# 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

Meetup

Start a new group | Explore | Messages


# Berlin Computer Vision Group

Berlin, Germany

384 members · Public group

Organized by **Antonio Rueda Toicen**

Share:

About | Events | Members | Photos | Discussions | More

Manage group | Create event

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

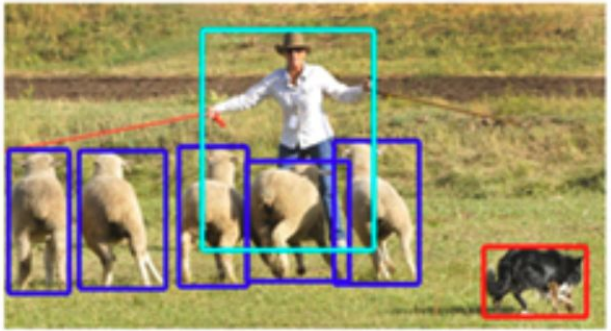
# About you

- 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

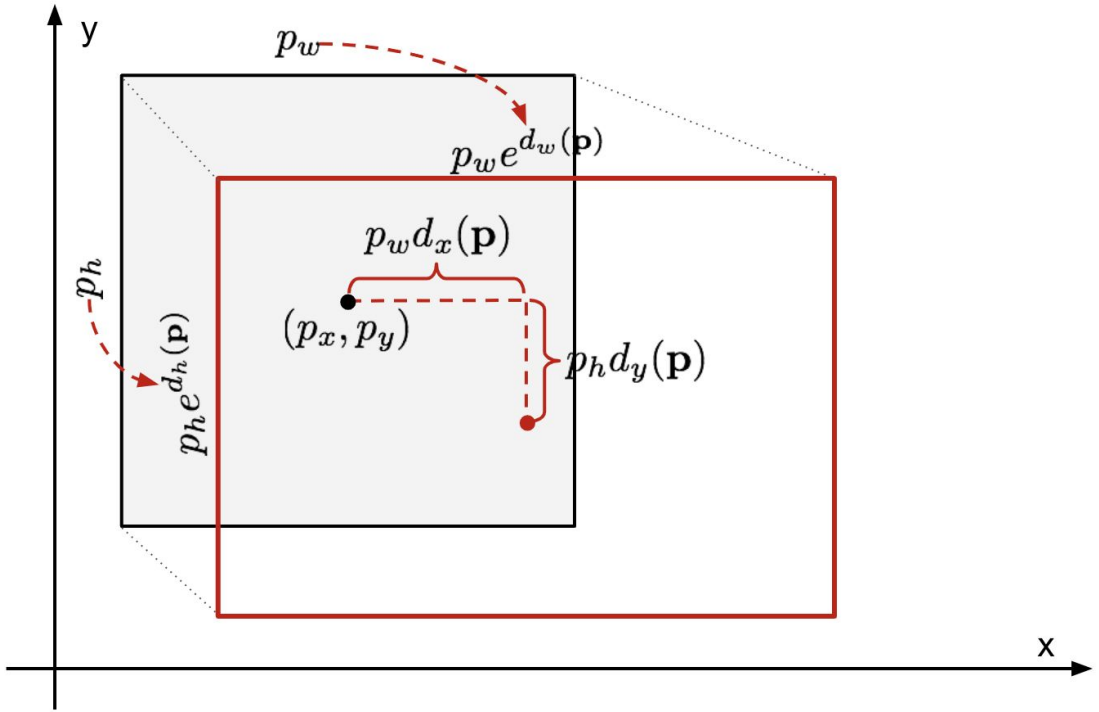
Classification refers to image-wide labels
Detection refers to localization of bounding boxes with labels
Segmentation refers to pixel-wise localization of the labels

# Goals of object detection

Given an input image we wish to obtain:

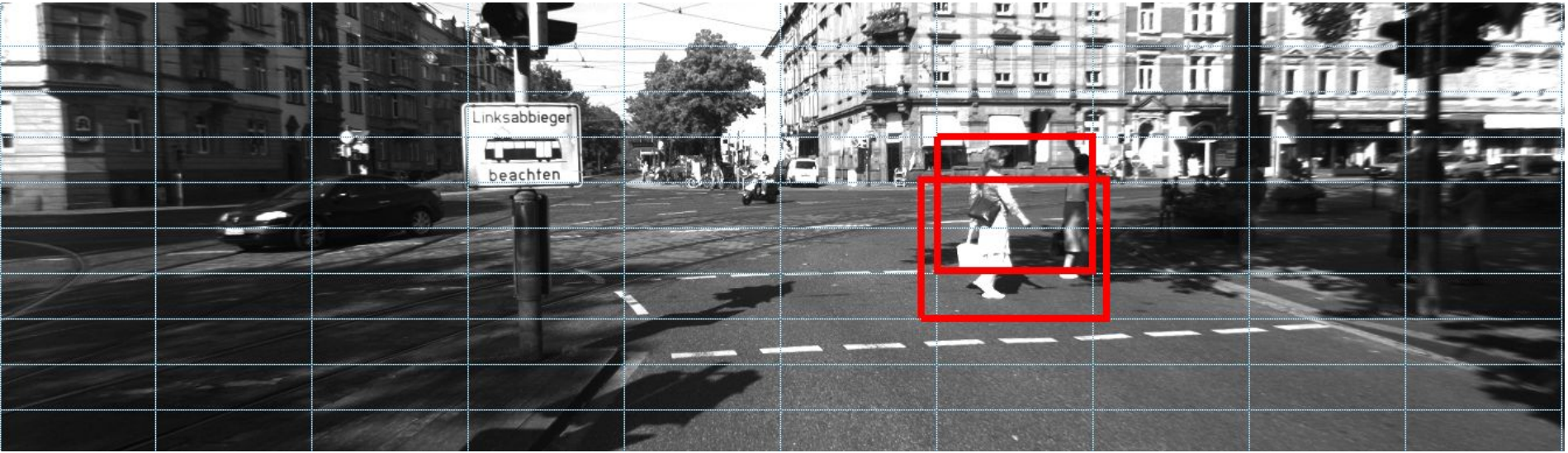
1. A **list of bounding boxes,** or the (*x*, *y*)-coordinates for each object in an image
2. A **class label** associated with each bounding box
3. The **probability** score associated with each bounding box and class label

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

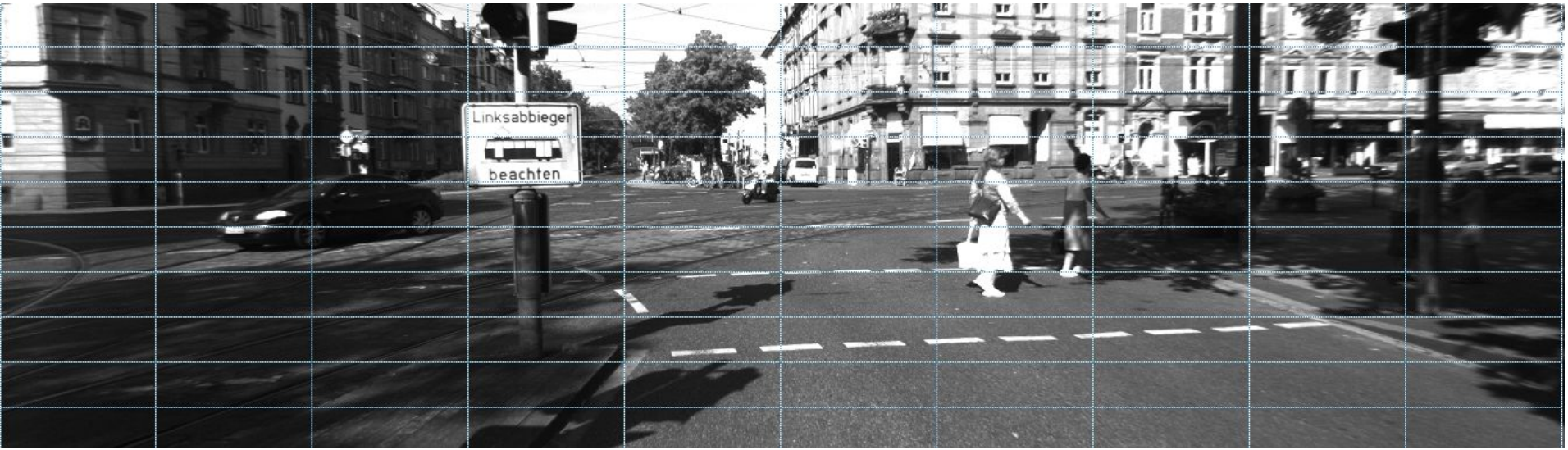



‘Regression’ means predicting a continuous value

# Breaking down an image into regions

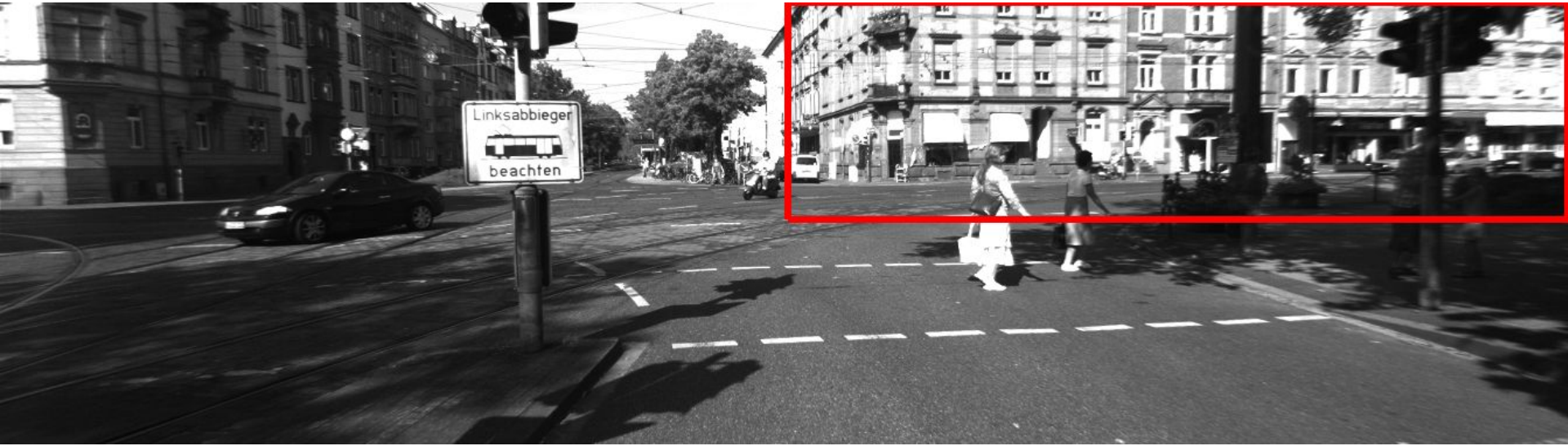


# Breaking down an image into regions

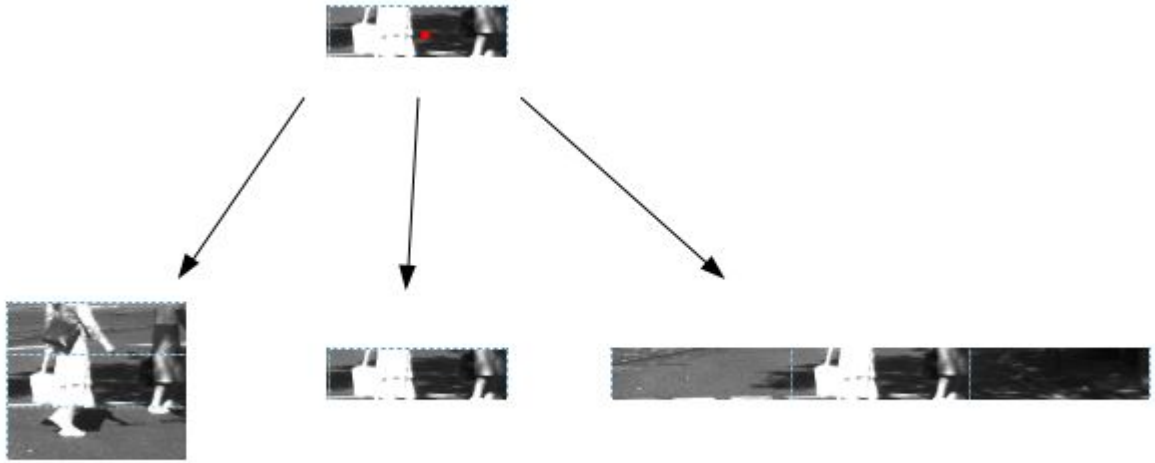



A sliding window approach partitions the image into discrete blocks

# Breaking down an image into regions

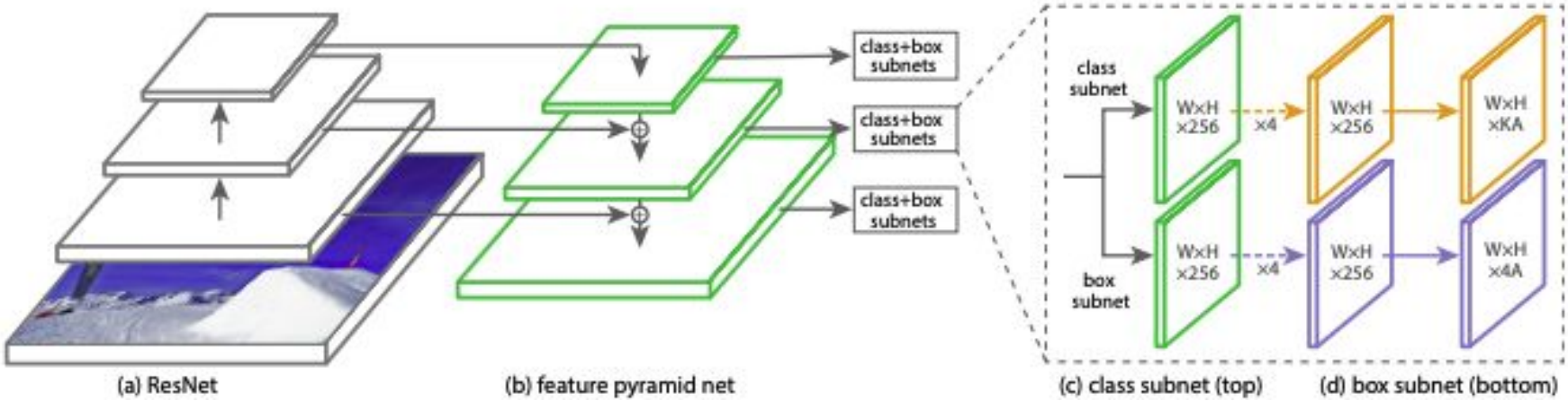


# Breaking down an image into regions

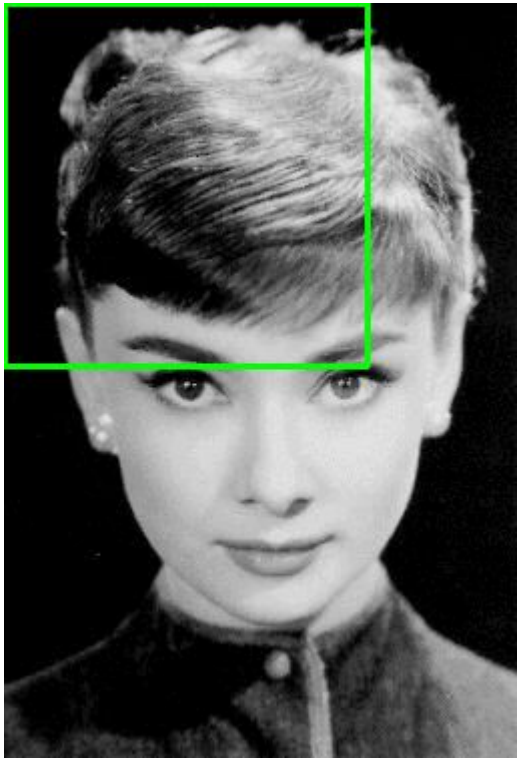


Finding the optimal image partitioning algorithm is not a trivial task

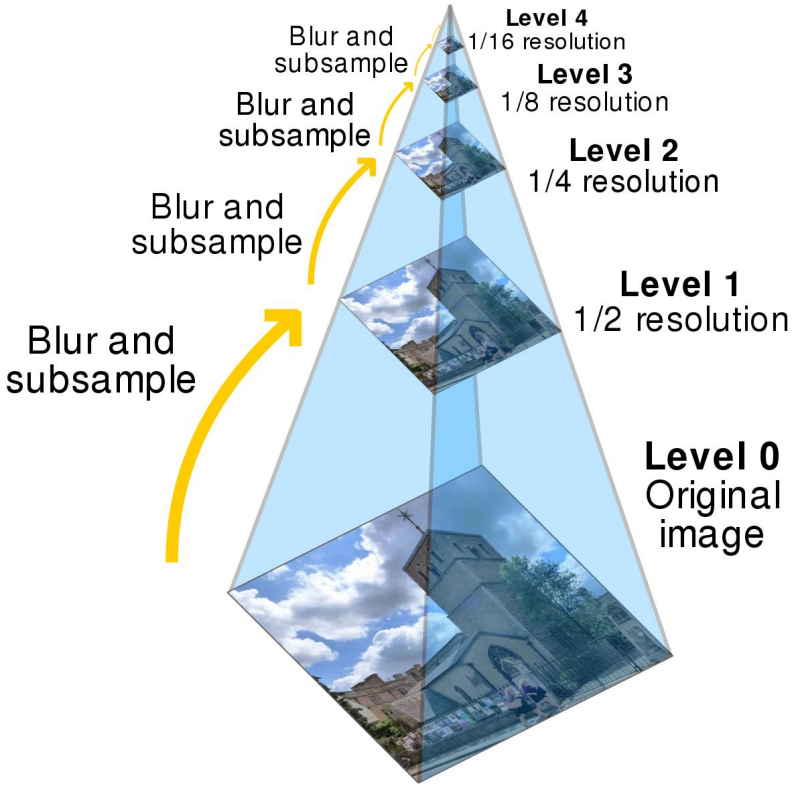
# RetinaNet's architecture



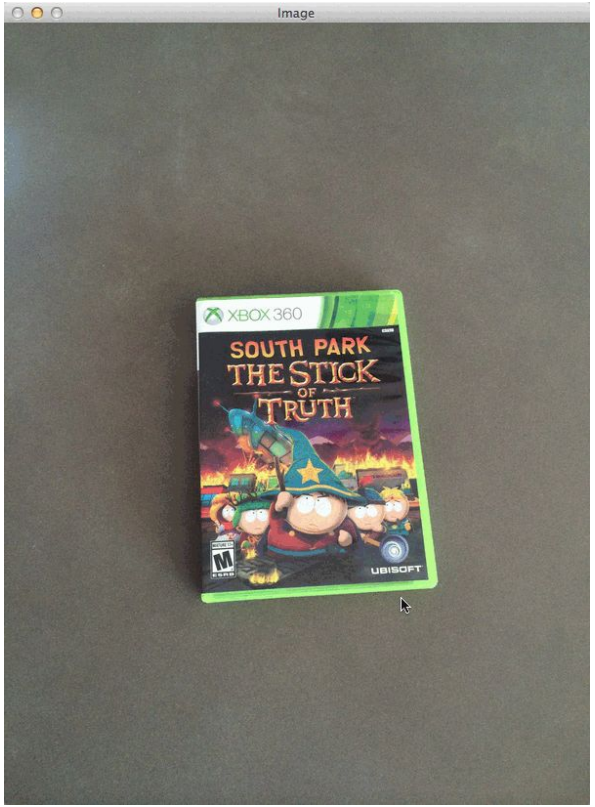
# Sliding windows



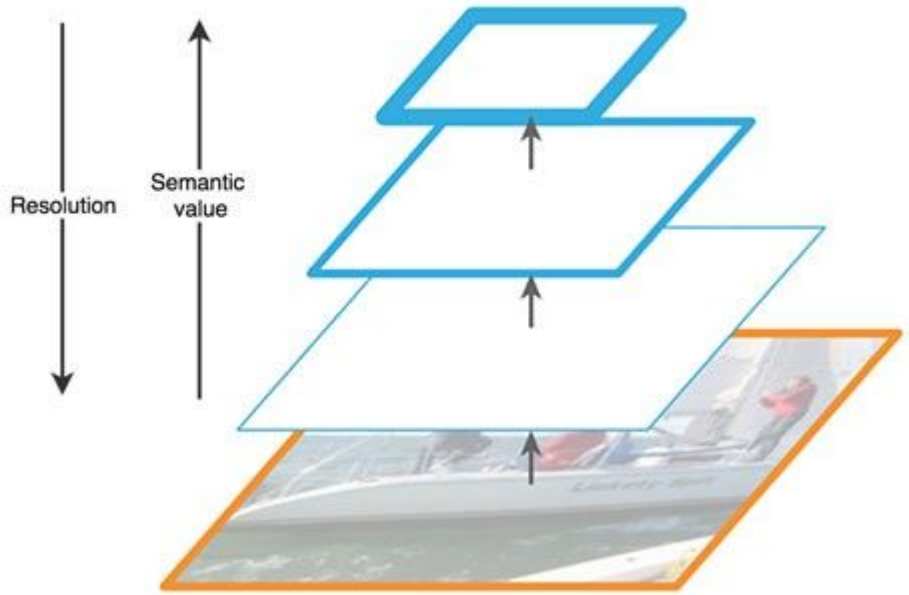
# Image pyramids



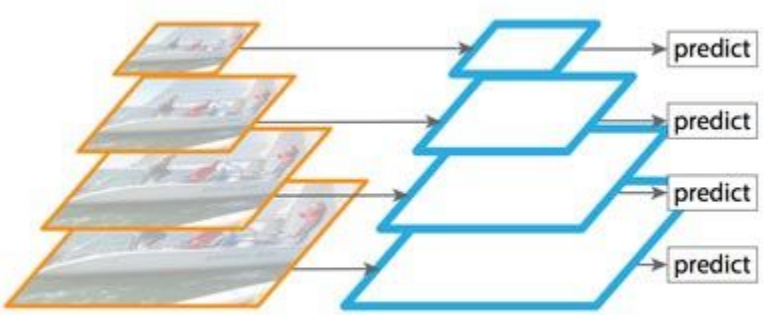
# Sliding windows on an image pyramid



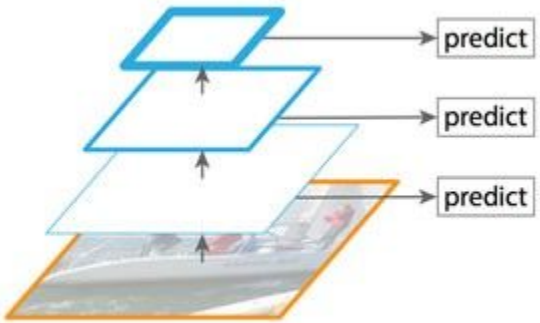
# Image Pyramids



# Image Pyramids

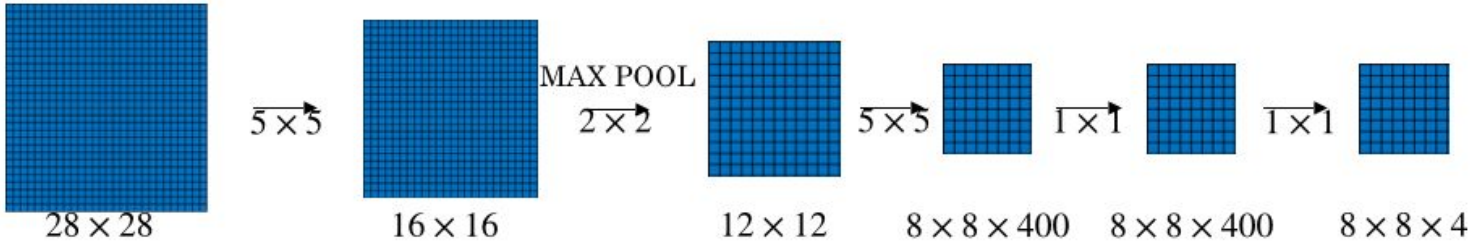


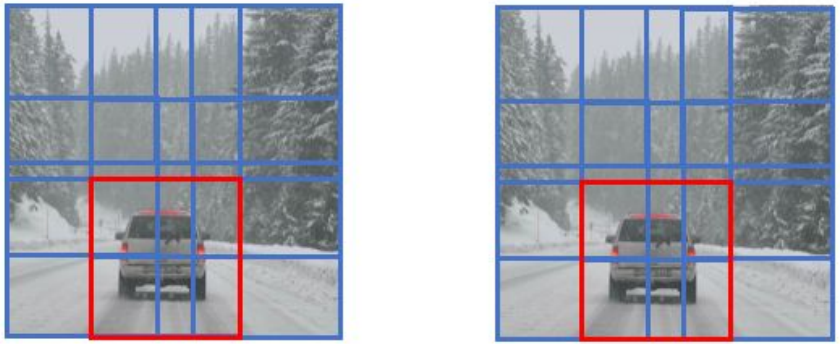

Pyramid of images




Pyramid of feature maps

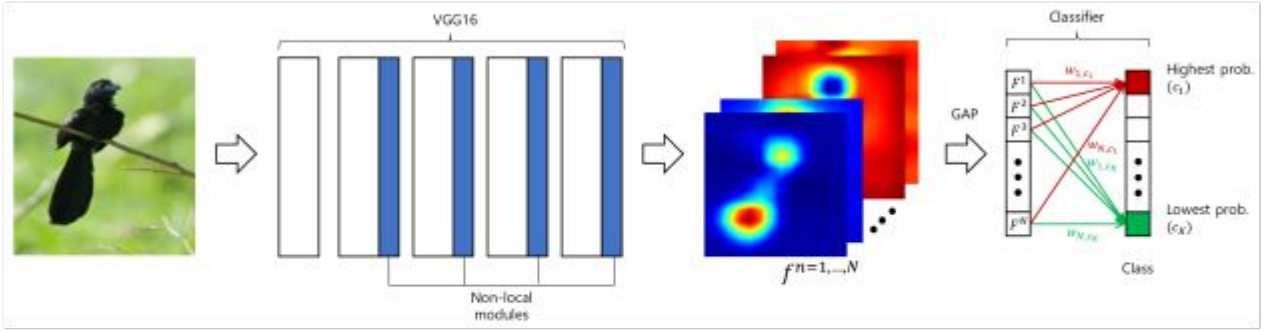
# Pyramids and sliding windows in CNNs



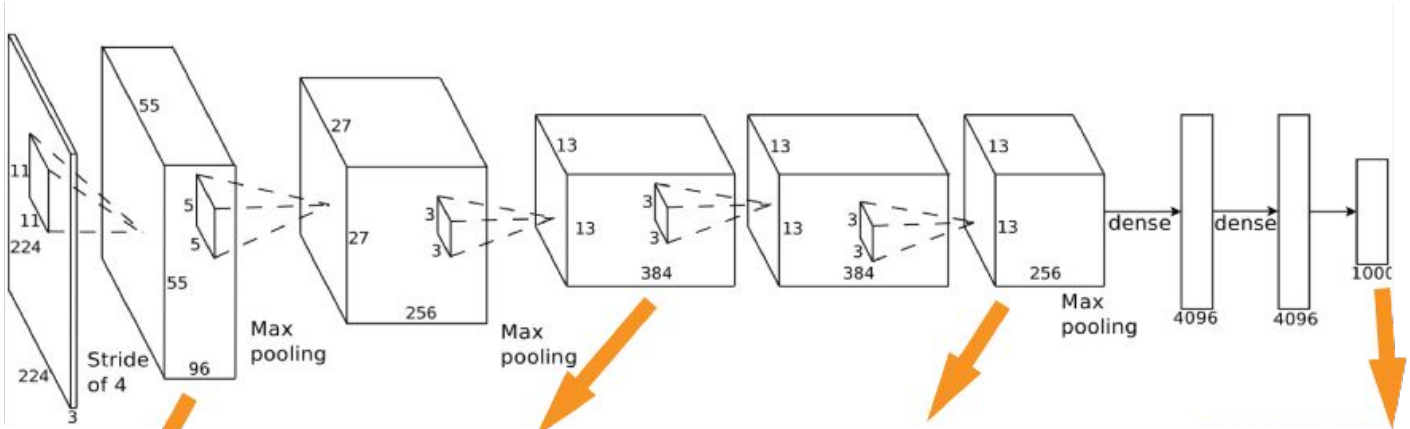
Andrew Ng

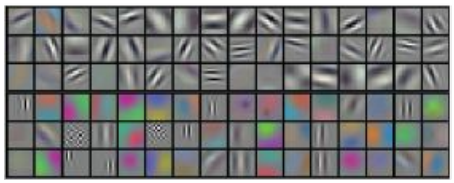
# Transfer learning of classification is currently used for feature extraction



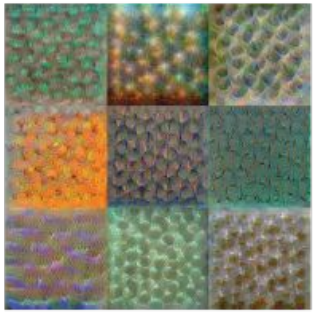

Source

# What convolutional networks learn



Conv 1: Edge+Blob



Conv 3: Texture

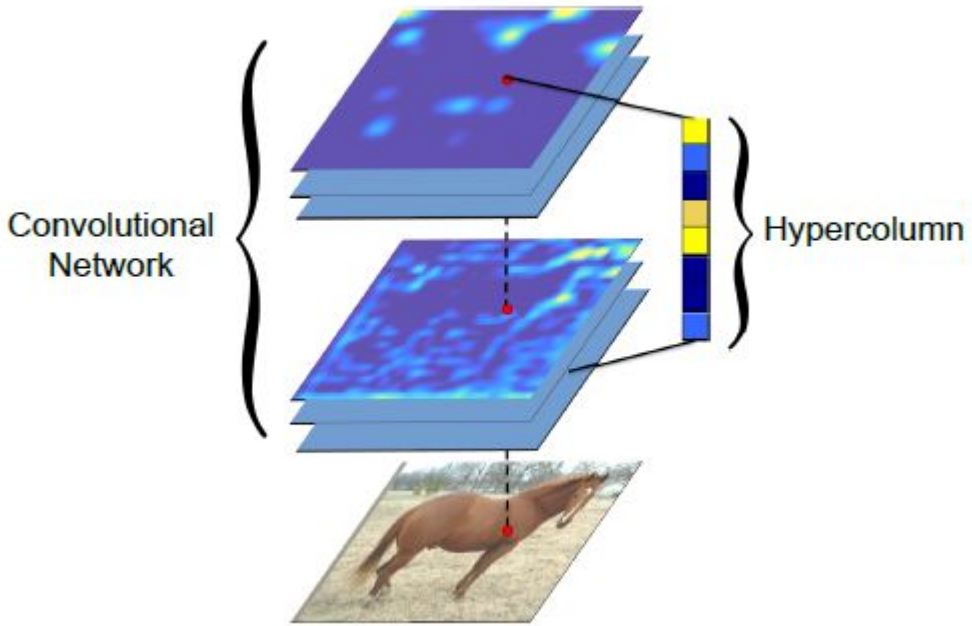
Numerical

Data-driven

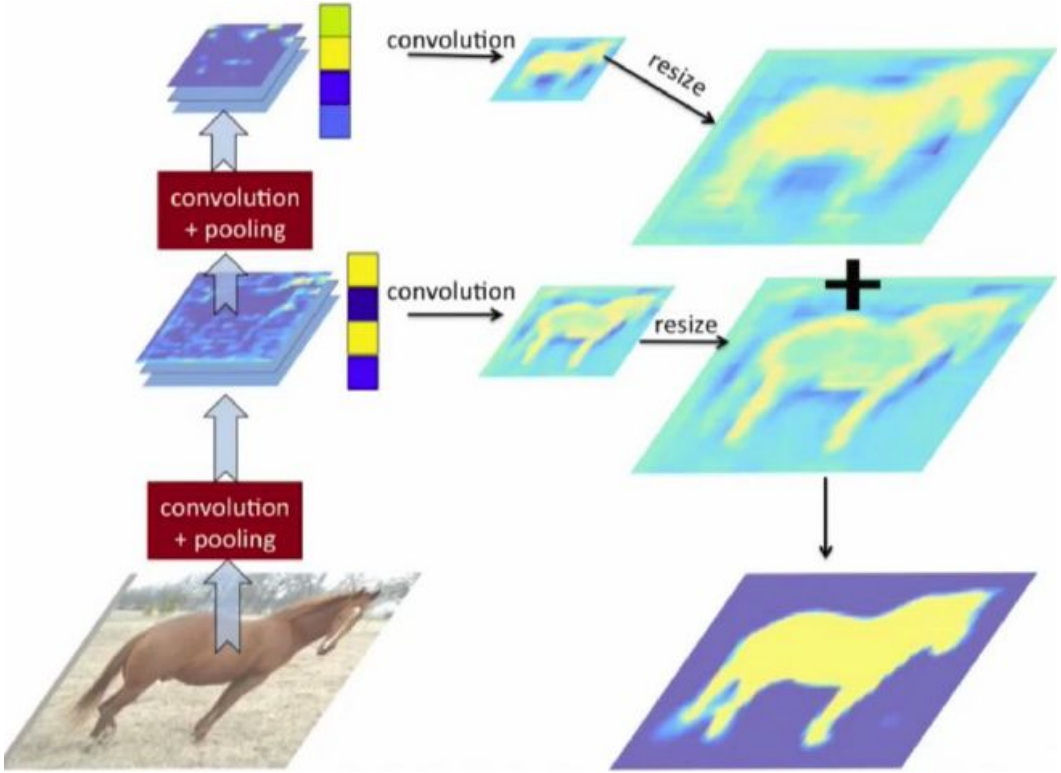
Conv 5: Object Parts


Fc8: Object Classes

# “Hypercolumns”



# “Hypercolumns”



# Anchor boxes

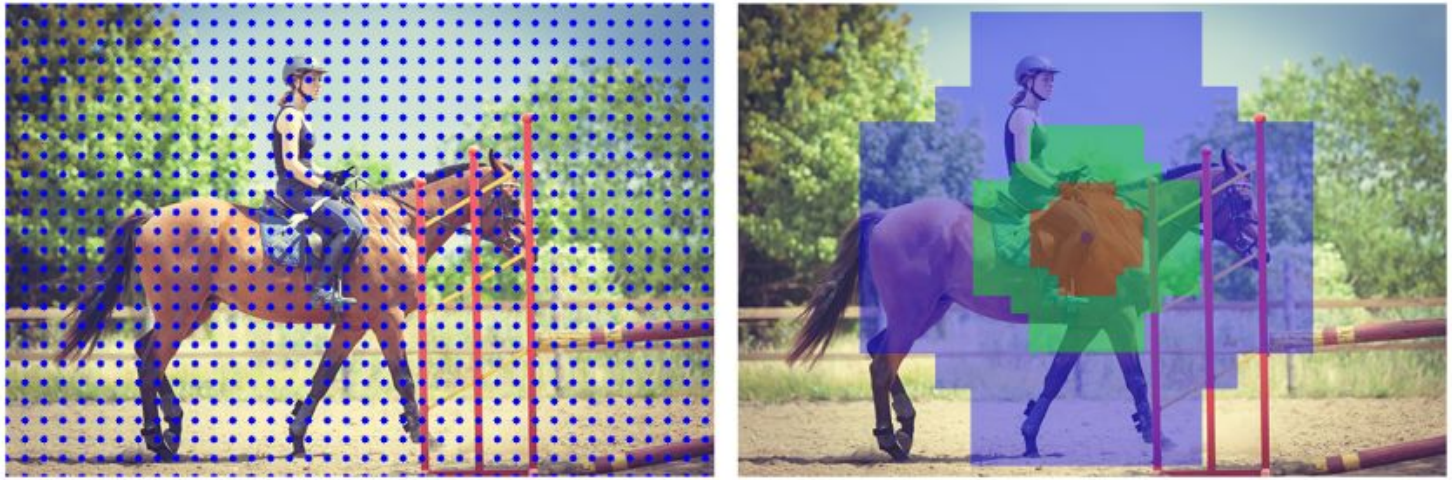
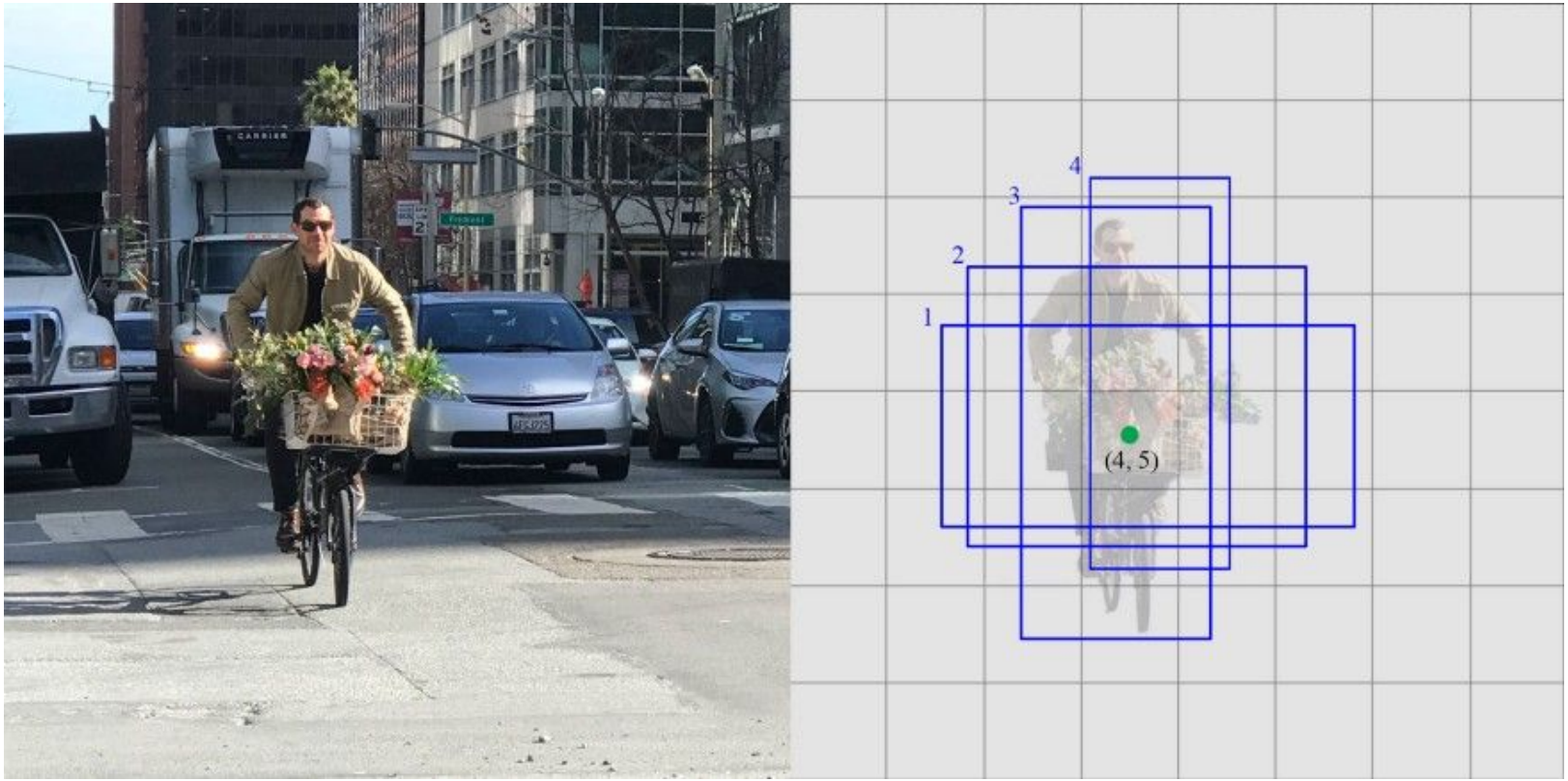


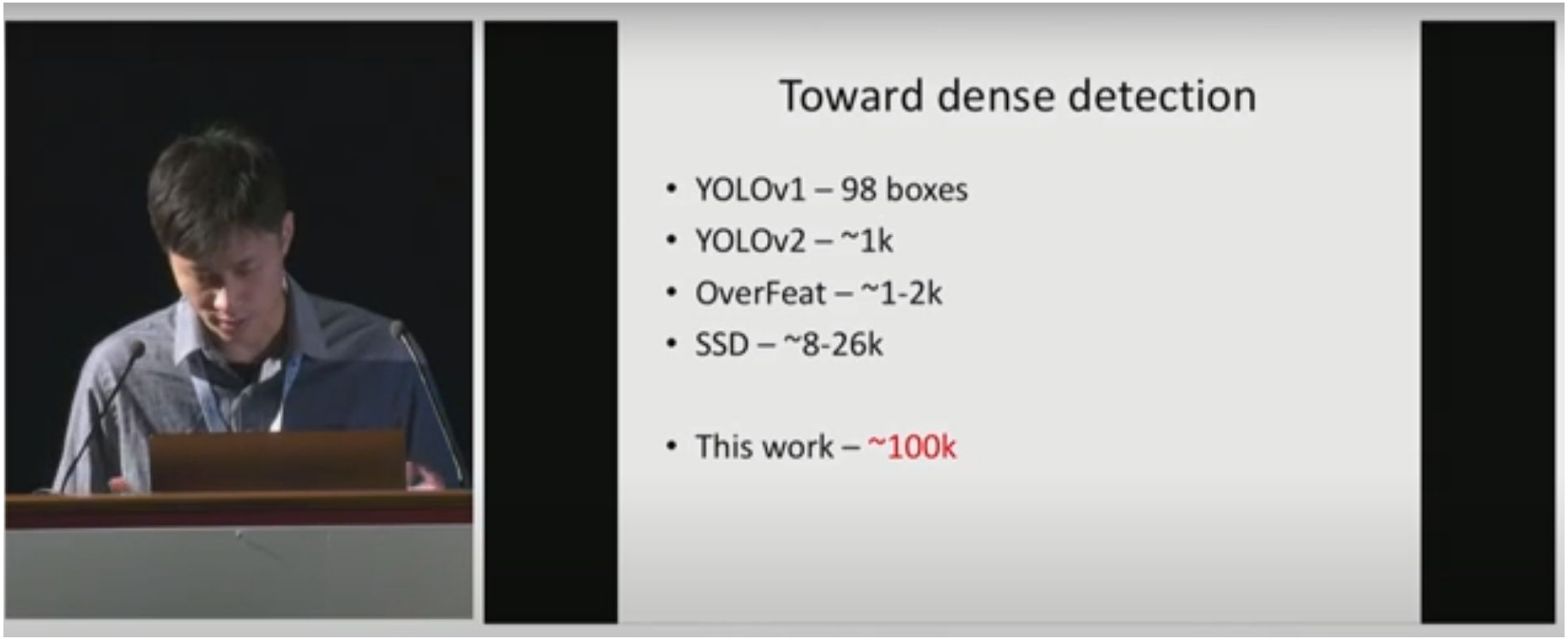
Figure 14.7: **Left:** Creating anchors starts with the process of sampling the coordinates of an image every $r$ pixels ($r = 16$ in the original Faster R-CNN implementation). **Right:** We create a total of nine anchors centered around *each* sampled $(x, y)$-coordinate. In this visualization, $x = 300, y = 200$ (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



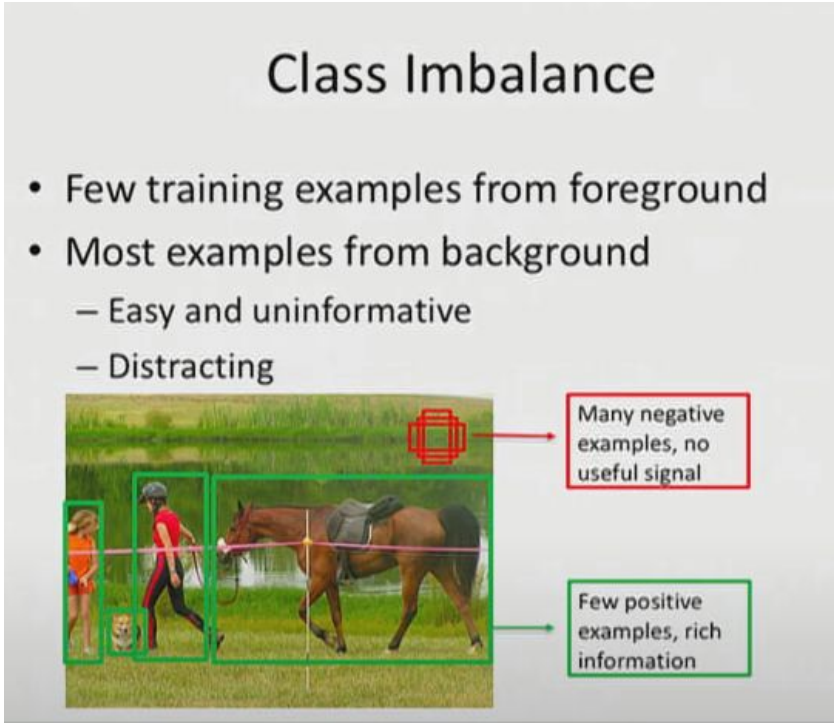
# Anchor boxes

# RetinaNet's high number of anchor boxes



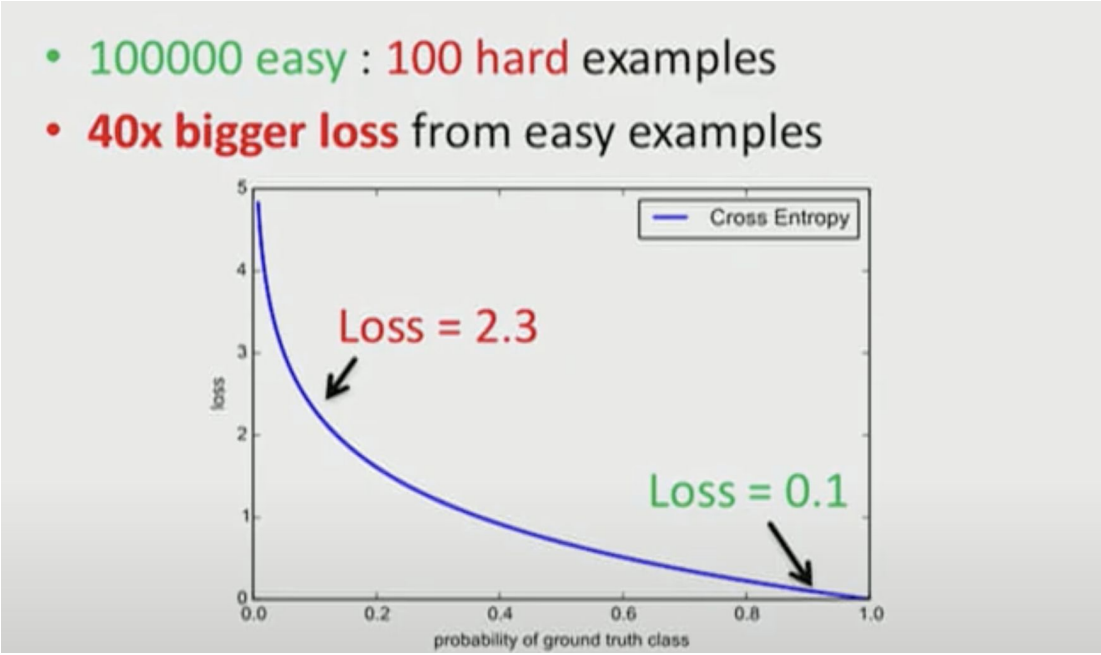

Focal Loss for Dense Object Detection (CVPR 2017 Presentation)

# Retinanet’s solution to the class imbalance problem




RetinaNet: how Focal Loss fixes Single-Shot Detection

# Retinanet's solution to the class imbalance problem



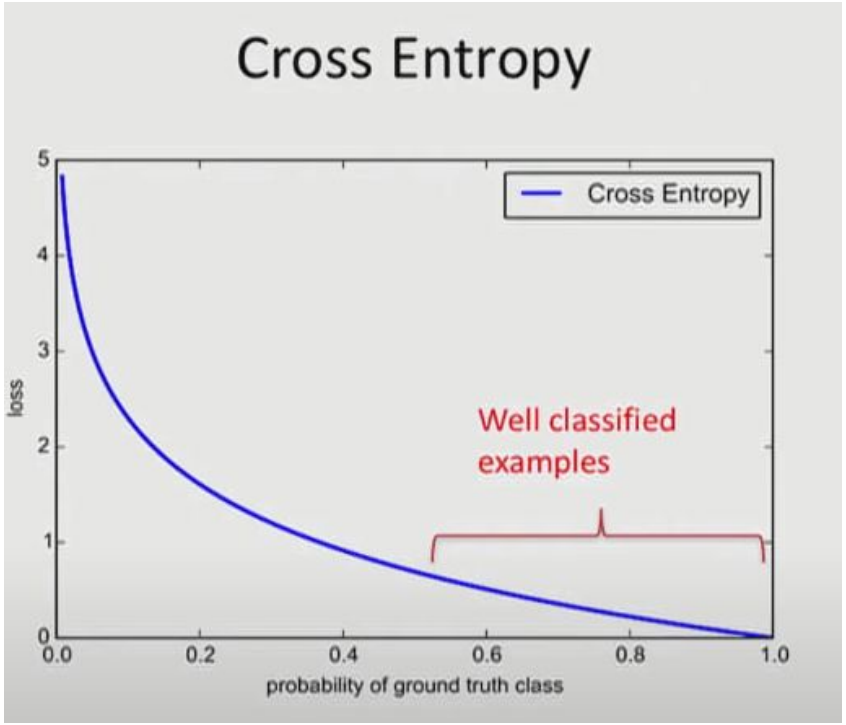

RetinaNet: how Focal Loss fixes Single-Shot Detection

# 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

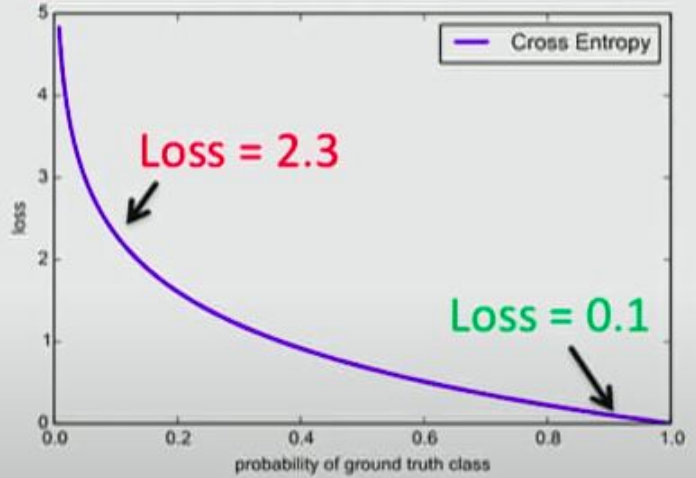
# Cross entropy at different confidence levels



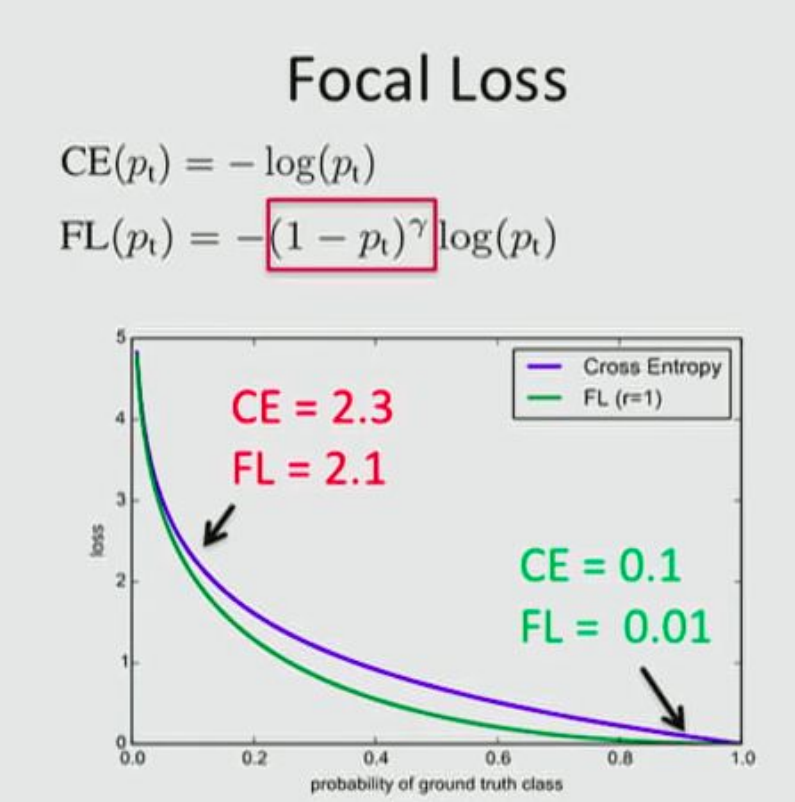

Andrew Ng's explanation of cross entropy loss

# Regular cross entropy gives too much importance to easy cases

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



# Retinanet's solution to the class imbalance problem



# Focal loss as modified cross-entropy loss

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

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

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

[Colab notebook to play with these equations]

# Tweaking gamma $\gamma$

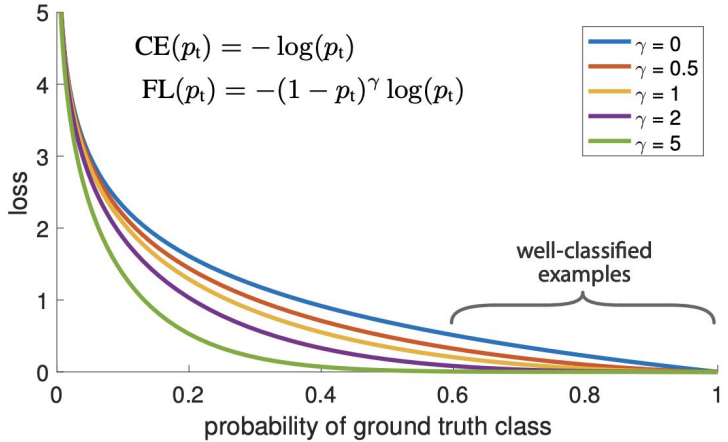
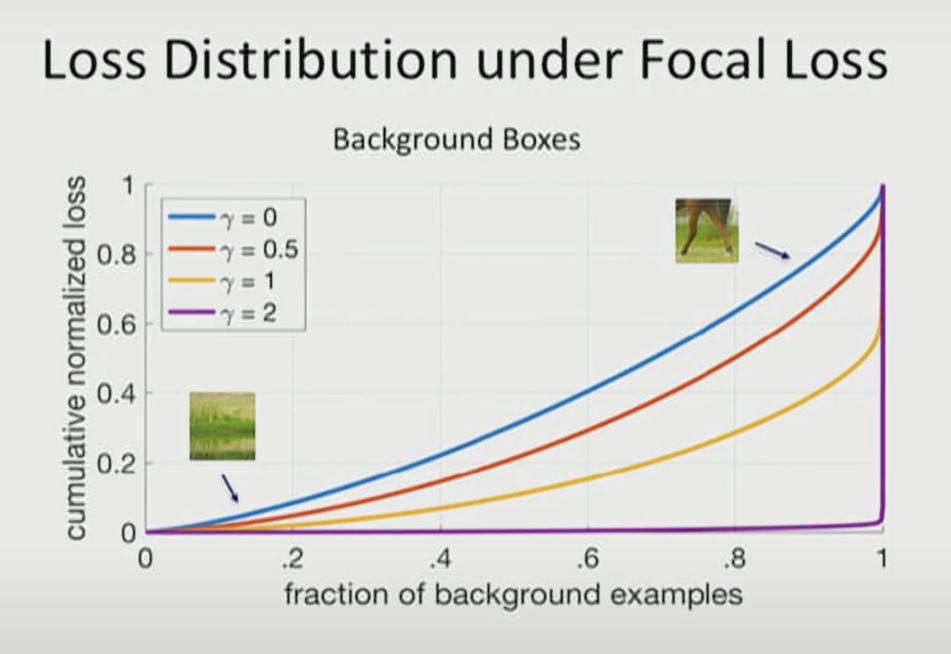



Figure 1. We propose a novel loss we term the *Focal Loss* that adds a factor $(1 - p_t)^{\gamma}$ to the standard cross entropy criterion. Setting $\gamma > 0$ reduces the relative loss for well-classified examples ($p_t > .5$), putting more focus on hard, misclassified examples. As our experiments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

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

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

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:

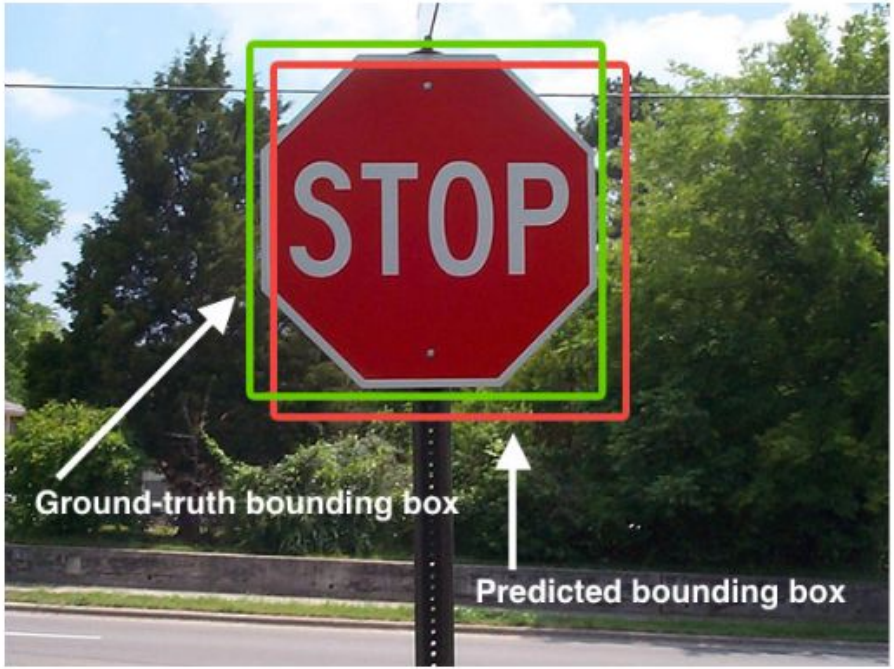
$$L_{1;smooth} = \begin{cases} |x| & \text{if } |x| > \alpha; \\ \frac{1}{|\alpha|}x^2 & \text{if } |x| \leq \alpha \end{cases}$$

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

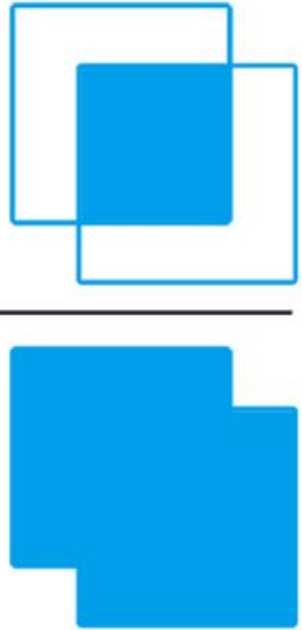
Smooth L1-loss combines the advantages of L1-loss (steady gradients for large values of $x$) and L2-loss (less oscillations during updates when $x$ is small).

https://stats.stackexchange.com/questions/351874/how-to-interpret-smooth-l1-loss

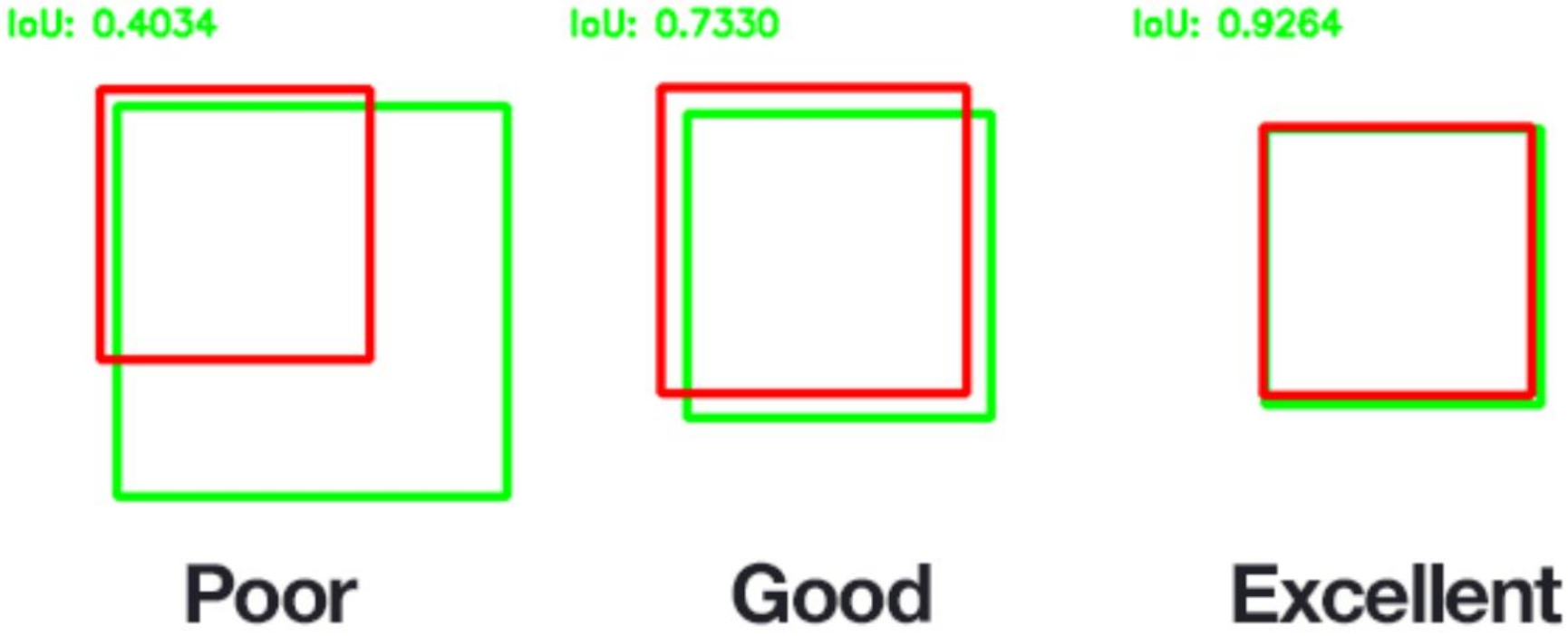
# Measuring performance with IoU



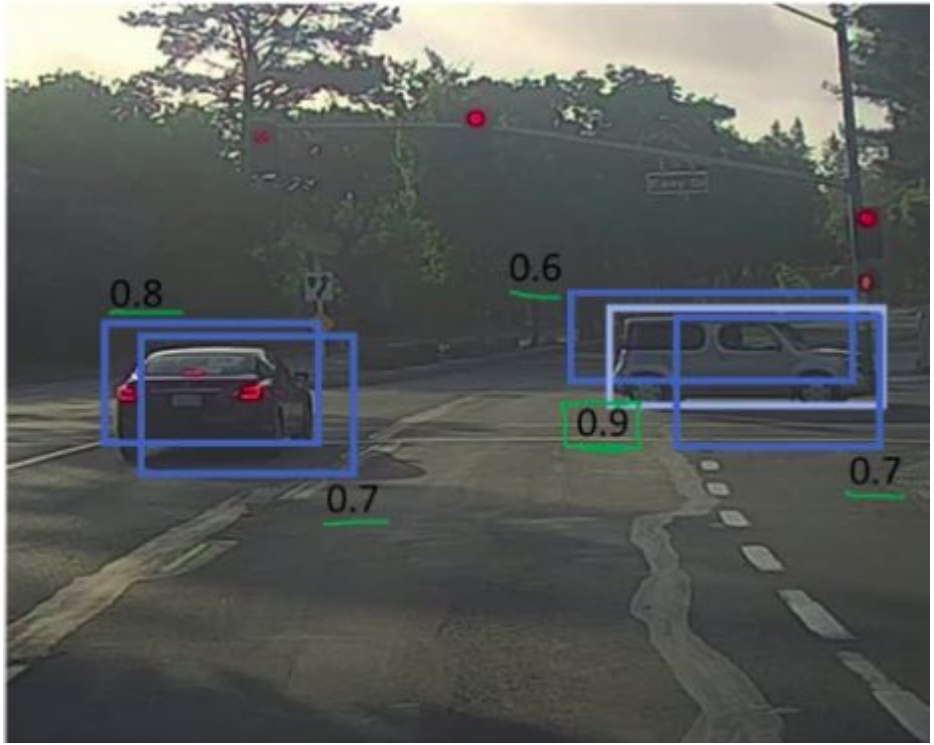

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



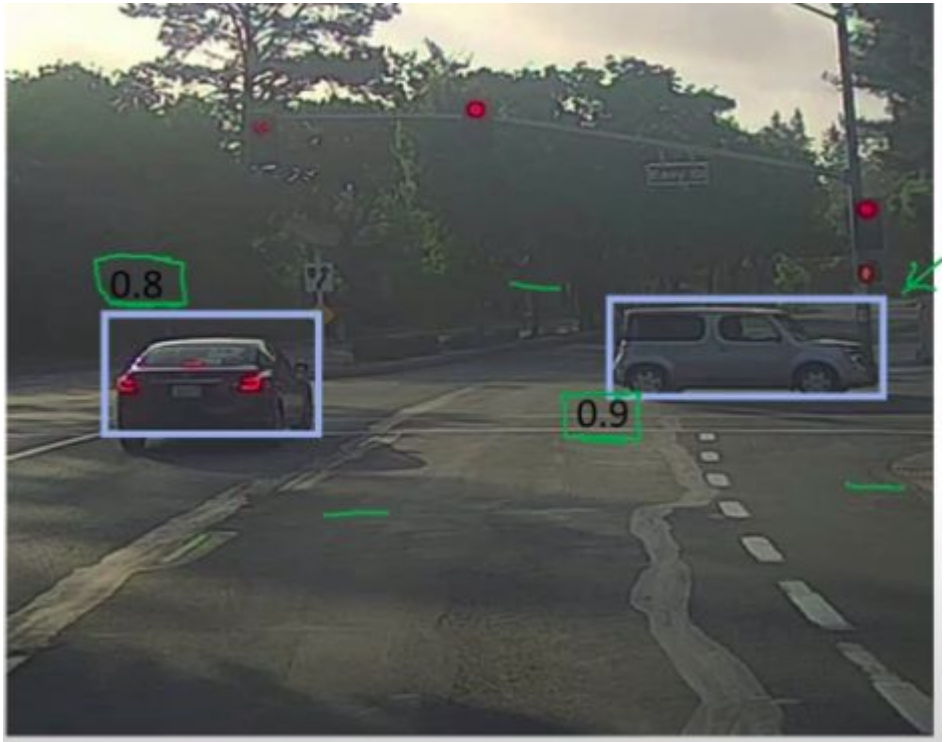
# Measuring performance with IoU



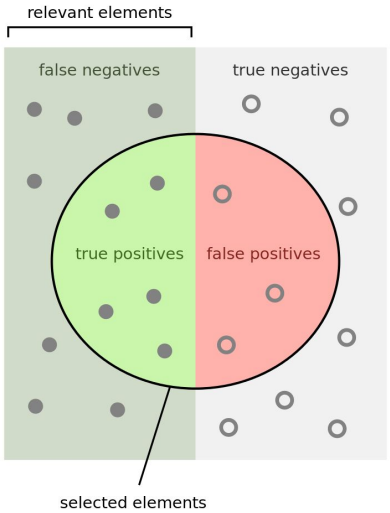
# Non-max suppression

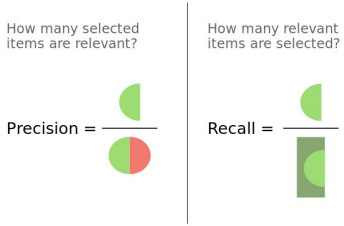


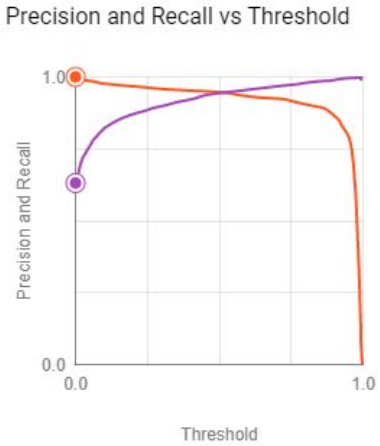
# Non-max suppression



# Precision, recall, and f1 metrics




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

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)

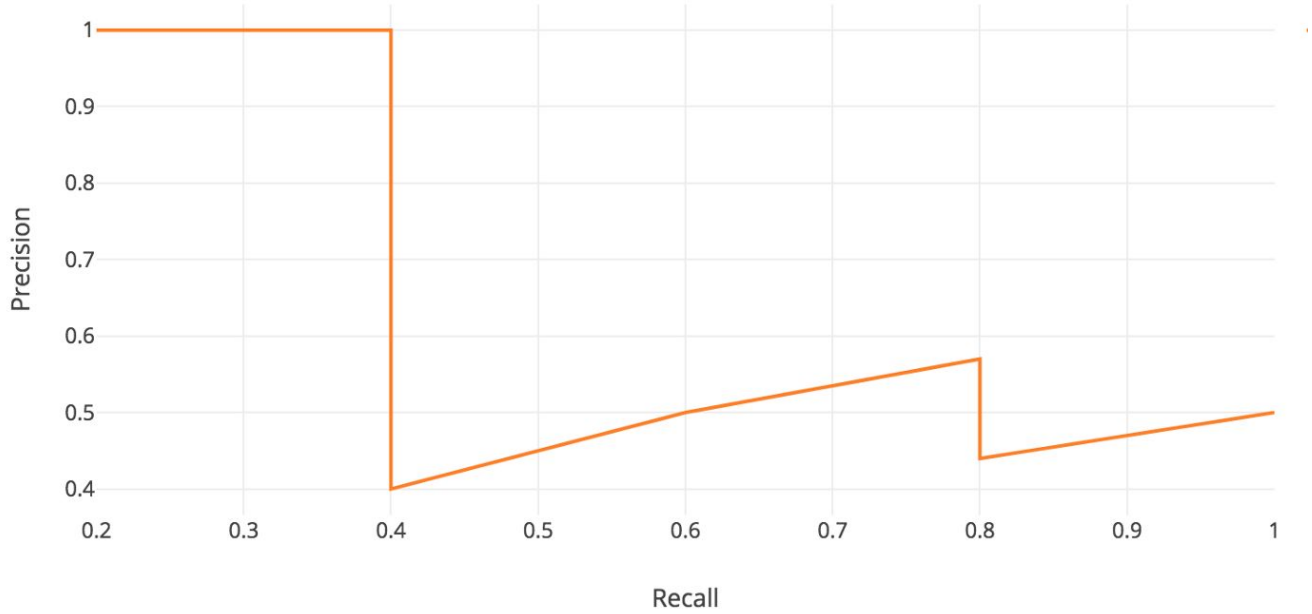
$$mAP = \frac{1}{n} \sum_{k=1}^{k=n} AP_k$$

$$AP_k = \text{the AP of class } k$$

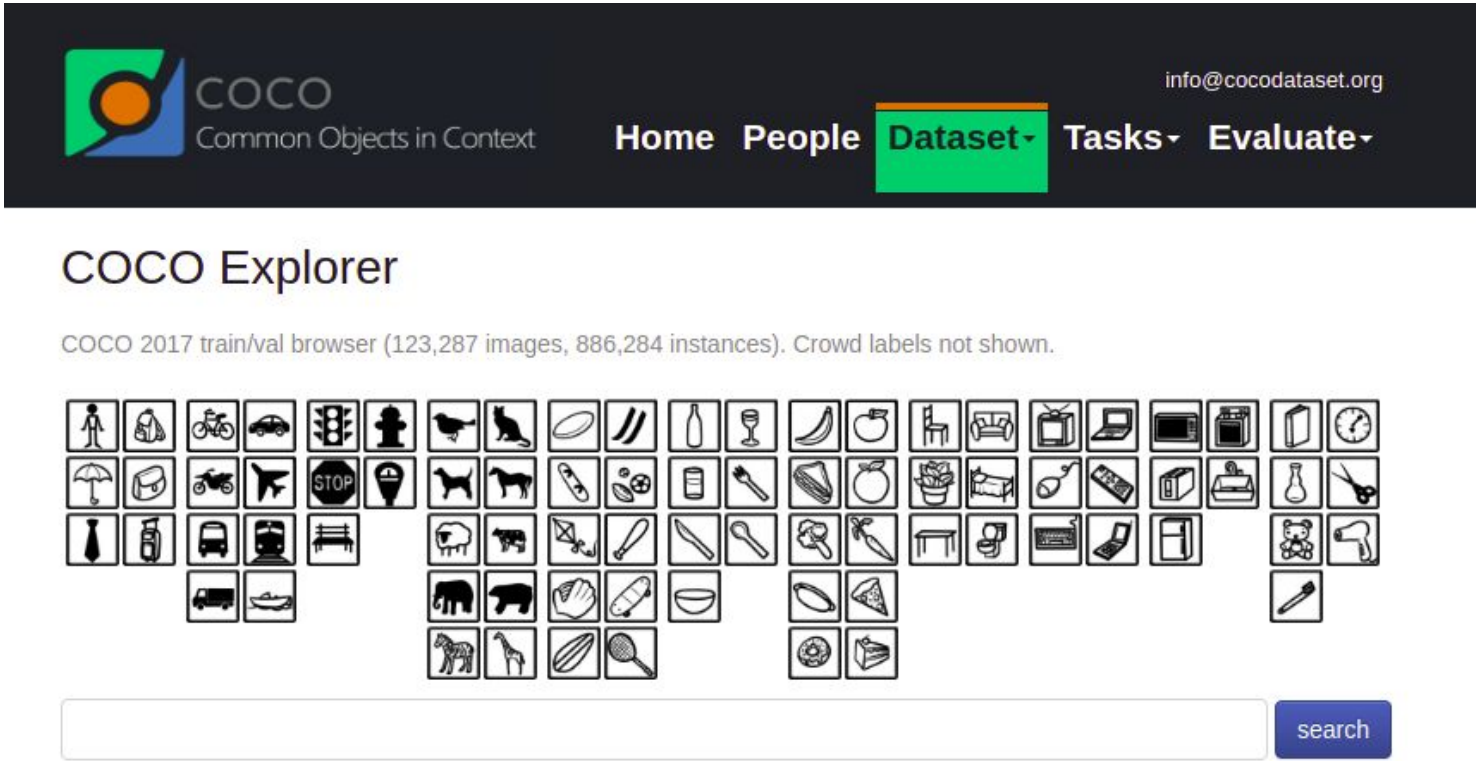
$$n = \text{the number of classes}$$

Mean Average Precision (mAP) Explained

# Understanding mean Average Precision (mAP)

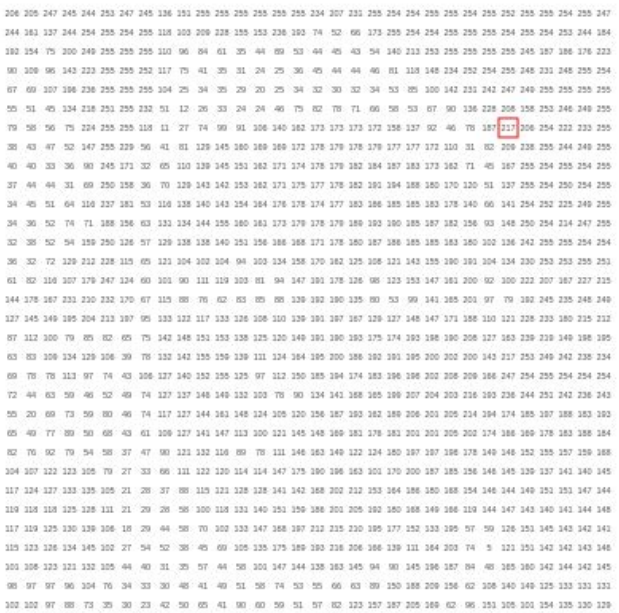



Mean Average Precision (mAP) Explained



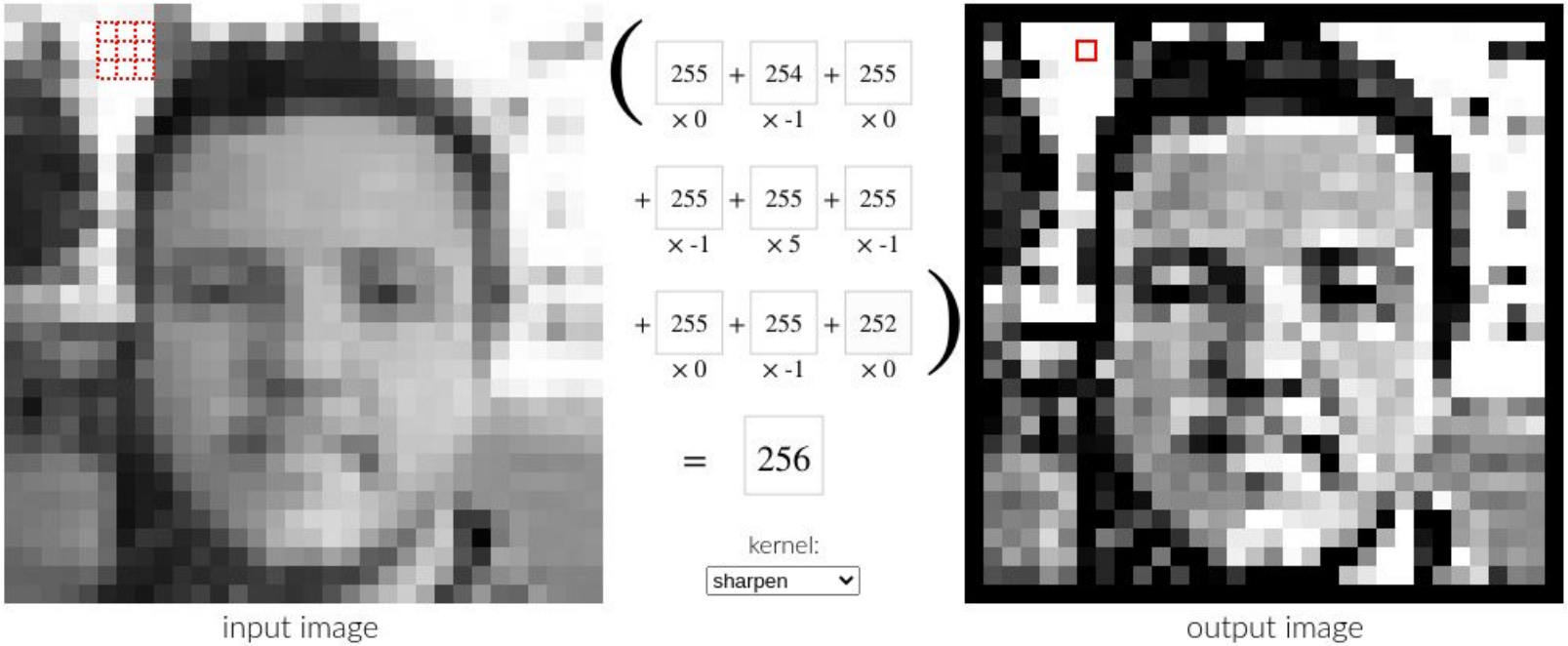

https://cocodataset.org/#explore

# Convolution kernels (aka filters)



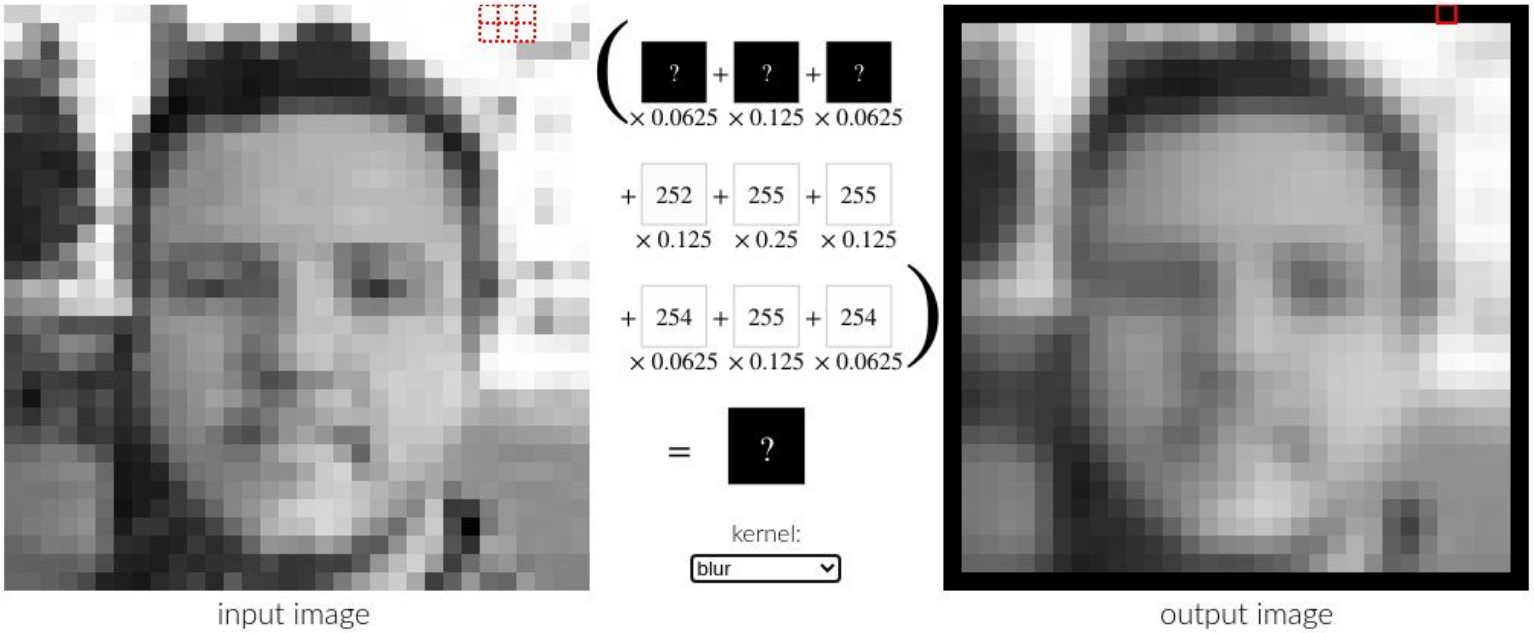
https://setosa.io/ev/image-kernels/

# What is a convolution filter?



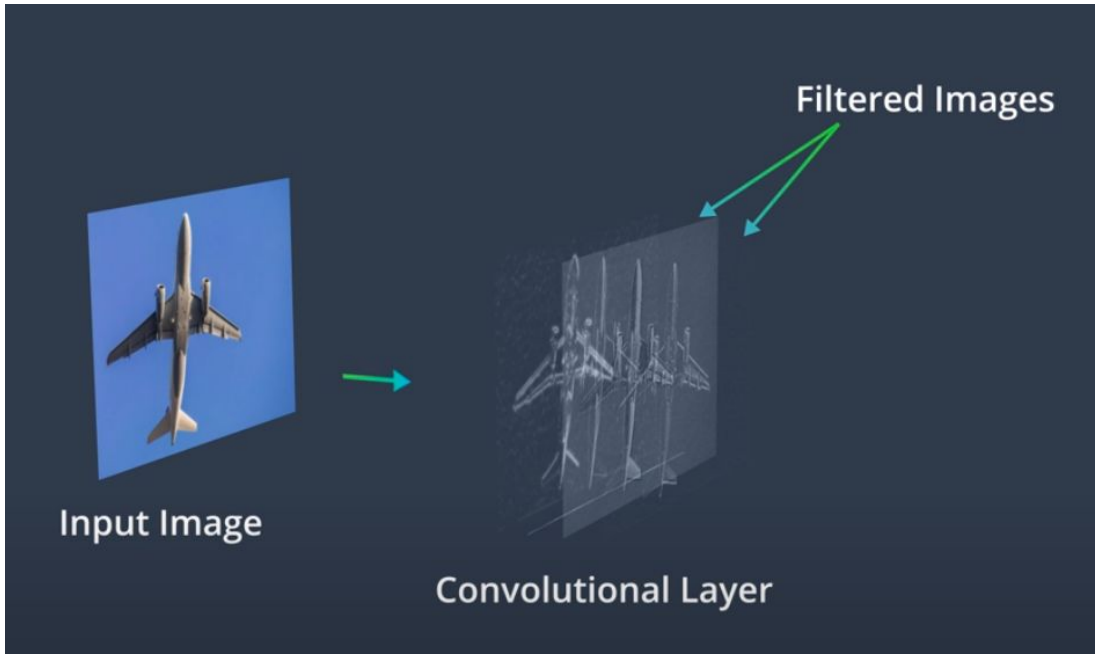

https://setosa.io/ev/image-kernels/

# What is a convolution filter?




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

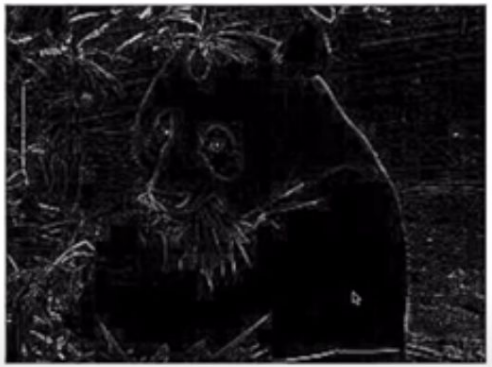
# What is a convolution filter?




Understanding the output of convolutional layers

# What is a convolution filter?

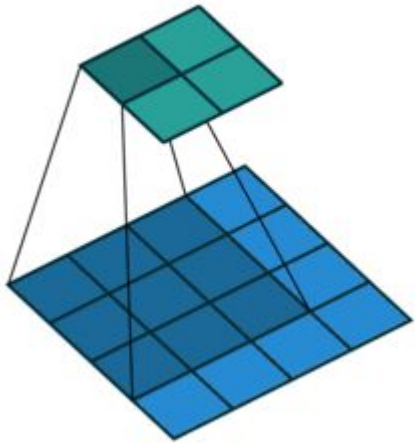
| | | |
|---|---|---|
| 0 | -1 | 0 |
| -1 | 4 | **-2** |
| 0 | -1 | 0 |



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

Visualizing convolution kernels

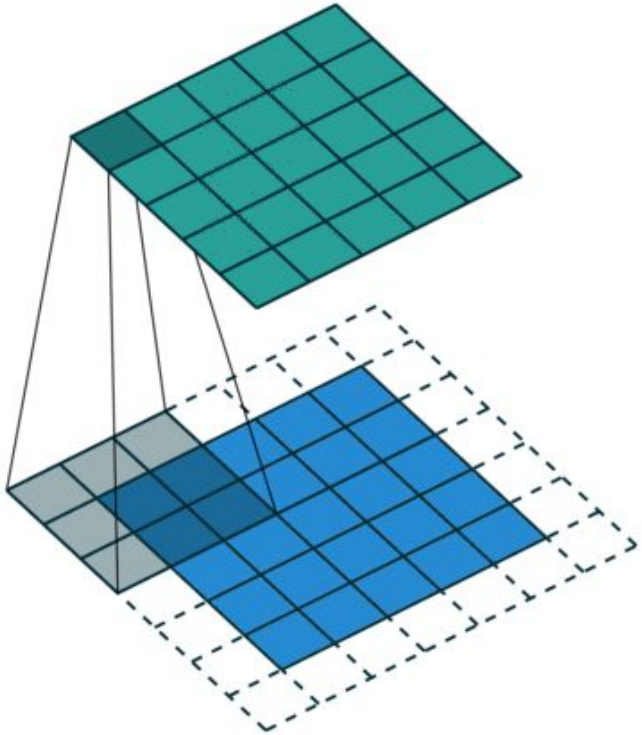
# What is a convolution filter?



Convolution of 3x3 and stride = 1 without padding

Effect: the output loses one pixel on each dimension

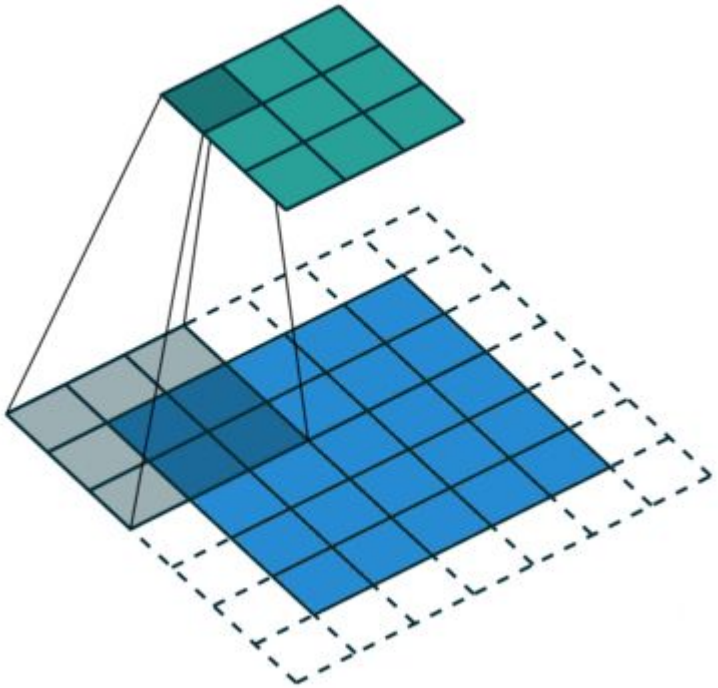
# What is a convolution filter?



Convolution of 3x3 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)

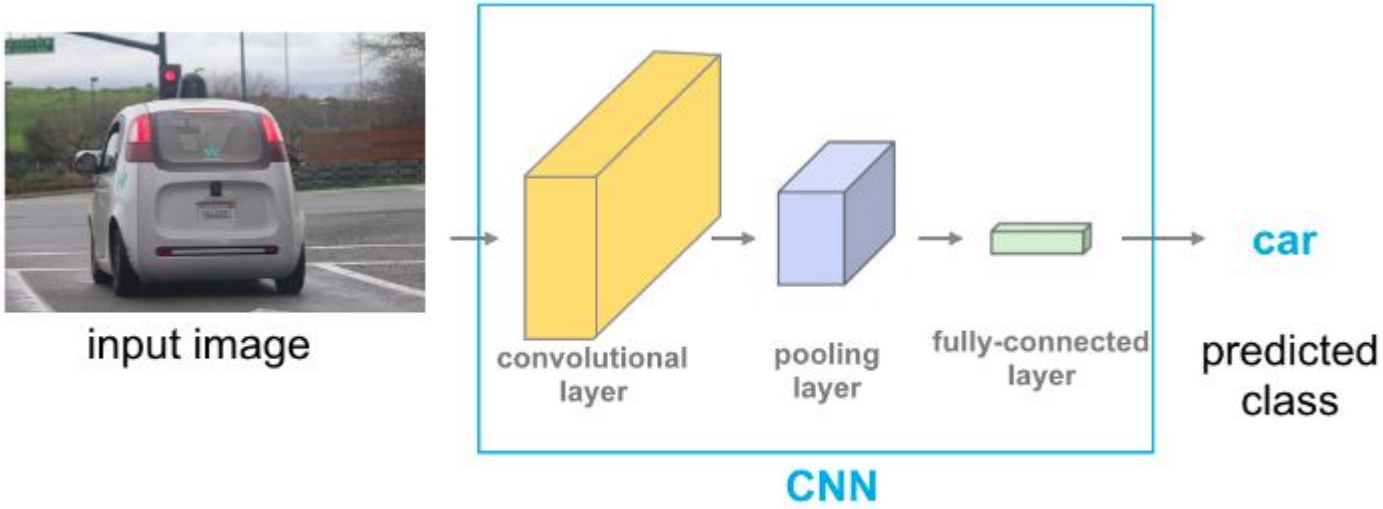
# Experiment: finding edges with convolutions

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

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

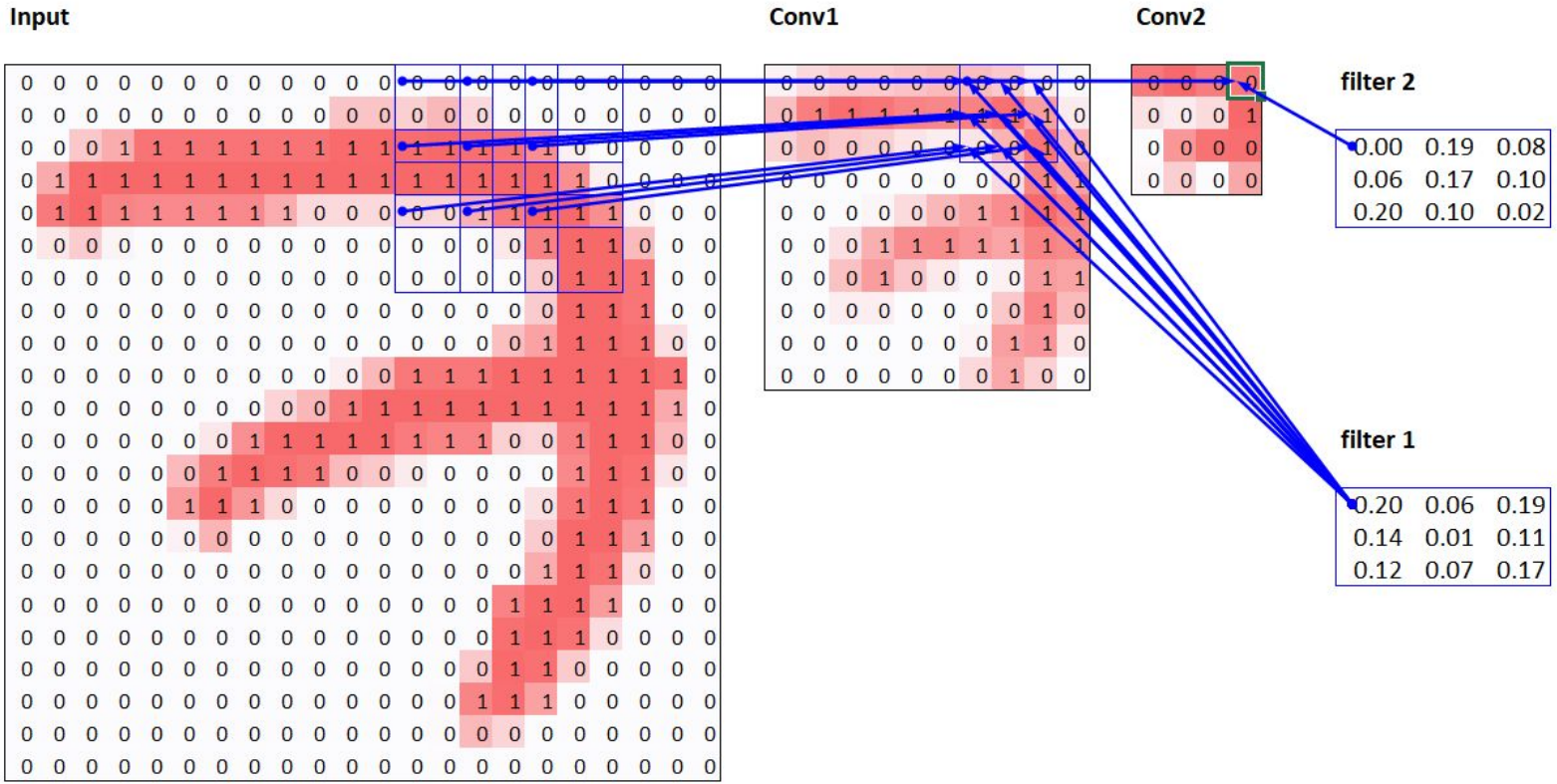
Finding edges with convolution kernels

# CNN architectures

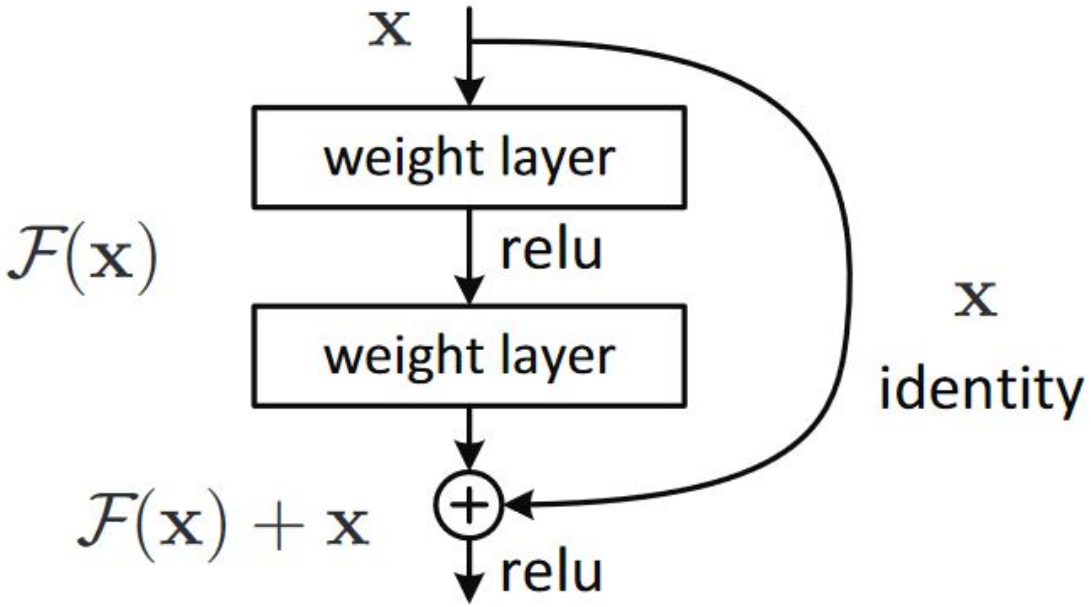



Layers in a CNN.

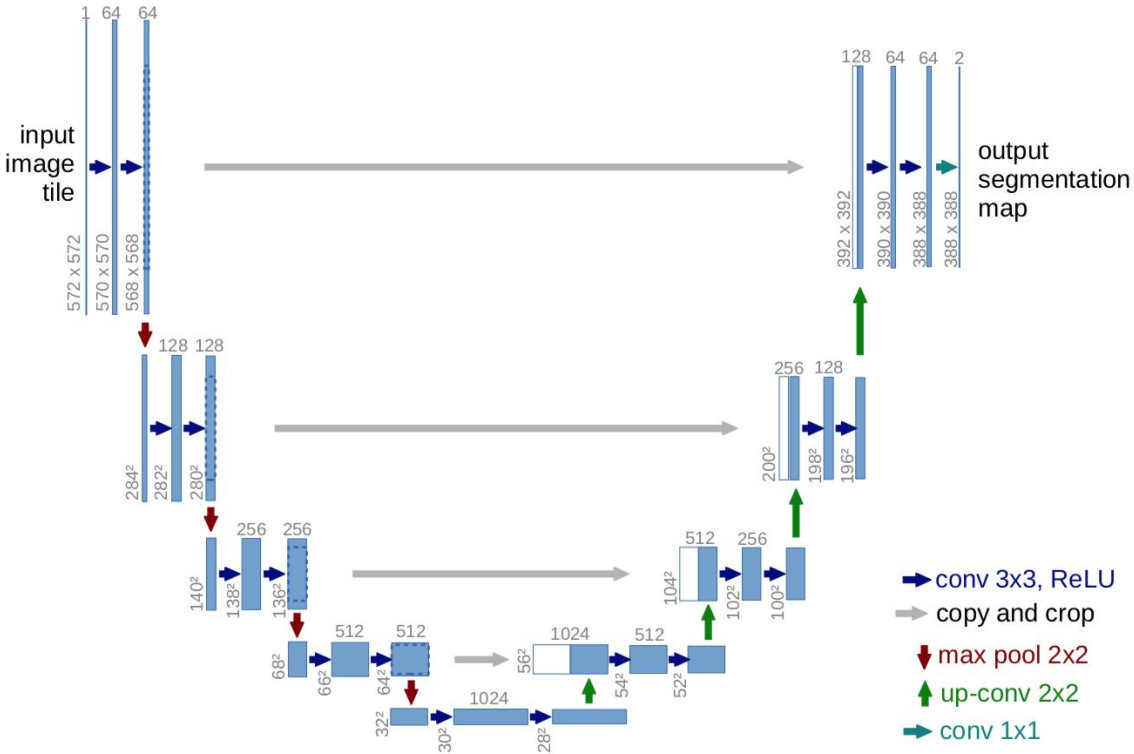
# Convolution kernels (aka filters)

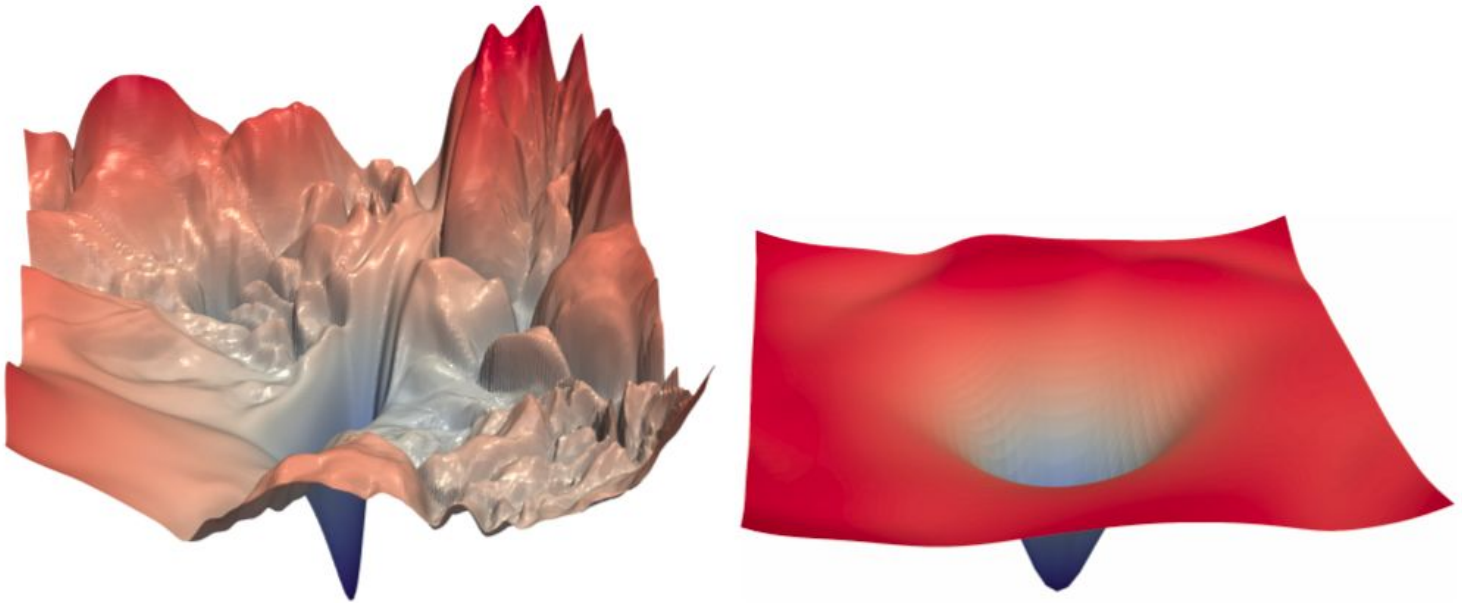


# Skip connections



# Skip connections





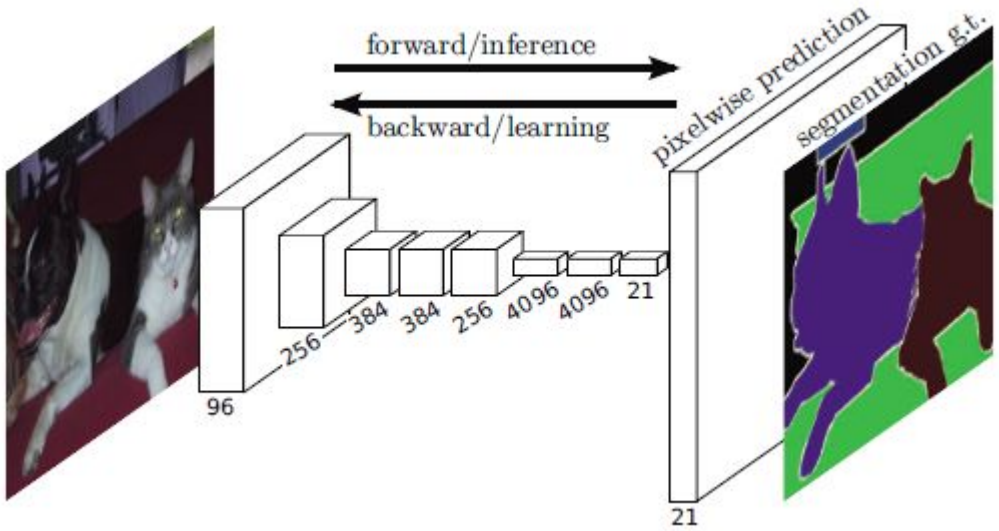
(a) without skip connections

(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.

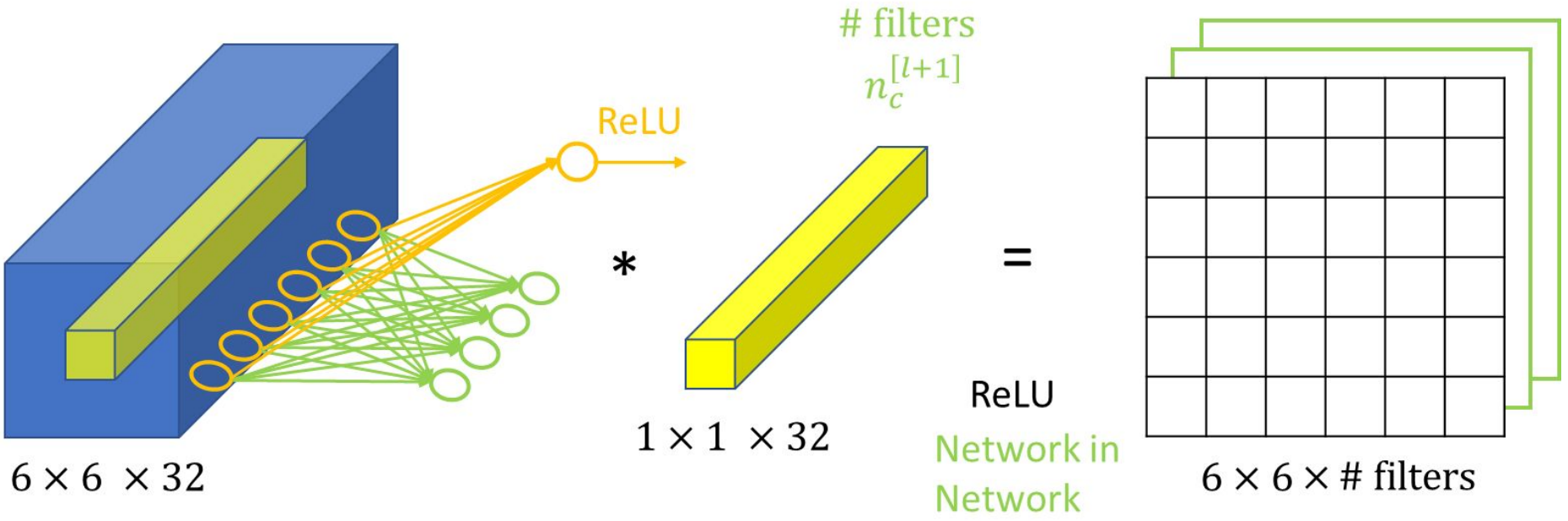
Visualizing the loss landscape of neural nets

# “Fully Convolutional Networks” allow pixelwise prediction



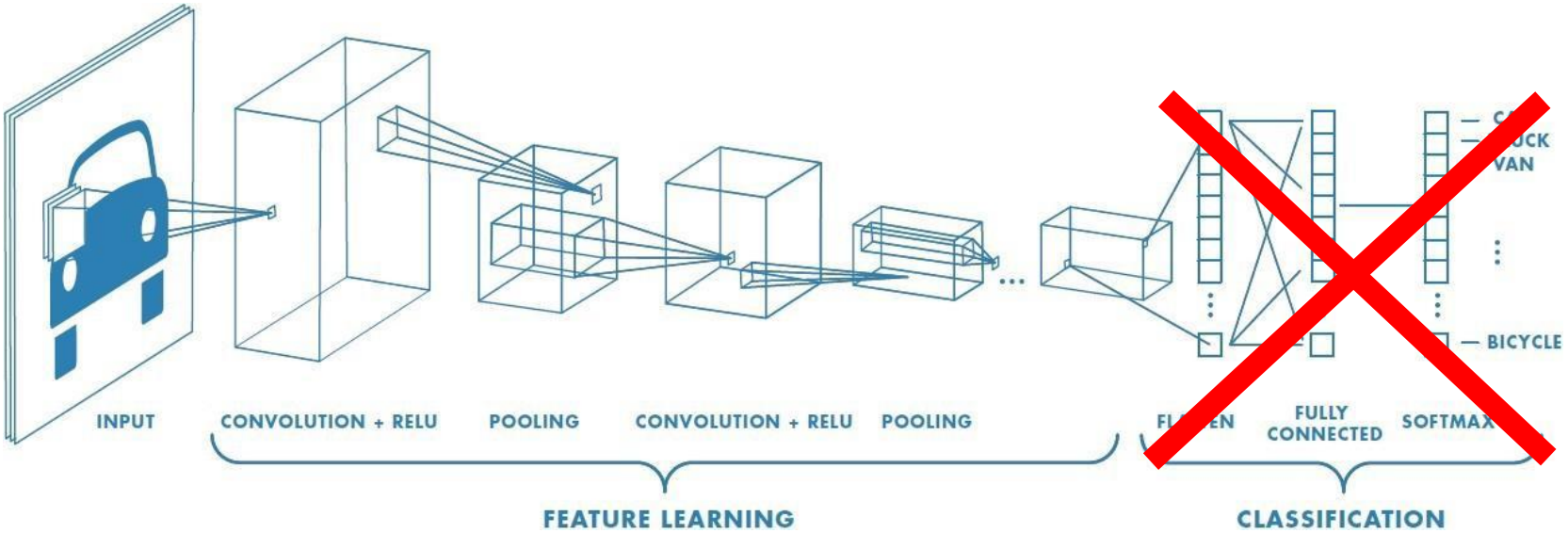

All layers in the network are convolutional, there is no fully connected (aka “dense”) layer like in most classifiers, we use the local info of the pixel neighborhood

# “1x1” convolutions




- A Gentle Introduction to 1×1 Convolutions
- Tutorial on 1 x 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

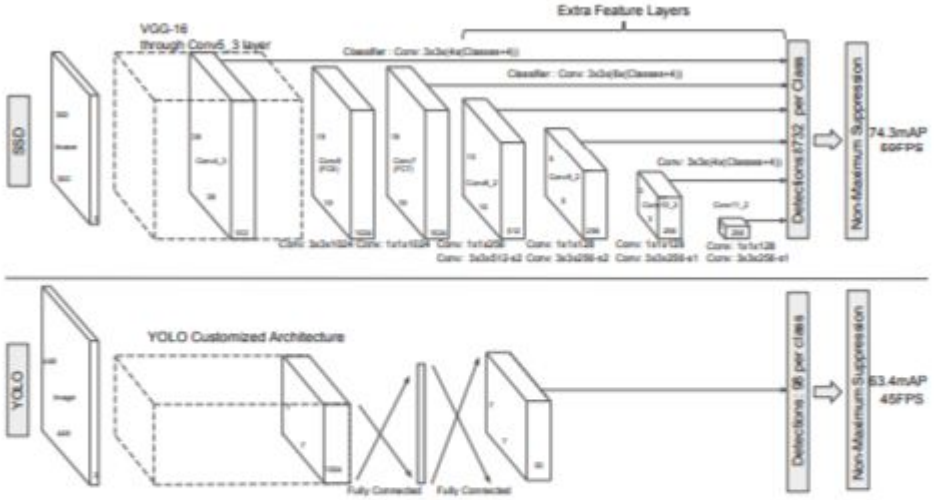



Fig. 2: A comparison between two single shot detection models: SSD and YOLO [5]. Our SSD model adds several feature layers to the end of a base network, which predict the offsets to default boxes of different scales and aspect ratios and their associated confidences. SSD with a $300 \times 300$ input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

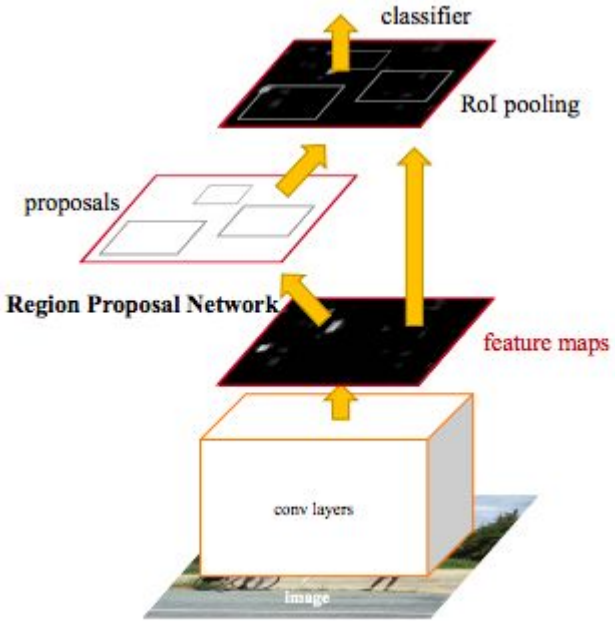
# Two stage detectors: R-CNN variants

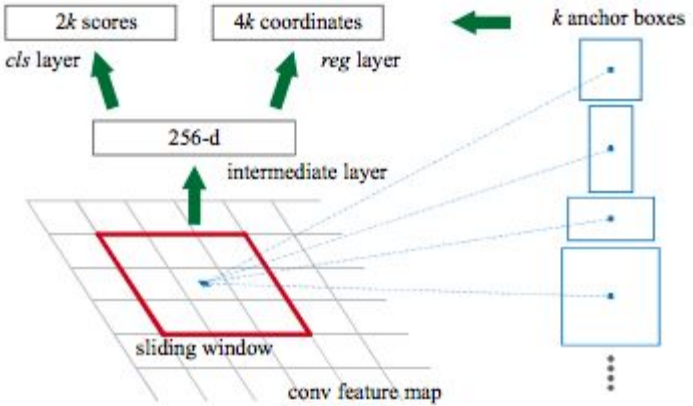
- **R-CNN**: Selective Search is run on the image, output segments from Selective Search are used for feature extraction and classification using a pre-trained CNN (two networks, separate training)
- **Fast R-CNN**: Uses the Selective Search algorithm to obtain region proposals, adding the Region of 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

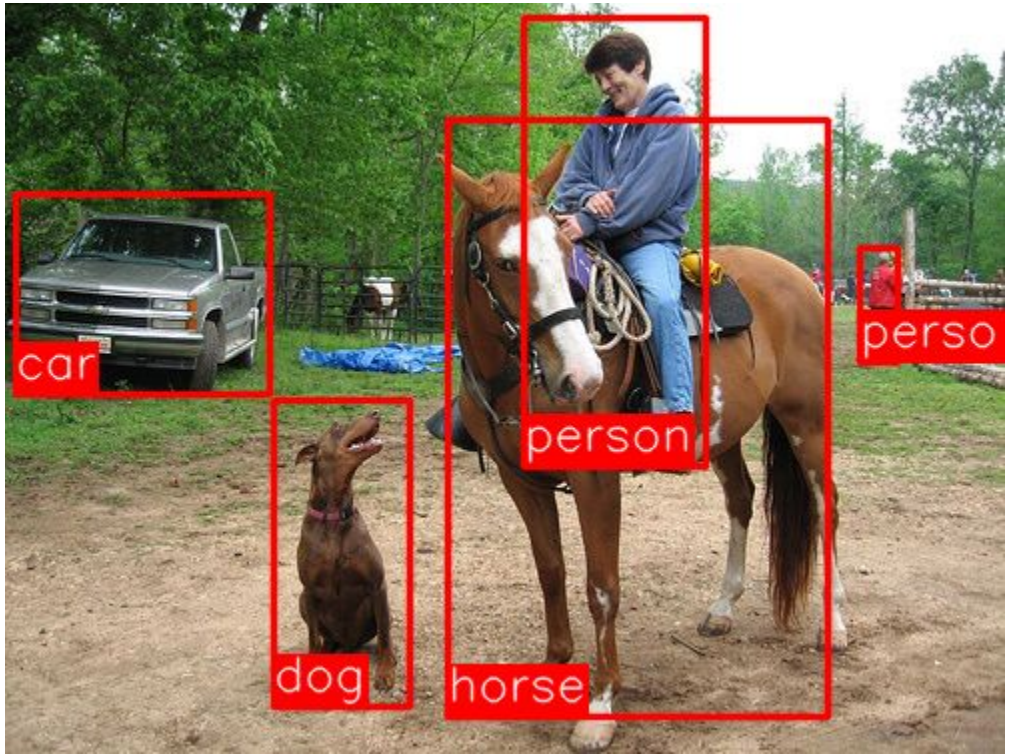
# Region Proposal Networks - Faster R-CNN



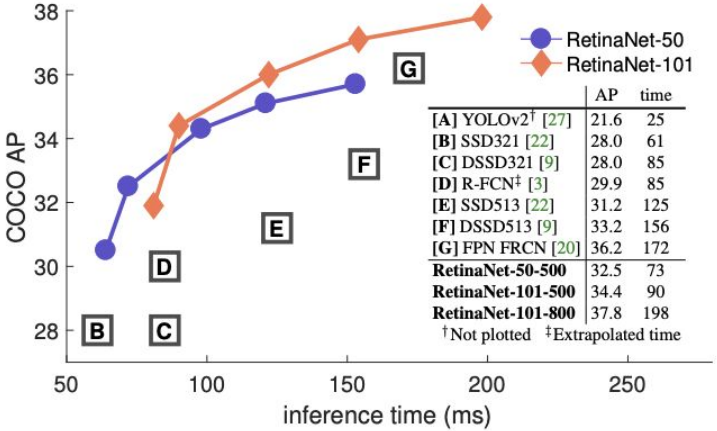




[Faster R-CNN breakdown]

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



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




| | AP | time |
|---|---|---|
| [A] YOLOv2† [27] | 21.6 | 25 |
| [B] SSD321 [22] | 28.0 | 61 |
| [C] DSSD321 [9] | 28.0 | 85 |
| [D] R-FCN‡ [3] | 29.9 | 85 |
| [E] SSD513 [22] | 31.2 | 125 |
| [F] DSSD513 [9] | 33.2 | 156 |
| [G] FPN FRCN [20] | 36.2 | 172 |
| **RetinaNet-50-500** | 32.5 | 73 |
| **RetinaNet-101-500** | 34.4 | 90 |
| **RetinaNet-101-800** | 37.8 | 198 |

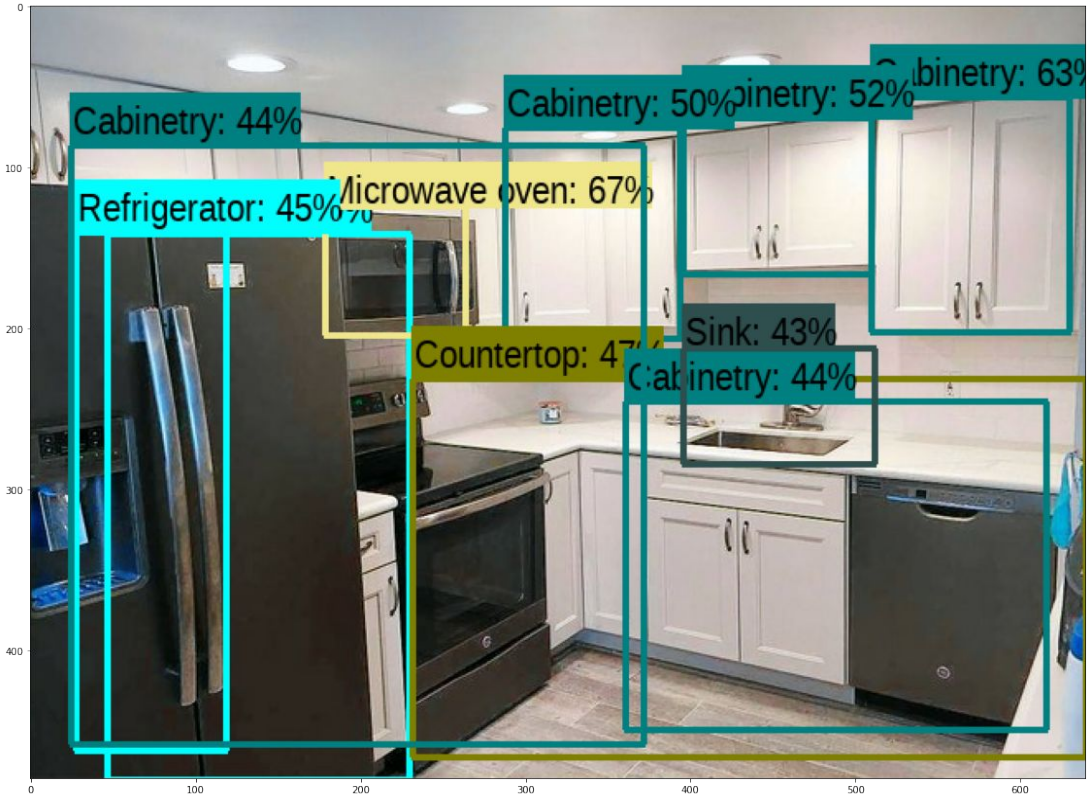
†Not plotted ‡Extrapolated time

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

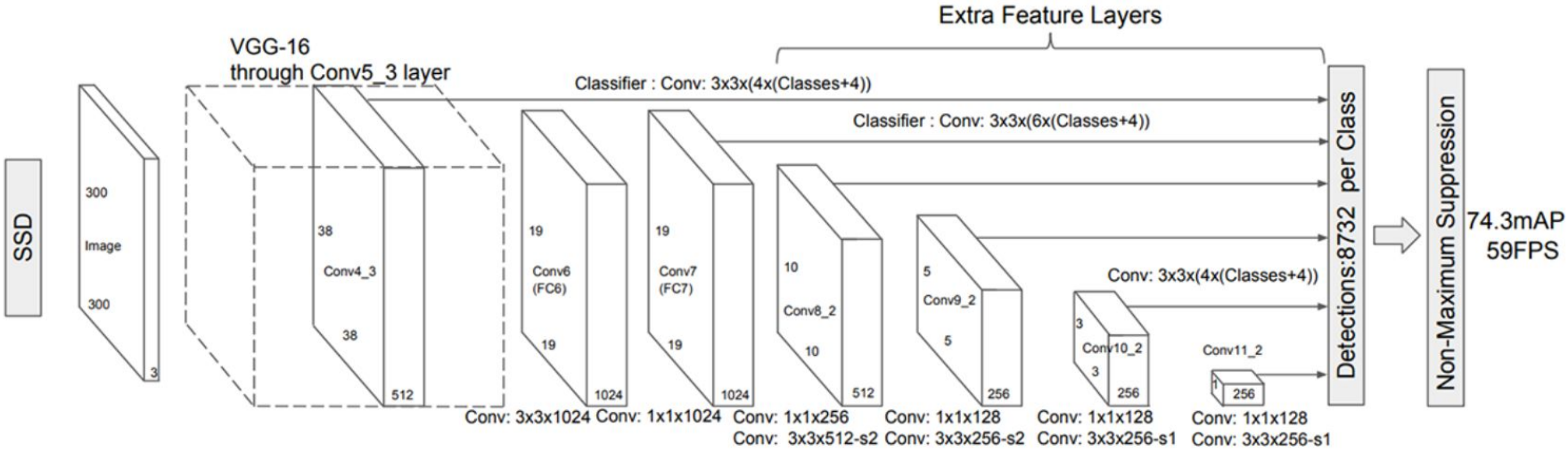
# SSD trained on Open Images Dataset

# 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**

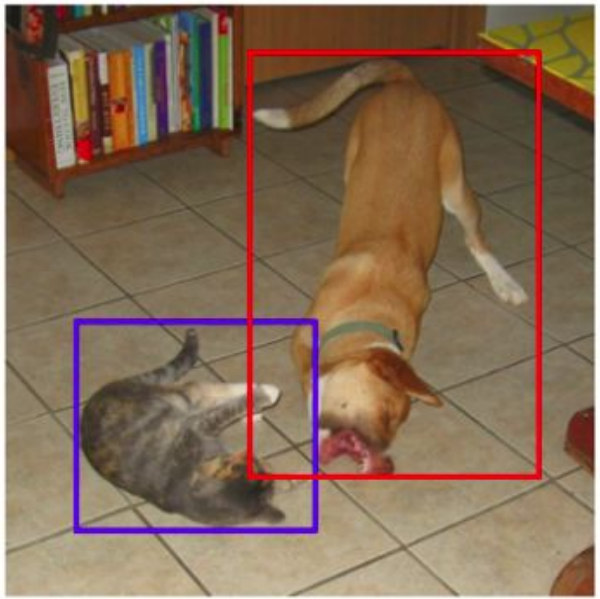
Small objects detection problem

# Single Shot Detectors

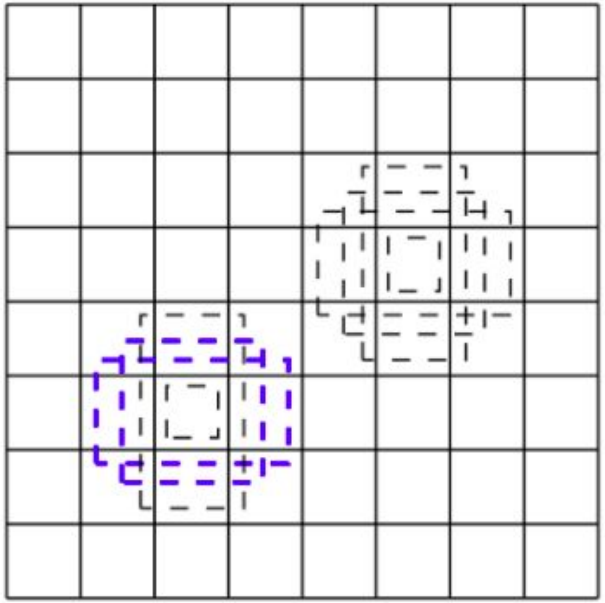



[Single Shot Multibox Detection]

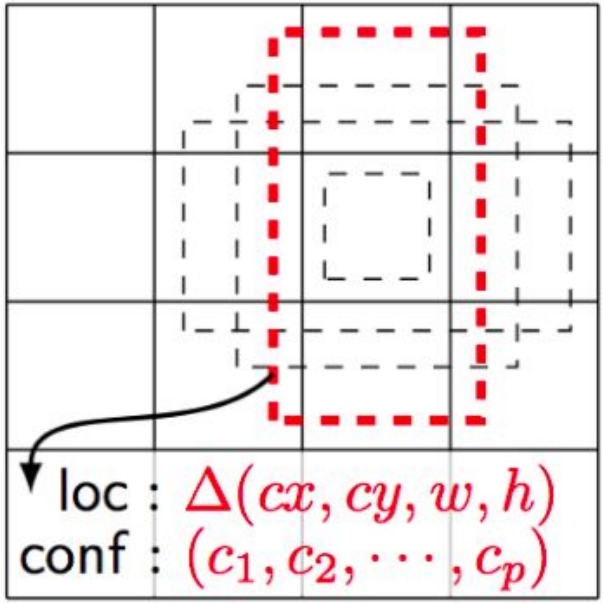
Single shot detectors make predictions of feature maps at different scales (different sizes for the bounding box) and compute cross entropy on a fully convolutional network that takes all these activation scales as input



(a) Image with GT boxes



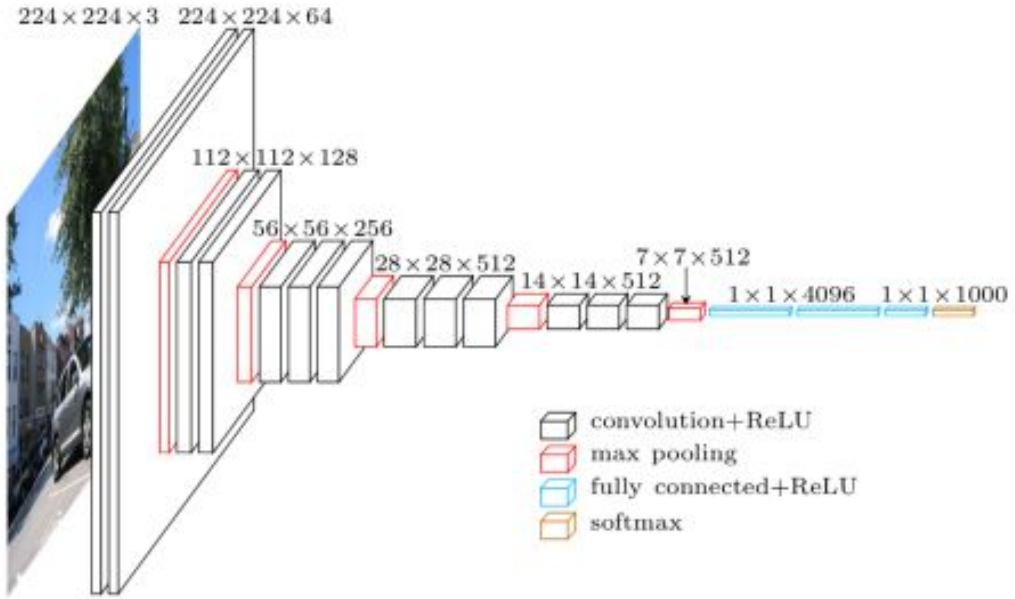
(b) 8 × 8 feature map




(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

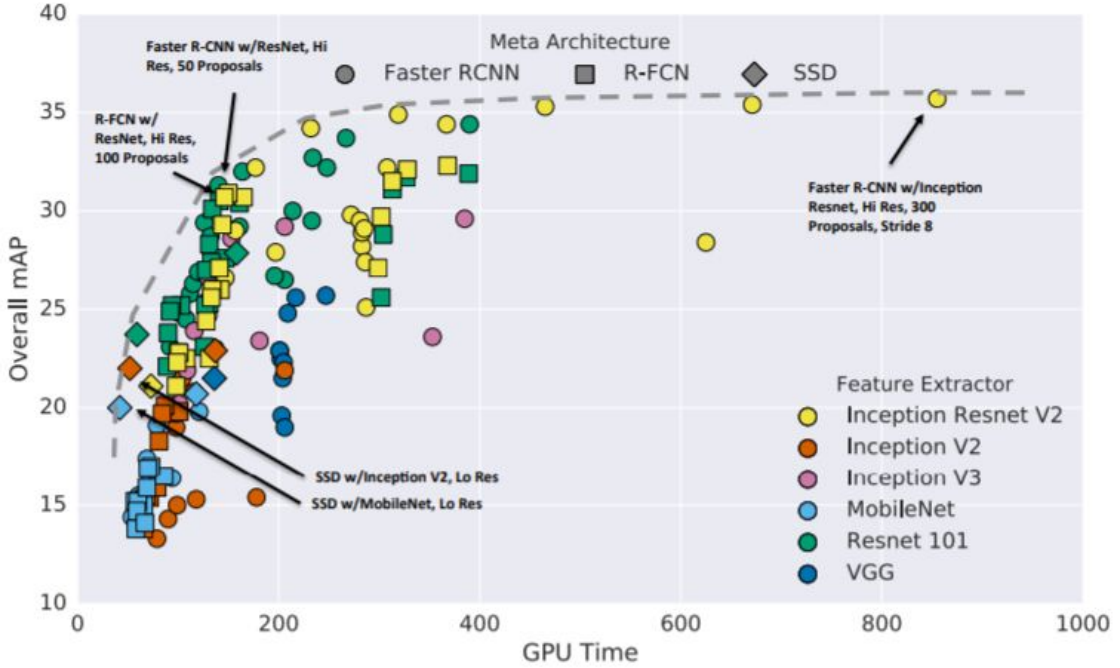



Figure 2: Accuracy vs time, with marker shapes indicating meta-architecture and colors indicating feature extractor. Each (meta-architecture, feature extractor) pair can correspond to multiple points on this plot due to changing input sizes, stride, etc.

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.

From Adrian Rosebrock's Deep Learning for Computer Vision with Python

# 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:**

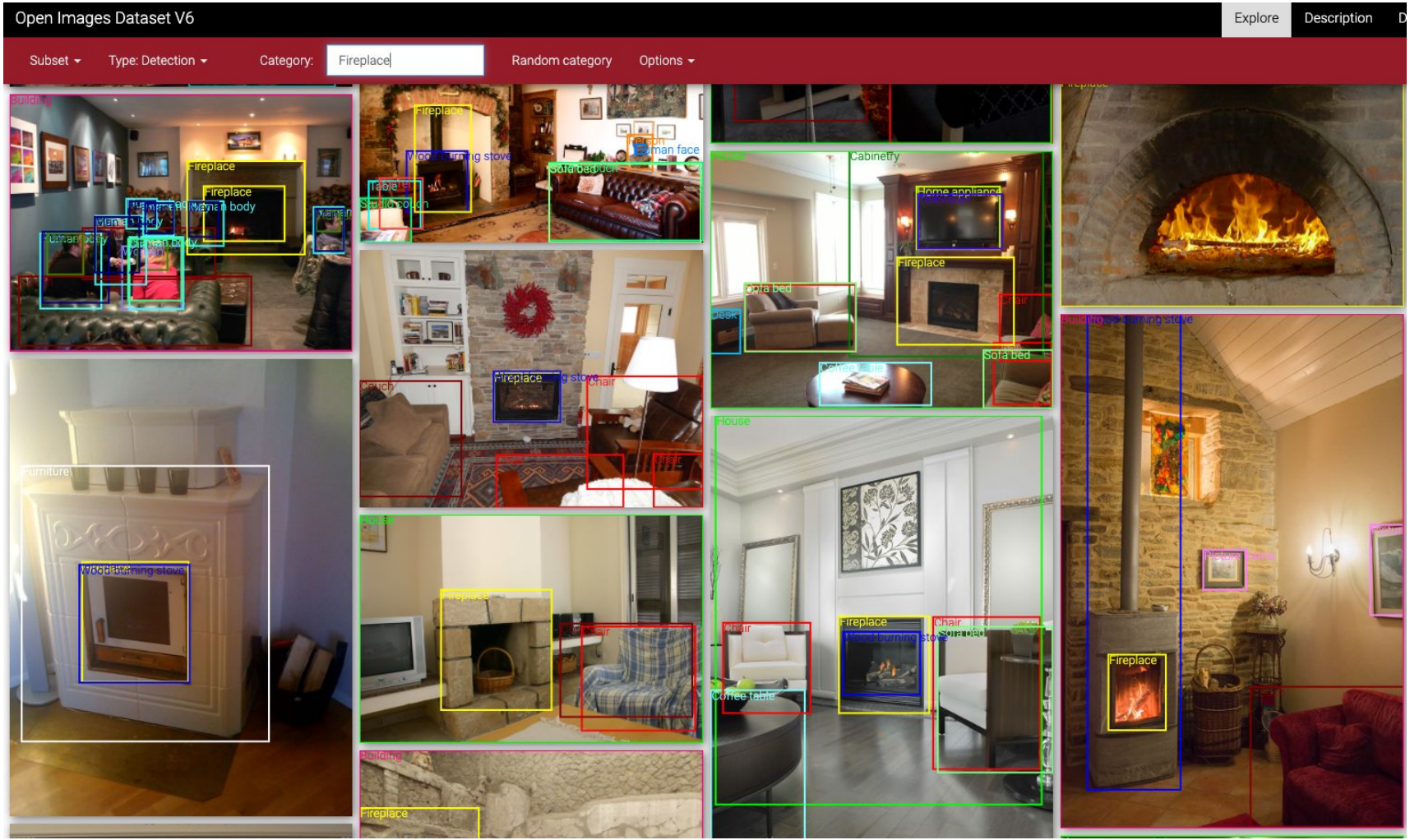
| Name | lr sched | train time (s/iter) | inference time (s/im) | train mem (GB) | box AP | model id | download |
|---|---|---|---|---|---|---|---|
| R50 | 1x | 0.205 | 0.041 | 4.1 | 37.4 | 190397773 | model \| metrics |
| R50 | 3x | 0.205 | 0.041 | 4.1 | 38.7 | 190397829 | model \| metrics |
| R101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


http://cocodataset.org/#explore


Open Images Dataset V6
Explore
Description
Subset
Type: Detection
Category:
Fireplace
Random category
Options
Building
Fireplace
Human body
Furniture
Wood-burning stove
Table
Sofa bed
Human face
Cabinetry
Home appliance
Chair
Desk
Coffee table
House
Couch

# Case study

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

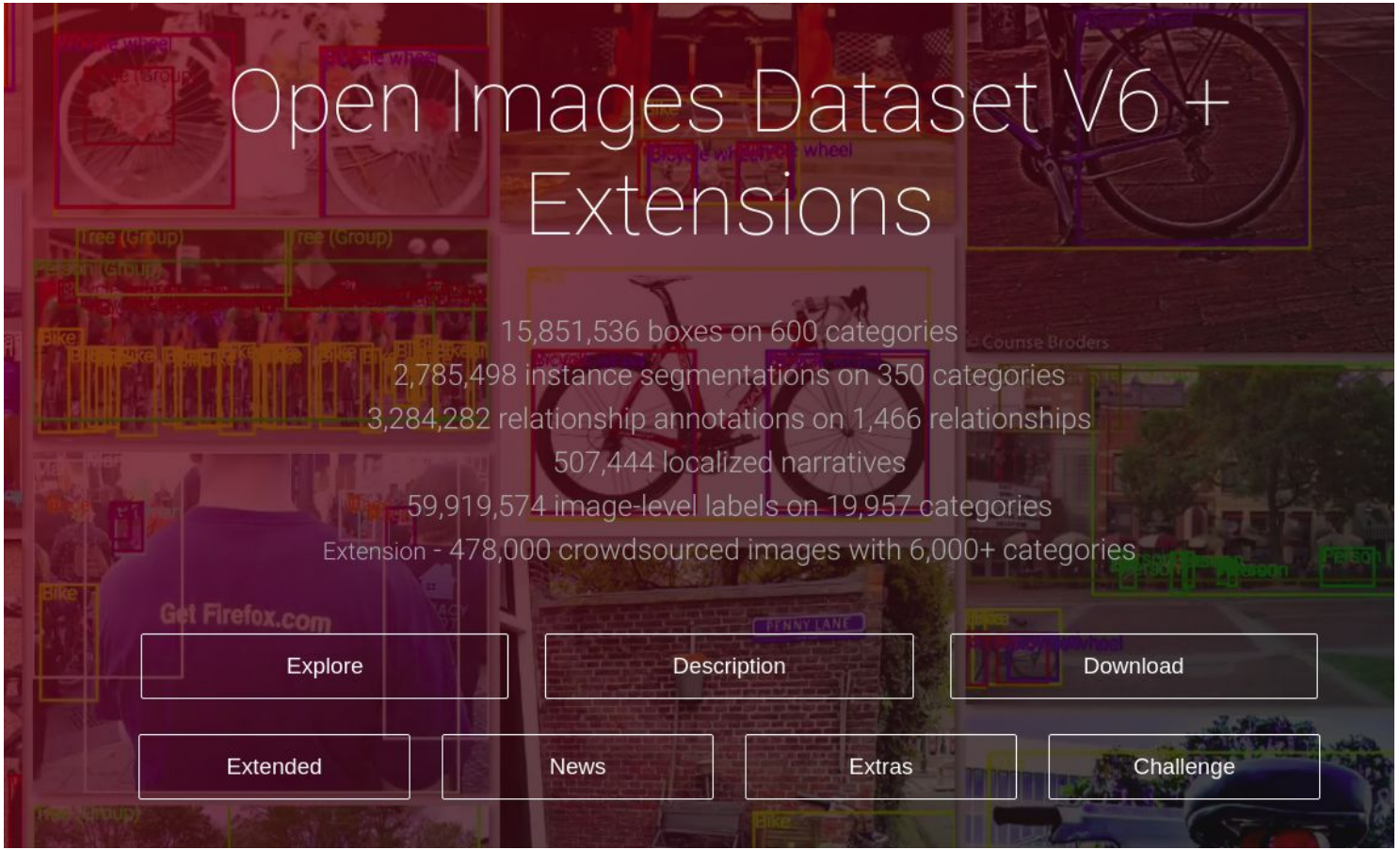
Build highly customized AI technologies into home-sharing products and help our guests belong anywhere.

Shijing Yao Follow

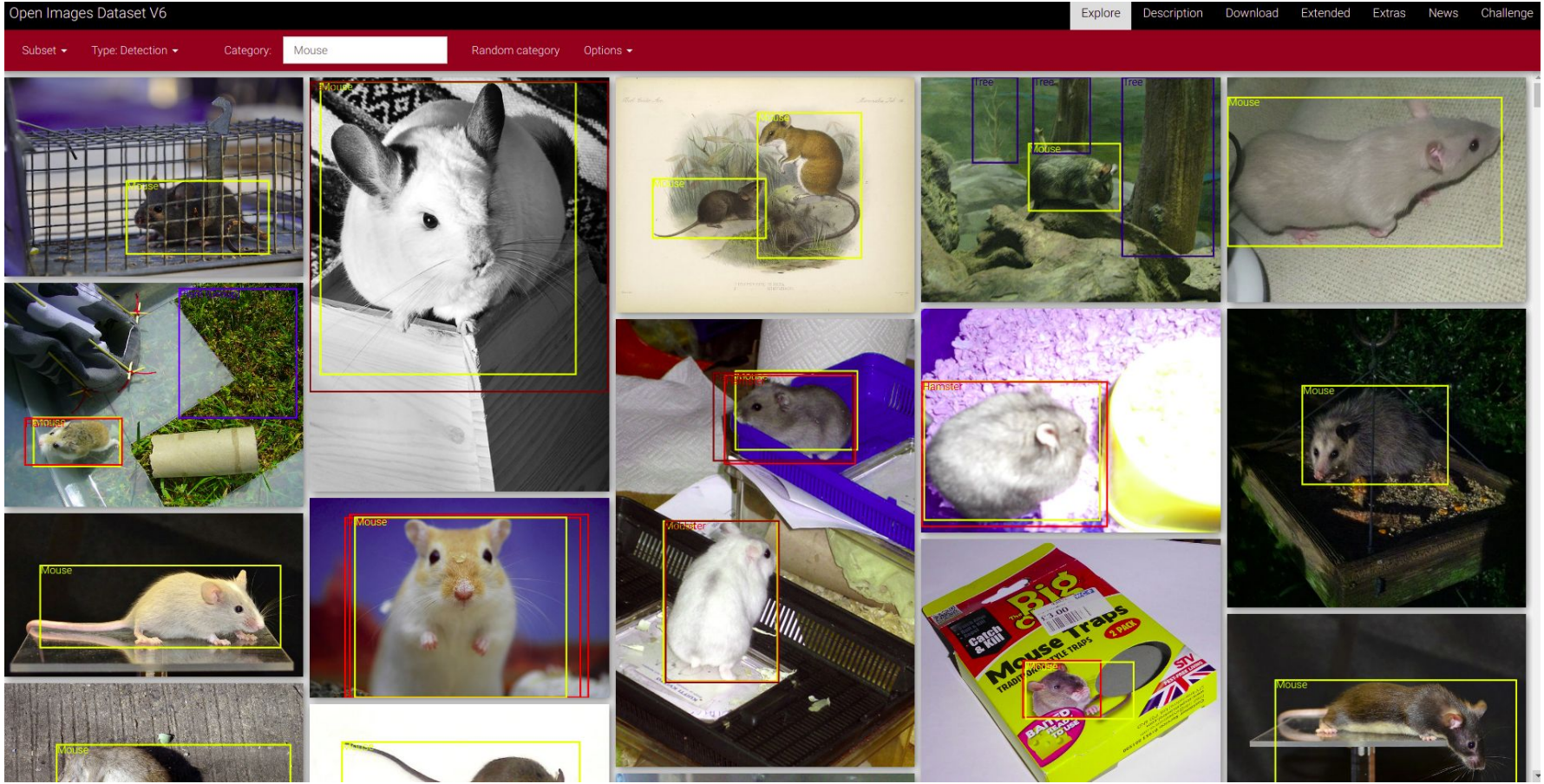
Jul 16, 2019 · 15 min read ★

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


Refrigerator - 0.96
Microwave oven - 0.85
Gas stove - 0.61
Oven - 0.61
Kitchen & dining room table - 0.55

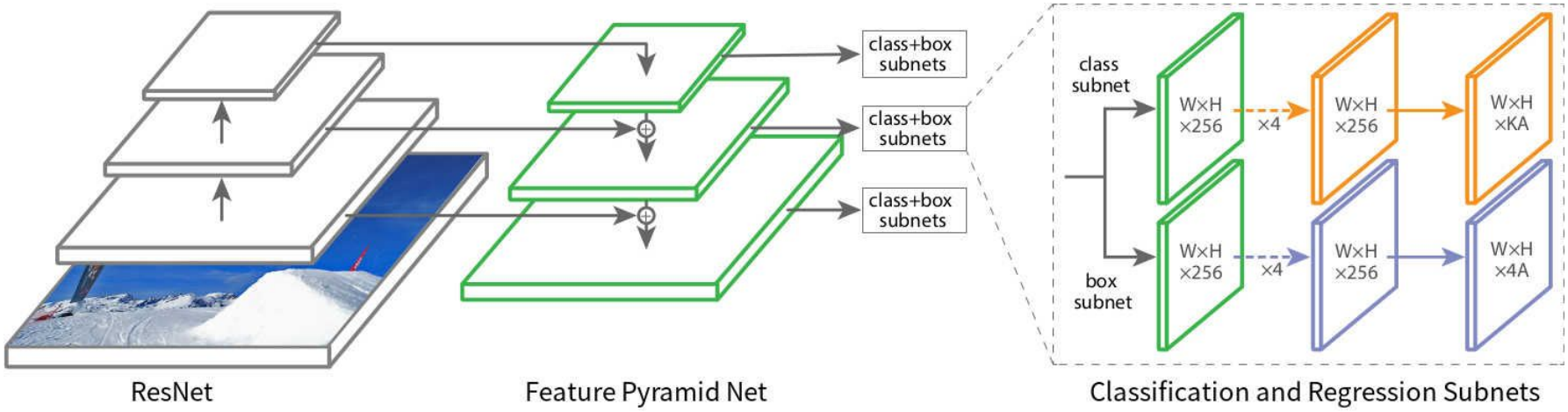



https://storage.googleapis.com/openimages/web/index.html




[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?