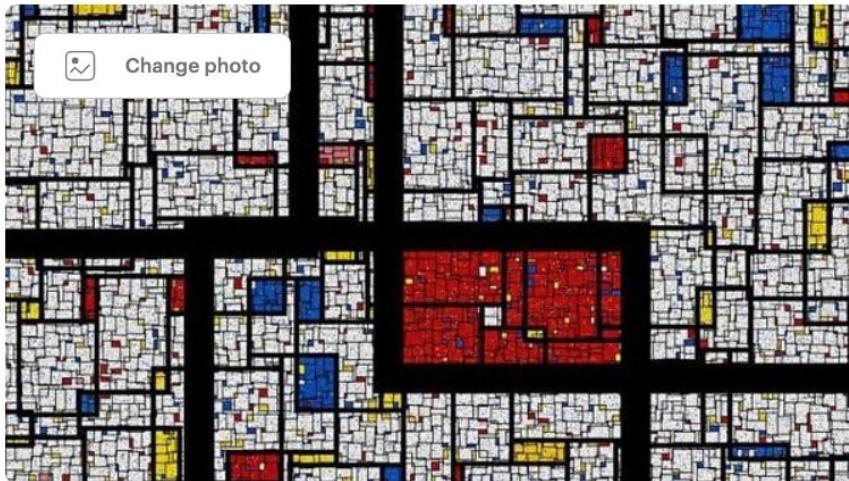


Object Detection and RetinaNet

Antonio Rueda-Toicen
Data Science Retreat
February 2022

About me

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



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/>

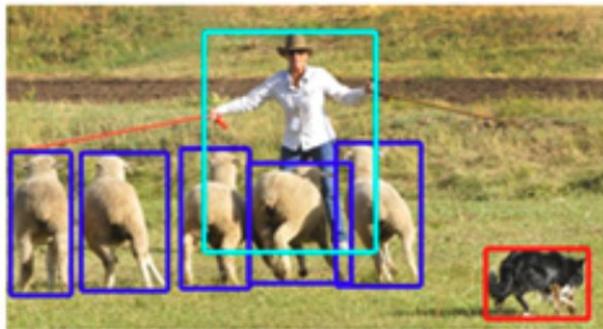
About you

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

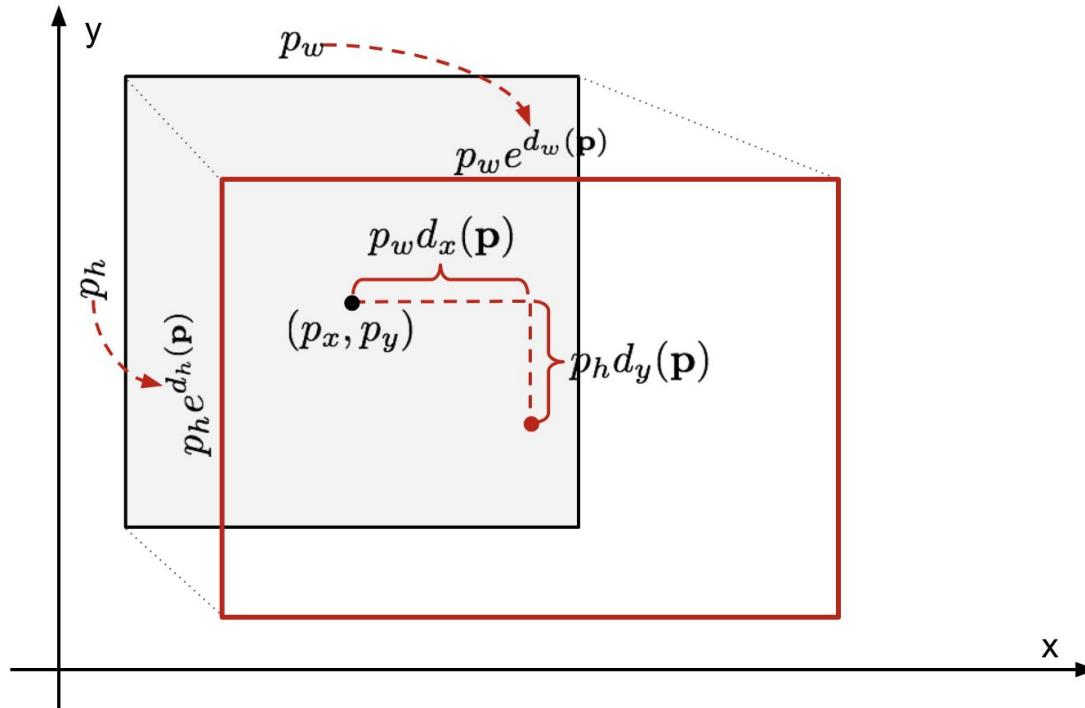
Segmentation refers to pixel-wise localization of the labels

Goals of object detection

Given an input image we wish to obtain:

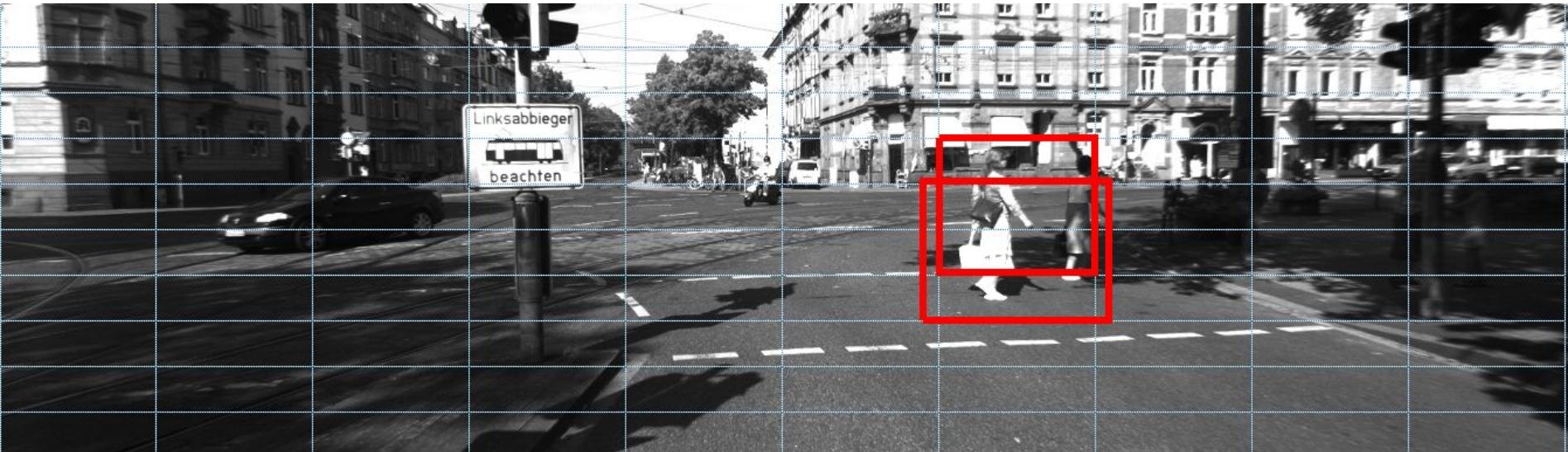
1. A **list of bounding boxes**, or the (x, y) -coordinates for each object in an image
2. A **class label** associated with each bounding box
3. The **probability** score associated with each bounding box and class label

Object detection as bounding box regression (1st step)

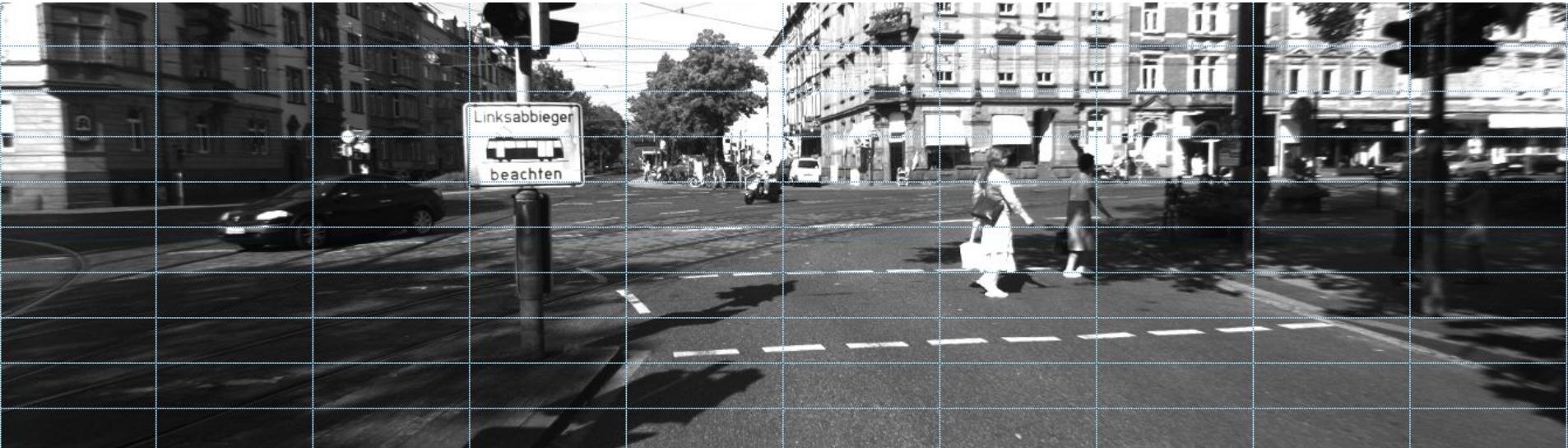


‘Regression’ means predicting a continuous value

Breaking down an image into regions



Breaking down an image into regions

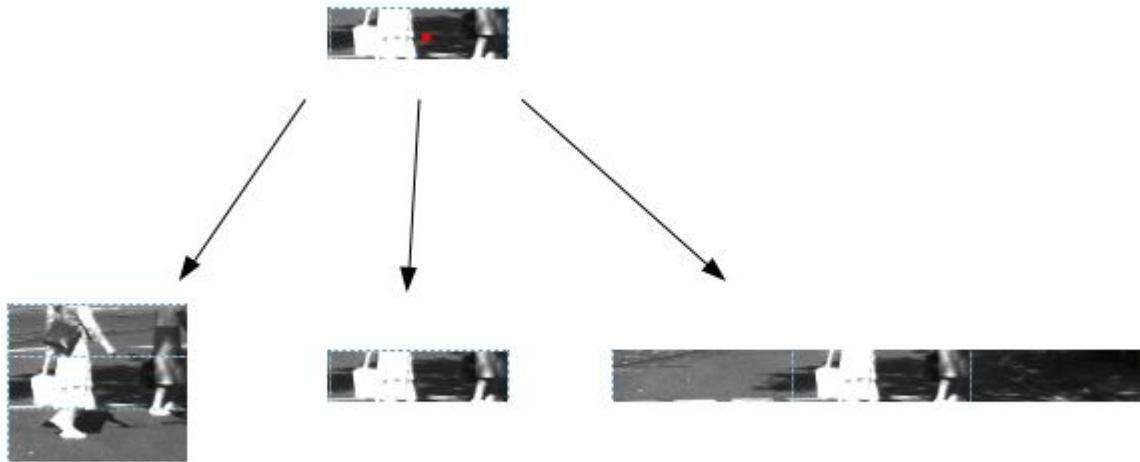


A sliding window approach partitions the image into discrete blocks

Breaking down an image into regions



Breaking down an image into regions



Finding the optimal image partitioning algorithm is not a trivial task

Sliding windows

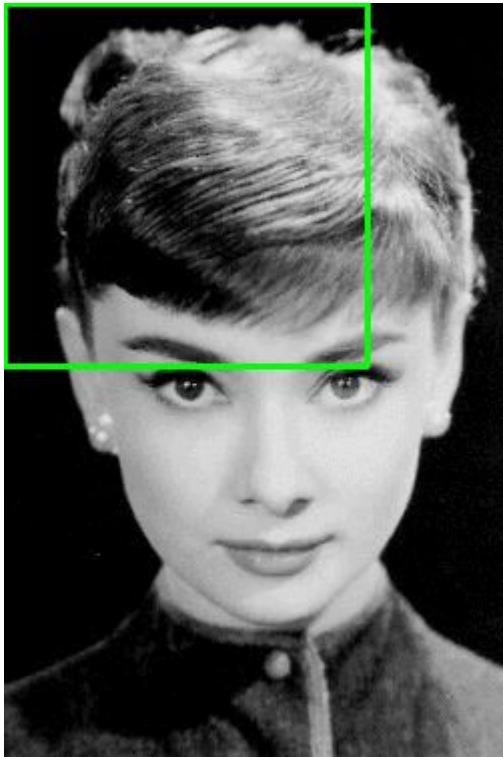
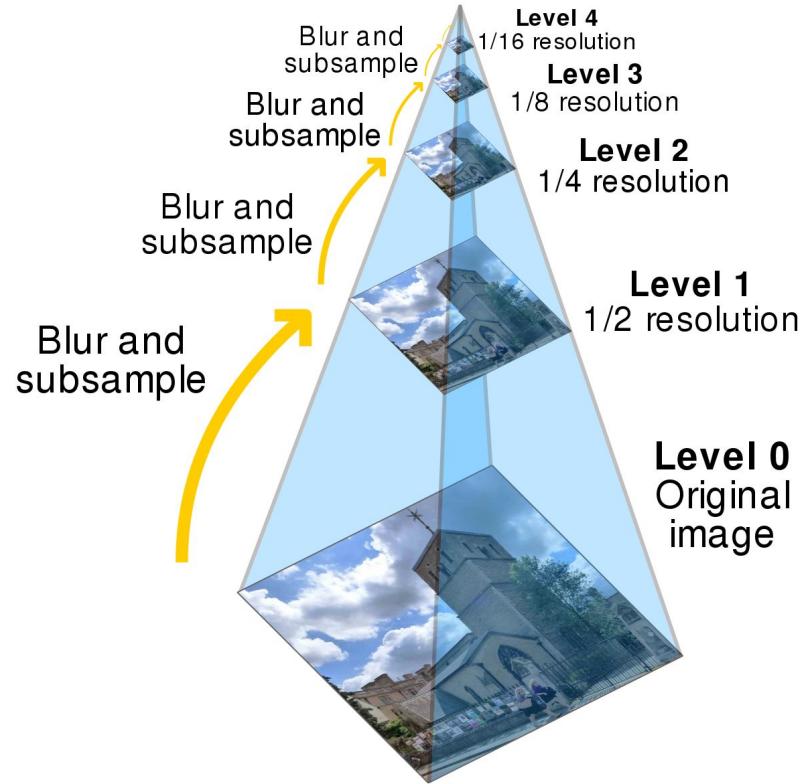


Image pyramids



Sliding windows on an image pyramid



Image Pyramids

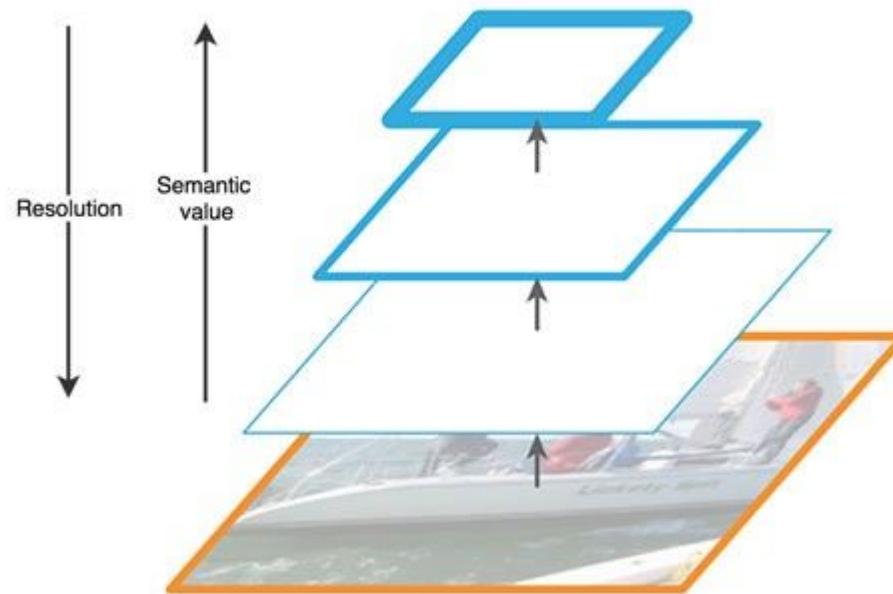
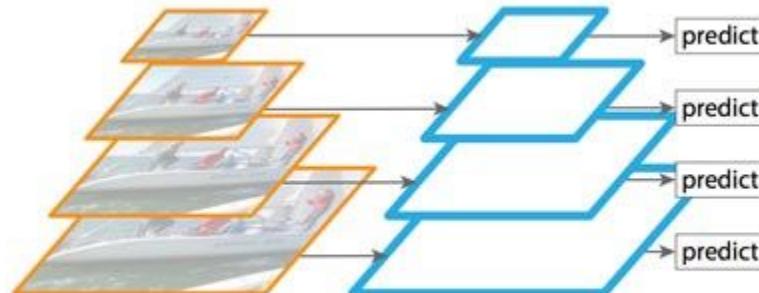
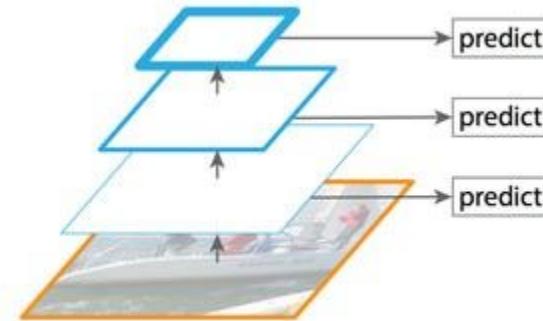


Image Pyramids

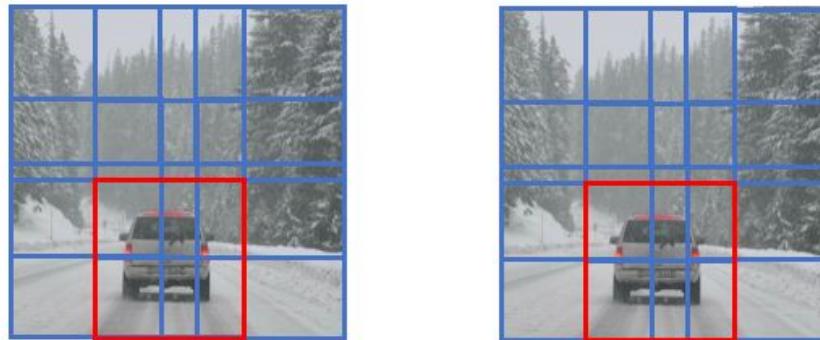
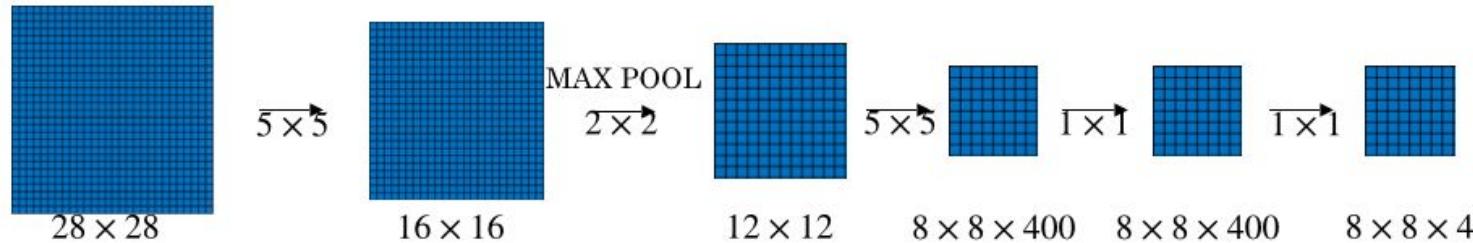


Pyramid of images



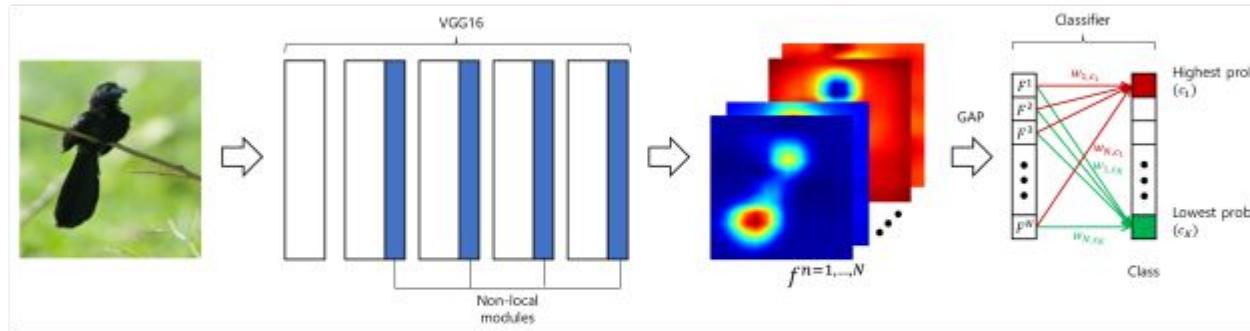
Pyramid of feature maps

Pyramids and sliding windows in CNNs



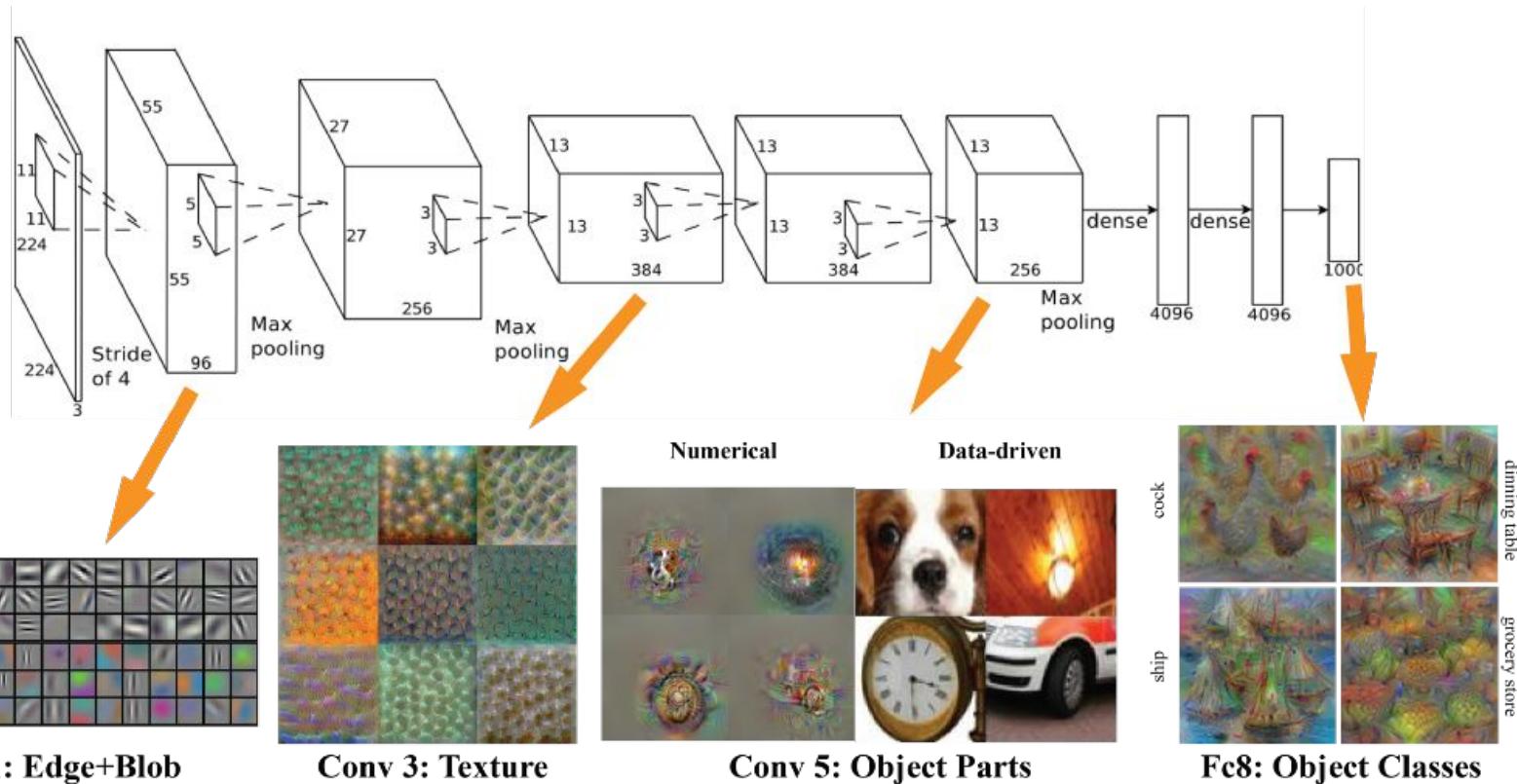
Andrew Ng

Transfer learning of classification is currently used for feature extraction

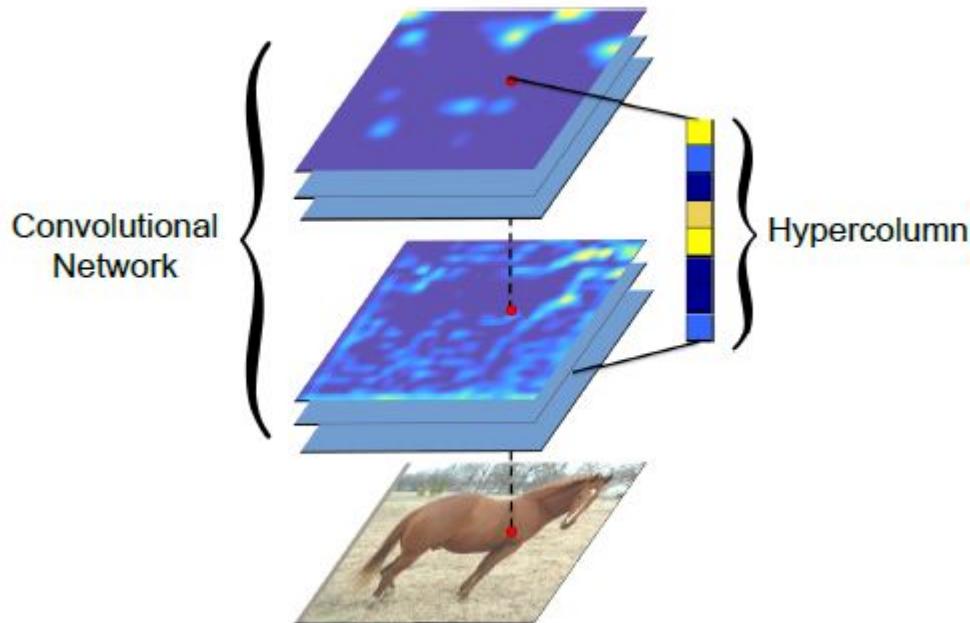


[Source](#)

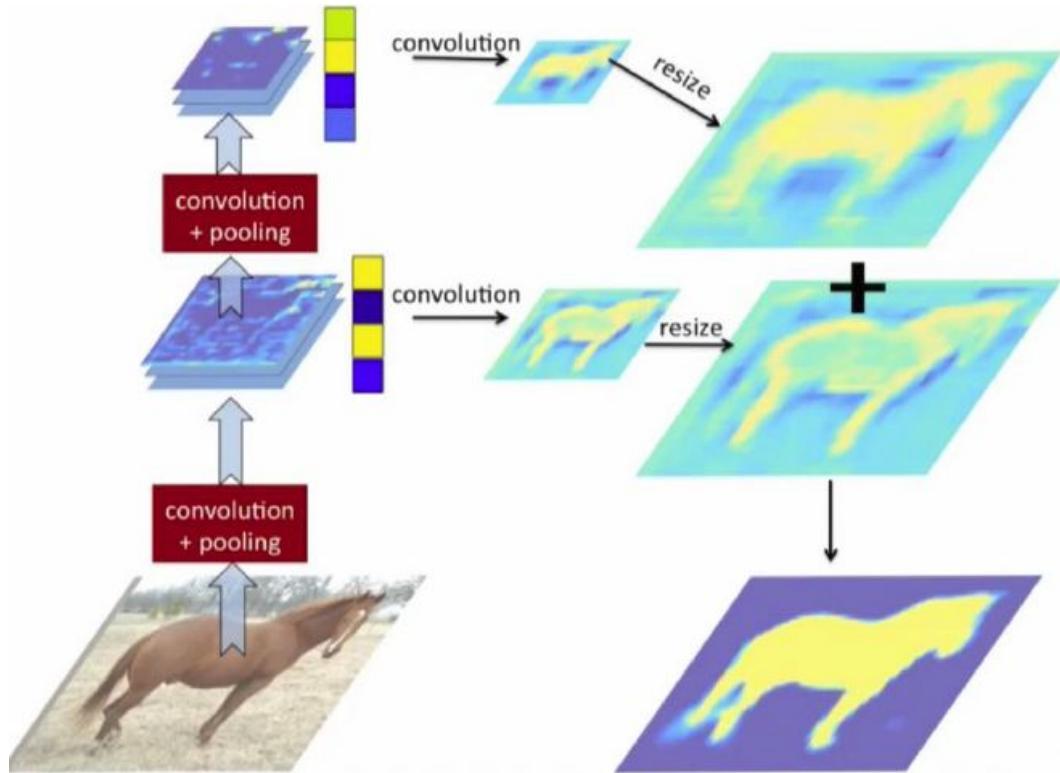
What convolutional networks learn



“Hypercolumns”



“Hypercolumns”



Anchor boxes

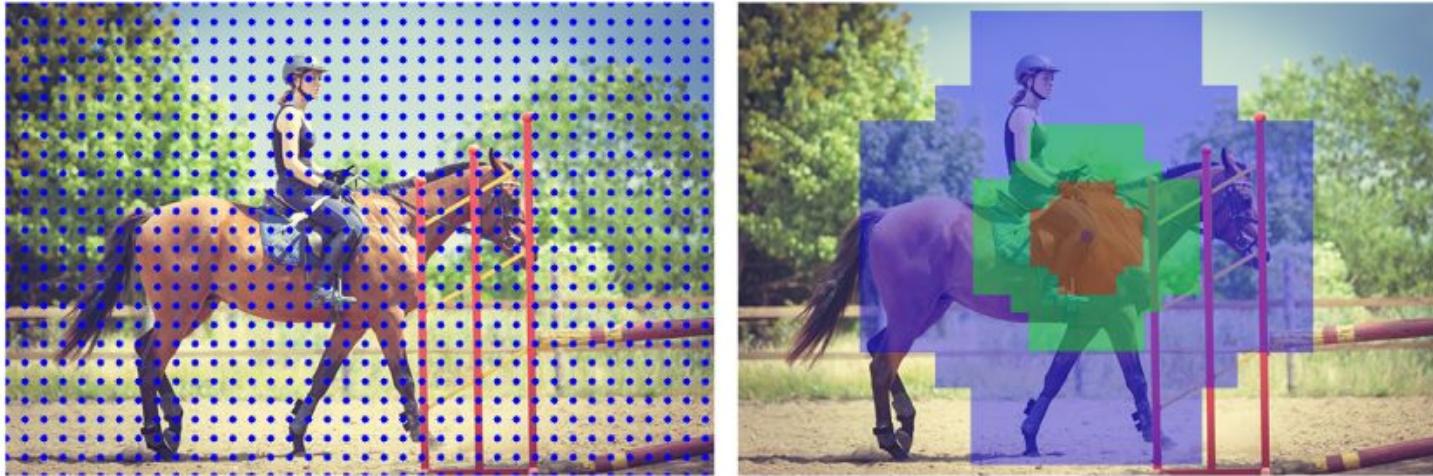
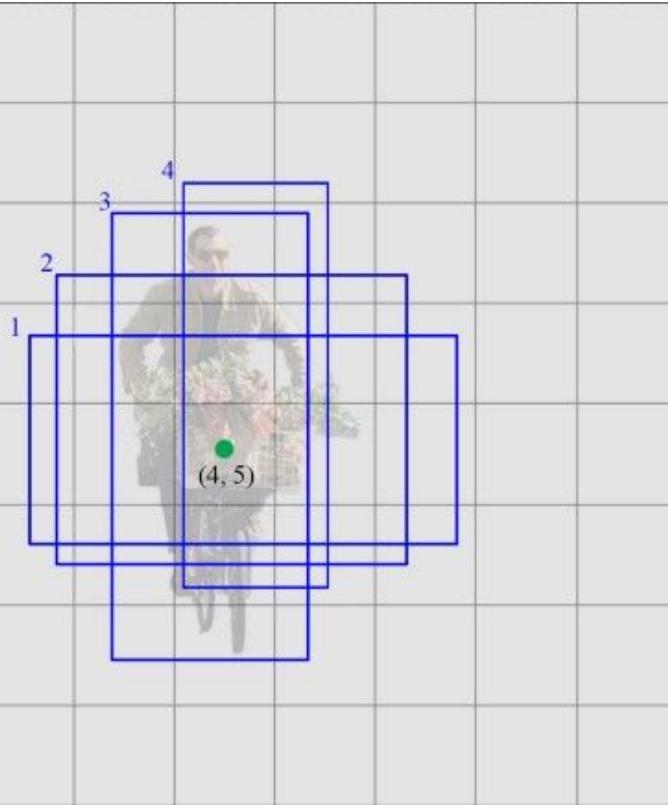


Figure 14.7: **Left:** Creating anchors starts with the process of sampling the coordinates of an image every r pixels ($r = 16$ in the original Faster R-CNN implementation). **Right:** We create a total of nine anchors centered around *each* sampled (x, y) -coordinate. In this visualization, $x = 300,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

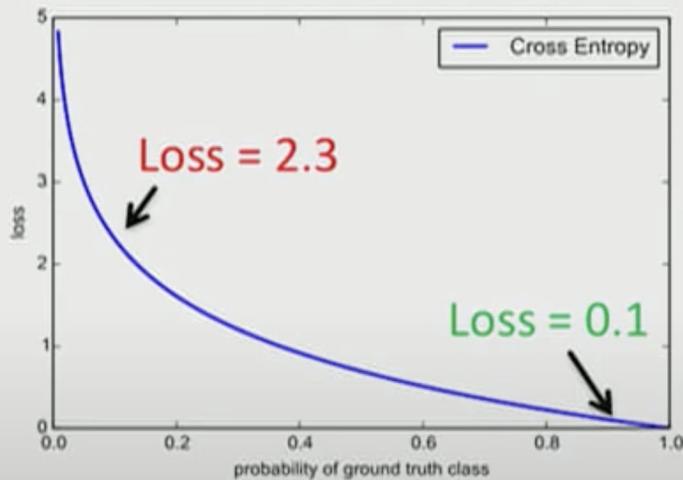
Many negative examples, no useful signal

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



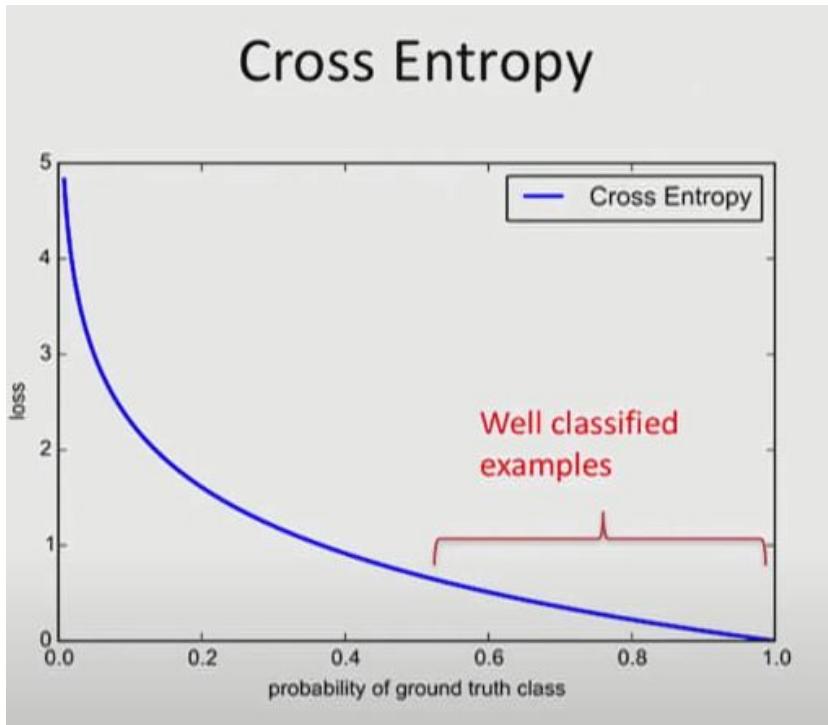
[RetinaNet: how Focal Loss fixes Single-Shot Detection](#)

Cross entropy loss

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

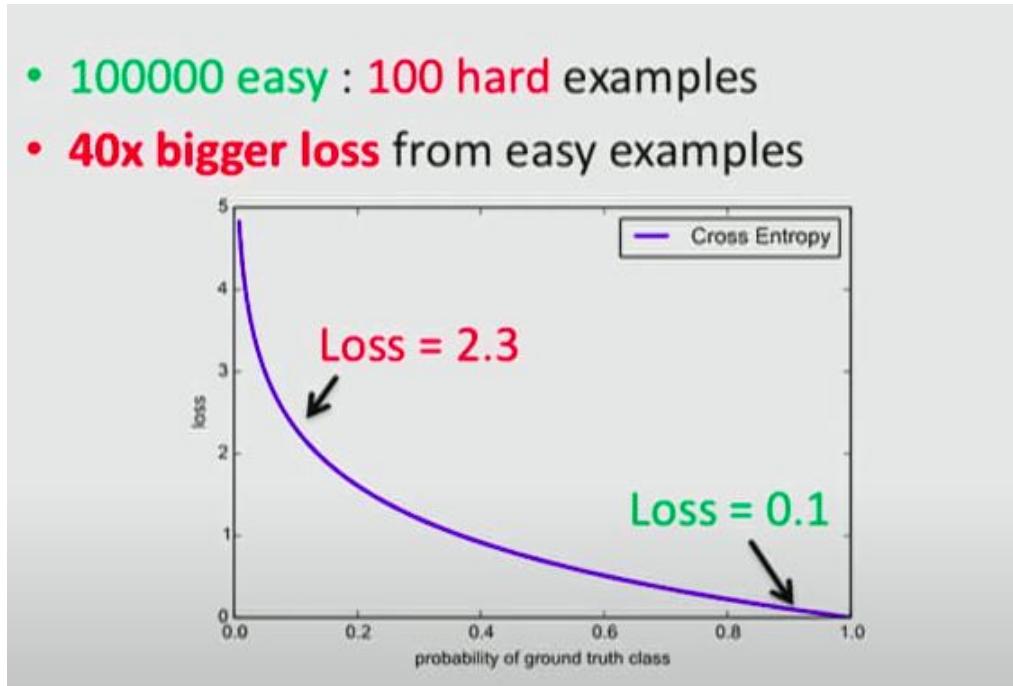
[Understanding cross entropy loss](#)

Cross entropy at different confidence levels

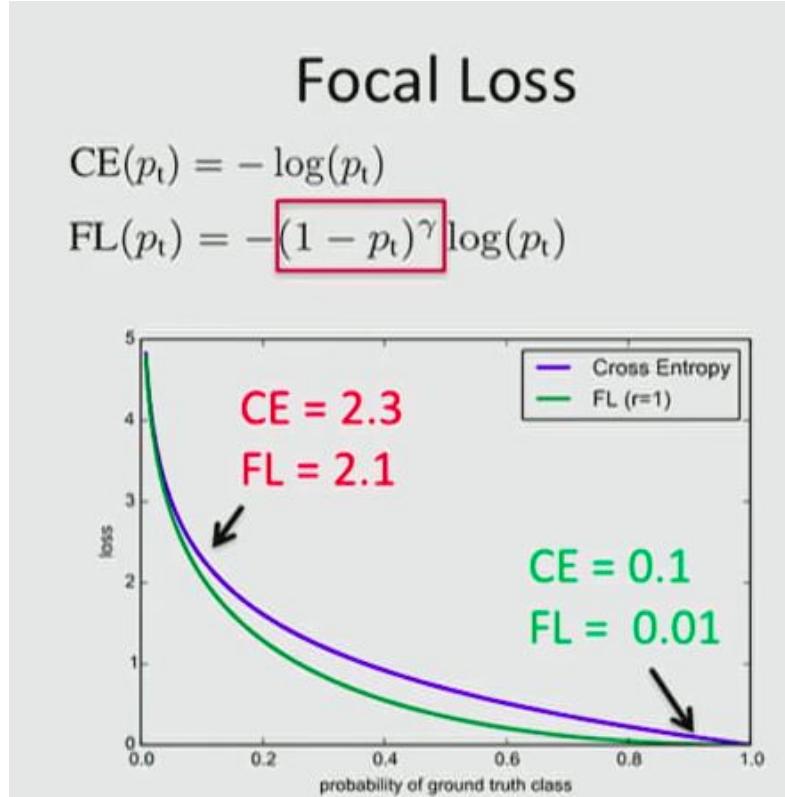


[Andrew Ng's explanation of cross entropy loss](#)

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 γ

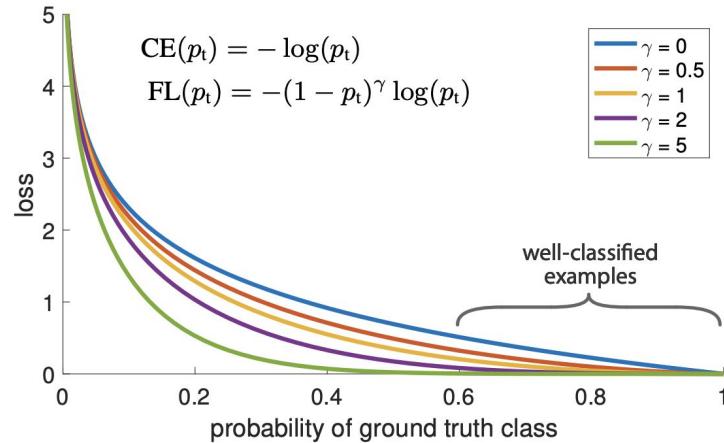
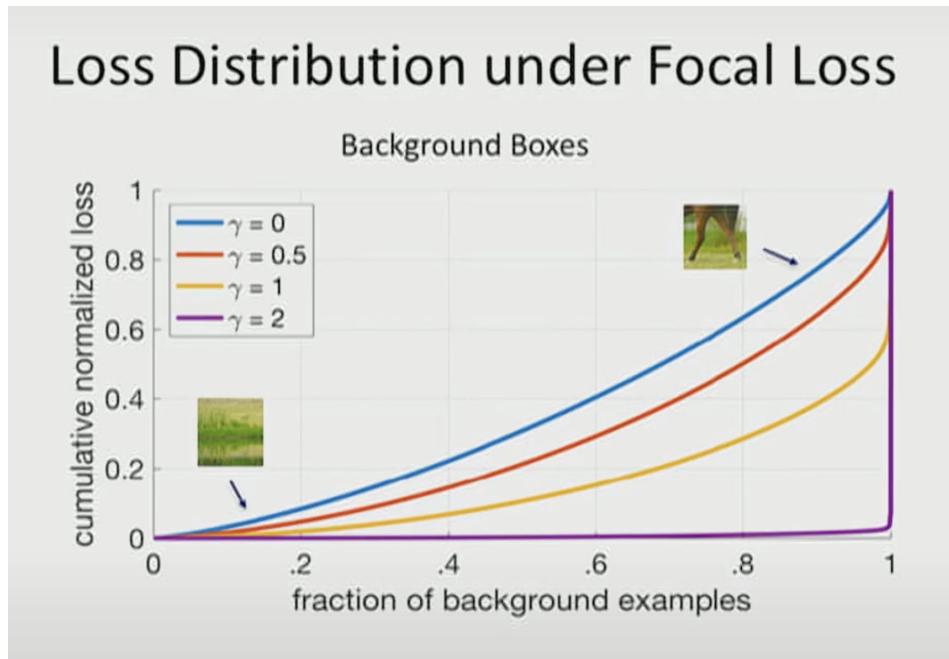


Figure 1. We propose a novel loss we term the *Focal Loss* that adds a factor $(1 - p_t)^\gamma$ to the standard cross entropy criterion. Setting $\gamma > 0$ reduces the relative loss for well-classified examples ($p_t > .5$), putting more focus on hard, misclassified examples. As our experiments will 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 γ



$\gamma = 0$ 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.

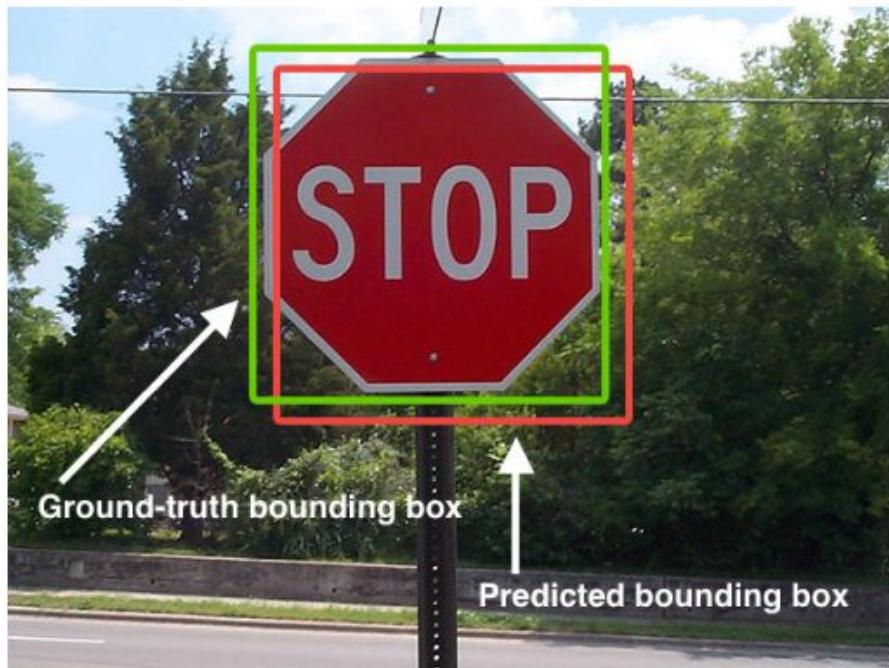
Anchor box x, y offset predictions. We tried using the normal anchor box prediction mechanism where you predict the x, y offset as a multiple of the box width or height using a linear activation. We found this formulation decreased model stability and didn't work very well.

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.

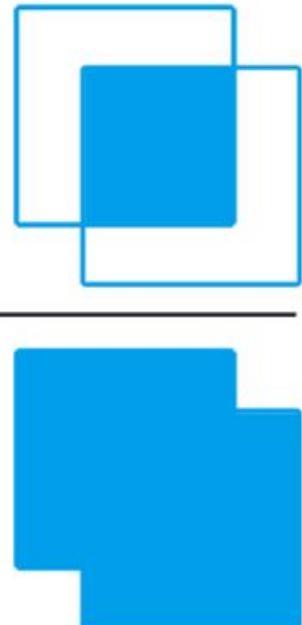
Focal loss. We tried using focal loss. It dropped our mAP about 2 points. YOLOv3 may already be robust to the problem focal loss is trying to solve because it has separate objectness predictions and conditional class predictions. Thus for most examples there is no loss from the class predictions? Or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

[Extract from the YOLOv3 paper](#)

Measuring performance with IoU

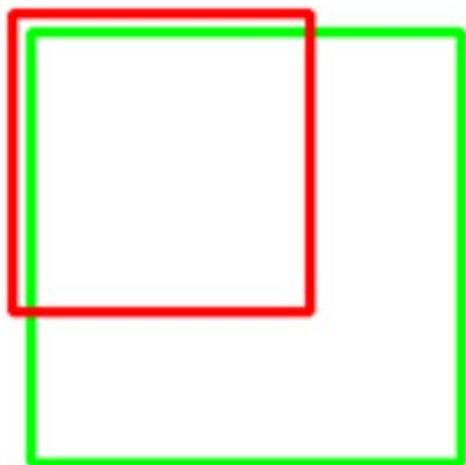


$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



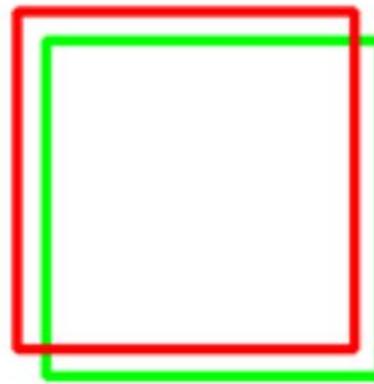
Measuring performance with IoU

IoU: 0.4034



Poor

IoU: 0.7330



Good

IoU: 0.9264

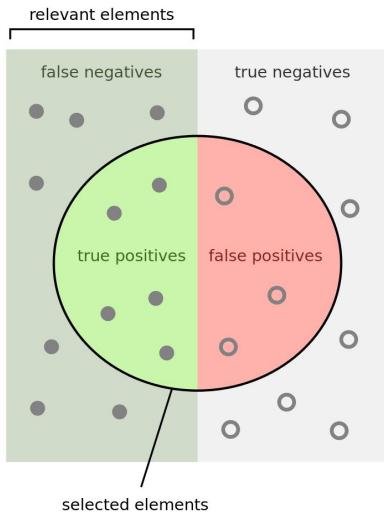


Excellent

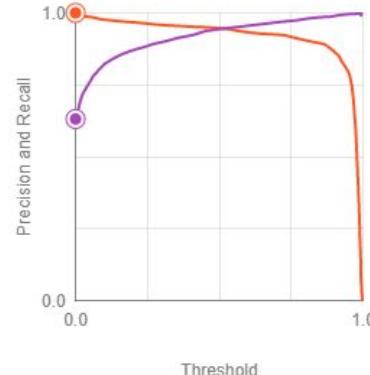
Measuring performance with IoU

$$class(IoU) = \begin{cases} Positive \rightarrow IoU \geq Threshold \\ Negative \rightarrow IoU < Threshold \end{cases}$$

Precision, recall, and f1 metrics



Precision and Recall vs Threshold



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

How many selected items are relevant?

How many relevant items are selected?

$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$

$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$

- 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)

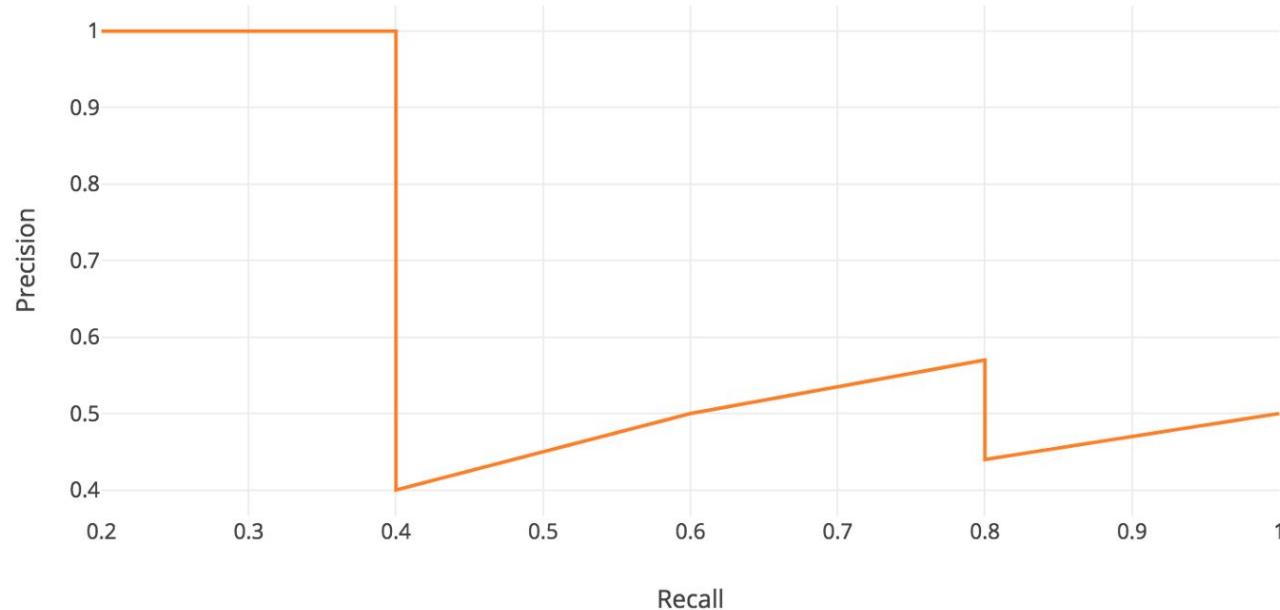
$$mAP = \frac{1}{n} \sum_{k=1}^{k=n} AP_k$$

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>

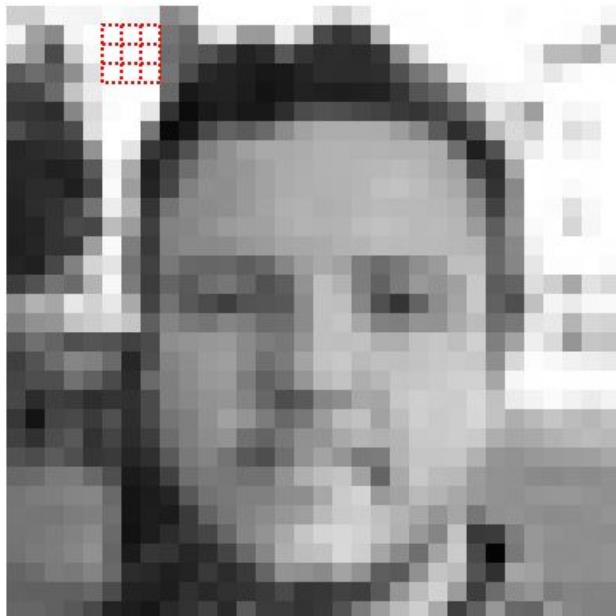
Convolution kernels (aka filters)

206 205 247 245 244 253 247 245 136 151 258 255 255 255 255 255 254 255 255 254 255 255 255 255 255 255 255 254 255 247
244 161 137 244 254 255 254 255 118 103 209 128 155 153 236 193 74 52 60 173 255 254 254 255 255 254 255 254 253 244 184
182 154 75 300 249 255 255 255 110 96 84 61 35 44 89 53 44 45 43 94 140 213 253 255 255 255 255 245 187 186 176 223
90 109 95 143 223 255 255 255 252 117 75 41 35 31 24 25 36 45 44 46 91 118 148 134 252 254 255 245 231 248 255 255 254 247
67 69 107 198 238 255 255 255 104 29 34 35 29 20 25 34 32 30 32 34 53 89 100 141 231 241 247 240 255 255 255 255 255
55 51 49 134 238 255 255 252 51 12 26 33 24 24 40 75 92 70 71 66 58 53 67 90 236 228 206 158 253 246 255 255
79 58 56 75 234 255 255 116 11 27 74 99 91 101 340 182 173 173 173 173 173 173 173 173 173 173 173 173 173 173 173 173 173
38 43 47 52 147 255 229 56 41 81 128 145 180 288 186 172 178 178 178 178 177 177 177 177 177 177 177 177 177 177 177 177 177
40 40 45 56 110 139 145 151 161 174 178 178 182 184 187 183 187 173 180 71 45 187 255 254 255 255 255 254 255
37 44 44 31 69 250 158 30 70 129 143 142 152 161 173 175 177 178 182 191 194 188 180 170 120 51 137 255 254 250 254 255
34 45 51 64 116 237 182 53 118 138 140 142 154 184 178 178 174 177 183 186 185 183 182 178 140 66 141 254 252 225 249 255
34 36 52 74 71 188 156 63 133 134 144 155 180 181 173 179 178 179 180 193 180 185 187 182 156 93 141 250 254 234 247 255
32 38 52 54 159 250 128 57 129 128 138 140 150 251 156 188 188 171 178 180 187 188 185 182 183 180 202 136 242 255 255 254 254
36 32 72 129 232 228 115 65 123 104 102 104 98 303 134 156 170 182 125 106 123 143 155 180 193 104 124 230 252 253 255 253
61 82 114 107 179 247 134 69 101 90 111 119 103 94 147 181 178 128 98 123 153 147 161 200 92 100 222 207 187 227 235
144 379 267 233 230 232 170 67 119 183 70 62 93 109 88 239 239 180 225 90 53 59 141 183 203 97 79 183 249 235 248 249
137 245 349 195 204 213 187 95 133 122 117 133 126 109 110 139 191 187 187 129 127 146 147 171 188 110 121 228 235 180 215 212
87 112 100 79 85 82 65 75 142 148 153 155 138 125 120 149 191 180 185 175 174 193 188 190 208 127 163 236 231 149 186 195
63 103 150 129 306 39 78 150 125 142 124 184 185 200 186 182 181 185 200 202 200 145 217 253 249 182 238 234
69 70 70 71 113 97 74 43 104 137 140 152 155 125 97 112 150 185 184 174 183 196 198 302 205 206 186 247 254 255 254 254 254
72 44 83 59 46 52 49 74 127 137 348 140 132 103 70 90 134 141 188 185 180 207 204 203 216 189 236 244 251 242 238 243
55 20 69 73 59 80 46 74 117 127 144 143 148 148 124 106 120 156 187 183 182 188 206 201 205 214 174 185 187 188 183 193
65 40 77 89 50 68 43 61 108 127 141 147 113 100 121 145 148 189 183 178 183 203 201 205 202 174 186 186 178 183 188 184
82 76 92 79 54 58 37 47 90 123 132 116 109 78 111 146 183 149 122 124 180 187 188 178 178 140 146 152 155 257 259 188
304 107 122 123 109 79 27 33 66 111 122 120 114 114 147 175 180 196 183 101 170 200 187 185 186 146 145 139 137 141 140 145
117 124 127 123 125 105 21 28 37 88 115 121 128 128 141 142 188 202 212 153 164 188 186 188 186 154 146 144 144 144 144 144 144 144
119 118 118 125 128 111 21 29 28 58 300 118 131 140 151 159 188 200 205 182 180 186 149 188 210 144 147 143 140 143 144 144 144
117 119 125 130 138 106 18 29 44 58 70 102 133 147 168 187 212 215 210 185 177 152 133 195 57 59 126 153 145 143 142 142
119 123 228 134 145 202 27 54 52 38 49 69 106 235 275 189 180 218 206 188 139 111 184 203 74 5 121 153 342 142 142 146
303 308 123 122 132 105 44 40 31 35 57 44 58 101 344 138 183 145 94 98 145 196 187 94 481 185 200 142 144 142 145
98 97 97 96 104 76 34 33 30 48 41 40 51 58 74 53 55 66 63 109 150 188 208 156 62 108 140 149 125 133 133 133
302 302 97 88 73 35 38 23 42 50 65 41 90 60 59 51 57 62 123 157 187 205 169 62 96 155 308 100 154 135 100 129



<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

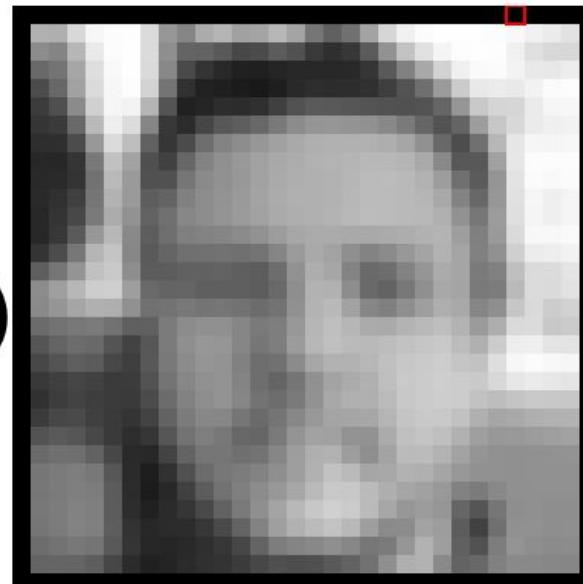
<https://setosa.io/ev/image-kernels/>

What is a convolution filter?



input image

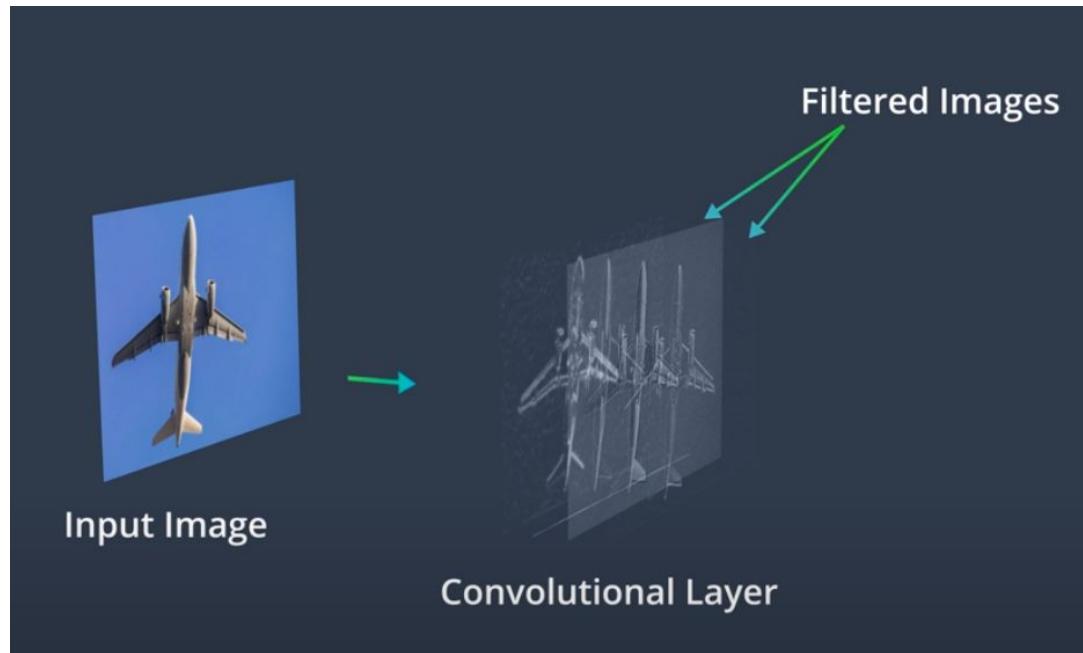
$$\begin{aligned} & \left(\begin{array}{ccc} ? & ? & ? \\ \times 0.0625 & \times 0.125 & \times 0.0625 \end{array} \right. \\ & + \begin{array}{ccc} 252 & 255 & 255 \\ \times 0.125 & \times 0.25 & \times 0.125 \end{array} \\ & + \left. \begin{array}{ccc} 254 & 255 & 254 \\ \times 0.0625 & \times 0.125 & \times 0.0625 \end{array} \right) \\ & = ? \\ & \text{kernel: } \text{blur} \end{aligned}$$



output image

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

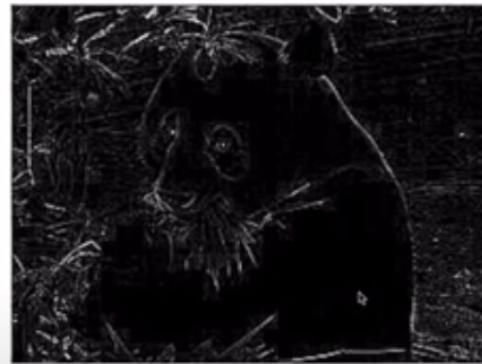
What is a convolution filter?



[Understanding the output of convolutional layers](#)

What is a convolution filter?

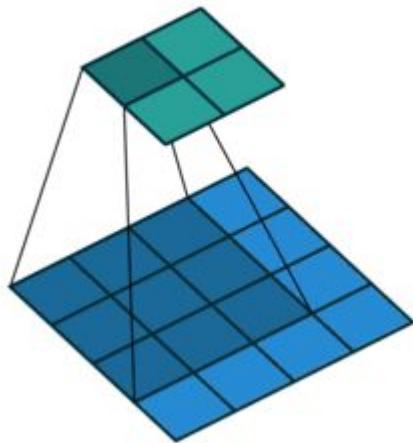
0	-1	0
-1	4	-2
0	-1	0



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

[Visualizing convolution kernels](#)

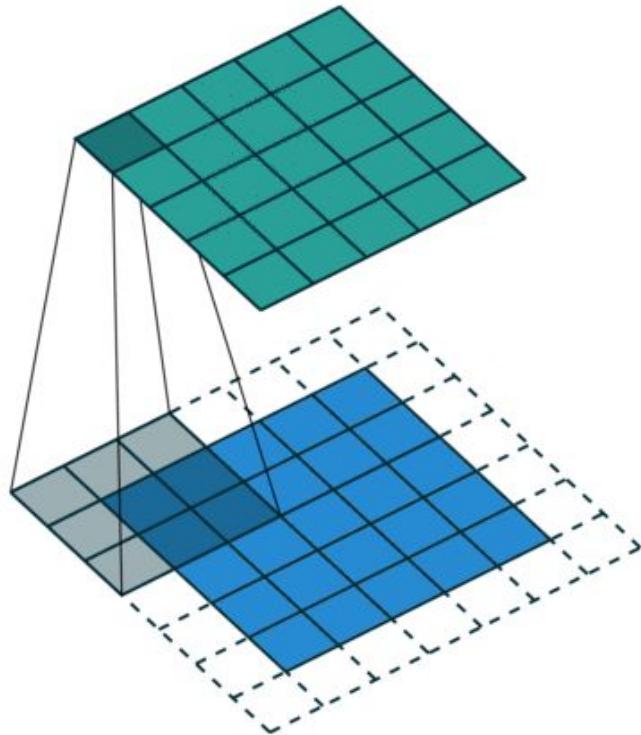
What is a convolution filter?



Convolution of 3×3 and stride = 1 without padding

Effect: the output loses one pixel on each dimension

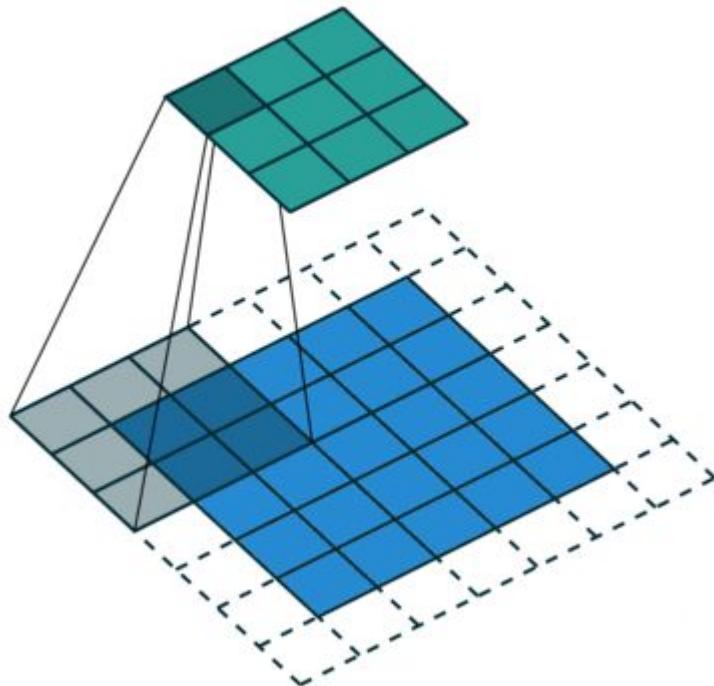
What is a convolution filter?



Convolution of 3×3 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

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

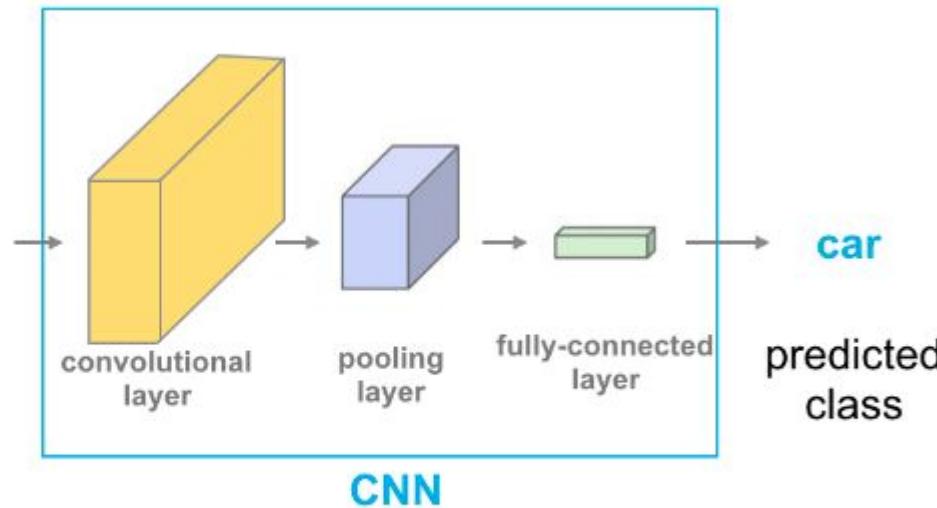
$$S_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

[Finding edges with convolution kernels](#)

CNN architectures

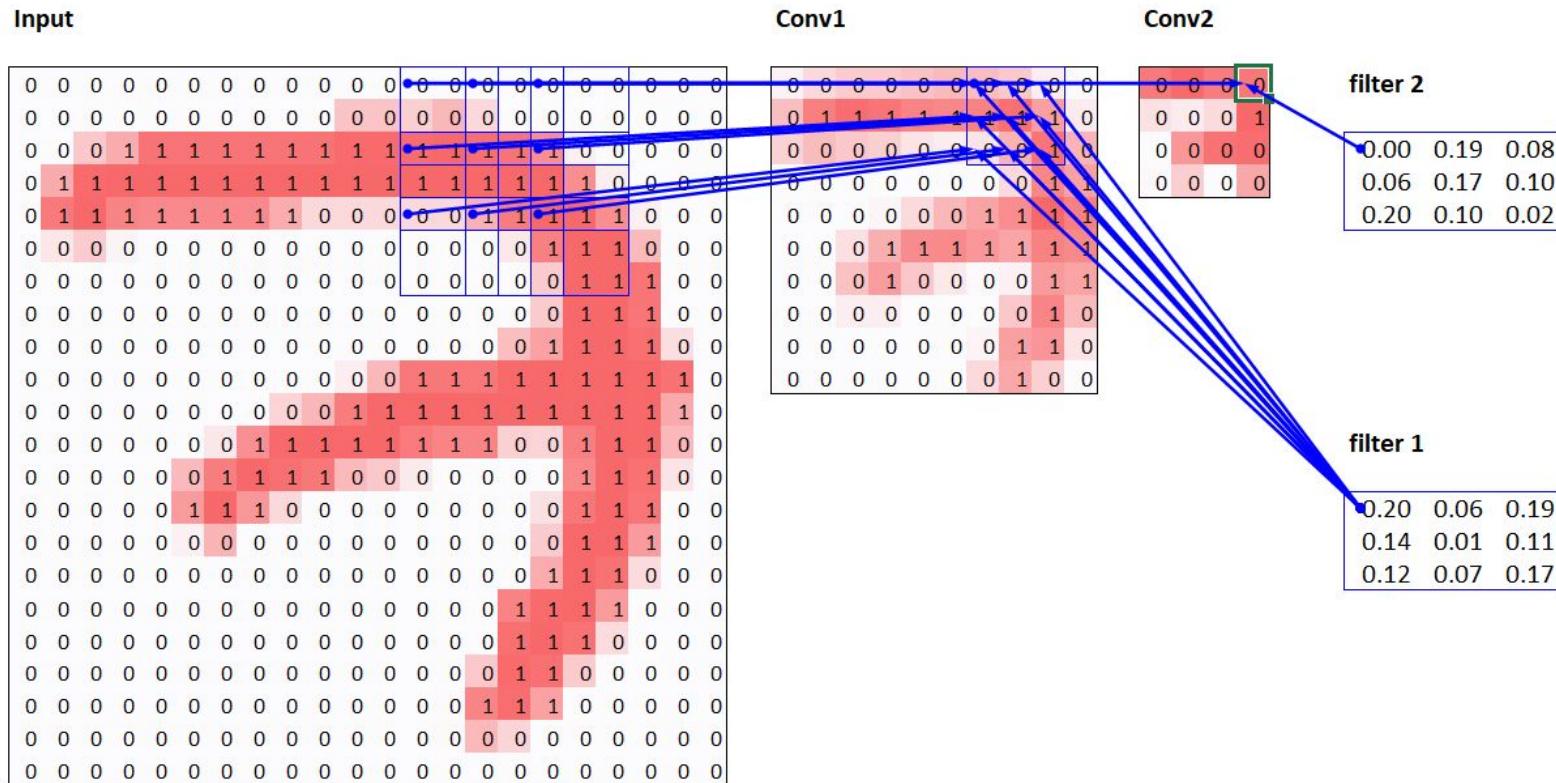


input image

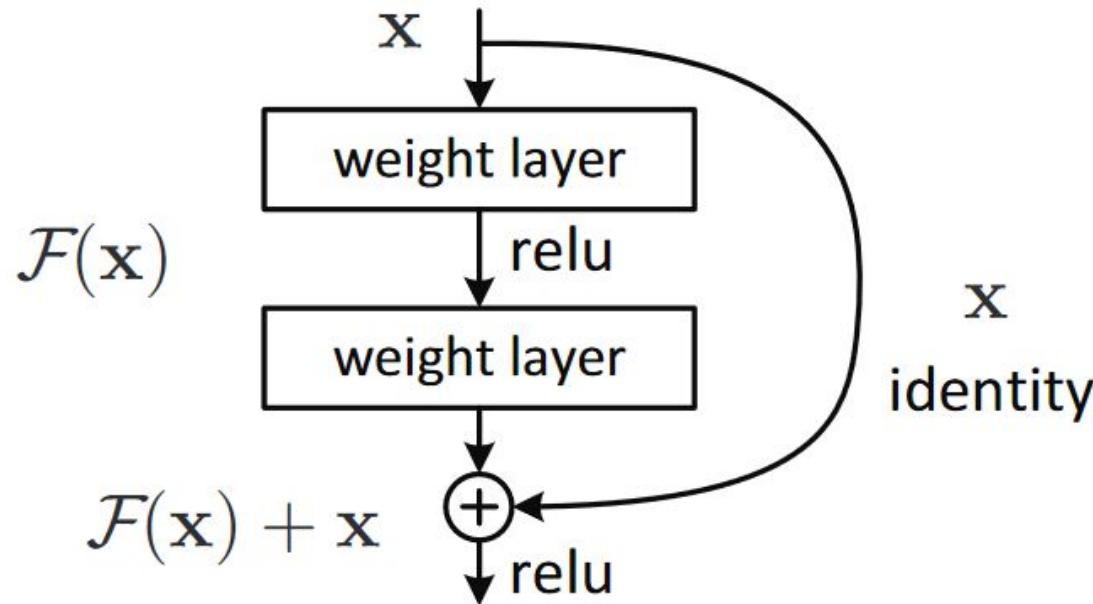


Layers in a CNN.

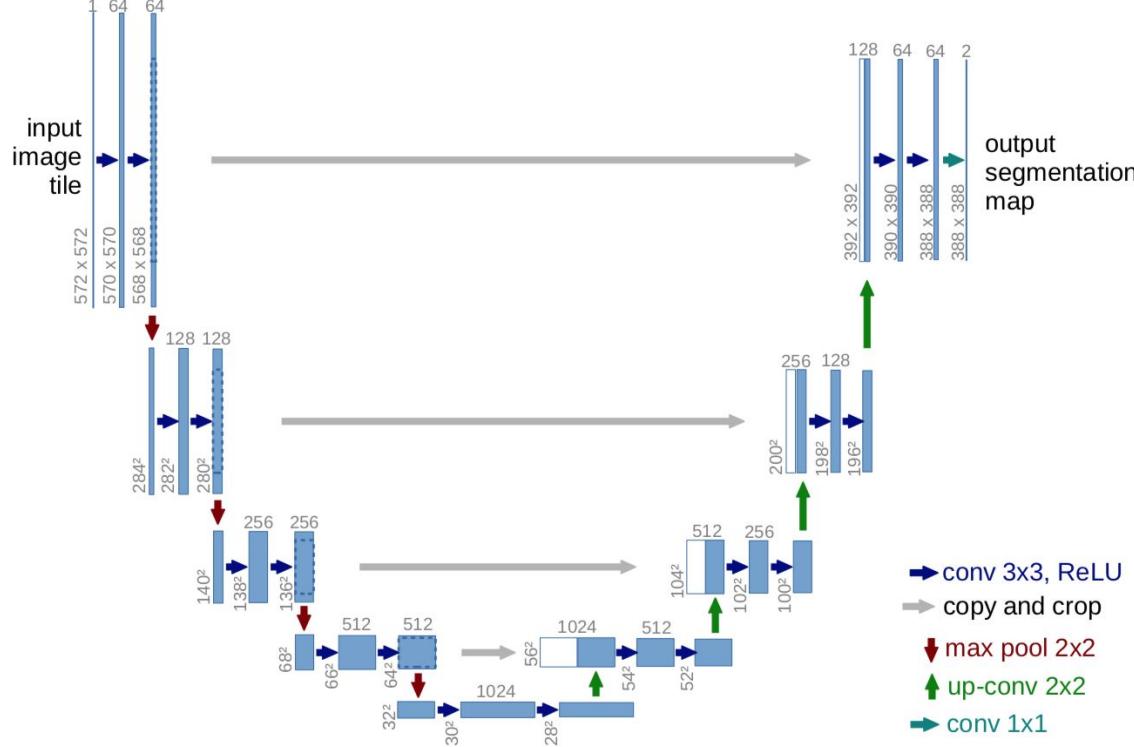
Convolution kernels (aka filters)

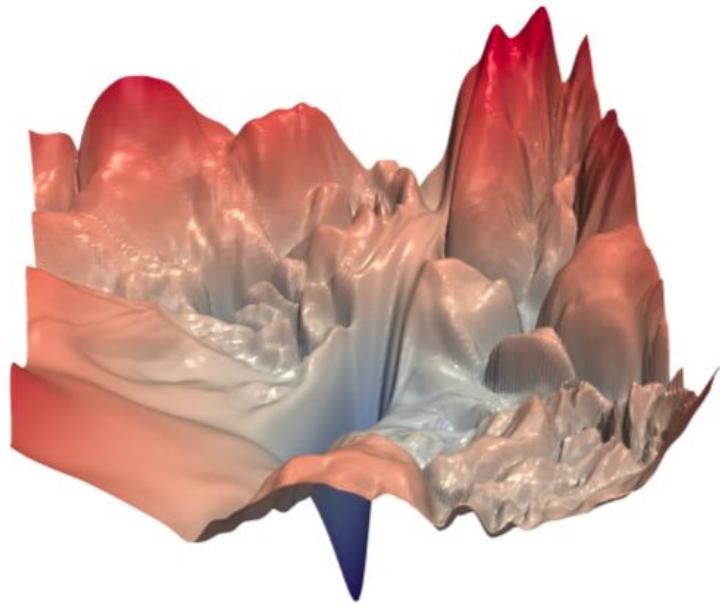


Skip connections

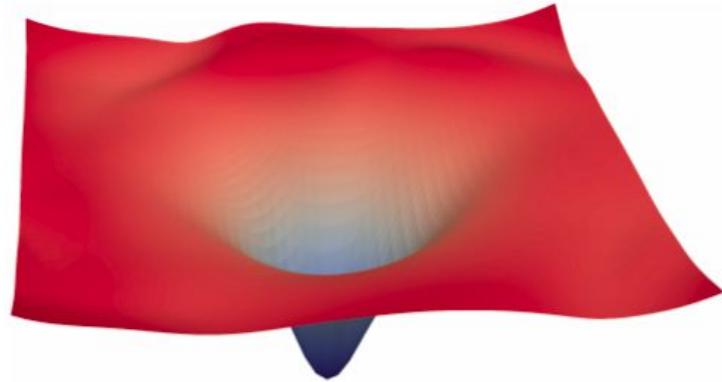


Skip connections





(a) without skip connections

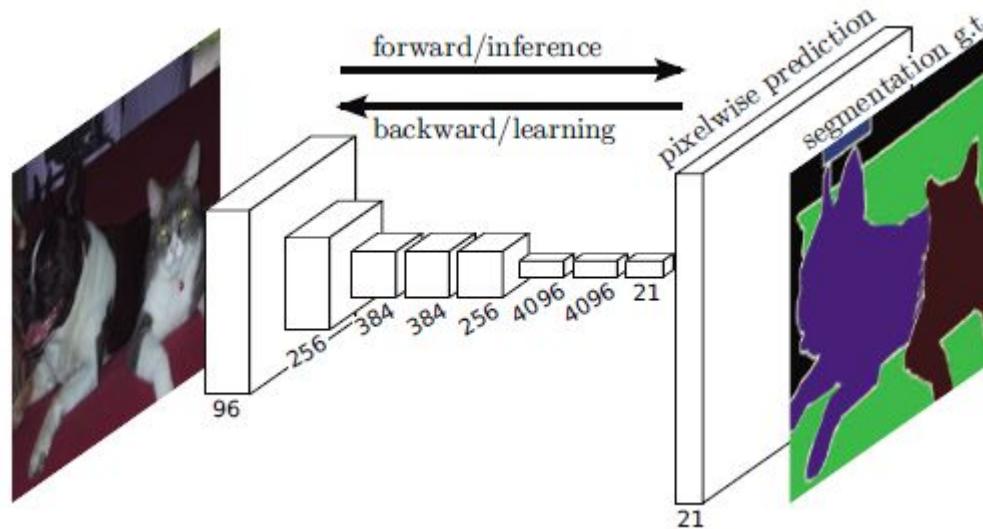


(b) with skip connections

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.

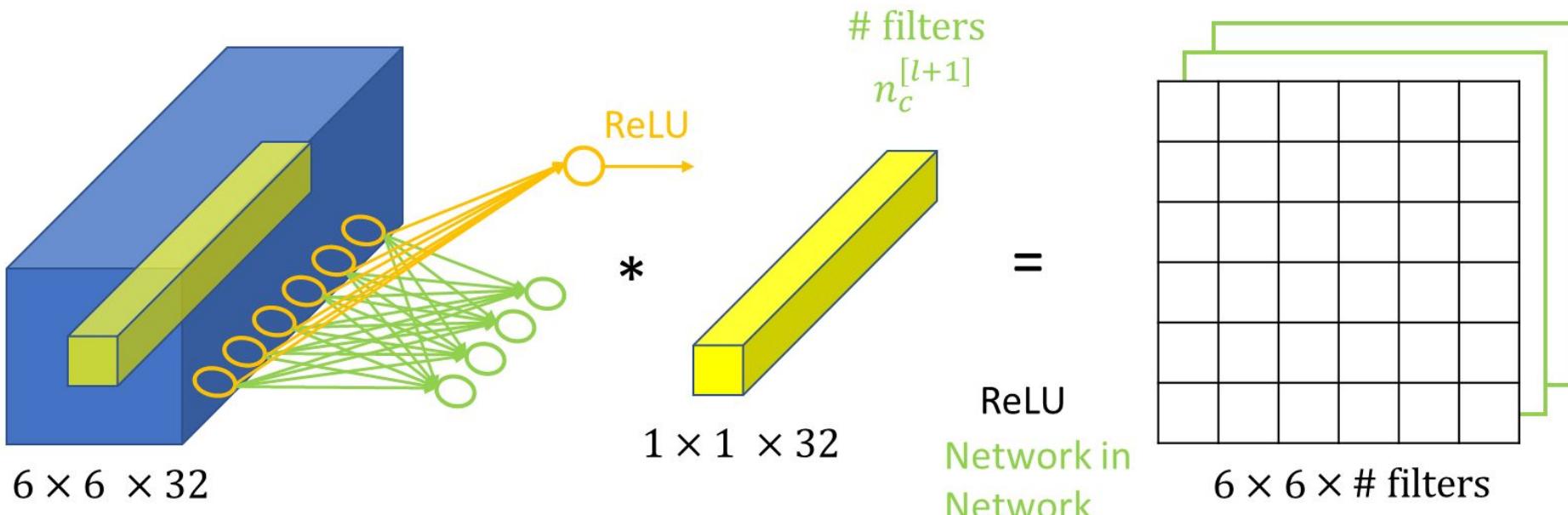
[Visualizing the loss landscape of neural nets](#)

“Fully Convolutional Networks” allow pixelwise prediction



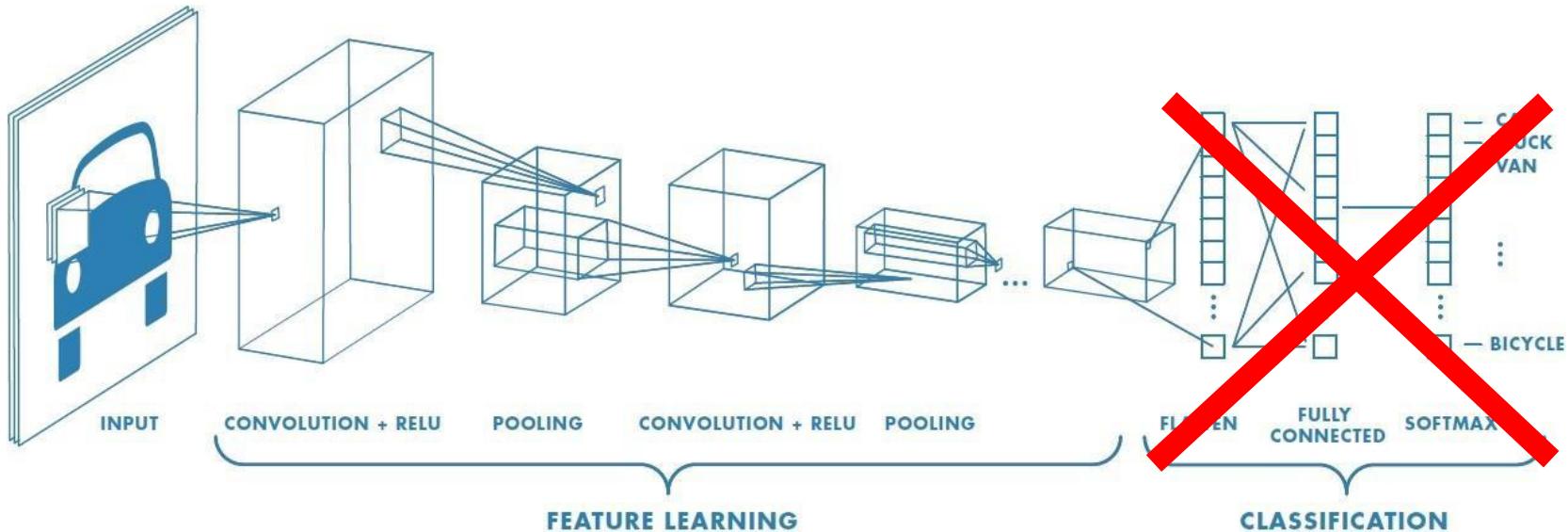
All layers in the network are convolutional, there is no fully connected (aka “dense”) layer like in most classifiers, we use the local info of the pixel neighborhood

“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

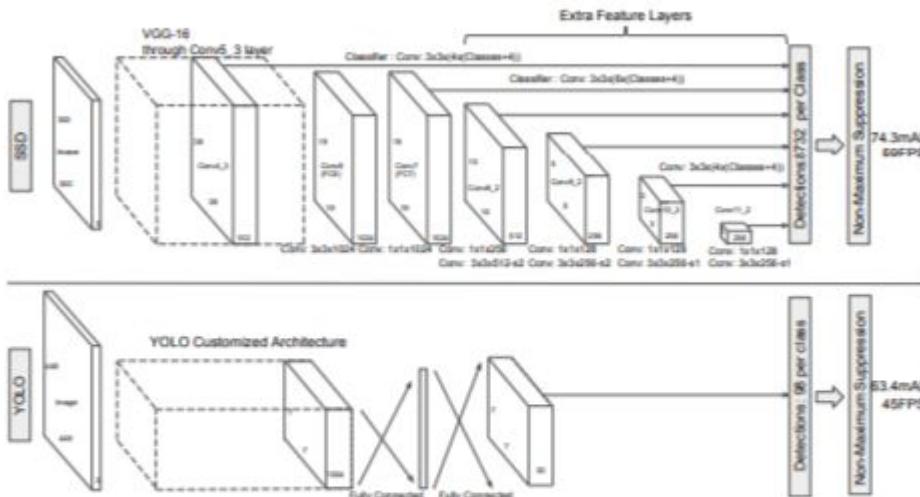


Fig. 2: A comparison between two single shot detection models: SSD and YOLO [5]. Our SSD model adds several feature layers to the end of a base network, which predict the offsets to default boxes of different scales and aspect ratios and their associated confidences. SSD with a 300×300 input size significantly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

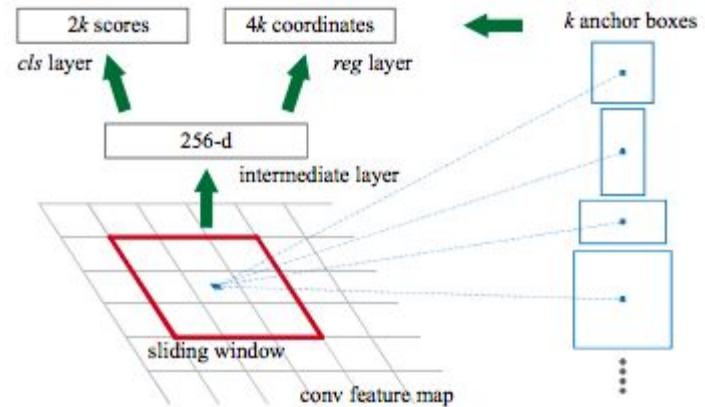
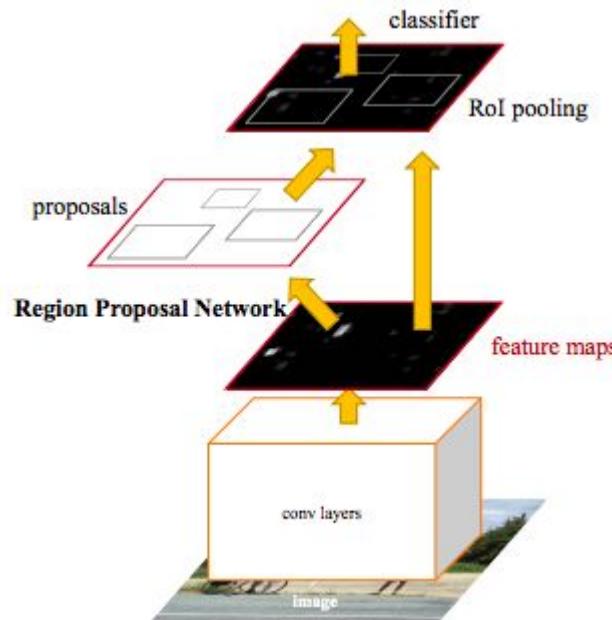
Two stage detectors: R-CNN variants

- **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

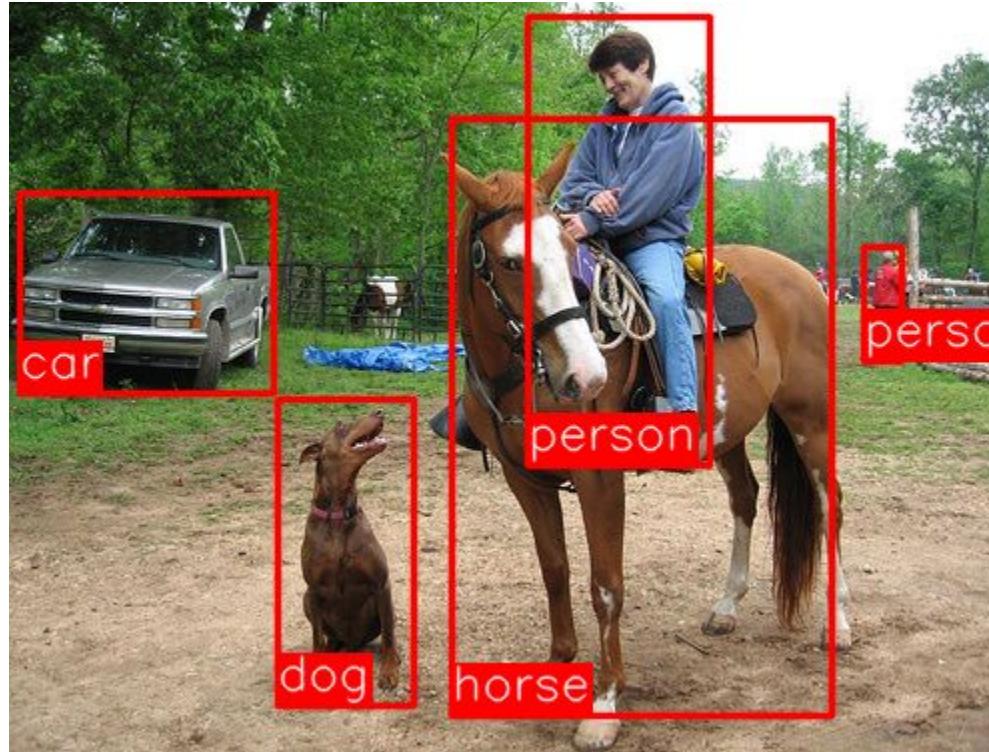
- **Step #1:** Input an image and segment it into blobs
- **Step #2:** Extract regions proposals using the Selective Search algorithm
- **Step #3:** Select the regions of interest with higher object activations using a pretrained network, warp them to standard size (e.g. 128x128) and output proposals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

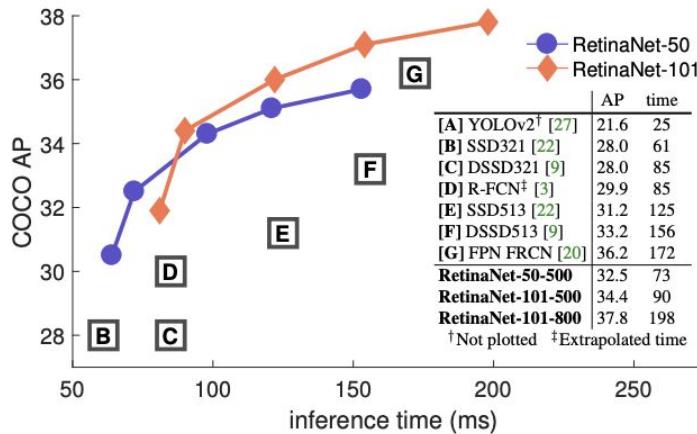
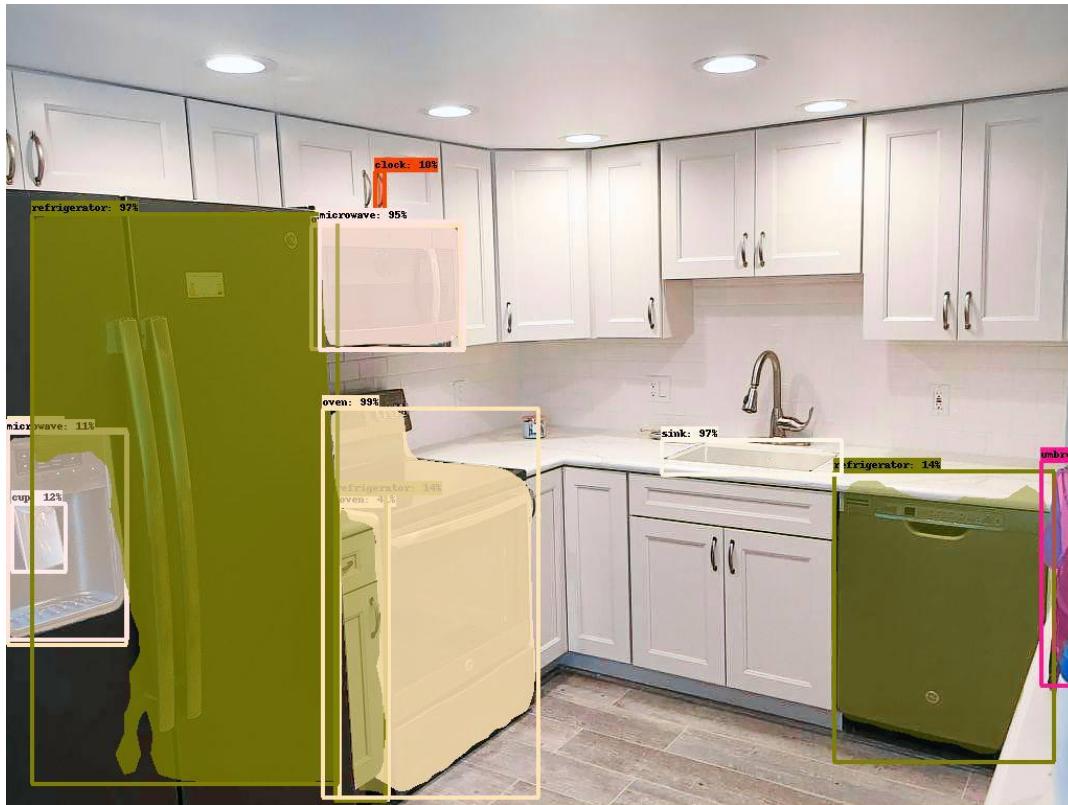


Figure 2. Speed (ms) versus accuracy (AP) on COCO test-dev. Enabled by the focal loss, our simple one-stage *RetinaNet* detector outperforms all previous one-stage and two-stage detectors, including the best reported Faster R-CNN [28] system from [20]. We show variants of RetinaNet with ResNet-50-FPN (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.

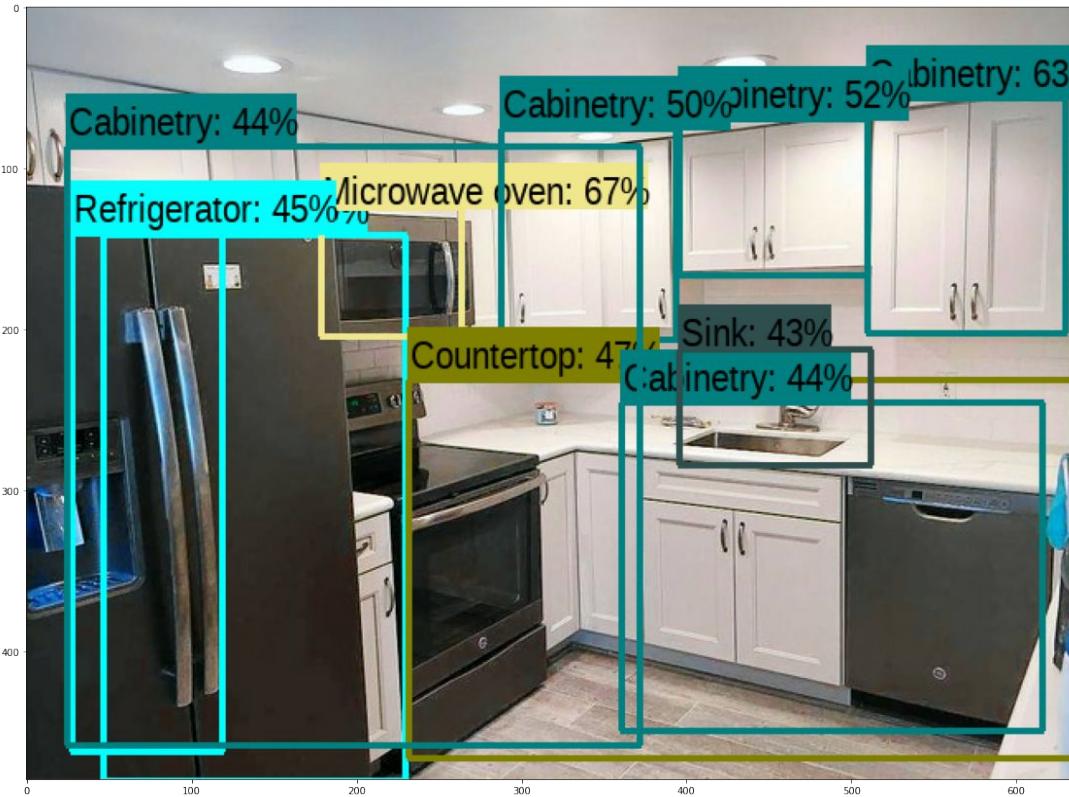
Input Image



Mask R-CNN pretrained on COCO (inventory image)



SSD pretrained on Google Open Images



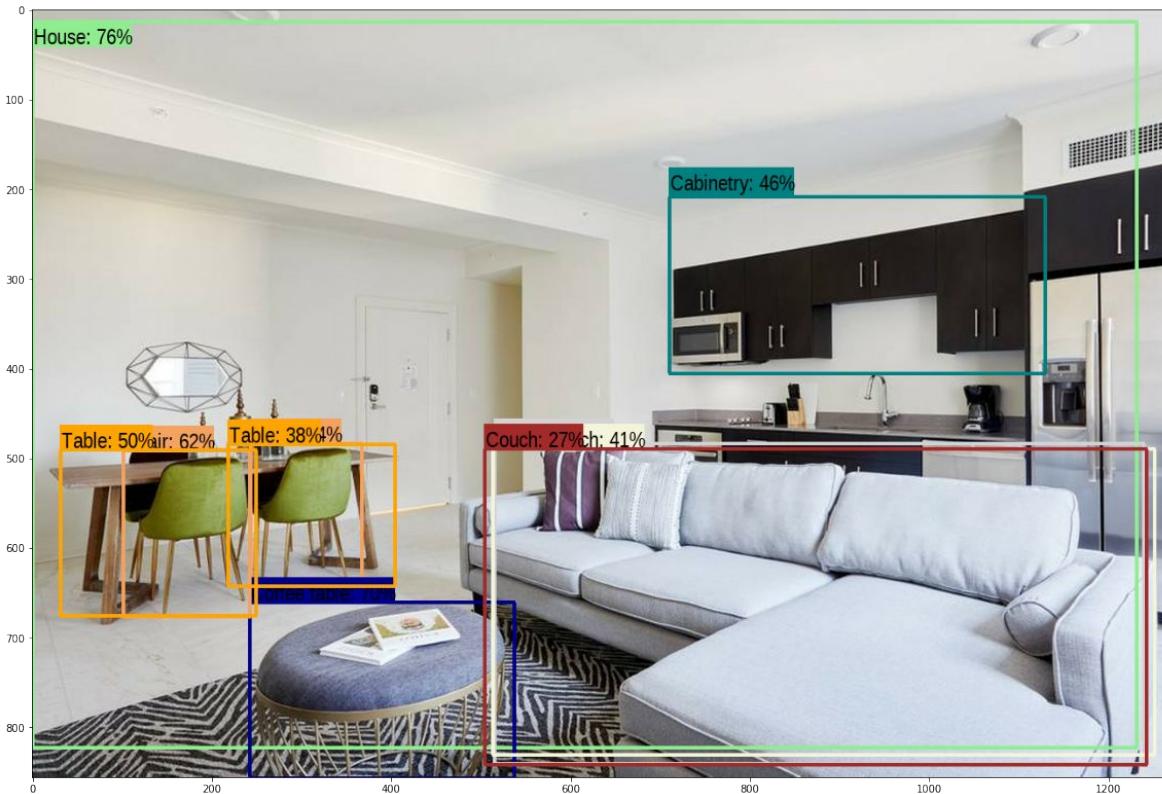
Input image



Mask R-CNN trained on COCO



SSD trained on Open Images Dataset

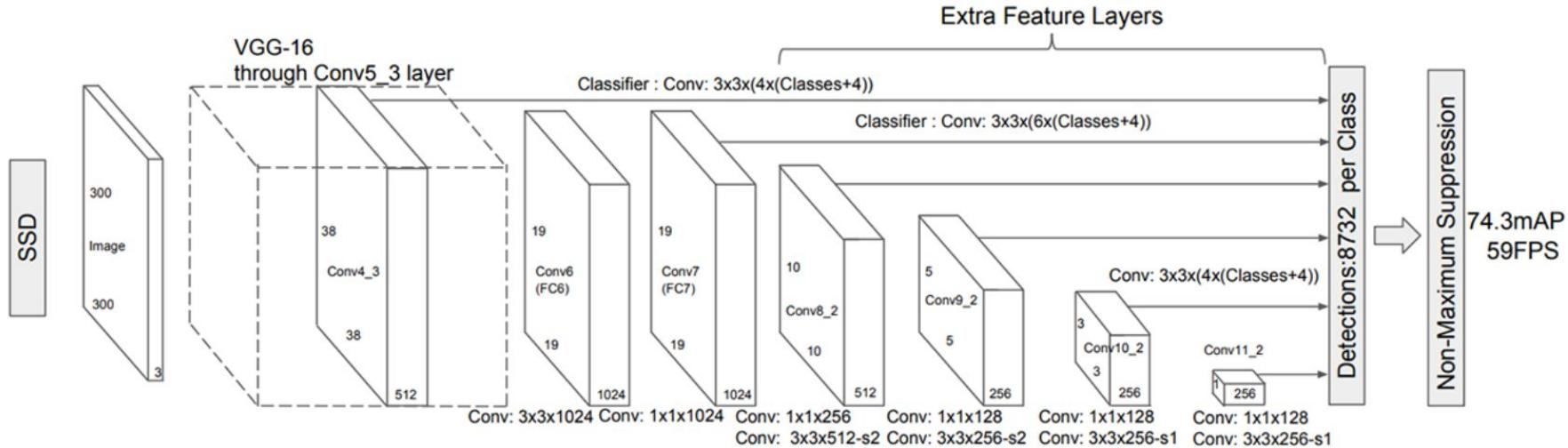


Tricks to improve accuracy on small objects

- 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

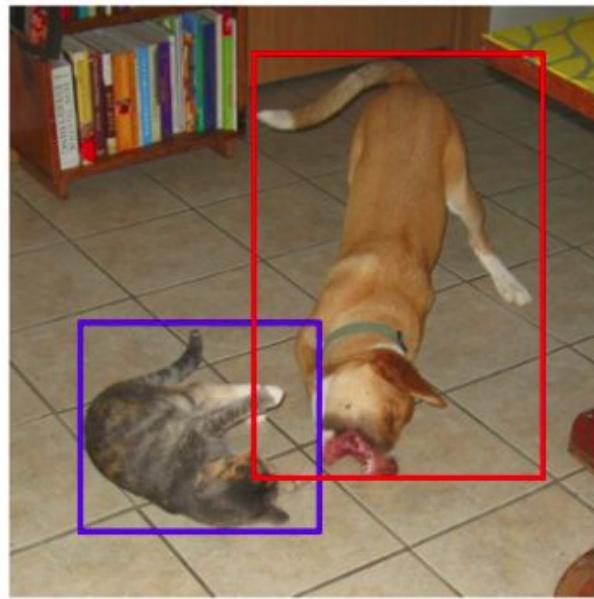
Small objects detection problem

Single Shot Detectors

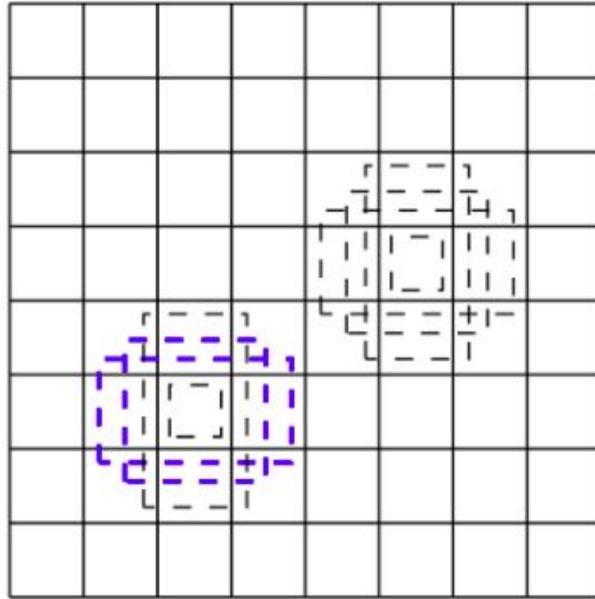


Single Shot Multibox Detection

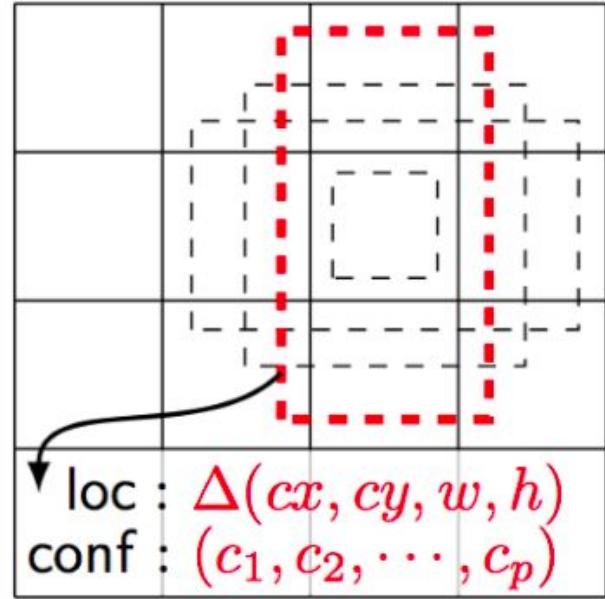
Single shot detectors make predictions of feature maps at different scales (different sizes for the bounding box) and compute cross entropy on a fully convolutional network that takes all these activation scales as input



(a) Image with GT boxes



(b) 8×8 feature map

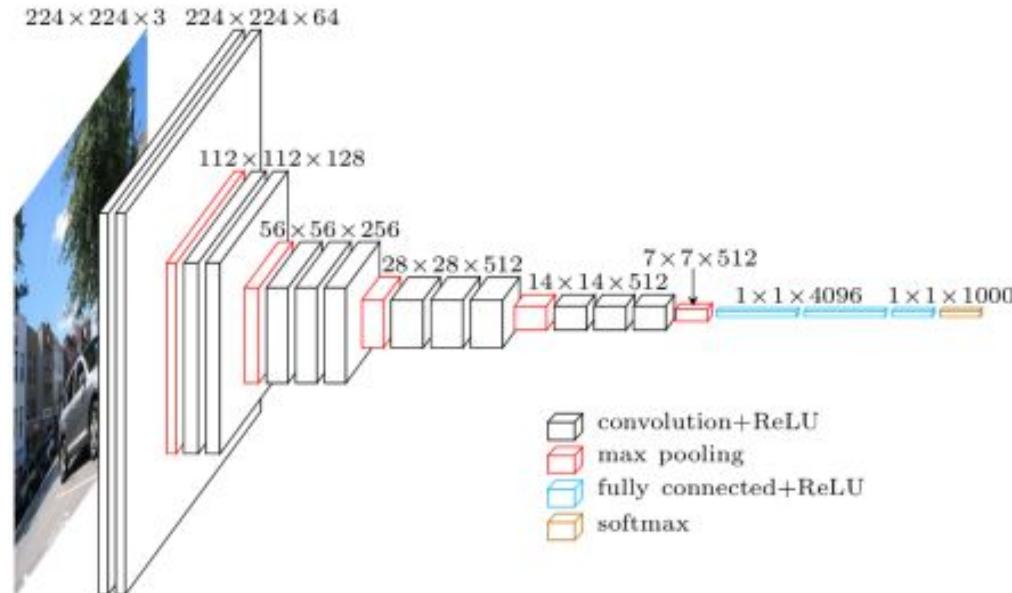


loc : $\Delta(cx, cy, w, h)$
conf : (c_1, c_2, \dots, c_p)

(c) 4×4 feature map

SSD default boxes at 8×8 and 4×4 feature maps

The VGG architecture



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

The speed vs mean average precision (mAP) tradeoff

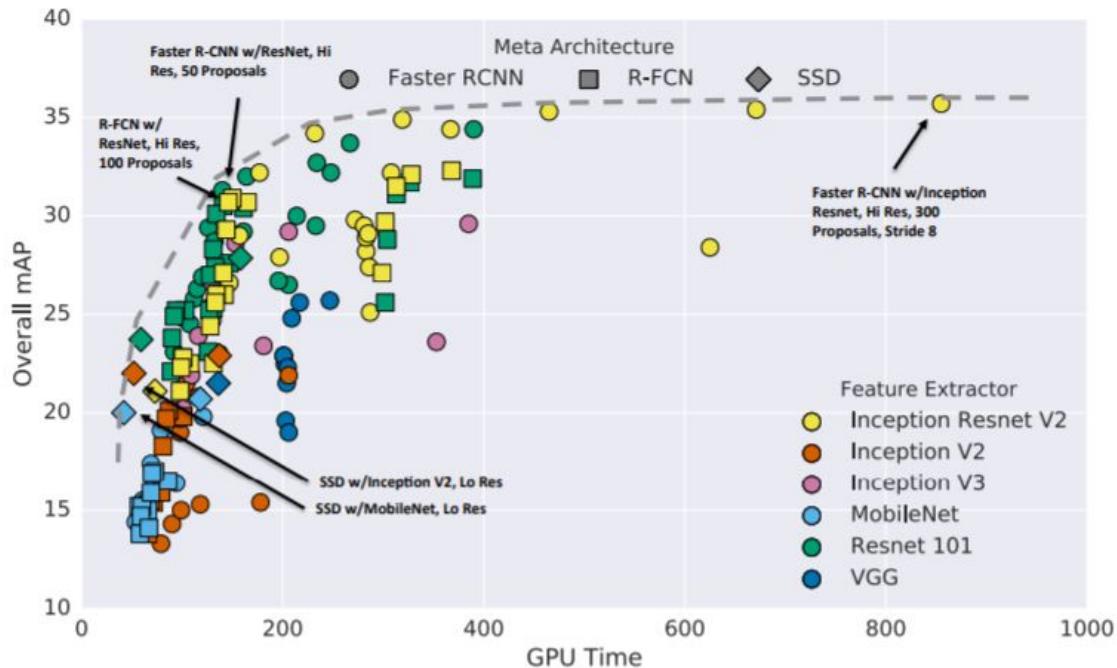


Figure 2: Accuracy vs time, with marker shapes indicating meta-architecture and colors indicating feature extractor. Each (meta-architecture, feature extractor) pair can correspond to multiple points on this plot due to changing input sizes, stride, etc.

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:

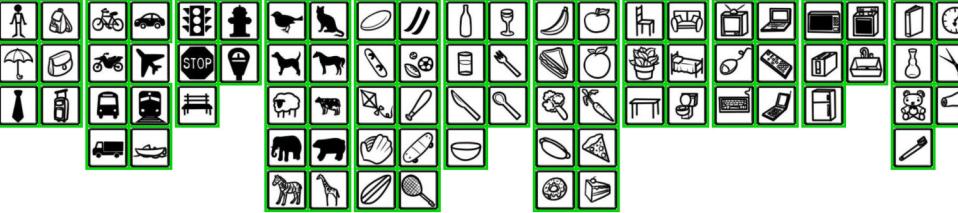
Name	lr sched	train time (s/iter)	inference time (s/im)	train mem (GB)	box AP	model id	download
R50	1x	0.205	0.041	4.1	37.4	190397773	model metrics
R50	3x	0.205	0.041	4.1	38.7	190397829	model metrics
R101	3x	0.291	0.054	5.2	40.4	190397697	model metrics

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	model metrics
RPN R50-FPN	1x	0.186	0.032	2.7		58.0	137258492	model metrics
Fast R-CNN R50-FPN	1x	0.140	0.029	2.6	37.8		137635226	model metrics

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 ▾

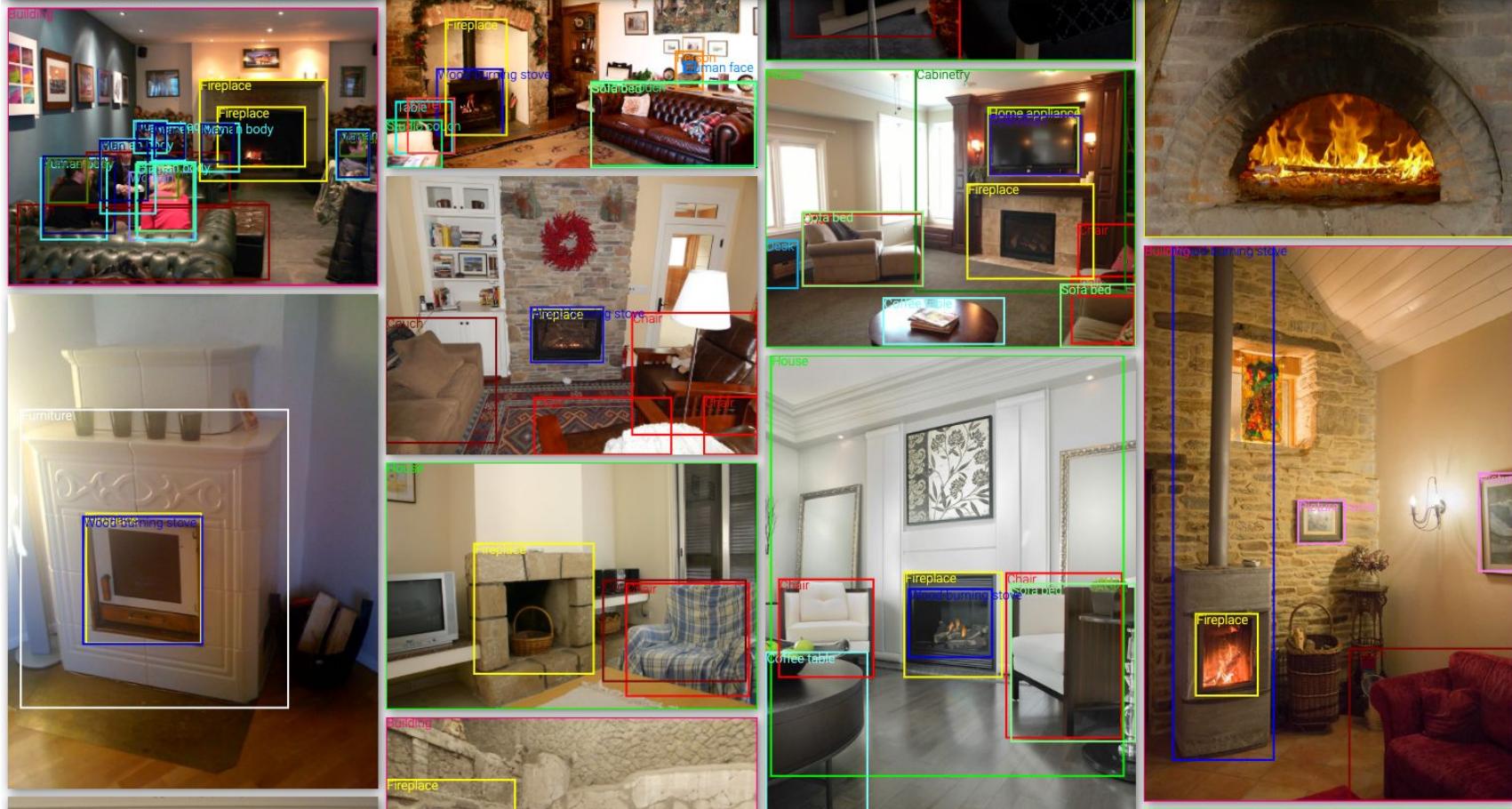
Type: Detection ▾

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

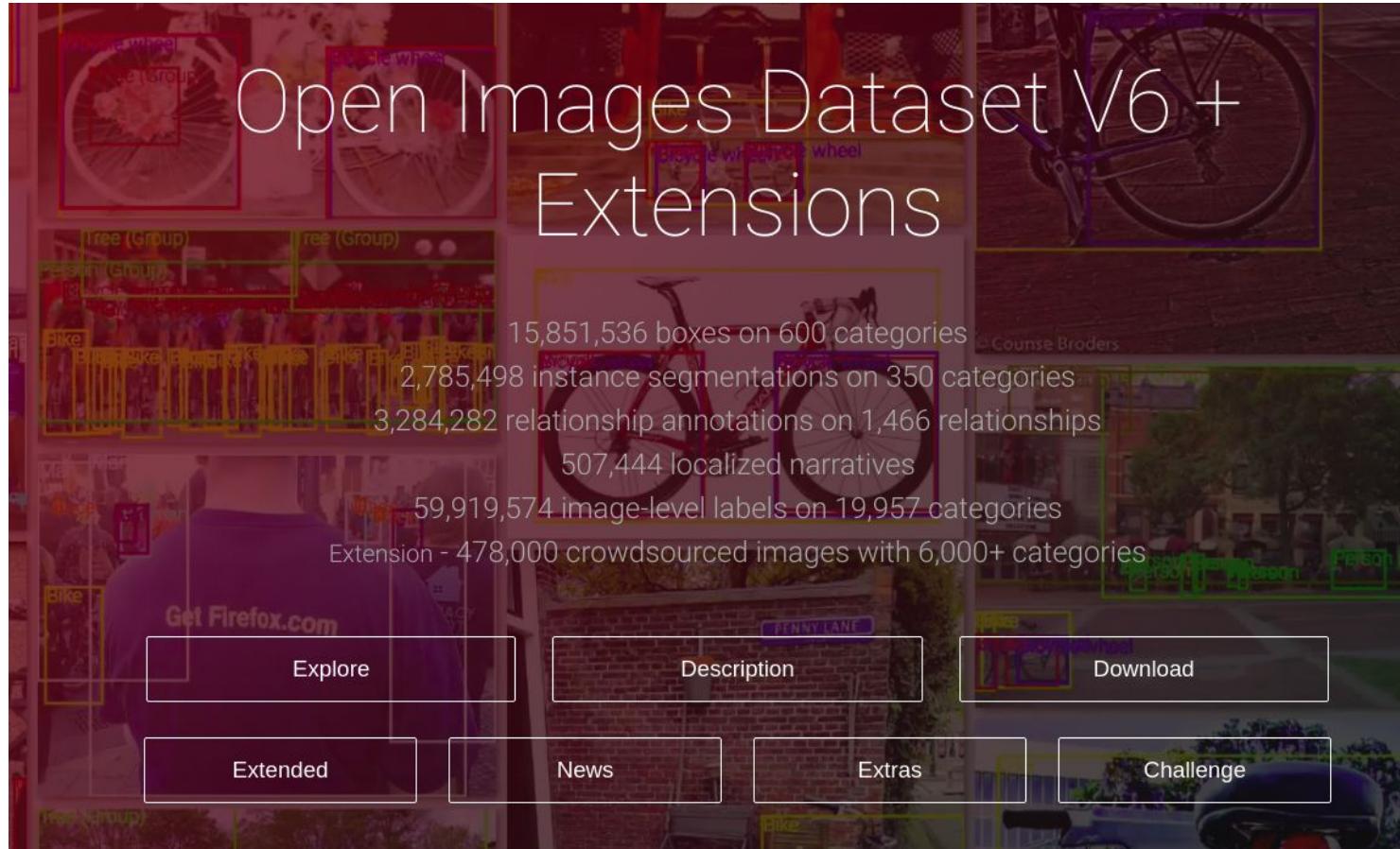
Follow

Jul 16, 2019 · 15 min read ★



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





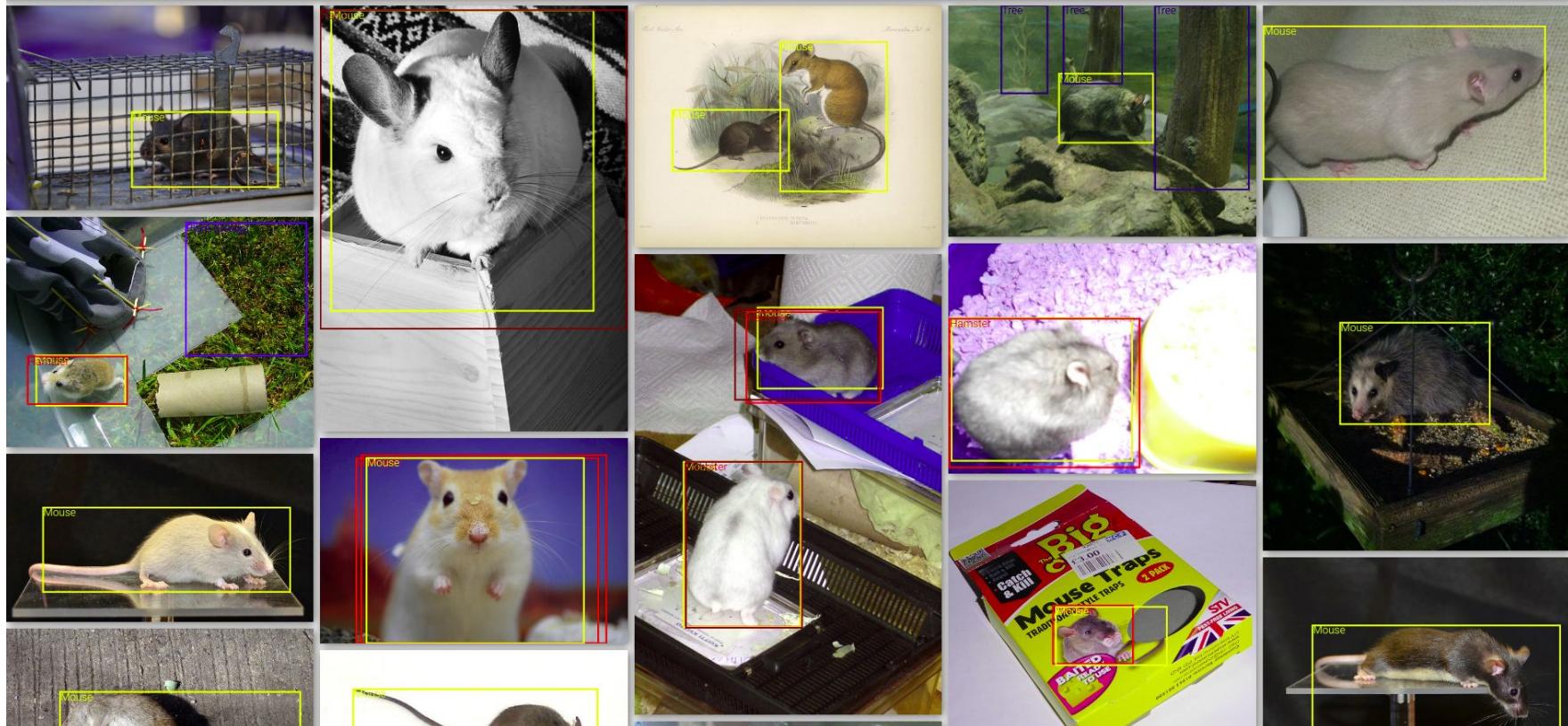
<https://storage.googleapis.com/openimages/web/index.html>

Subset ▾ Type: Detection ▾

Category: Mouse

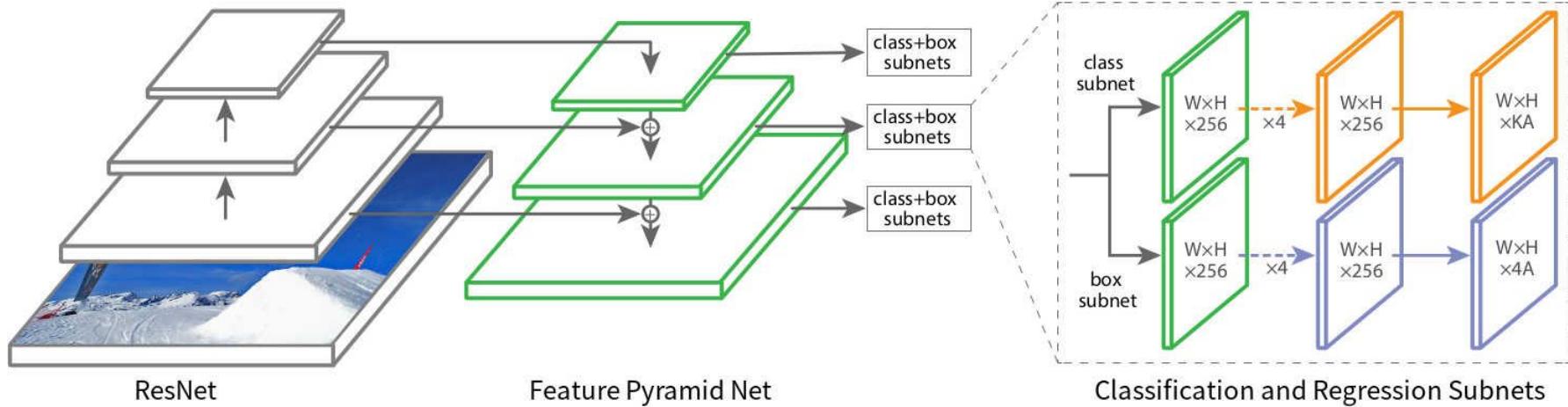
Random category

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?