	118	118	125	128	111	21	29	26	58	100	118	131	140	151	150	188	203	209	192	180	185	189	184	186	119	144	147	143	140	144	148
117	119	125	130	130	106	18	24	59	80	102	103	128	147	188	187	212	215	230	235	177	175	133	133	106	95	126	151	143	145	142	146	
125	126	126	134	145	106	27	54	58	49	60	105	135	178	195	193	196	196	186	111	134	183	184	184	7	5	121	153	142	143	146	146	
103	308	123	121	132	132	125	44	40	31	35	57	44	58	203	247	344	338	182	145	94	90	145	196	187	94	48	185	260	142	344	342	345
98	97	98	104	104	104	74	34	30	33	48	41	50	51	74	53	55	66	63	105	150	185	196	152	108	140	145	123	125	123	123	123	
102	100	97	87	85	81	39	30	23	42	50	60	41	60	51	51	57	123	127	157	205	208	196	152	96	105	131	104	151	124	130	129	



<https://setosa.io/ev/image-kernels/>

# What is a convolution filter?



input image

$$\left( \begin{array}{c} 255 \times 0 + 254 \times -1 + 255 \times 0 \\ + 255 \times -1 + 255 \times 5 + 255 \times -1 \\ + 255 \times 0 + 255 \times -1 + 252 \times 0 \\ = 256 \end{array} \right)$$

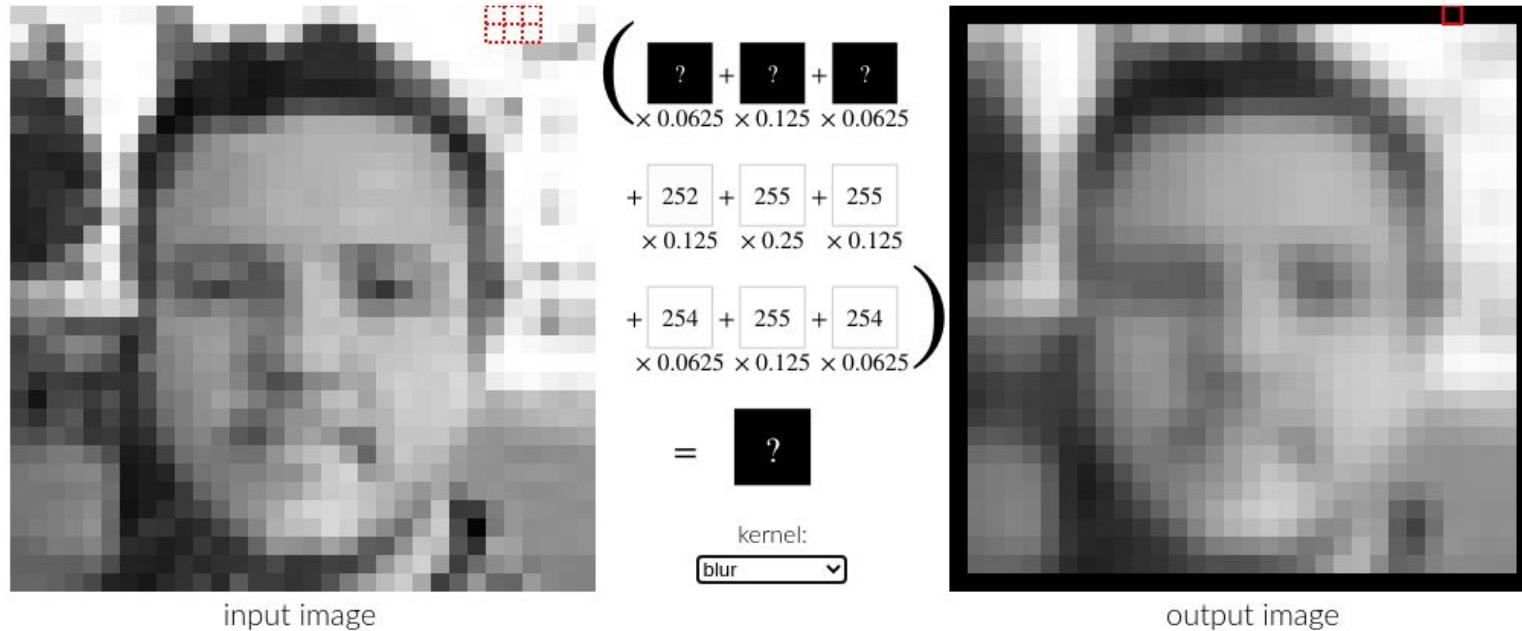
kernel:  
sharpen



output image

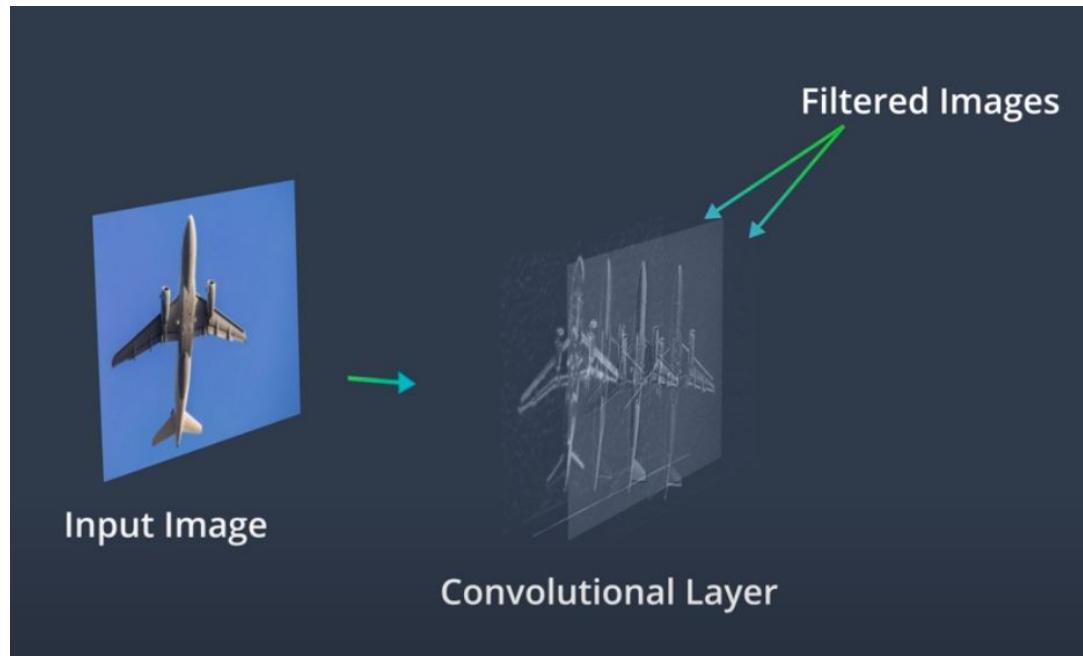
<https://setosa.io/ev/image-kernels/>

# What is a convolution filter?



<https://setosa.io/ev/image-kernels/>

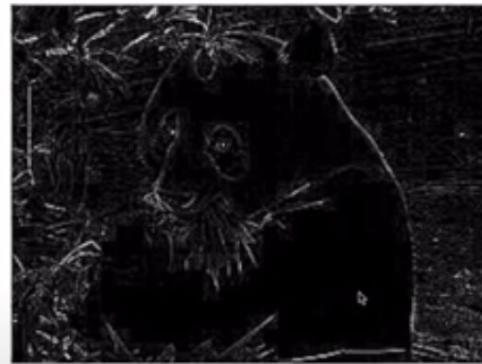
# What is a convolution filter?



[Understanding the output of convolutional layers](#)

# What is a convolution filter?

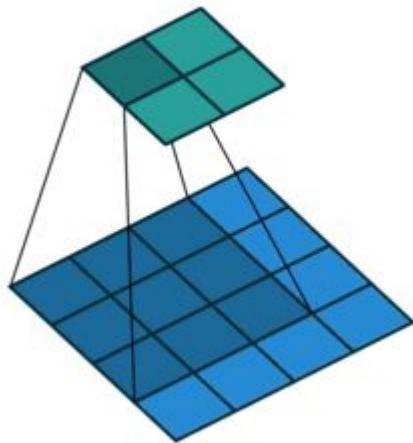
0	-1	0
-1	4	<b>-2</b>
0	-1	0



$$0 + -1 + 0 + -1 + 4 + \mathbf{-2} + 0 + -1 + 0 = -1$$

[Visualizing convolution kernels](#)

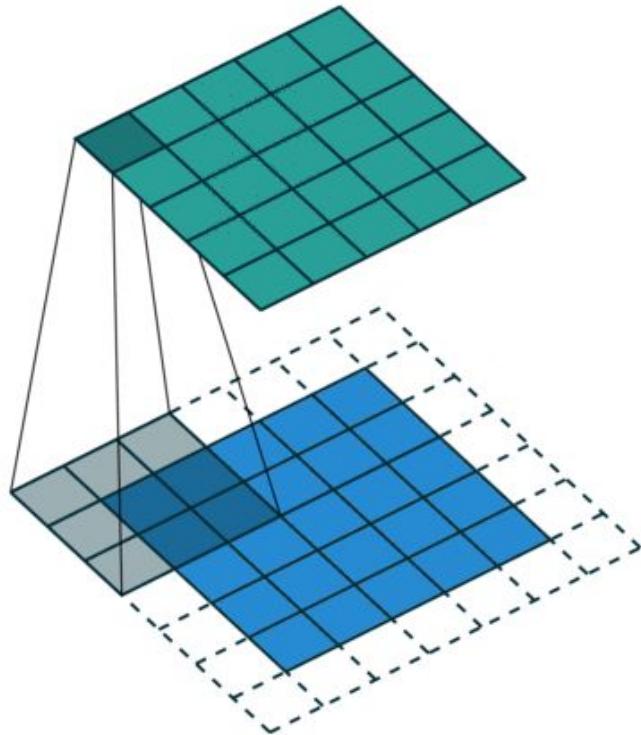
# What is a convolution filter?



Convolution of  $3 \times 3$  and stride = 1 without padding

Effect: the output loses one pixel on each dimension

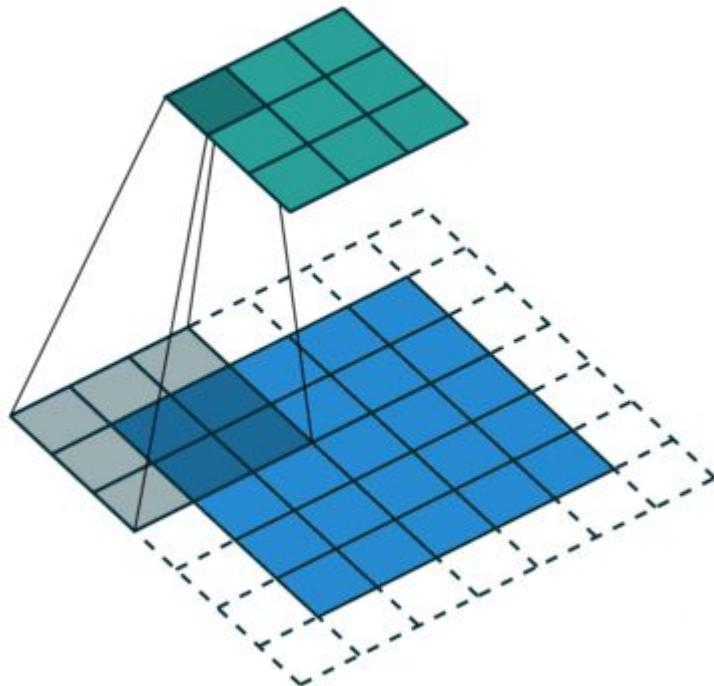
# What is a convolution filter?



Convolution of  $3 \times 3$  and stride = 1 with zero padding of one pixel

Effect: the output preserves original image size

# What is a convolution filter?



Convolution of 3x3 and stride = 2 with zero padding

Effect: the output is downsampled to about half its size

[Activity: Conv2D in Pytorch \(Colab notebook\)](#)

# Experiment: finding edges with convolutions

$$S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

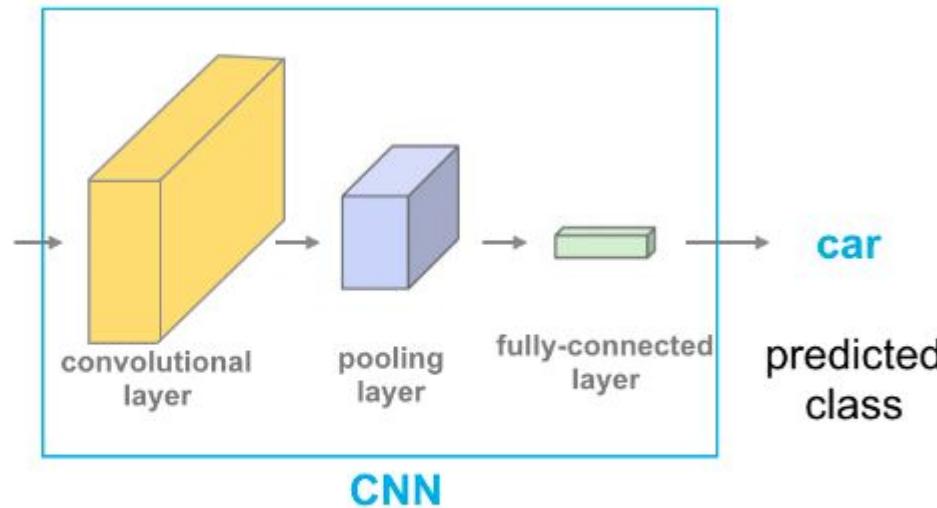
$$S_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

[Finding edges with convolution kernels](#)

# CNN architectures

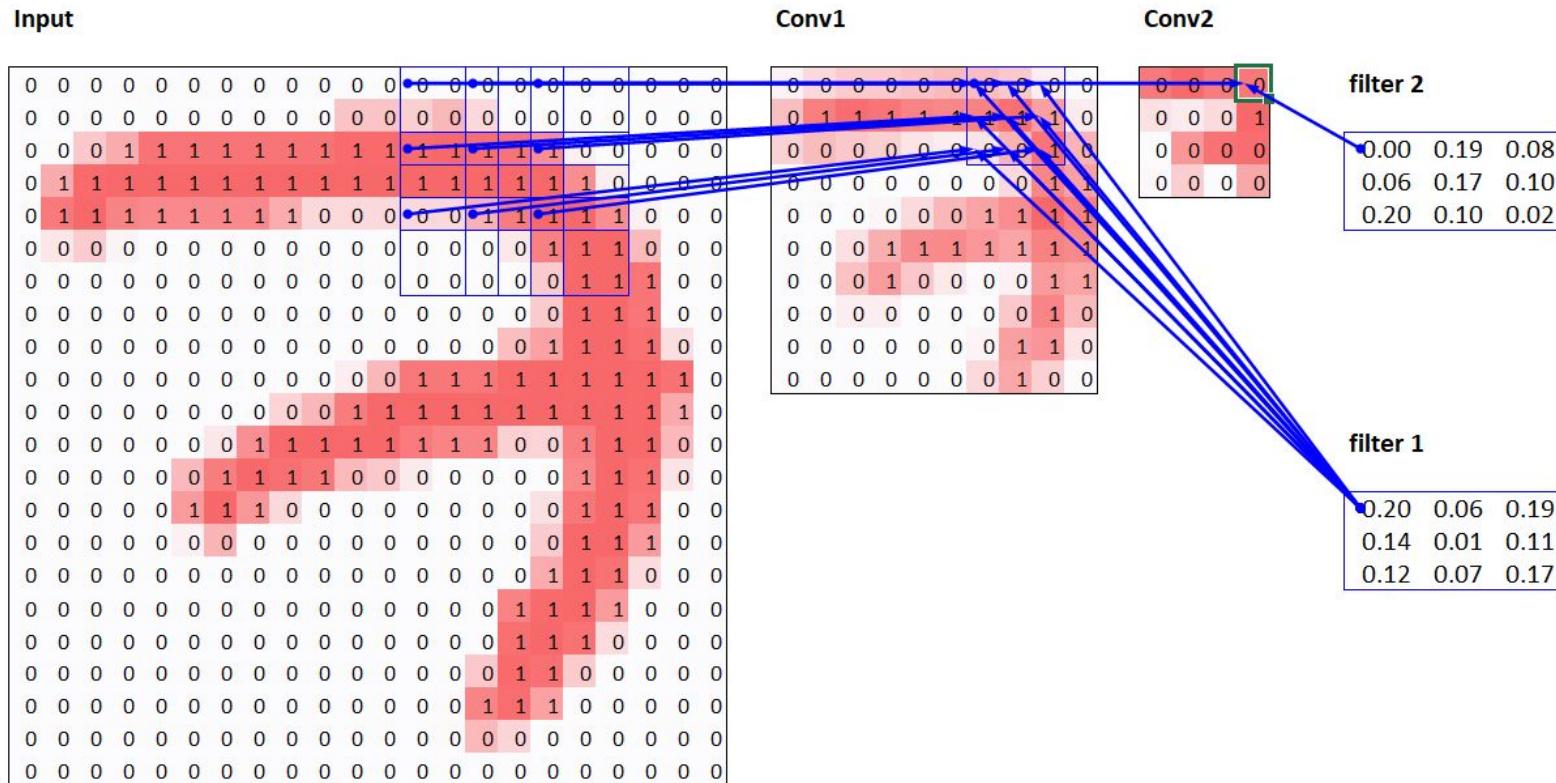


input image

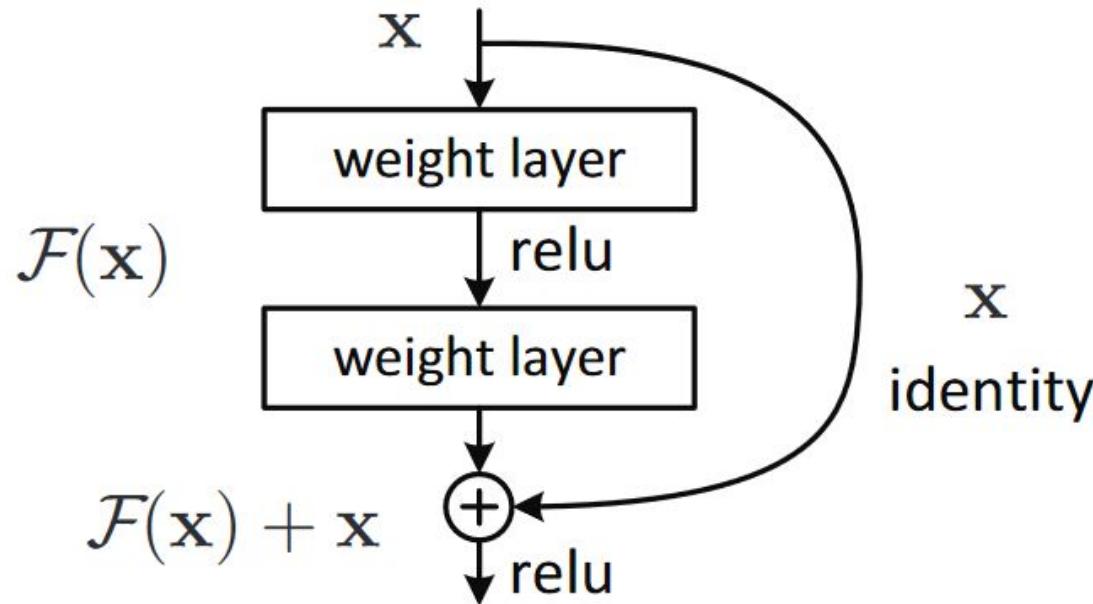


Layers in a CNN.

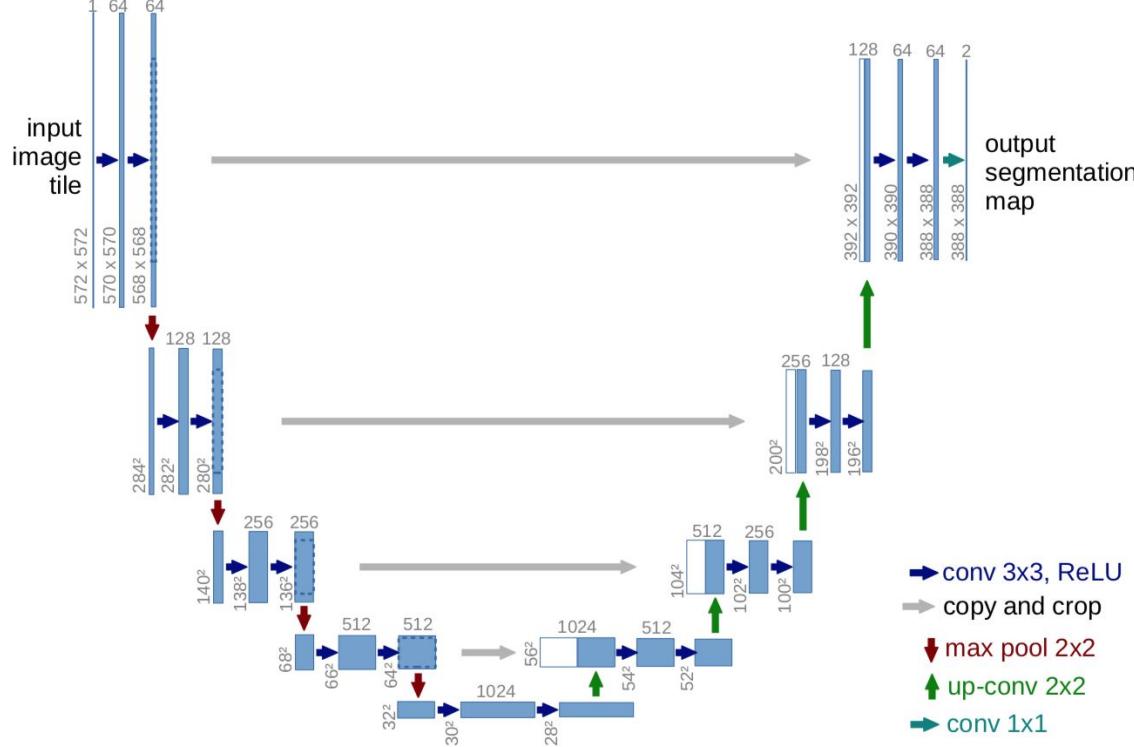
# Convolution kernels (aka filters)

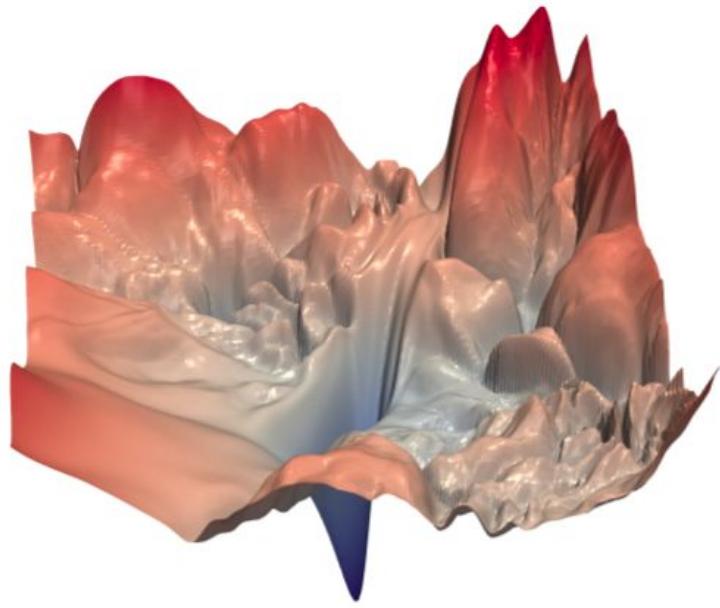


# Skip connections

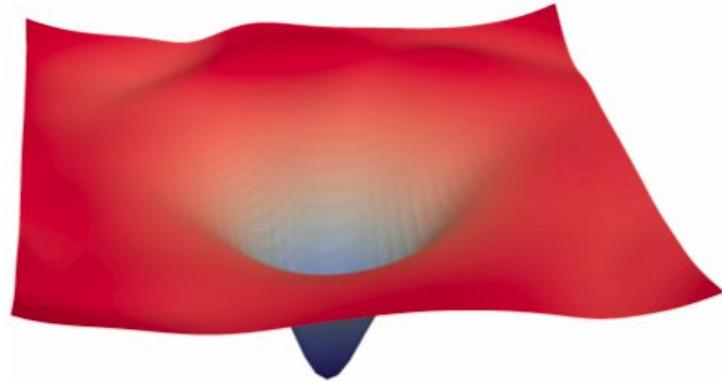


# Skip connections





(a) without skip connections

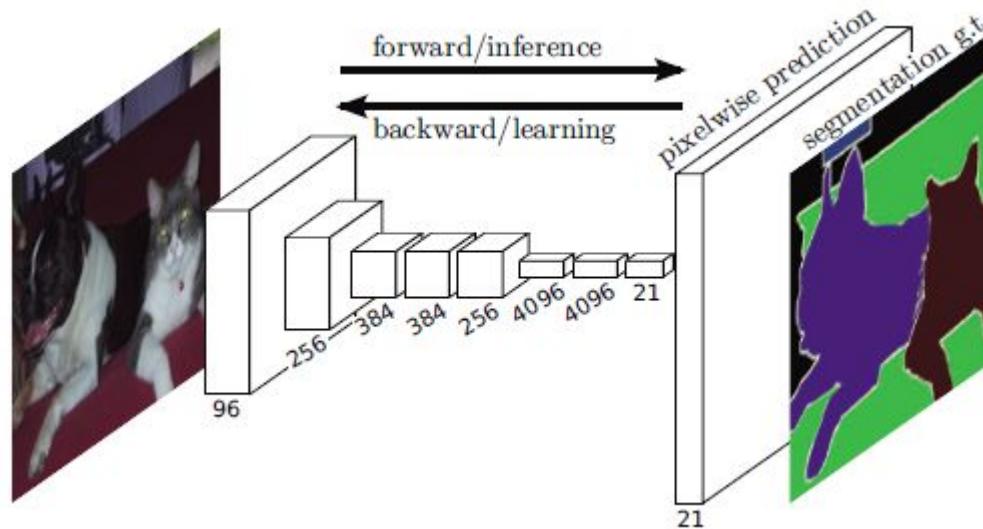


(b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

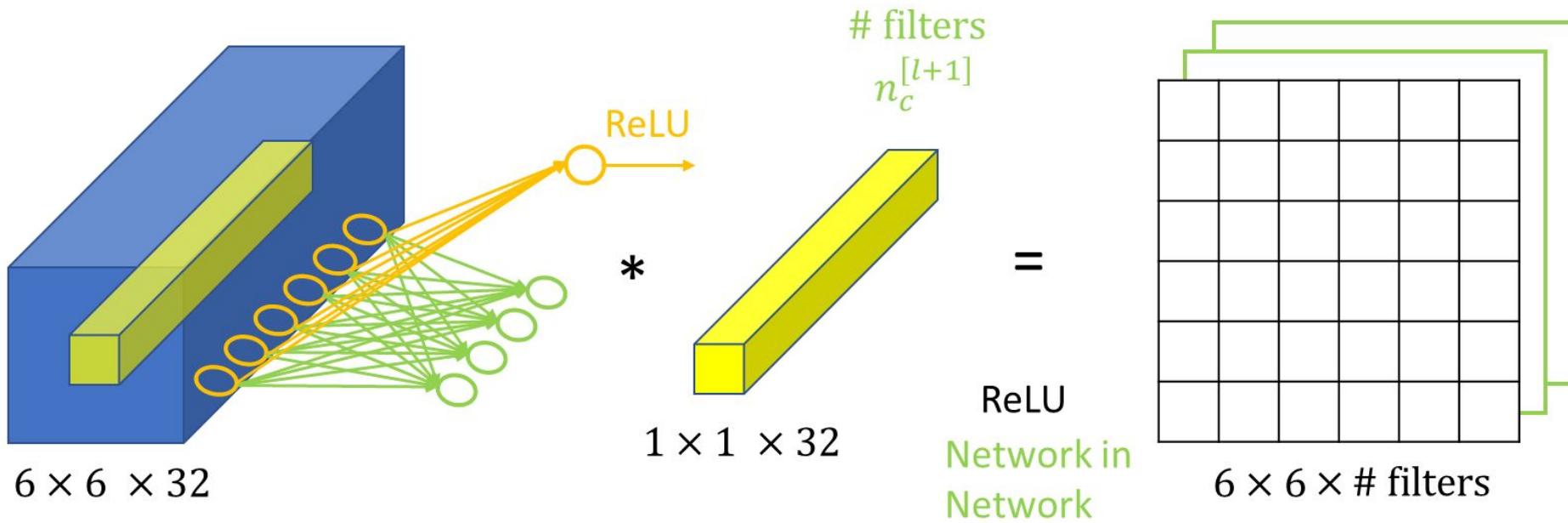
[Visualizing the loss landscape of neural nets](#)

# “Fully Convolutional Networks” allow pixelwise prediction



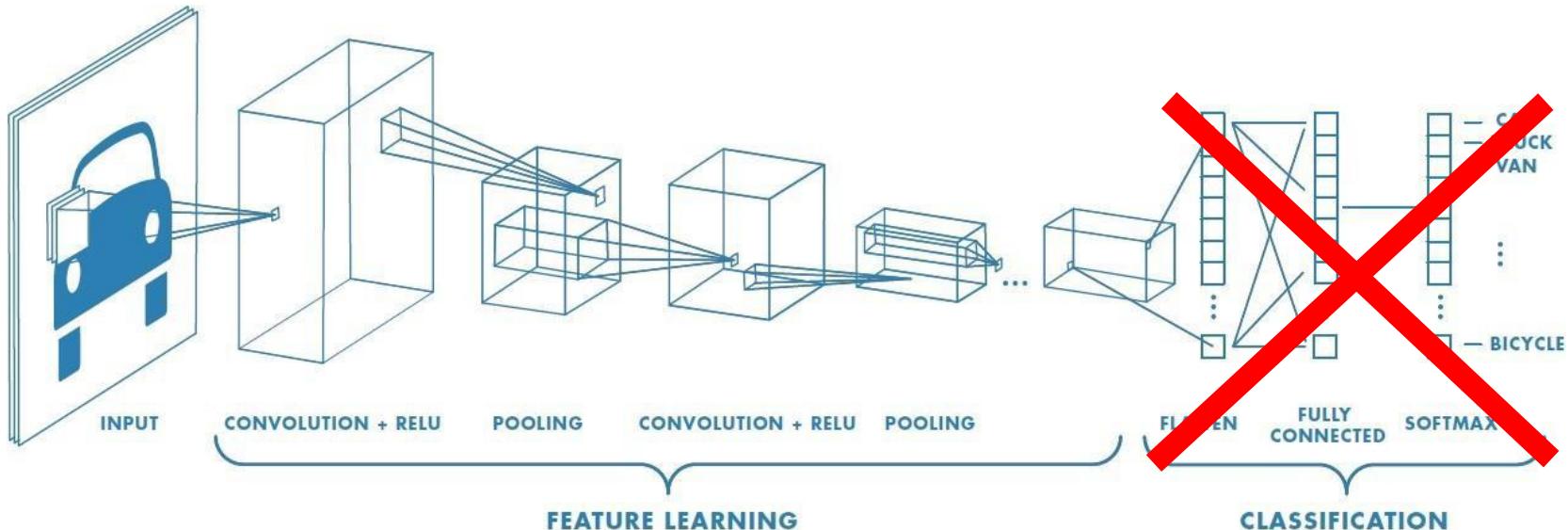
All layers in the network are convolutional, there is no fully connected (aka “dense”) layer like in most classifiers, we use the local info of the pixel neighborhood

# “1x1” convolutions



- [A Gentle Introduction to  \$1 \times 1\$  Convolutions](#)
- [Tutorial on  \$1 \times 1\$  Convolutions by Andrew Ng](#)

# “Fully Convolutional” networks localize pixels



All layers in the network are convolutional, there is no fully connected (aka “dense”) layer like in most classifiers, we use the local info of the pixel neighborhood

# SSD as a fully convolutional networks

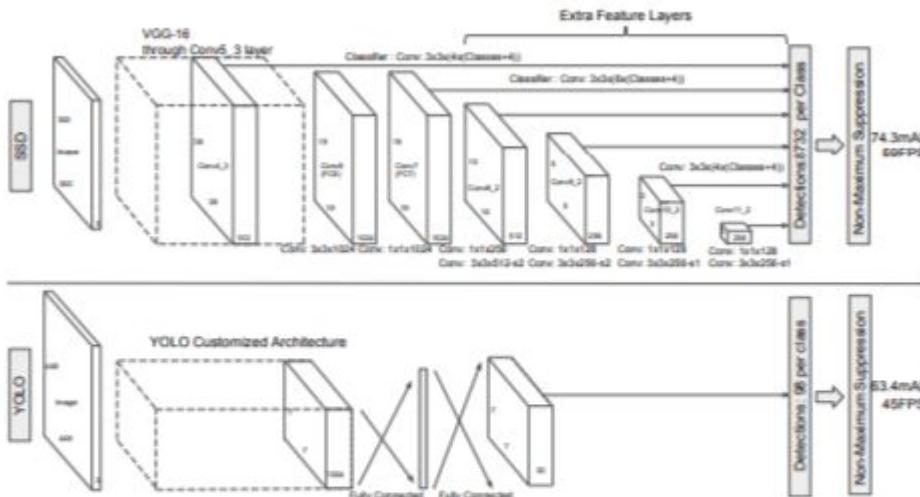


Fig. 2: A comparison between two single shot detection models: SSD and YOLO [5]. Our SSD model adds several feature layers to the end of a base network, which predict the offsets to default boxes of different scales and aspect ratios and their associated confidences. SSD with a  $300 \times 300$  input size significantly outperforms its  $448 \times 448$  YOLO counterpart in accuracy on VOC2007 test while also improving the speed.

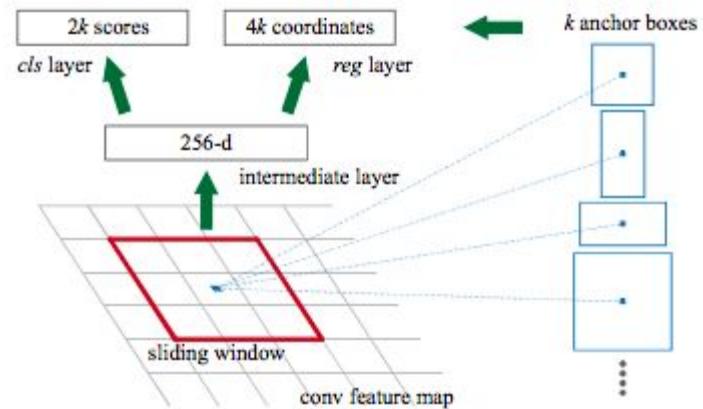
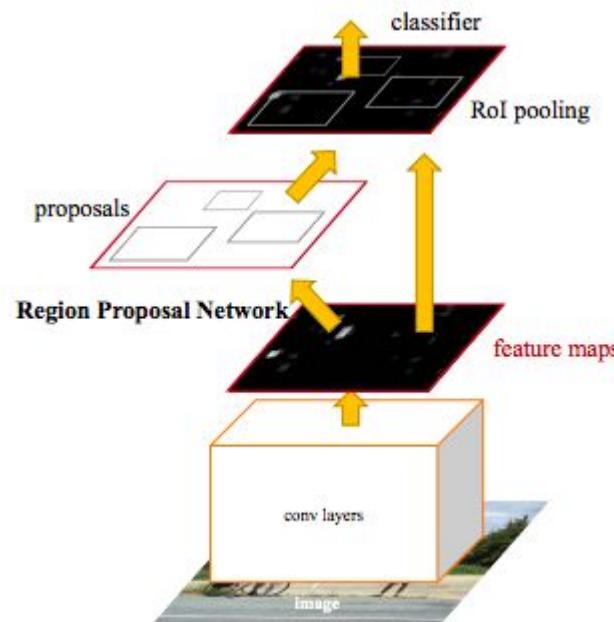
# Two stage detectors: R-CNN variants

- **R-CNN**: Selective Search is run on the image, output segments from Selective Search are used for feature extraction and classification using a pre-trained CNN (two networks, separate training)
- **Fast R-CNN**: Uses the Selective Search algorithm to obtain region proposals, adding the Region of Interest (ROI) Pooling module. Extracts a fixed-size window from the feature map and uses the features to obtain the class label and bounding box. The network is now *end-to-end trainable*
- **Faster R-CNN**: Introduces the Regional Proposal Network (RPN) that puts the region proposal *directly* into the architecture, alleviating the need for the Selective Search algorithm. Produces better results at lower training and inference time.
- **Mask R-CNN**: adds a convolution mask filter to the Faster R-CNN architecture to generate instance segmentation masks

# R-CNN

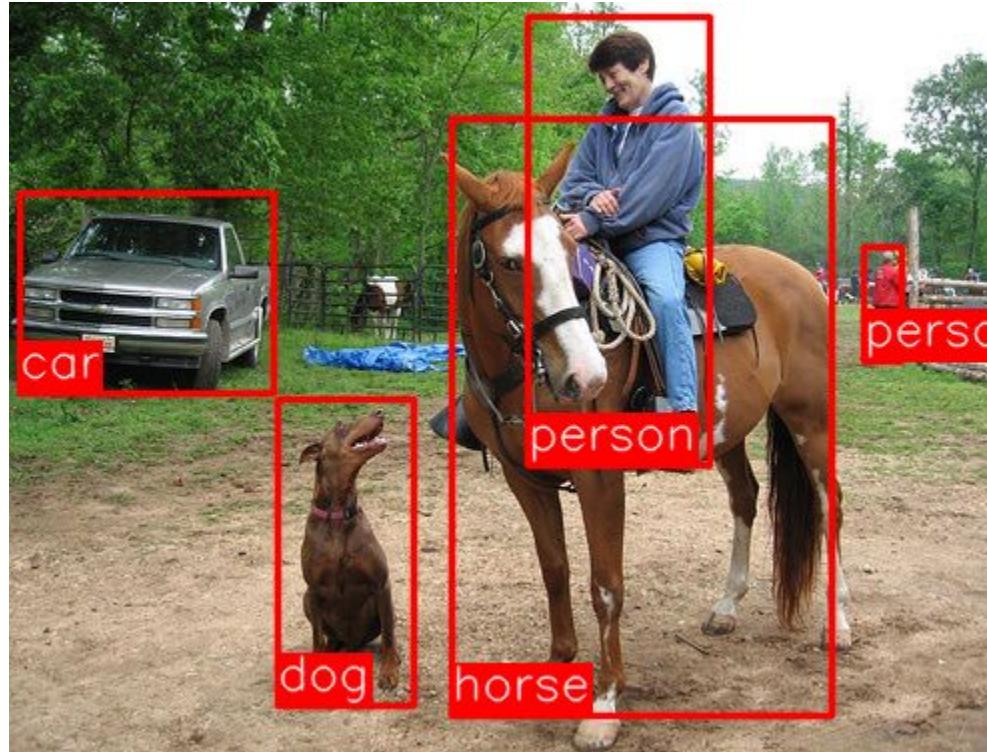
- **Step #1:** Input an image and segment it into blobs
- **Step #2:** Extract regions proposals using the Selective Search algorithm
- **Step #3:** Select the regions of interest with higher object activations using a pretrained network, warp them to standard size (e.g. 128x128) and output proposals of objects
- **Step #3** Classify each proposal using the extracted features with a Support Vector Machine

# Region Proposal Networks - Faster R-CNN



[Faster R-CNN breakdown](#)

Methods with RPNs “**usually**” handle bbox variance better



# RetinaNet was the first SSD to beat a Region Proposal Network

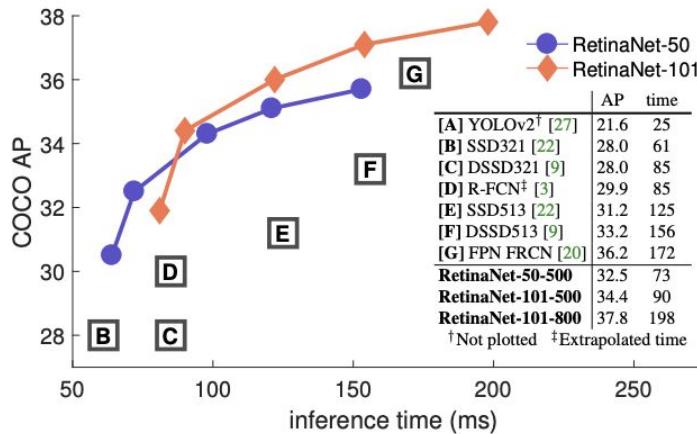


Figure 2. Speed (ms) versus accuracy (AP) on COCO test-dev. Enabled by the focal loss, our simple one-stage *RetinaNet* detector outperforms all previous one-stage and two-stage detectors, including the best reported Faster R-CNN [28] system from [20]. We show variants of RetinaNet with ResNet-50-FPN (blue circles) and ResNet-101-FPN (orange diamonds) at five scales (400-800 pixels). Ignoring the low-accuracy regime ( $AP < 25$ ), RetinaNet forms an upper envelope of all current detectors, and an improved variant (not shown) achieves 40.8 AP. Details are given in §5.

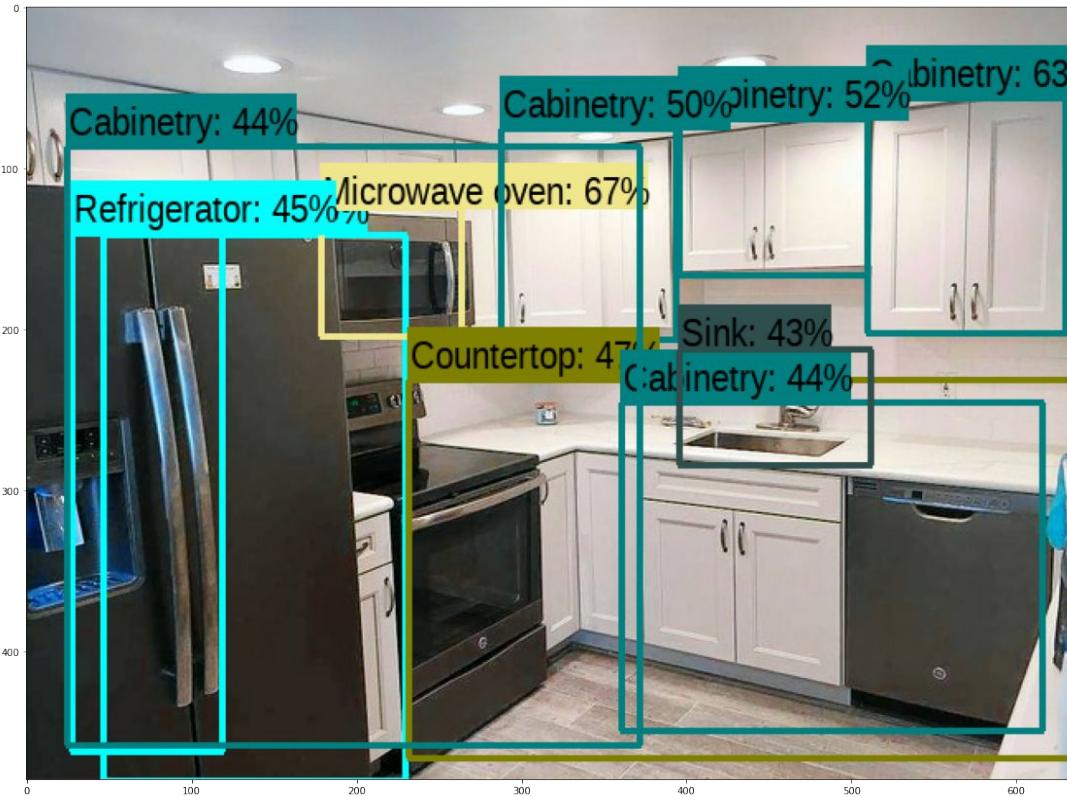
# Input Image



# Mask R-CNN pretrained on COCO (inventory image)



# SSD pretrained on Google Open Images



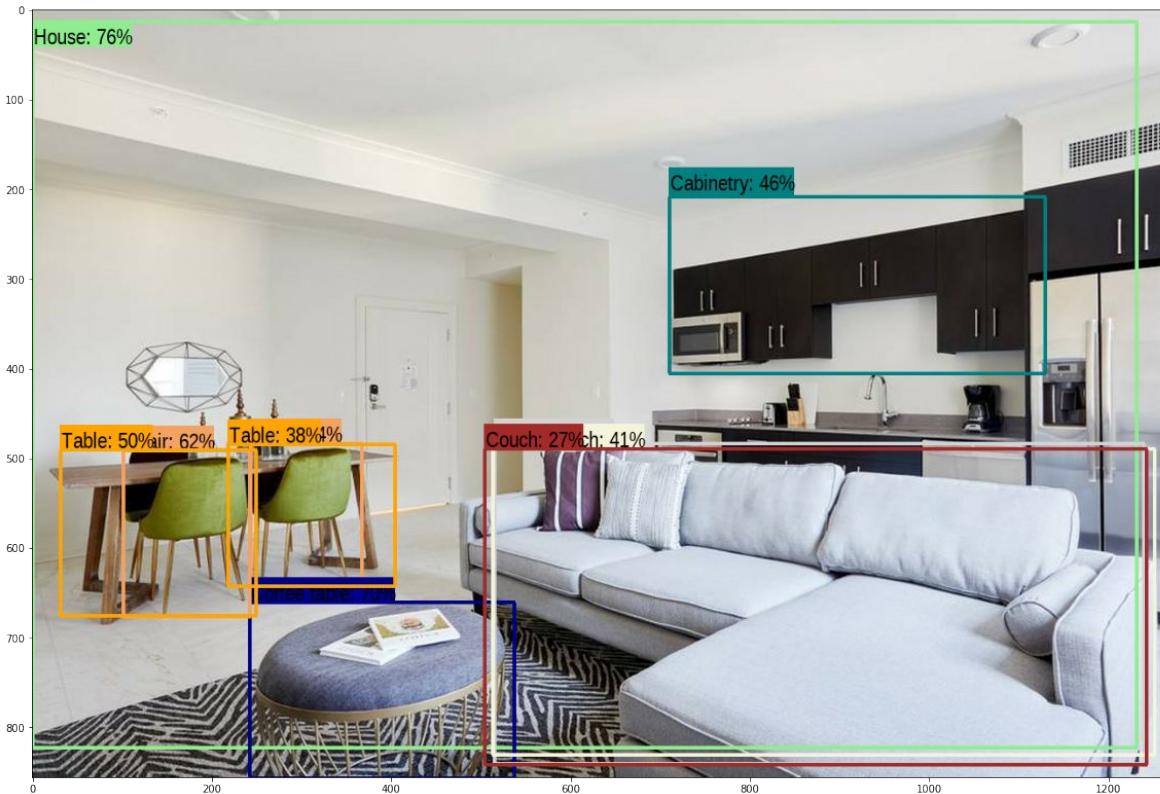
# Input image



# Mask R-CNN trained on COCO



# SSD trained on Open Images Dataset

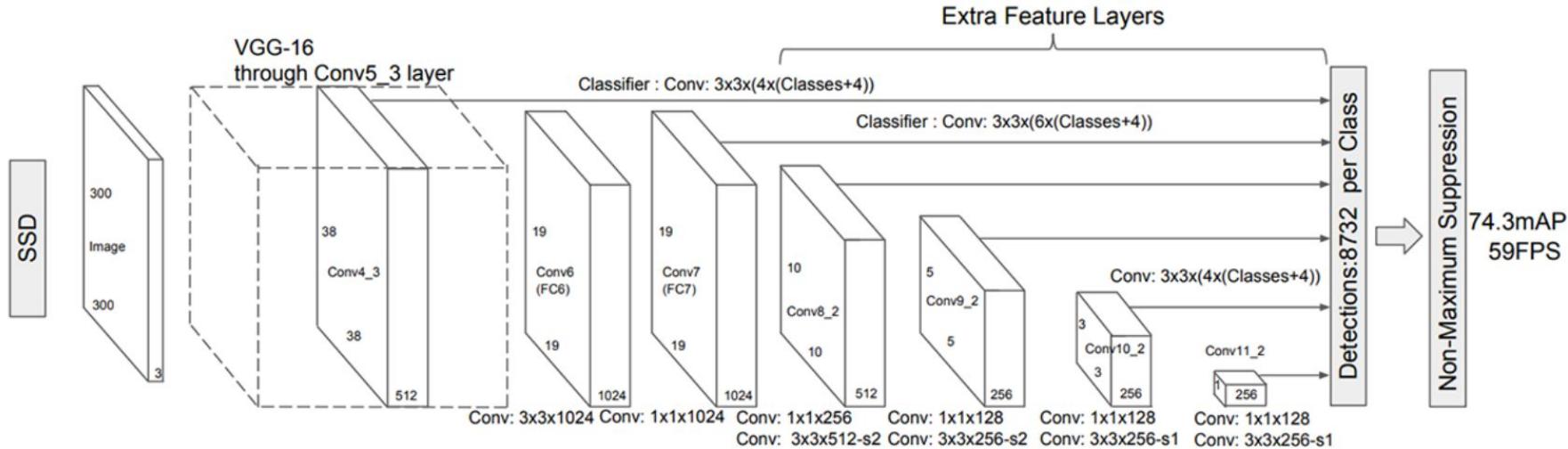


# Tricks to improve accuracy on small objects

- Split the image into tiles (good for high-res images)
- Use focal-loss
- Fine-tune the amount and size of anchor boxes
  - These tricks apply to both two-stage and single stage detectors

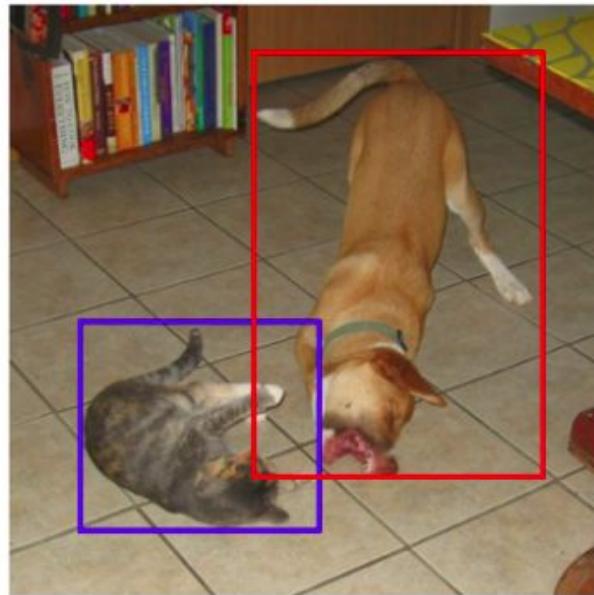
Small objects detection problem

# Single Shot Detectors

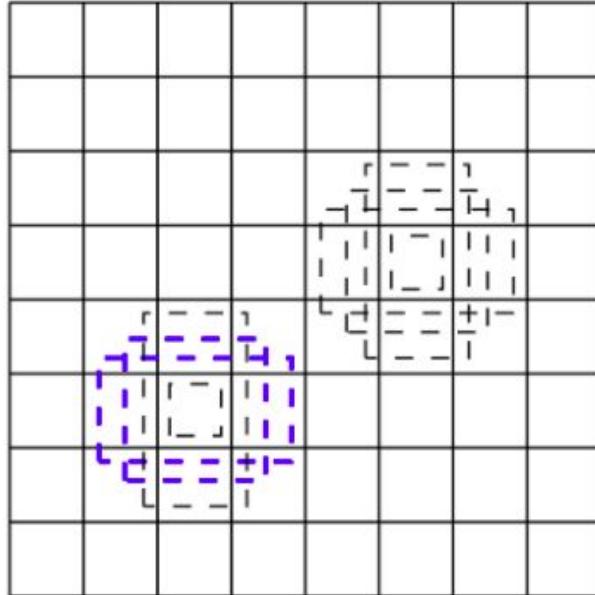


## Single Shot Multibox Detection

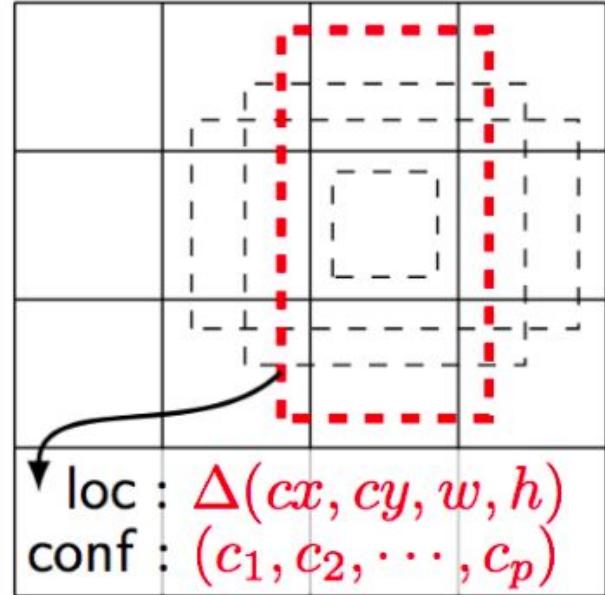
Single shot detectors make predictions of feature maps at different scales (different sizes for the bounding box) and compute cross entropy on a fully convolutional network that takes all these activation scales as input



(a) Image with GT boxes



(b)  $8 \times 8$  feature map

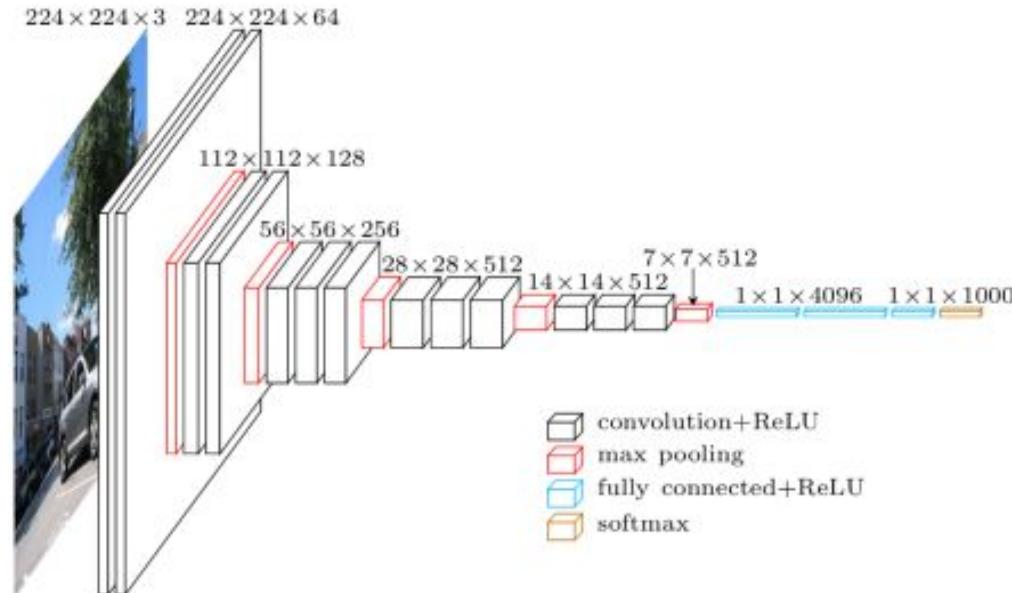


loc :  $\Delta(cx, cy, w, h)$   
conf :  $(c_1, c_2, \dots, c_p)$

(c)  $4 \times 4$  feature map

SSD default boxes at  $8 \times 8$  and  $4 \times 4$  feature maps

# The VGG architecture



[VGG, Resnet, and Inception in Keras](#)

# The speed vs mean average precision (mAP) tradeoff

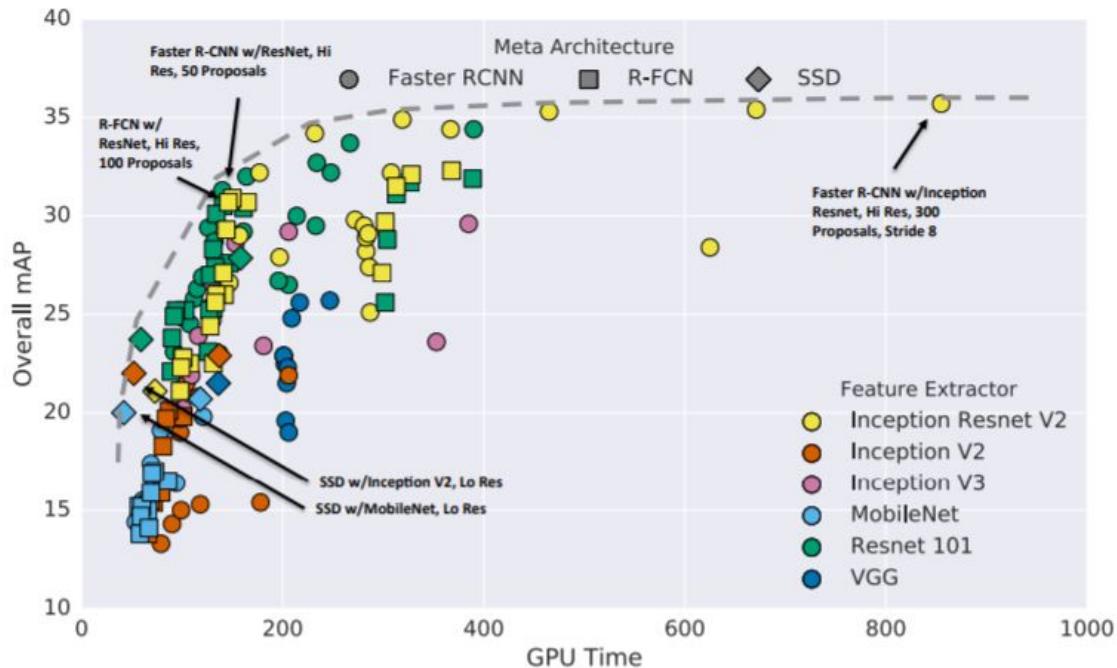


Figure 2: Accuracy vs time, with marker shapes indicating meta-architecture and colors indicating feature extractor. Each (meta-architecture, feature extractor) pair can correspond to multiple points on this plot due to changing input sizes, stride, etc.

Source: [Speed vs Accuracy Tradeoffs for Modern Convolutional Neural Networks, by Google Brain](#)

# For quick experiments use an API

## Installing the TensorFlow Object Detection API

When putting this book together I evaluated *many* deep learning-based object detection implementations, including pure Keras-based libraries, mxnet-based packages, Caffe implementations, and even Torch libraries.

Object detection is not only much harder to *train* a network on, but significantly more challenging to *implement* as well, as there are many more components, some of which require custom layers and loss functions. After reading Chapter 14 on the fundamentals of Faster R-CNNs, it should be clear there are many modules that would need to be implemented by hand.

Implementing the entire Faster R-CNN architecture is not something that can be covered in this book, for a number of reasons, including:

1. Implementing and explaining the Faster R-CNN architecture by hand using the same style used throughout the rest of the book (code blocks and detailed explanations) would take hundreds (if not more) of pages.
2. Object detection libraries and packages tend to be fragile in their nature as custom layers and loss methods are used. When a new version of their associated backend library is released, the risk of breaking such a custom module is high.

# Detectron2



# Detectron2

[detectron2/MODEL\\_ZOO.md at master · facebookresearch/detectron2 · GitHub](#)

[Inference with different object detection models](#)

# Detectron2 config files

master [detectron2 / configs /](#) [Go to file](#) [Add file ▾](#)

ppwwyyxx and **facebook-github-bot** Move PointRend logic to ... [...](#) 4 days ago History

..

	COCO-Detection	Initial commit	13 months ago
	COCO-InstanceSegment...	Configurable loss for rpn box regression and giou sup...	5 months ago
	COCO-Keypoints	Initial commit	13 months ago
	COCO-PanopticSegment...	set FILTER_EMPTY_ANNOTATIONS to False for pan...	5 months ago

# Check Detectron models before building your own

RetinaNet:

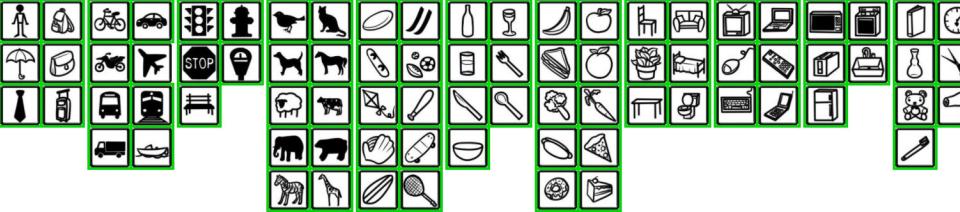
Name	lr sched	train time (s/iter)	inference time (s/im)	train mem (GB)	box AP	model id	download
R50	1x	0.205	0.041	4.1	37.4	190397773	<a href="#">model</a>   <a href="#">metrics</a>
R50	3x	0.205	0.041	4.1	38.7	190397829	<a href="#">model</a>   <a href="#">metrics</a>
R101	3x	0.291	0.054	5.2	40.4	190397697	<a href="#">model</a>   <a href="#">metrics</a>

RPN & Fast R-CNN:

Name	lr sched	train time (s/iter)	inference time (s/im)	train mem (GB)	box AP	prop. AR	model id	download
RPN R50-C4	1x	0.130	0.034	1.5		51.6	137258005	<a href="#">model</a>   <a href="#">metrics</a>
RPN R50-FPN	1x	0.186	0.032	2.7		58.0	137258492	<a href="#">model</a>   <a href="#">metrics</a>
Fast R-CNN R50-FPN	1x	0.140	0.029	2.6	37.8		137635226	<a href="#">model</a>   <a href="#">metrics</a>

# The COCO dataset

COCO 2017 train/val browser (123,287 images, 886,284 instances). Crowd labels not shown.



toothbrush x teddy bear x vase x book x clock x scissors x hair drier x  
person x tie x umbrella x backpack x handbag x suitcase x bicycle x  
motorcycle x bus x truck x car x train x boat x traffic light x stop sign x  
bench x fire hydrant x parking meter x bird x dog x sheep x elephant x  
zebra x cat x horse x cow x bear x giraffe x frisbee x snowboard x kite x  
baseball glove x surfboard x sports ball x skis x baseball bat x skateboard x  
tennis racket x bottle x cup x knife x bowl x banana x wine glass x fork x  
spoon x sandwich x broccoli x hot dog x donut x apple x orange x carrot x  
pizza x cake x chair x potted plant x dining table x couch x bed x toilet x  
tv x mouse x keyboard x refrigerator x remote x laptop x microwave x  
toaster x cell phone x sink x oven x

<http://cocodataset.org/#explore>

Subset ▾ Type: Detection ▾

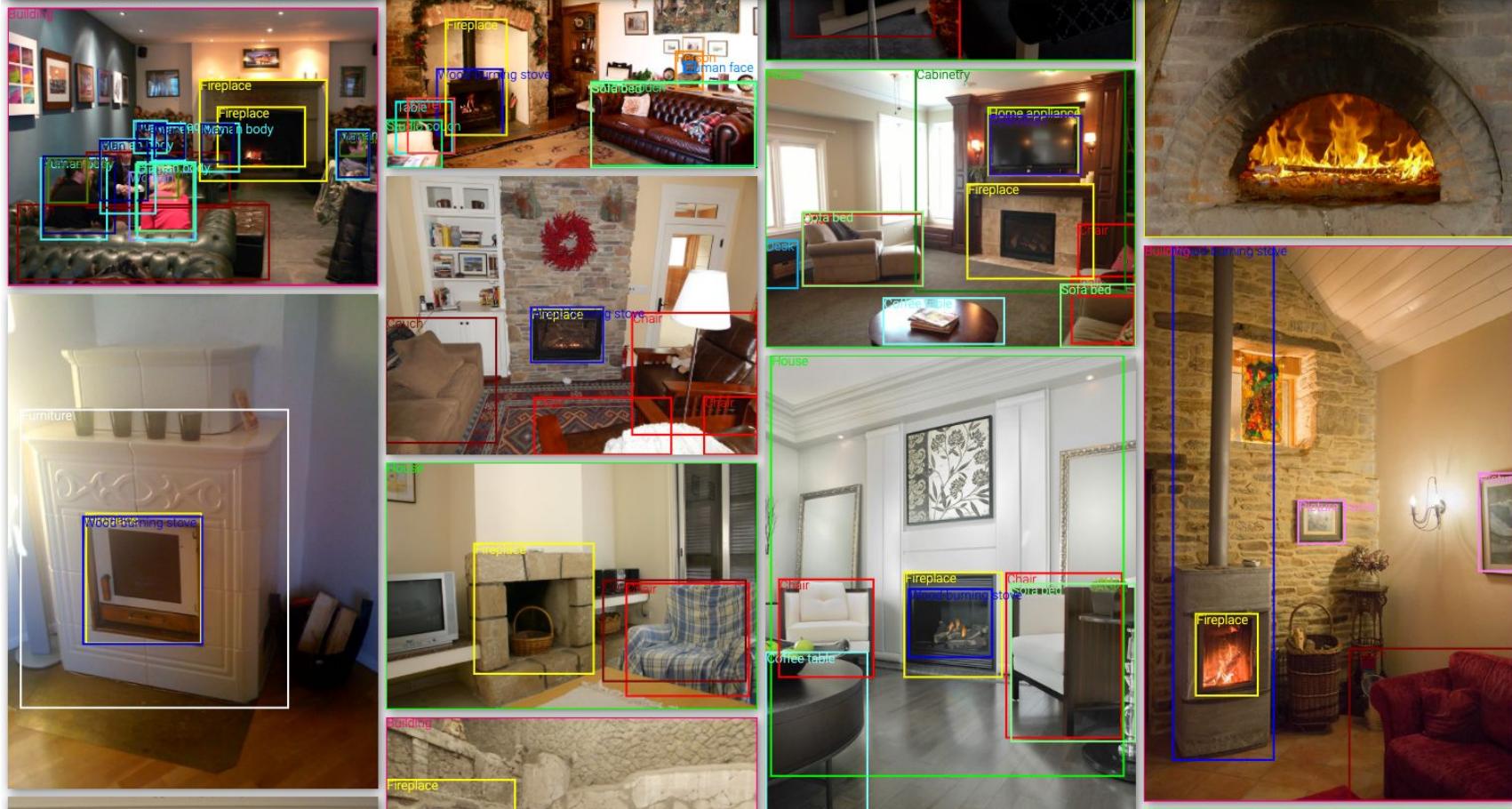
Type: Detection ▾

**Category:**

Fireplace|

Random category

Options ▾



## Case study

# Amenity Detection and Beyond — New Frontiers of Computer Vision at Airbnb

Build highly customized AI technologies into home-sharing products and help our guests belong anywhere.



Shijing Yao

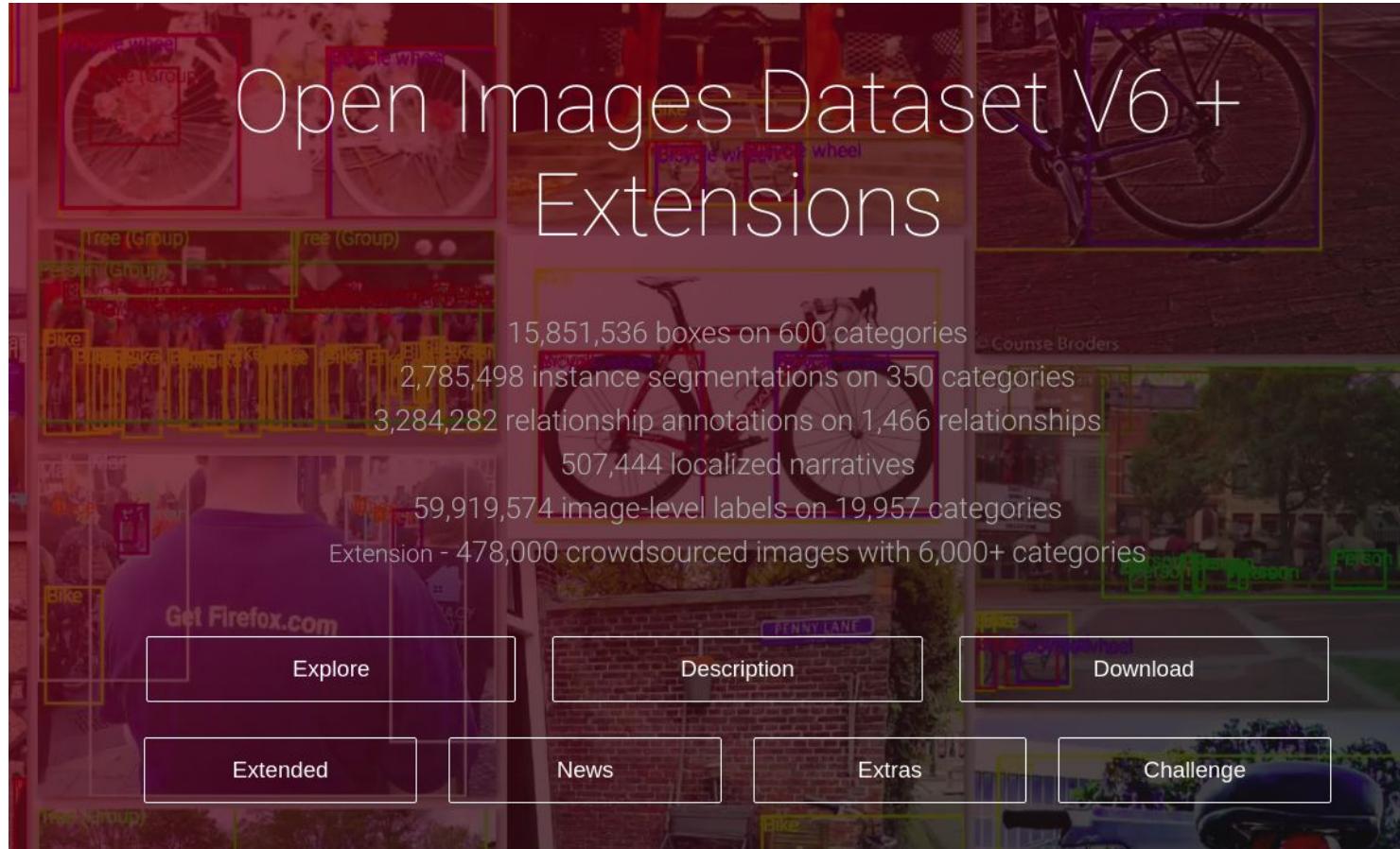
Follow

Jul 16, 2019 · 15 min read ★



**Authors:** *Shijing Yao, Dapeng Li, Shawn Chen*





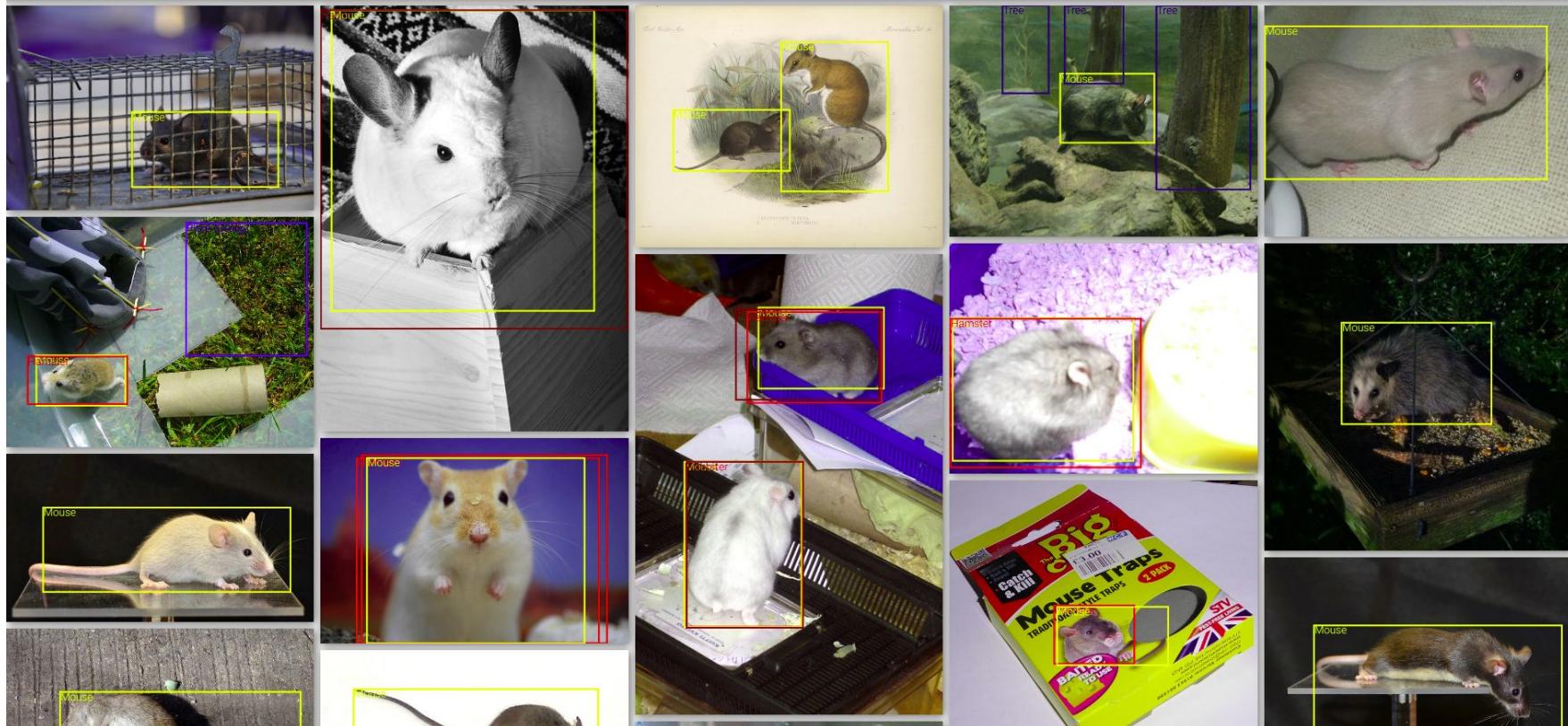
<https://storage.googleapis.com/openimages/web/index.html>

Subset ▾ Type: Detection ▾

Category: Mouse

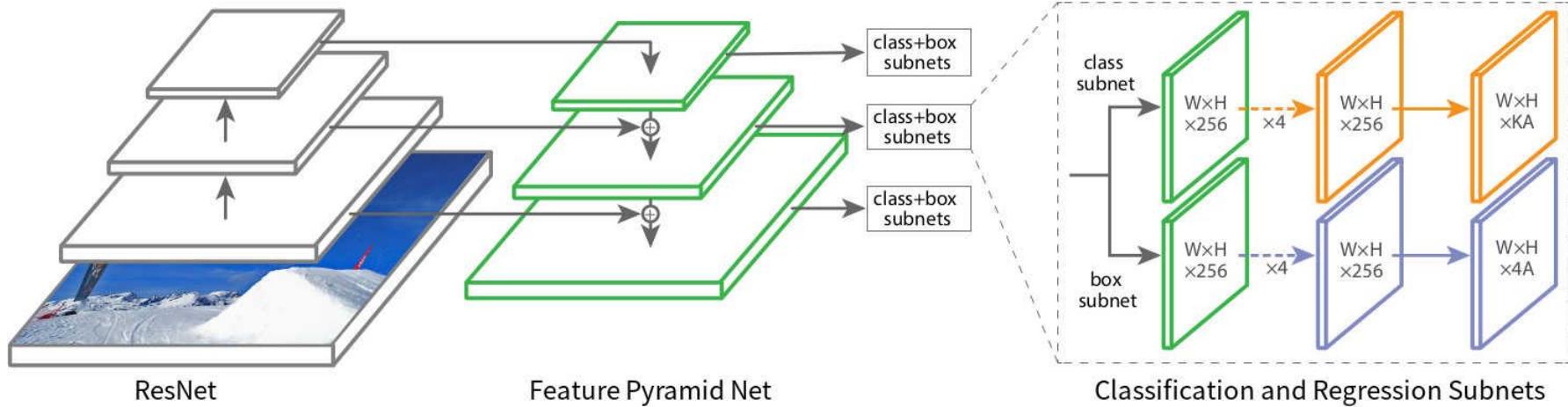
Random category

Options ▾



[Exploring object detection in the Google Open Images Dataset](#)

# RetinaNet's Feature Pyramid Network



# Building your own RetinaNet in Keras

- <https://keras.io/examples/vision/retinanet/#object-detection-with-retinanet>
- [Try it in Google Colab](#)

# Suggested code exercises

- Compute AP and IoU for faces in the dataset
  - Report these metrics [during the training loop](#)
- Experiment with different sizes of anchor boxes
- Change the network architecture, try using a U-net style or Original SSD style architecture
- Experiment with data augmentation using [albumentations](#)
- Retrain with/without [transfer learning](#) in the convolutional base
- Implement the Resnet block with element-wise addition instead of concatenation
- Add [non-max suppression](#)
- Try a RetinaNet, Faster R-CNN, or Mask R-CNN in the same dataset
  - The [detectron2](#) library will allow you to do these experiments quickly
- Use [focal-loss](#) instead of cross entropy for the bounding boxes
- Train the network using the [one-cycle policy](#)
- Fine tune the training with [hard-negative mining](#)

# Review questions

- If you were to choose a performance metric to optimize ‘tightness of bounding boxes’ would you choose mAP, AP, or IoU?
- Suppose that your object detector for pedestrians detects the 5 pedestrians that are present in an image but generates 995 boxes labeled as human that shouldn’t be there. Does this object detector have perfect recall? What is its precision?
- When choosing between two-stage and single stage detectors, can we know for certain that a given model will outperform the other in terms of accuracy? How about in terms of speed? (assume no pruning, distillation, or quantization)

# Review questions

- Do single shot detectors have region proposal networks?
- Does RetinaNet use the selective search algorithm?
- How does focal loss differ from cross entropy?
- Why is object detection both a regression and classification problem?
- What are the two subnetworks in RetinaNet?
- What is an anchor box?
- What is the speed/accuracy tradeoff related to the number and shape of anchor boxes?
- What is a feature pyramid network?
- Does RetinaNet have fully connected layers?
- What does it mean for a network to be ‘fully convolutional’?
- How do residual blocks help us tackle the vanishing gradient problem?