Computer Vision for Data Scientists

Antonio Rueda-Toicen

Data Science Retreat

November 2021

About me

- Machine Learning Engineer 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

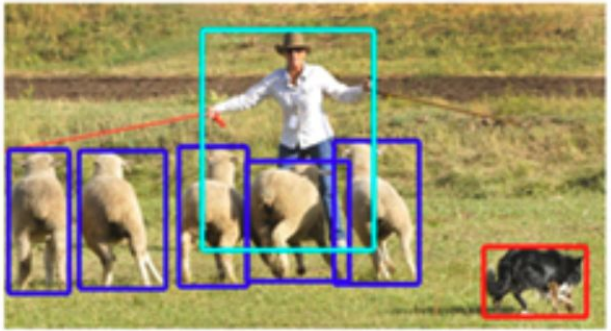
Manage group | Create event

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

Agenda

- Training image classifiers using FastAI and the one-cycle policy
- Semantic segmentation with U-net
- Transfer learning for image similarity
  - Image clustering
- Object detection
  - Bounding box regression
  - Single shot detectors and YOLO
  - RCNN family of object detection methods
- Image segmentation
  - Instance vs semantic segmentation
  - Panoptic segmentation
  - Fully convolutional networks
- Detectron2 framework
  - Case study: amenity detection in AirBnB

Classification, detection, and segmentation

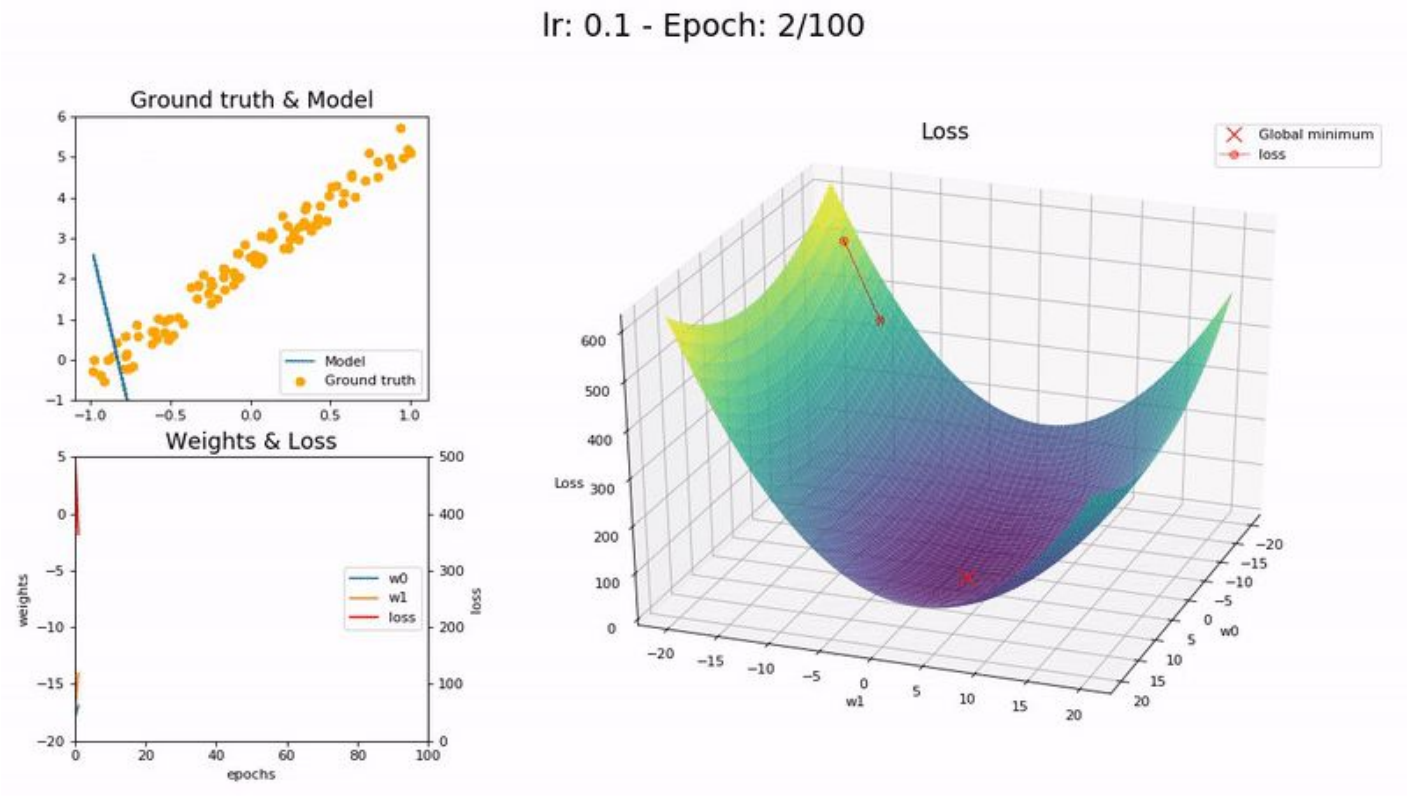


Classification refers to image-wide labels
Detection refers to localization of bounding boxes with labels
Segmentation refers to pixel-wise localization of the labels

Fastai 2

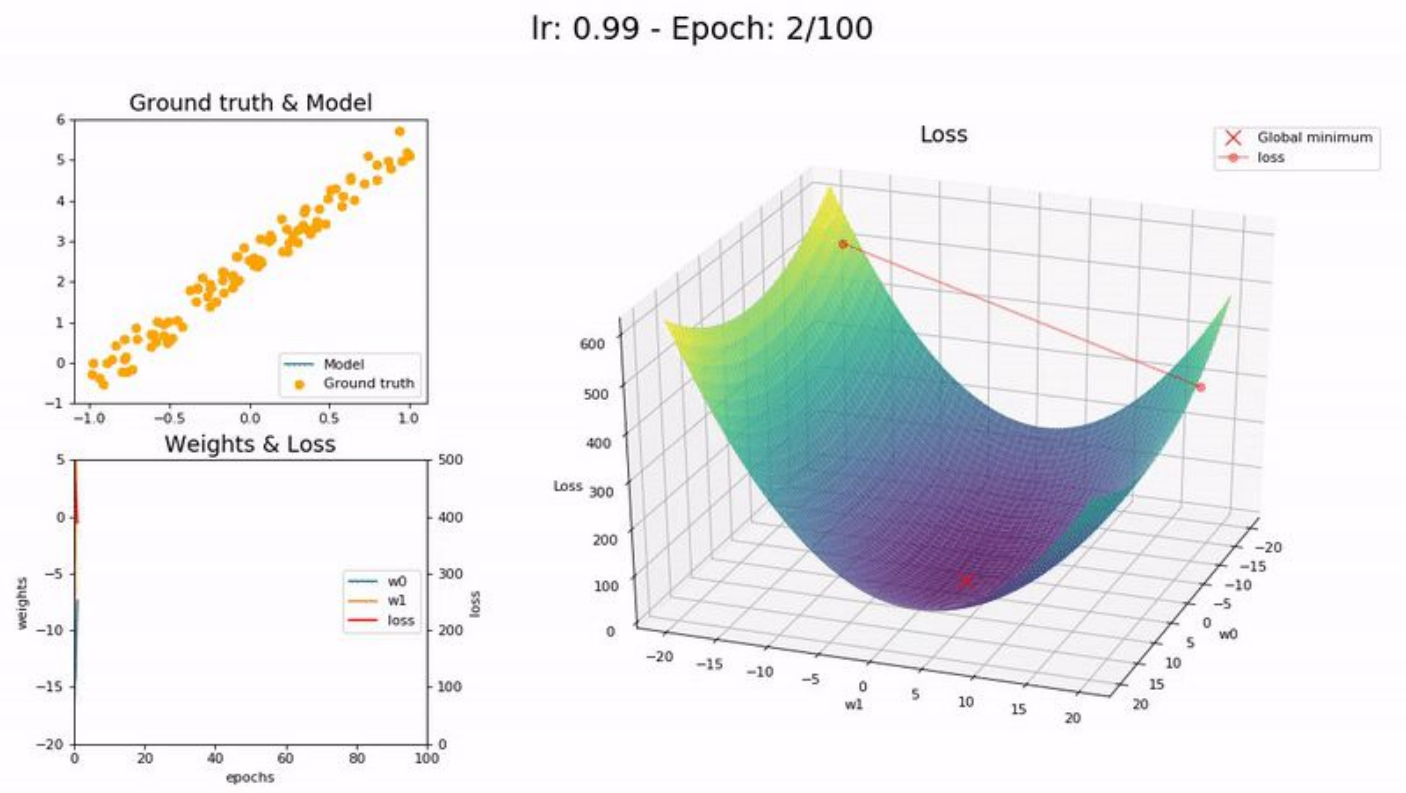
- PyTorch library from Jeremy Howard (former Kaggle president and #1 competitor)
- Very active community effort (use its forums!)
- It's the 'Keras for PyTorch'
  - Close to PyTorch lightning (but non-scammy)
  - Originally written in Tensorflow, rewritten in PyTorch in 2017
- Completely open license
  - Can be used for commercial projects without restriction
- Implements state-of-the-art techniques
  - One-cycle policy
  - Extensive image augmentations: presizing, mixups, progressive, resizing, pixel shuffle, and others
  - GradCam
  - Unet
- V2 documented in the Deep Learning for Coders O'reilly book and the last version of their MOOC, however **it will break, be prepared to Google every single error**
- Please read this blog post about the 'top-down' teaching approach of FastAI

The importance of the learning rate



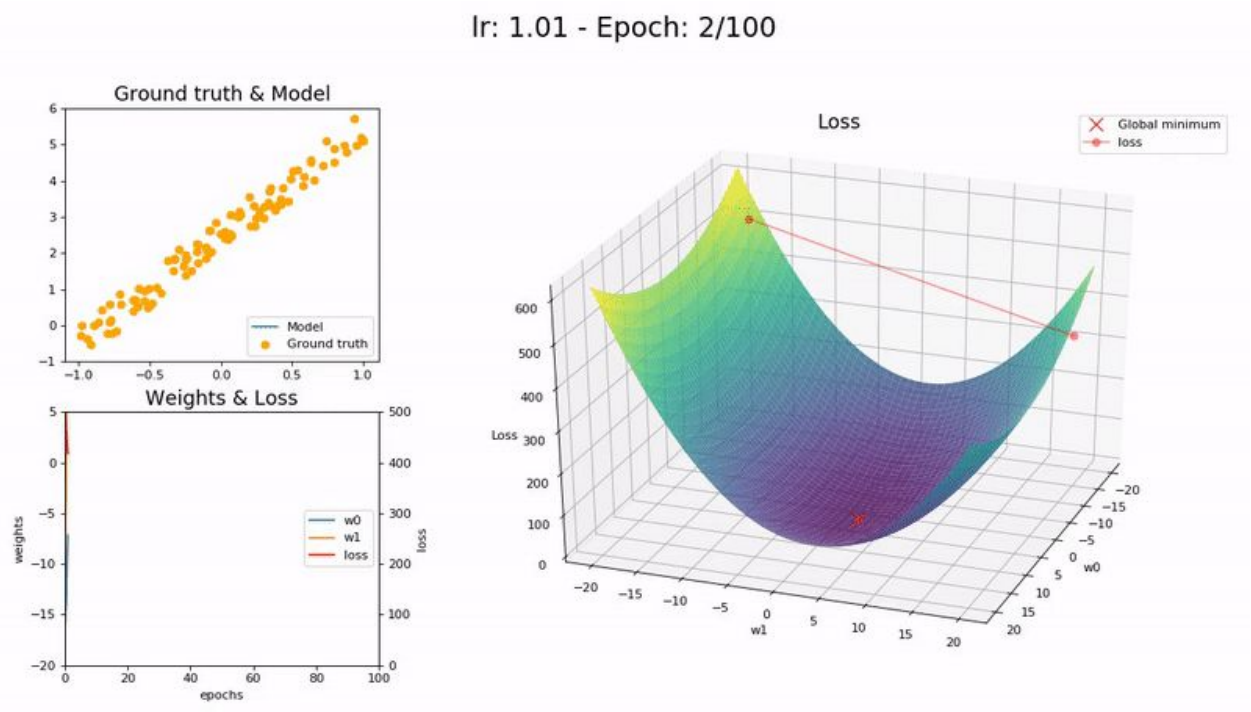
https://iconof.com/1cycle-learning-rate-policy/

The importance of the learning rate



https://iconof.com/1cycle-learning-rate-policy/

The importance of the learning rate



https://iconof.com/1cycle-learning-rate-policy/

The one cycle policy as a learning rate finder

A DISCIPLINED APPROACH TO NEURAL NETWORK HYPER-PARAMETERS: PART 1 – LEARNING RATE, BATCH SIZE, MOMENTUM, AND WEIGHT DECAY

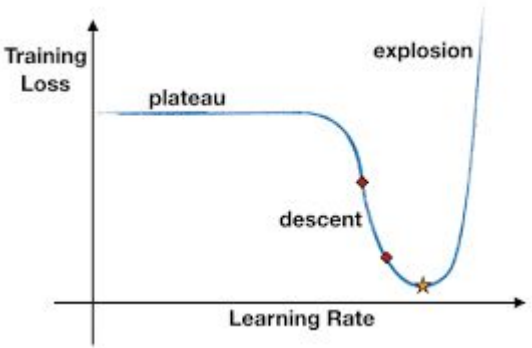
Leslie N. Smith
US Naval Research Laboratory
Washington, DC, USA
leslie.smith@nrl.navy.mil

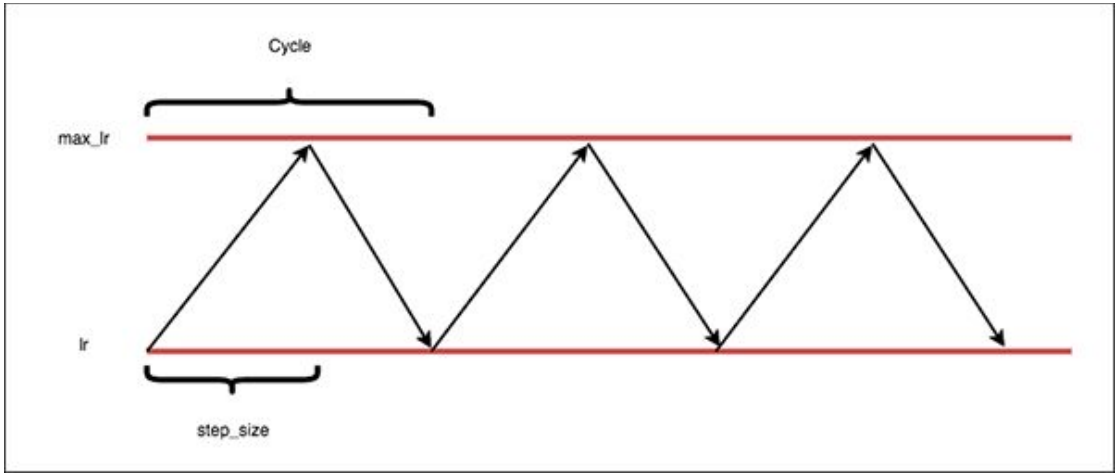
ABSTRACT

Although deep learning has produced dazzling successes for applications of image, speech, and video processing in the past few years, most trainings are with suboptimal hyper-parameters, requiring unnecessarily long training times. Setting the hyper-parameters remains a black art that requires years of experience to acquire. This report proposes several efficient ways to set the hyper-parameters that significantly reduce training time and improves performance. Specifically, this report shows how to examine the training validation/test loss function for subtle clues of underfitting and overfitting and suggests guidelines for moving toward the optimal balance point. Then it discusses how to increase/decrease the learning rate/momentum to speed up training. Our experiments show that it is crucial to balance every manner of regularization for each dataset and architecture. Weight decay is used as a sample regularizer to show how its optimal value is tightly coupled with the learning rates and momentum. Files to help replicate the results reported here are available at https://github.com/lnsmith54/hyperParam1.

https://arxiv.org/pdf/1803.09820.pdf

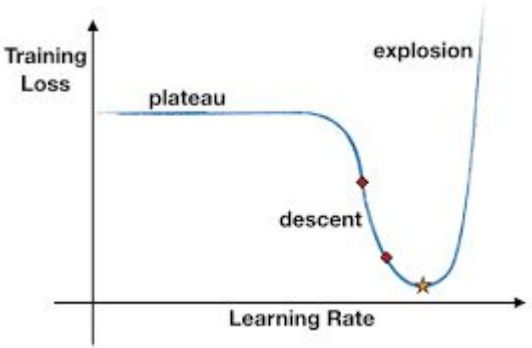
The one cycle policy as a learning rate finder

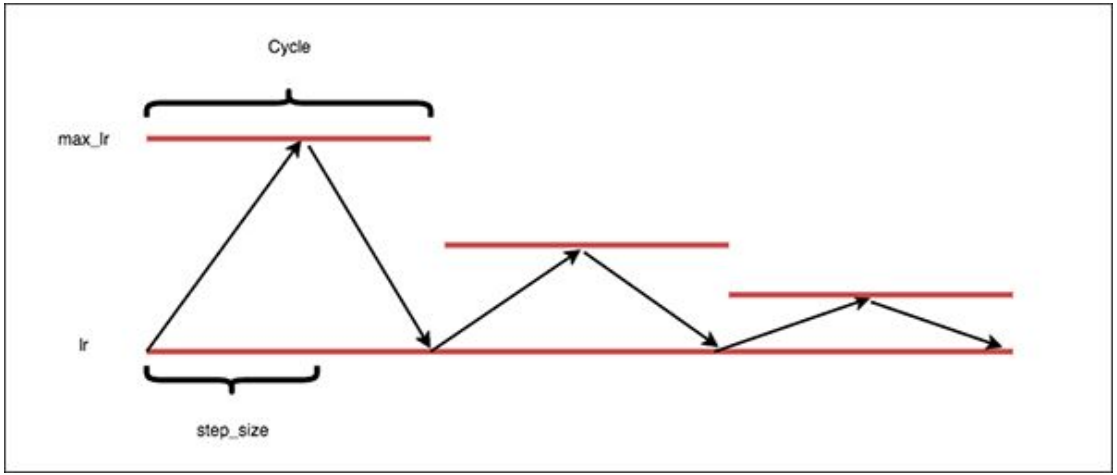




"CLRs leads to *far fewer* learning rate tuning experiments along with near identical accuracy to exhaustive hyperparameter tuning" source

The one cycle policy as a learning rate finder





"CLRs leads to *far fewer* learning rate tuning experiments along with near identical accuracy to exhaustive hyperparameter tuning" source

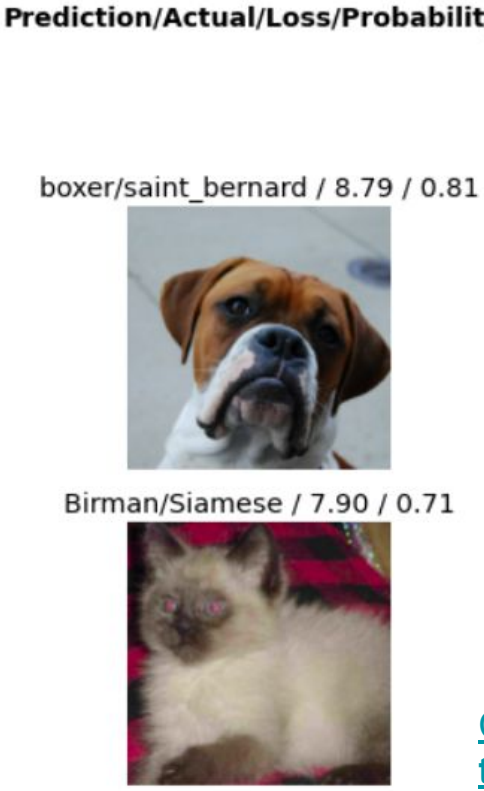
Image classification: pet breeds

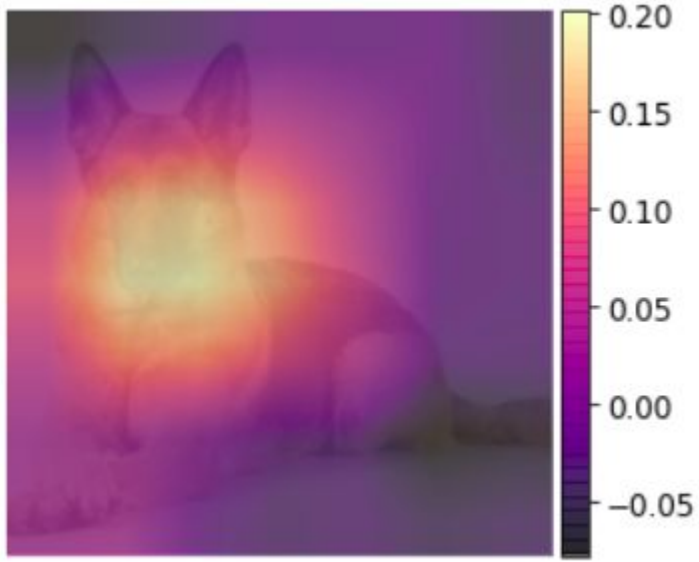
- Please read (in Google Colab) everything under the section “improving our model”
  - https://github.com/fastai/fastbook/blob/master/05_pet_breeds.ipynb

Review questions on the learning rate

- What is the one-cycle policy doing?
- What is the learning rate finder doing?
- What are discriminative learning rates?
  - Why do we use discriminative learning rates?

Image classification: pet breeds



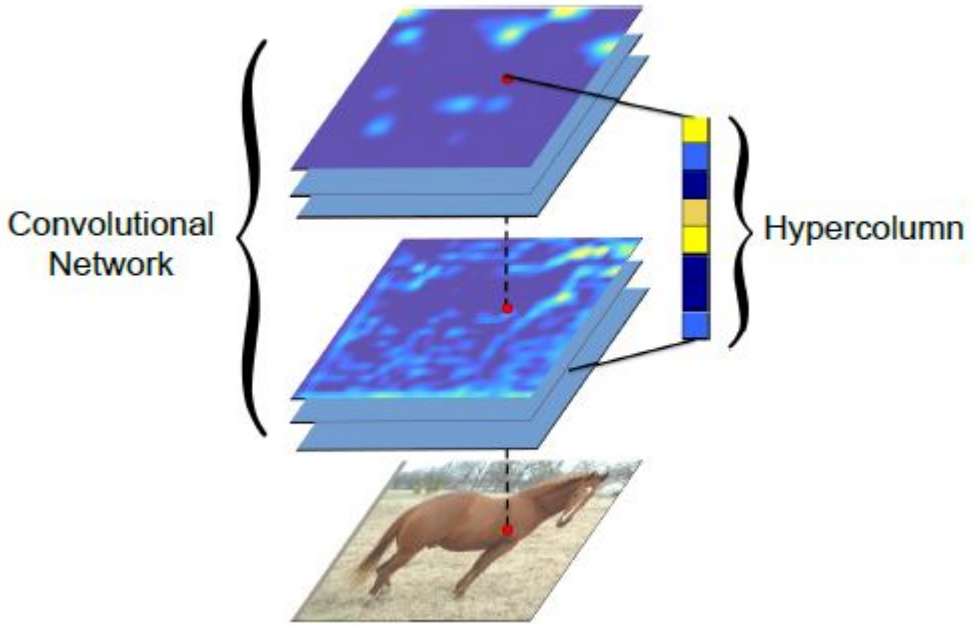


Chapter 5 of the FastAI book (try opening it on Google Colab **broken, please try to fix it before consulting the solution notebook**)

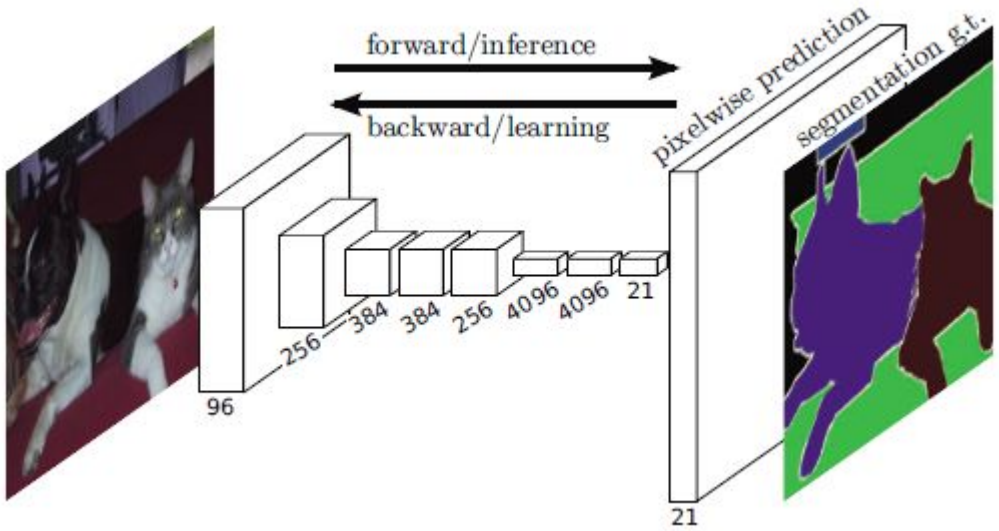
Image classification: pet breeds

- Exercise
- Solution

“Hypercolumns”

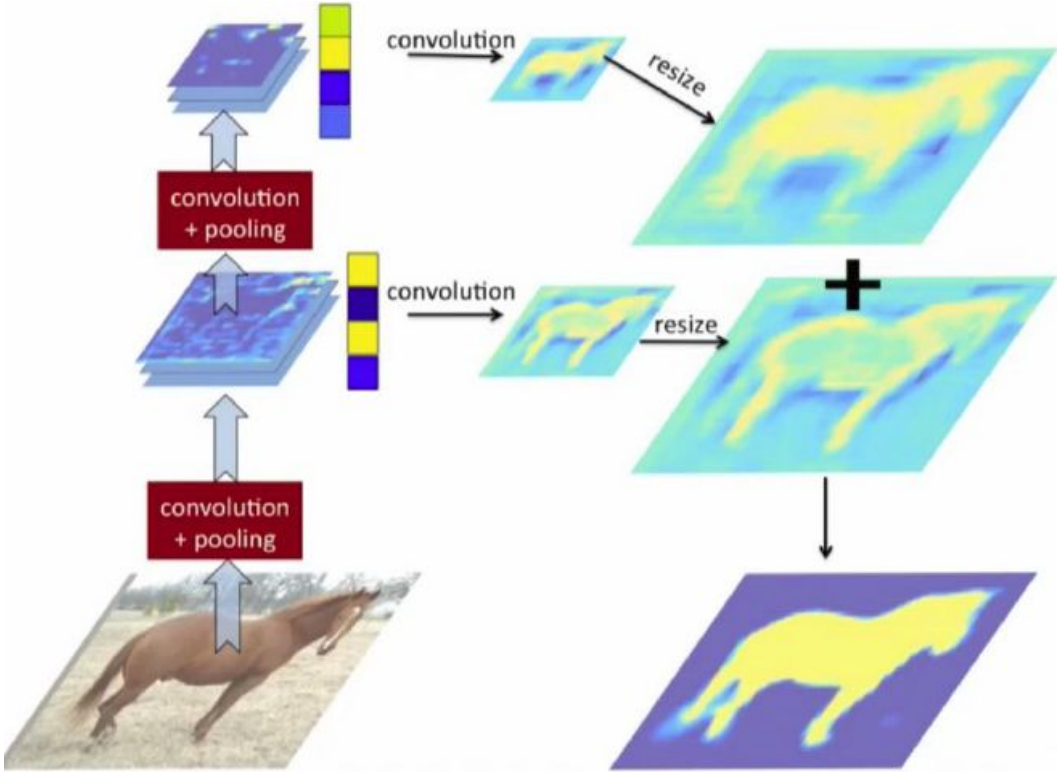


“Fully Convolutional” networks ‘localize’ labels

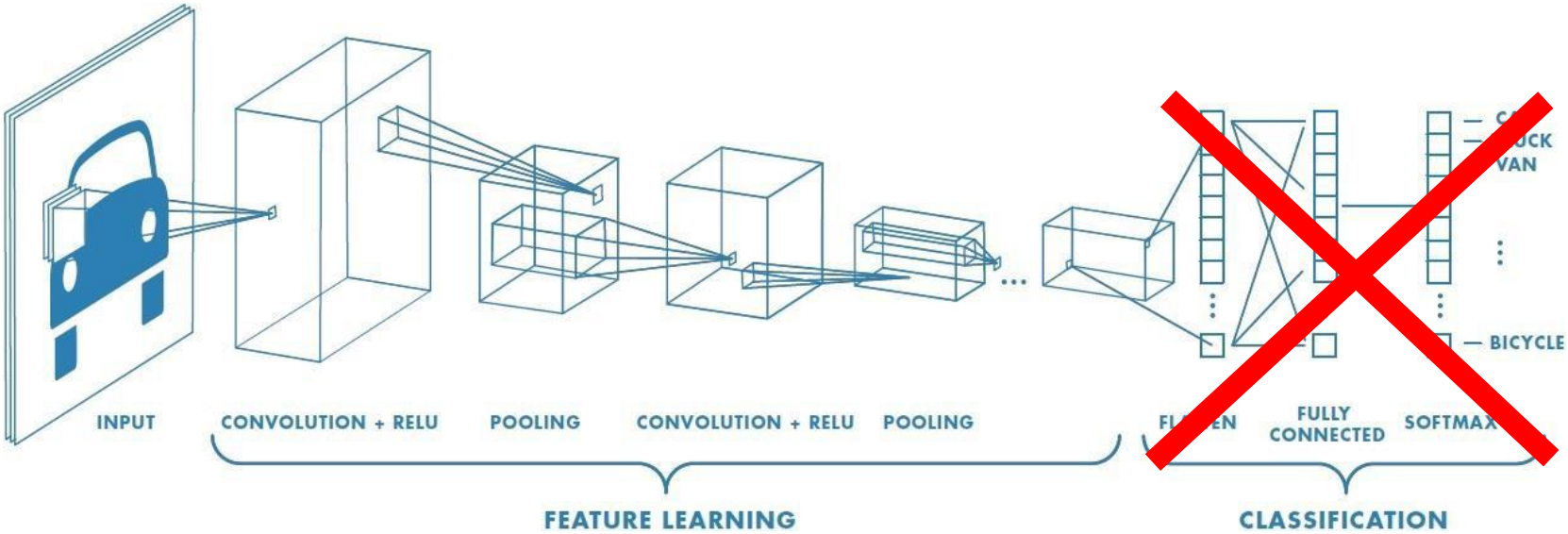


All layers in the network are convolutional, there is no fully connected (aka “dense”) layer like in most classifiers, we use the local info of the pixel neighborhood

“Hypercolumns”

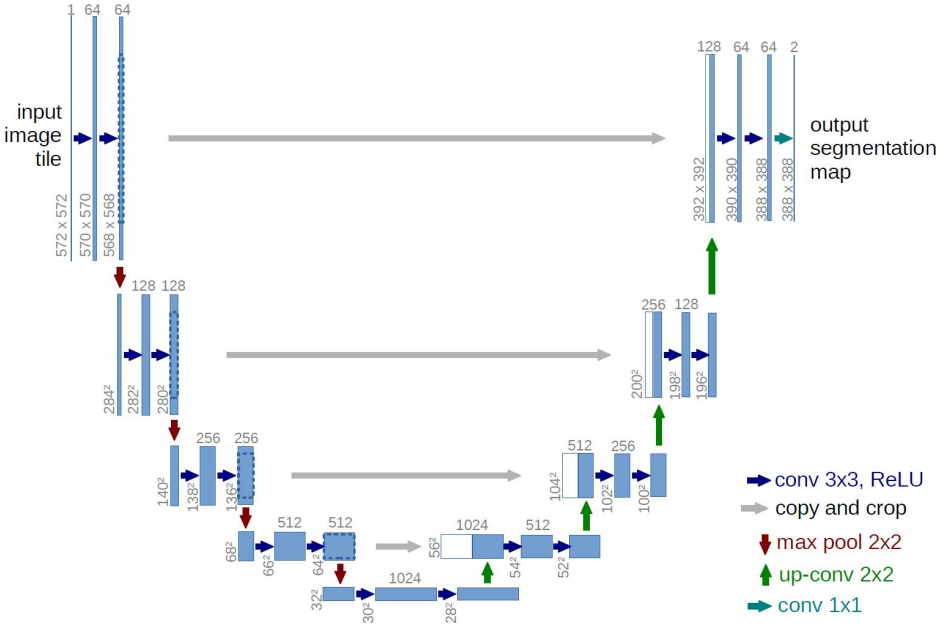


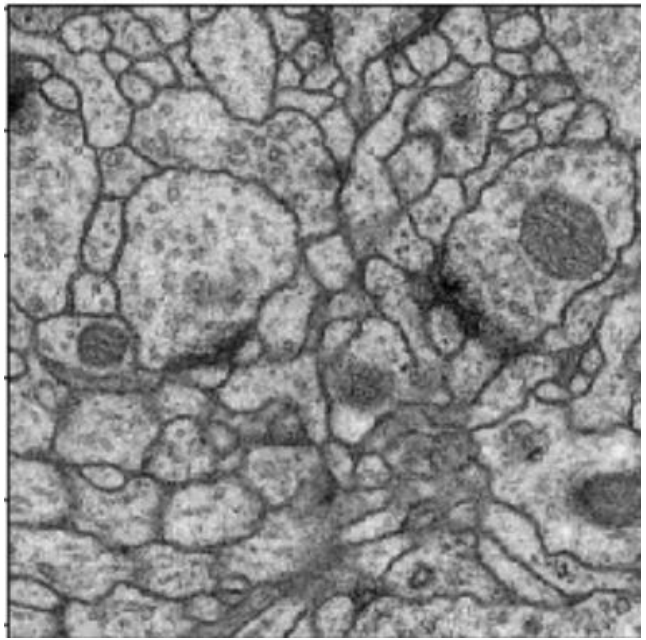
“Fully Convolutional” networks draw segmentation masks



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

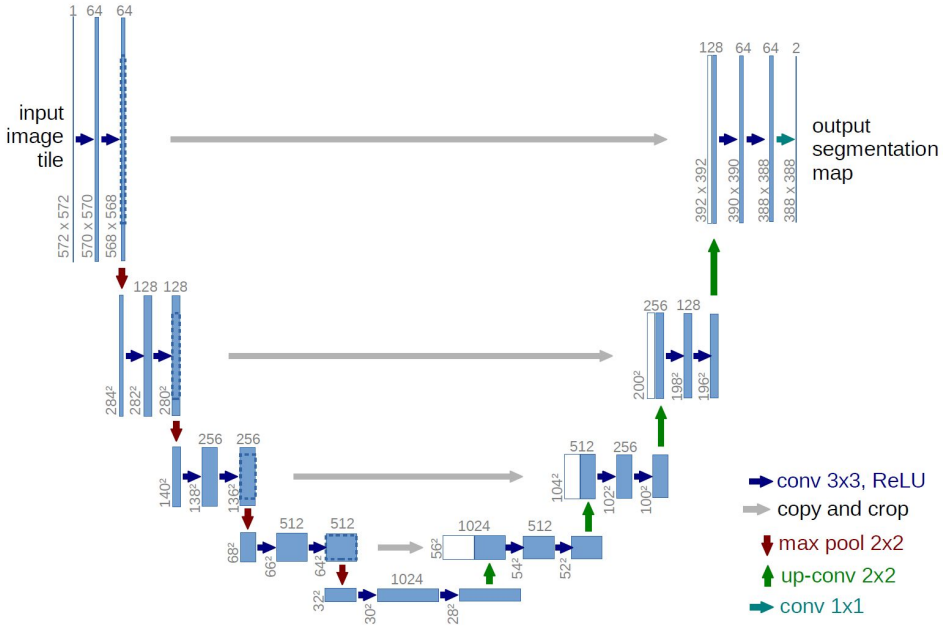
U-net for binary semantic segmentation





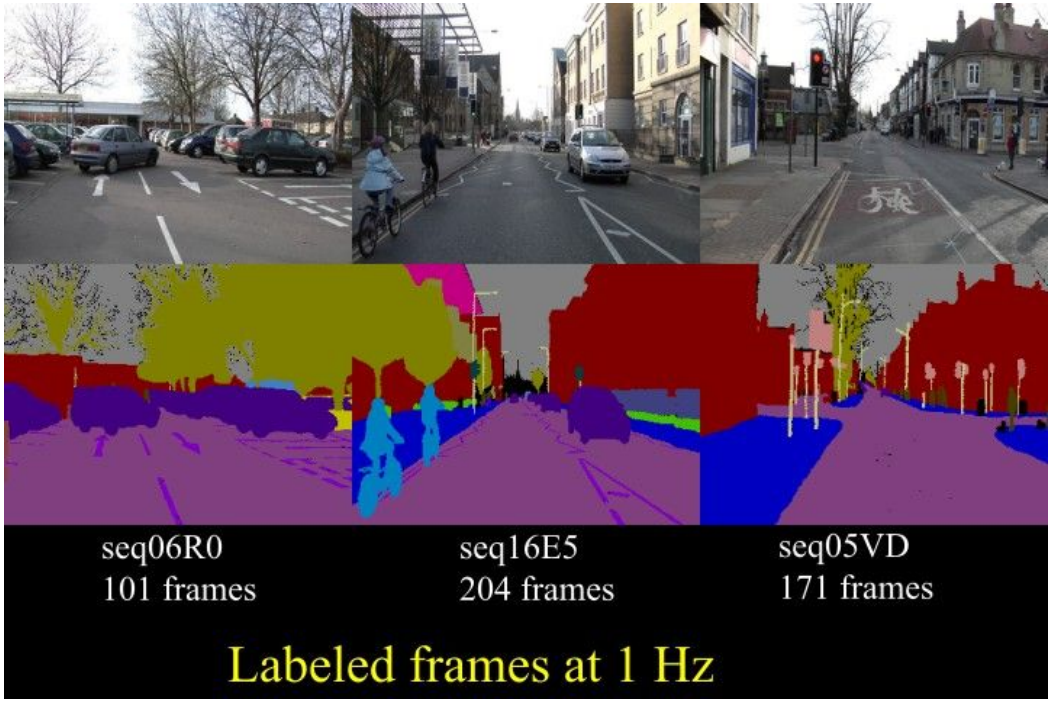
All layers in the network are convolutional, there is no fully connected (aka “dense”) layer like in most classifiers, we need this fully convolutional architecture to label images pixel by pixel preserving their local info

U-net for binary semantic segmentation



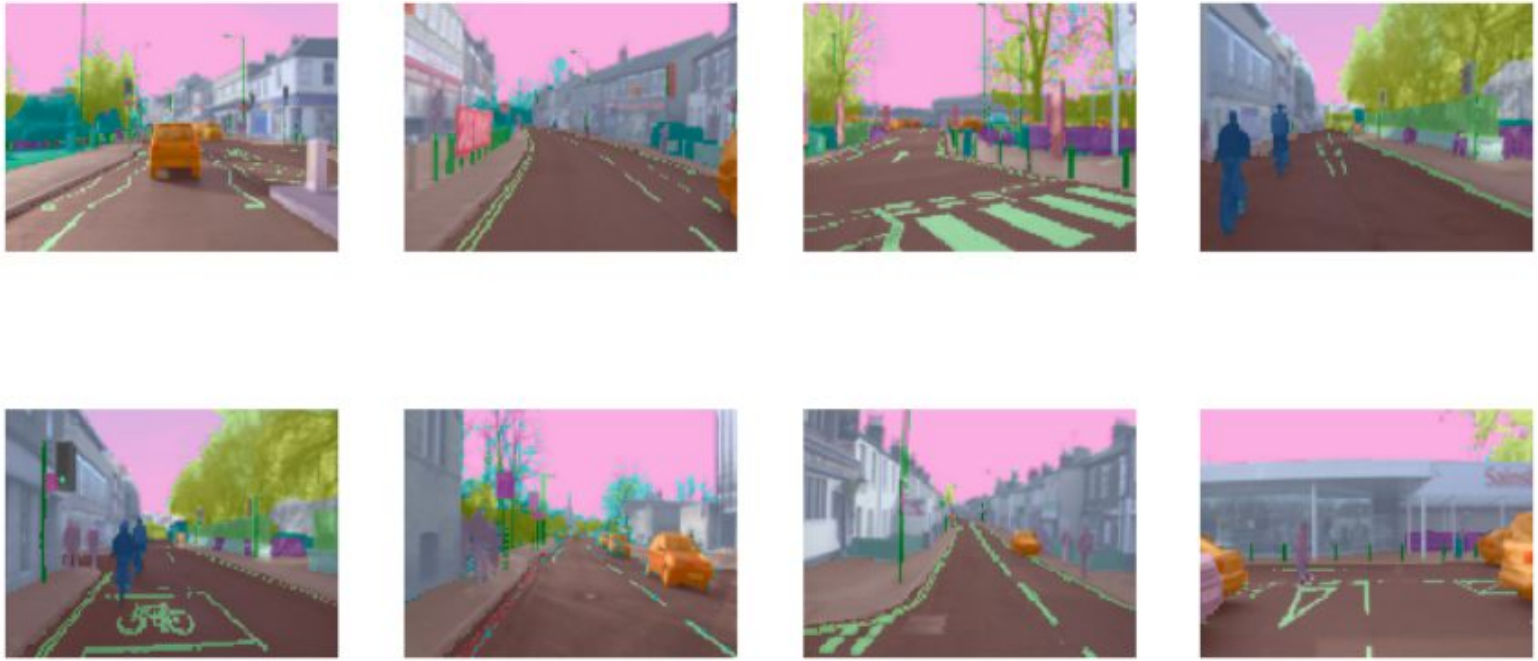
All layers in the network are convolutional, there is no fully connected (aka “dense”) layer like in most classifiers, we need this fully convolutional architecture to label images pixel by pixel preserving their local info

The CAMVID segmentation dataset



http://mi.eng.cam.ac.uk/research/projects/VideoRec/CamVid/

Semantic segmentation with Unet: Camvid



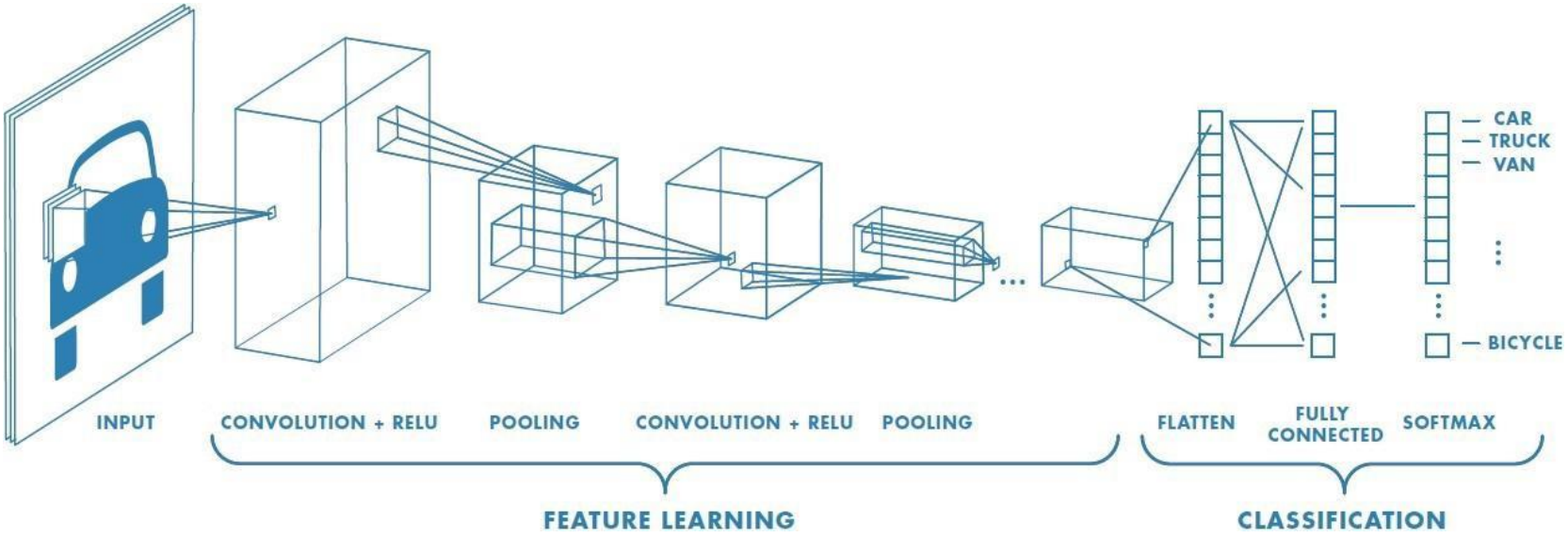
http://mi.eng.cam.ac.uk/research/projects/VideoRec/CamVid/

Semantic segmentation with Unet: Camvid

- Exercise
- Solution (with small images)
- Solution with large images

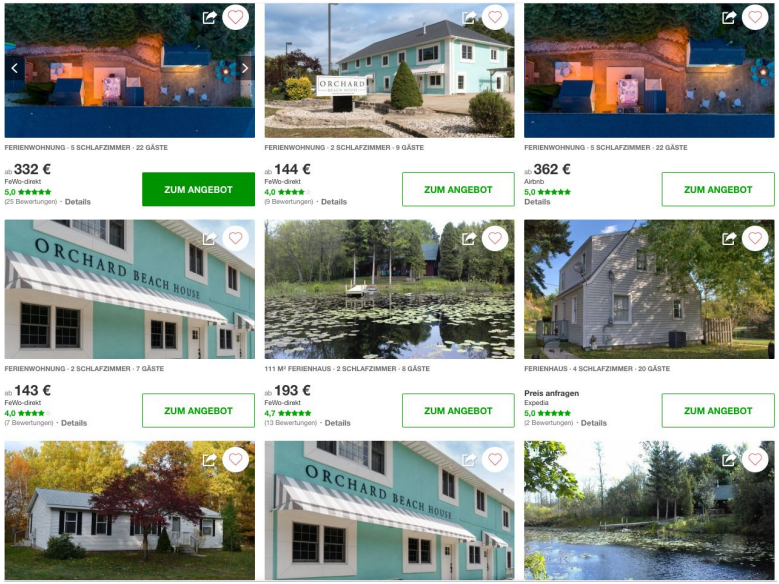
  - Why is the quality of the results so different among these two versions of the Camvid dataset?
  - In the small dataset, would using a bigger model (e.g. ResNet50) help?
  - Why did we use smaller batch sizes when running the larger dataset?

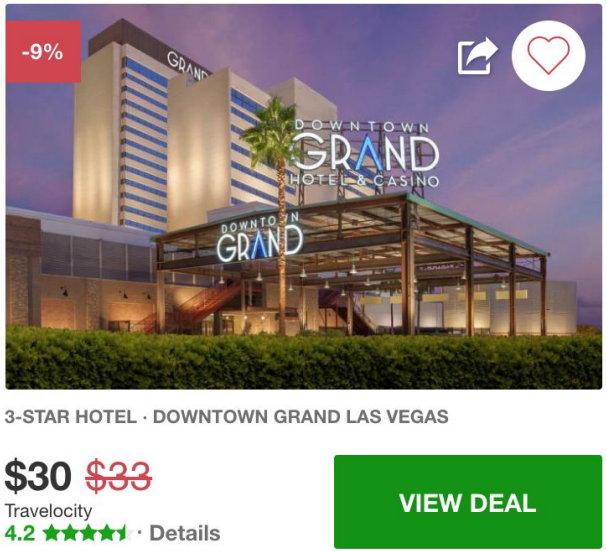
Convolutional networks as embeddings creators

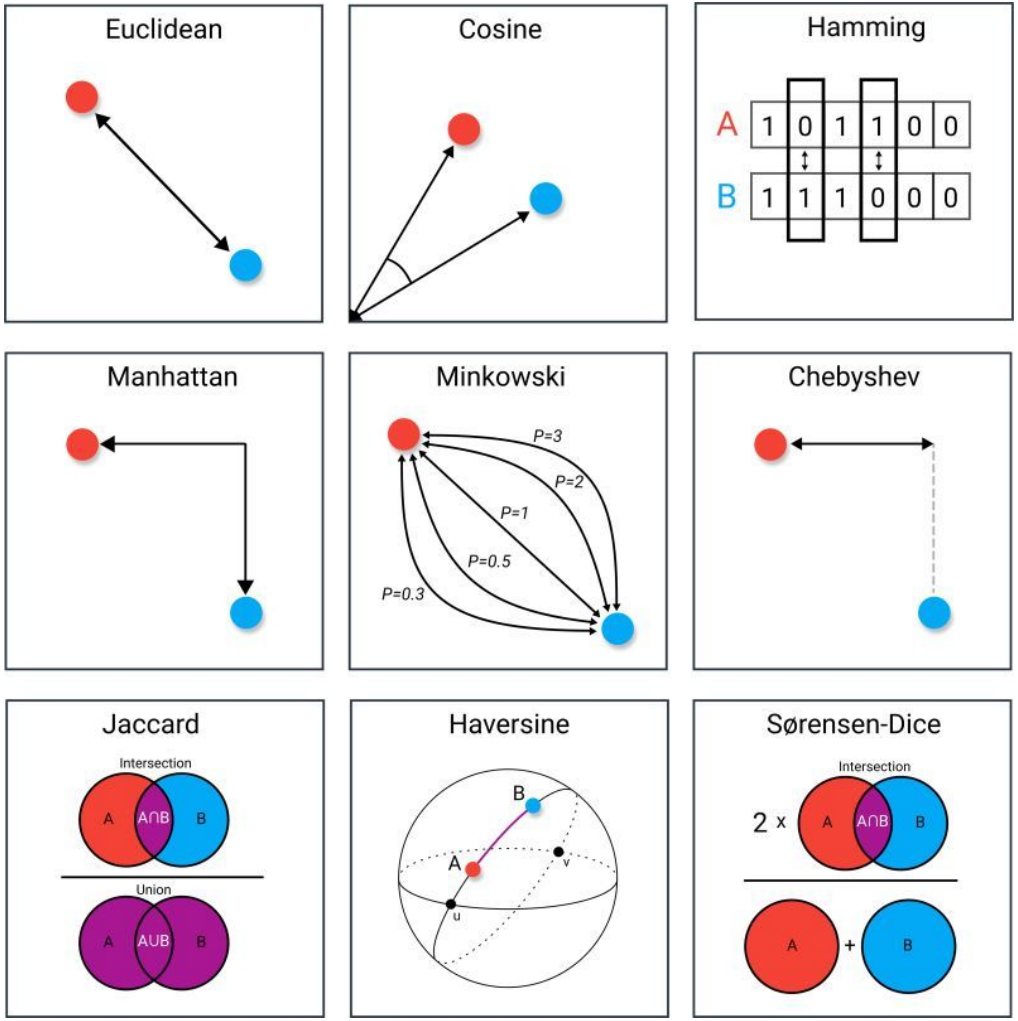


Primary image matching at Hometogo

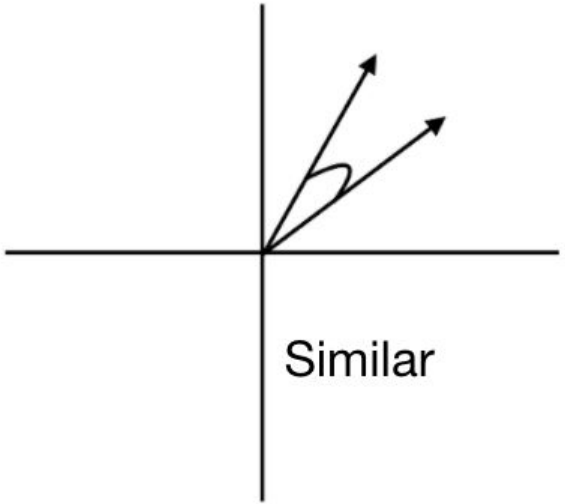
- Inventory understanding (500 million images)
- Providing the best deals to users (sample use case: strike prices)

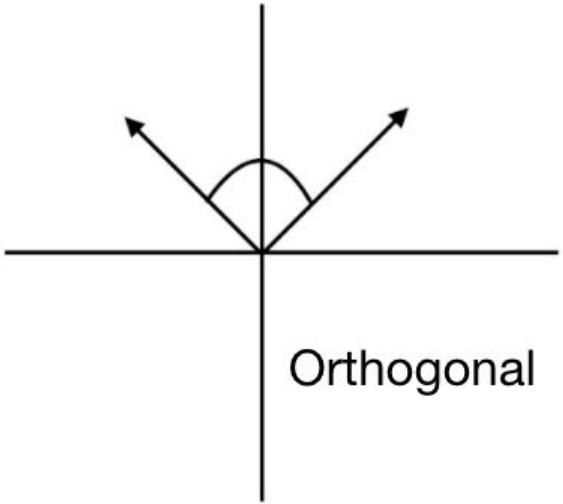





Euclidean
Cosine
Hamming
A 1 0 1 1 0 0
B 1 1 1 0 0 0
Manhattan
Minkowski
P=3
P=2
P=1
P=0.5
P=0.3
Chebyshev
Jaccard
Intersection
A
A∩B
B
Union
A
A∪B
B
Haversine
B
A
v
u
Sørensen-Dice
Intersection
2 x
A
A∩B
B
A
+
B

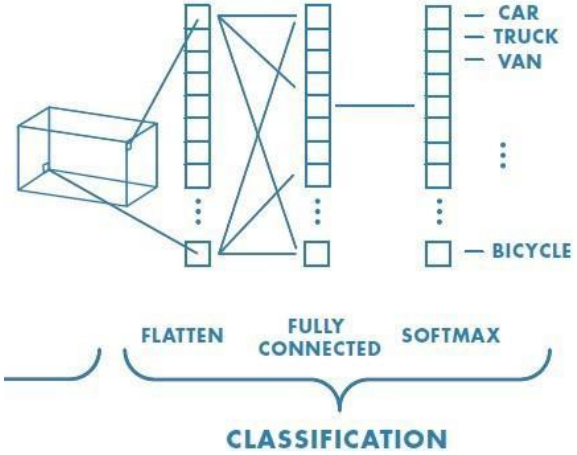
Cosine Similarity





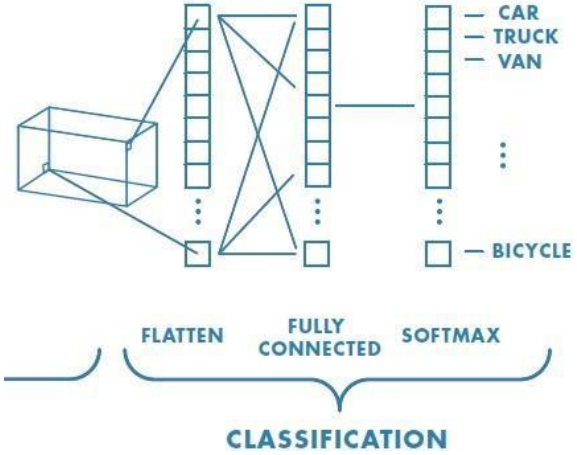
$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Convolutional networks as embeddings creators



Cosine similarity = 0.65

Convolutional networks as embeddings creators



Cosine similarity = 0.99

README.md

Art Recommendation system

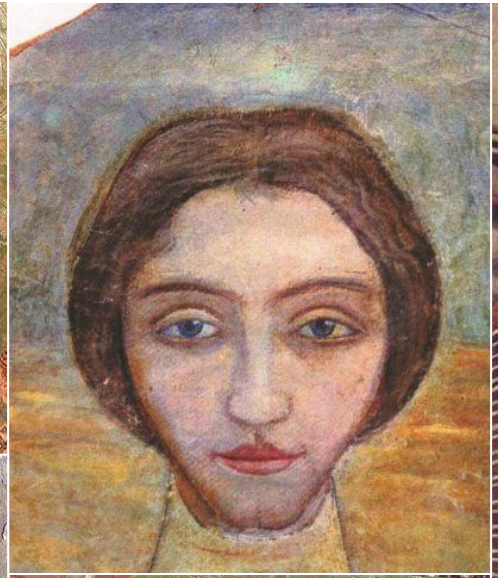
This is the repository of a portfolio project at DSR. This project aims to identify similar images using pre-trained computer vision networks. For an explanation of the technology see the technology section.

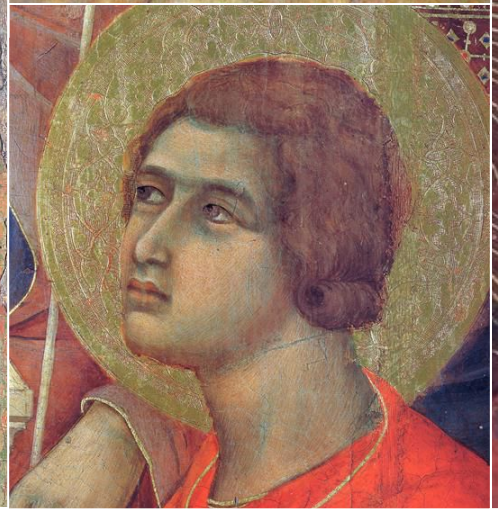
Contributors

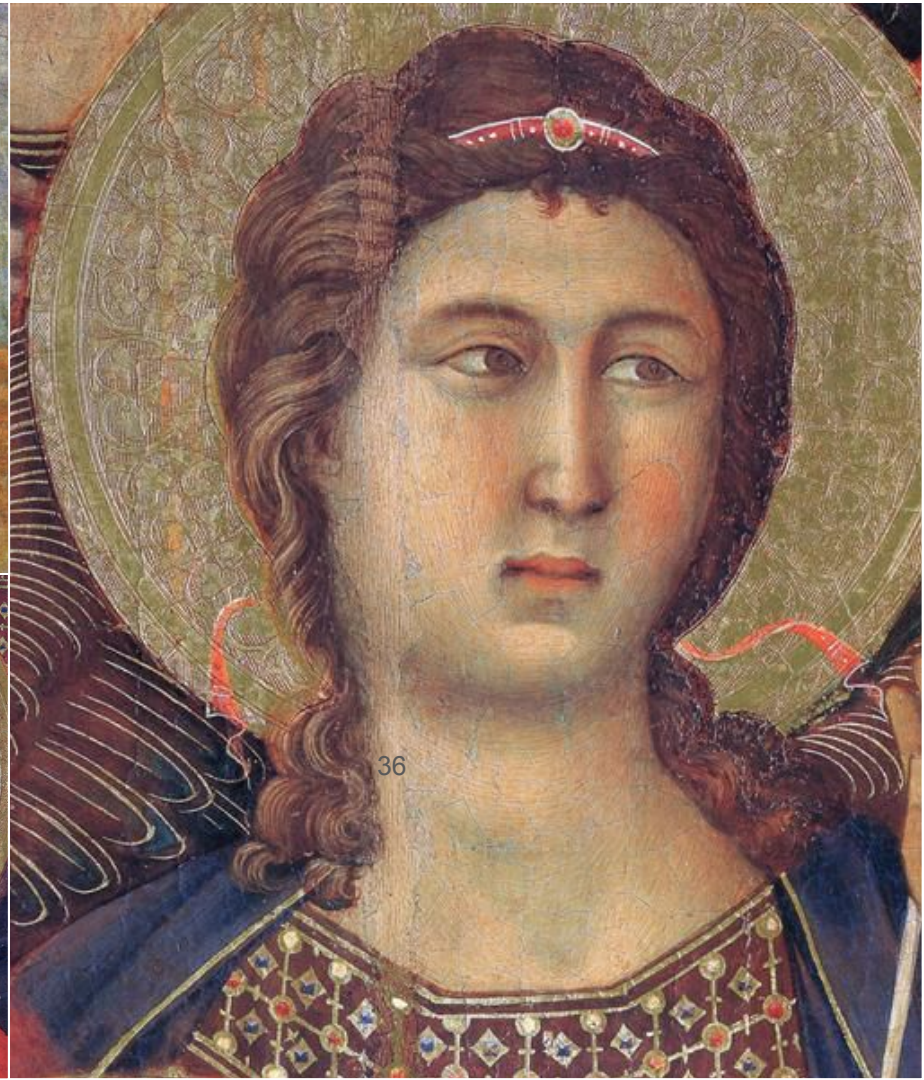
- Catarina Ferreira
- Gargi Maheshwari

https://github.com/gargimaheshwari/Wikiart-similar-art


12.8.1945







Perceptual hashing

94088af86c038327
14ee7fe587860078a1109033318bd986

94088af86c038327
07aaedb9b75e88a6051184f01be5cc50

Perceptual hashing

Perceptual hashing

94088af86c038327	1	0	0	0	0	0	1	1	0	0	1	0	**0**	**1**	**1**	**1**
94088af86c038328	1	0	0	0	0	0	1	1	0	0	1	0	**1**	**0**	**0**	**0**

Distance is **4 bit**

Image similarity

- Exercise
- Solution

K-medoids clustering

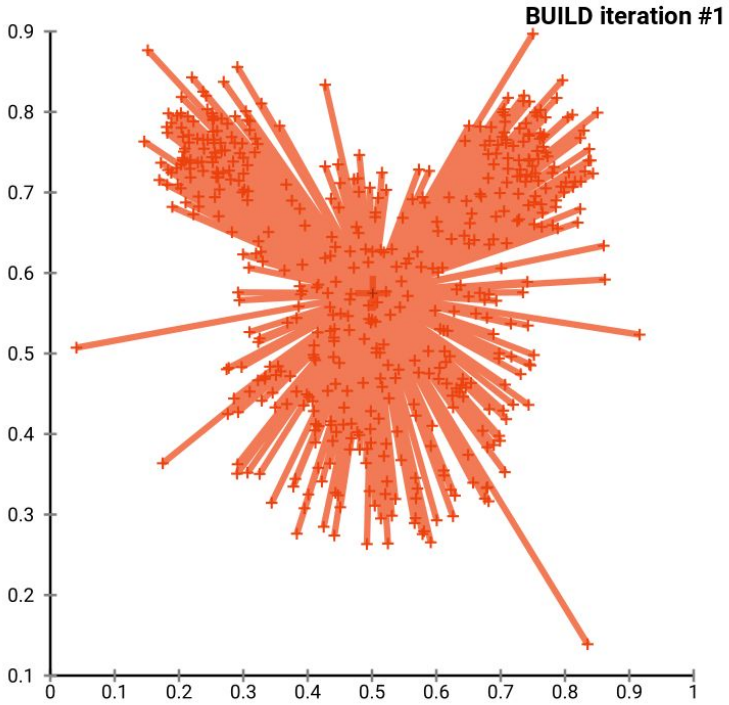
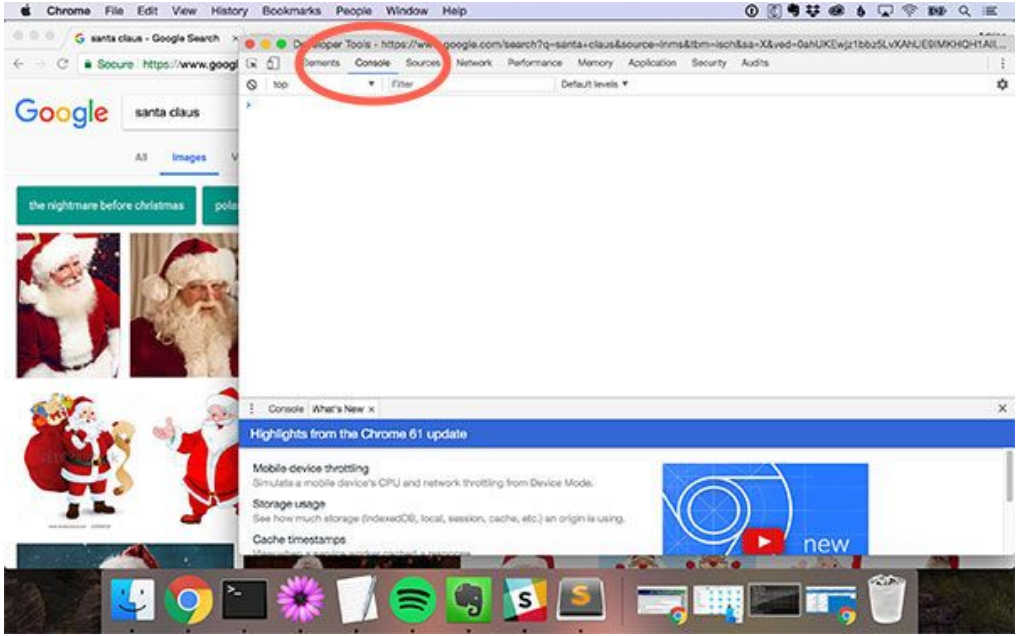


Image clustering

- Exercise
- Solution
- V2 Exercise (all images)
- V2 Solution (all images)

Scraping a dataset from Google Images



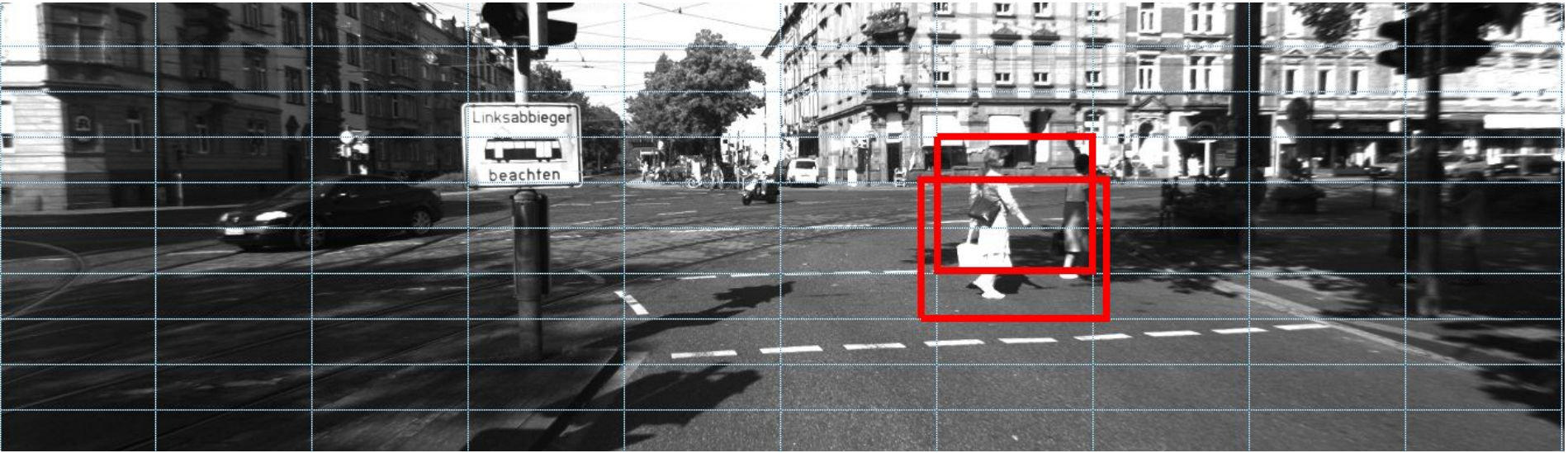
[PyImageSearch guide]

Goals of object detection

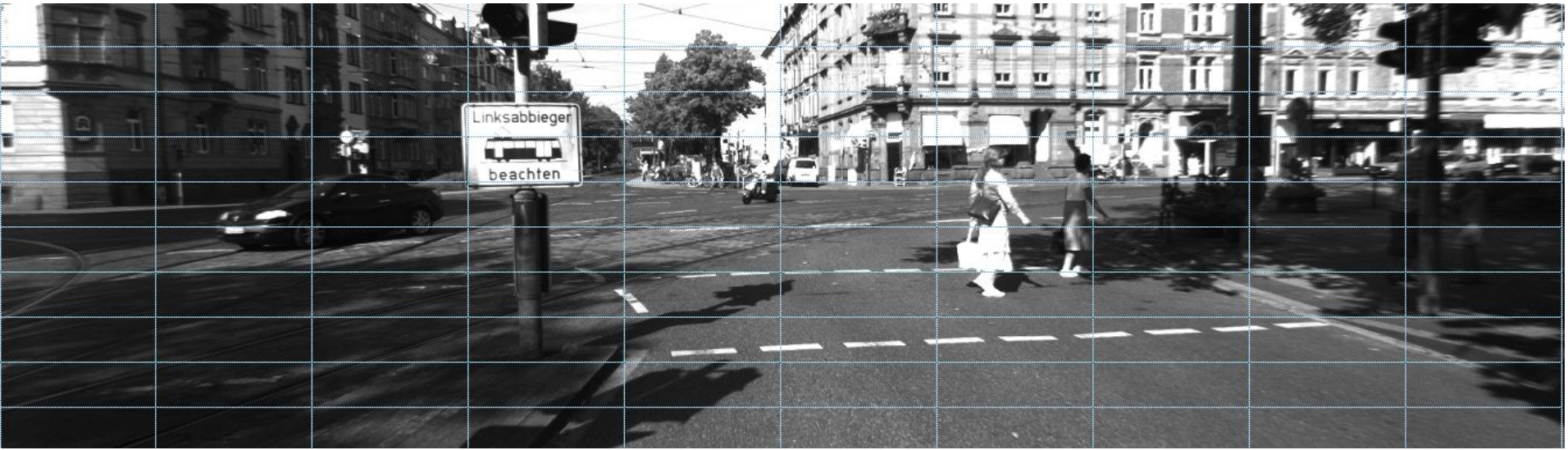
Given an input image we wish to obtain:

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

Breaking down an image into regions

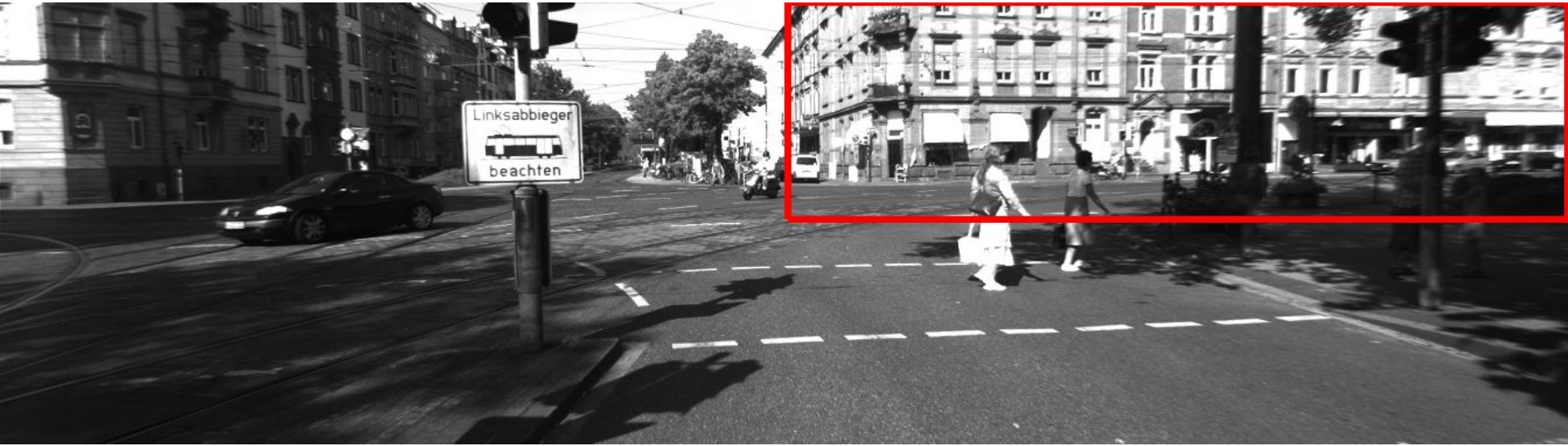


Breaking down an image into regions

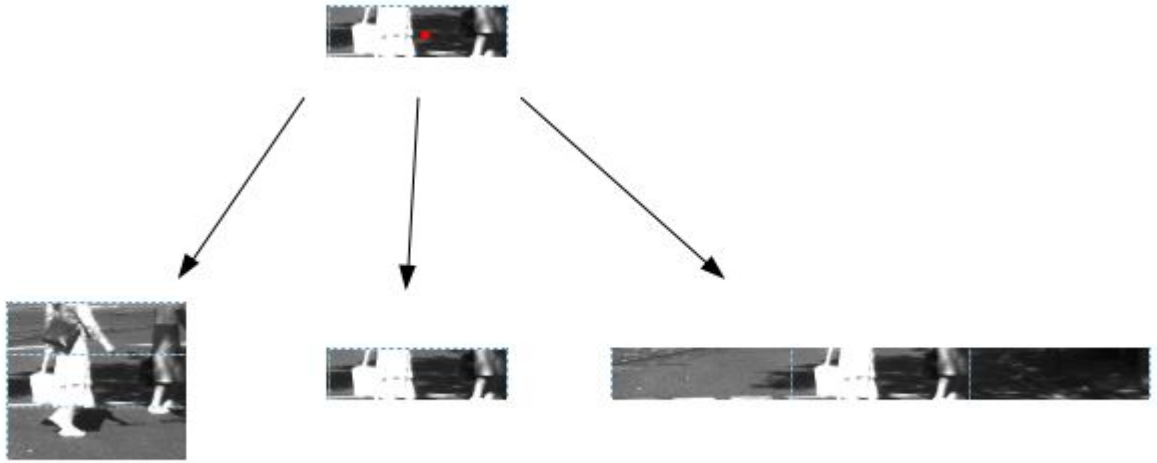


A sliding window approach partitions the image into discrete blocks

Breaking down an image into regions

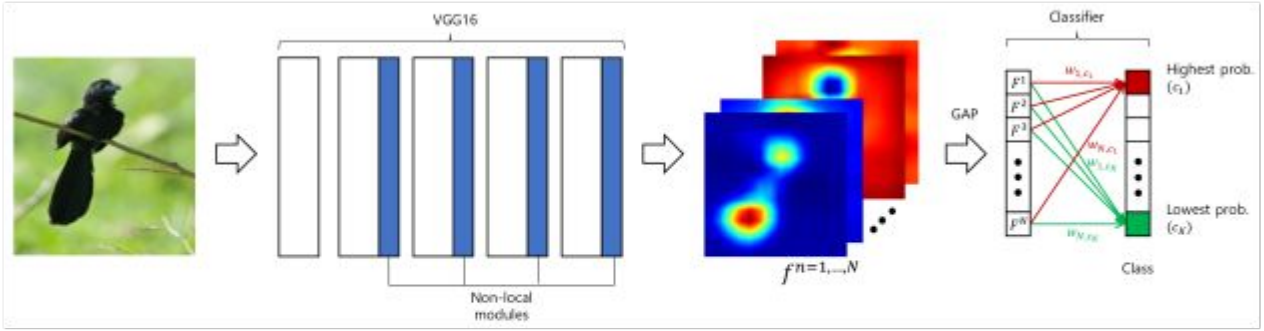


Breaking down an image into regions



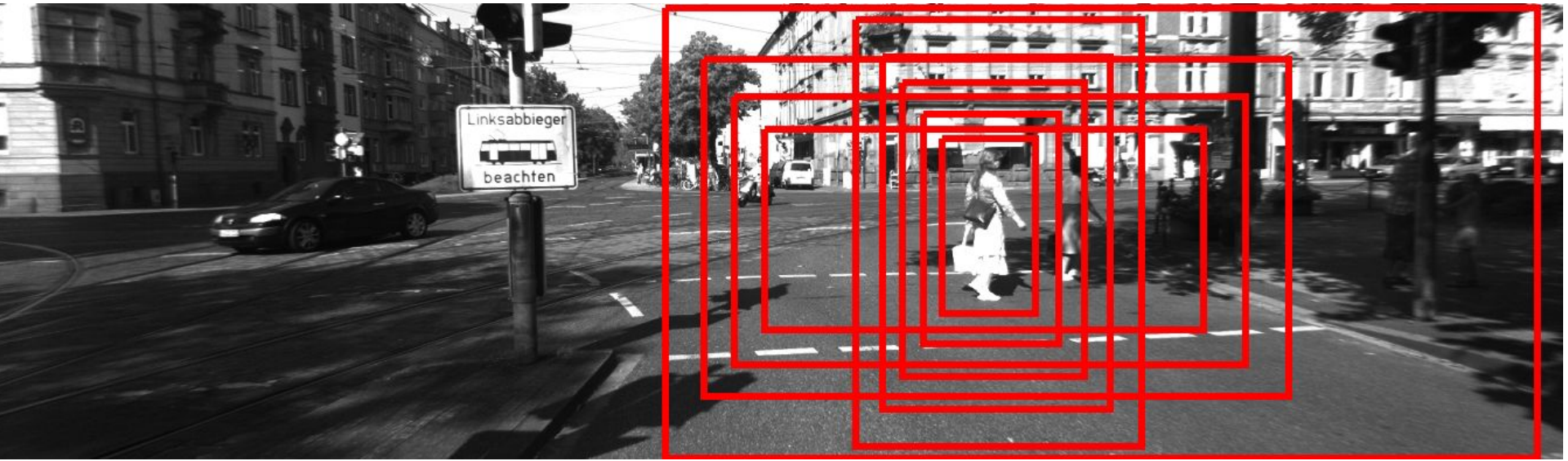
Finding the optimal image partitioning algorithm is not a trivial task

Transfer learning of classification is currently used for region proposals

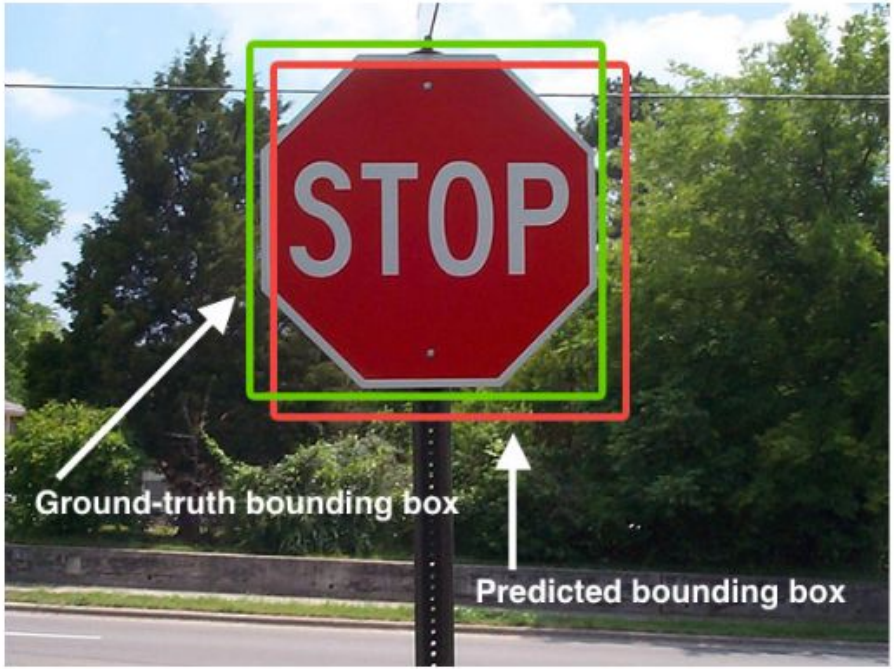


Source

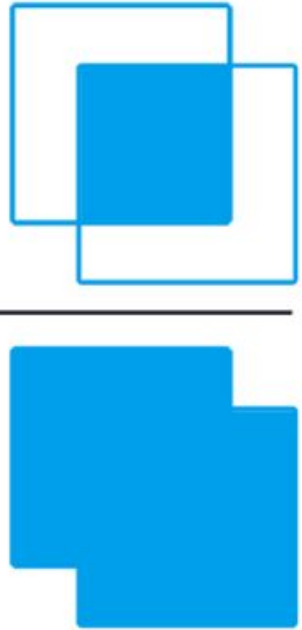
Breaking down an image into regions



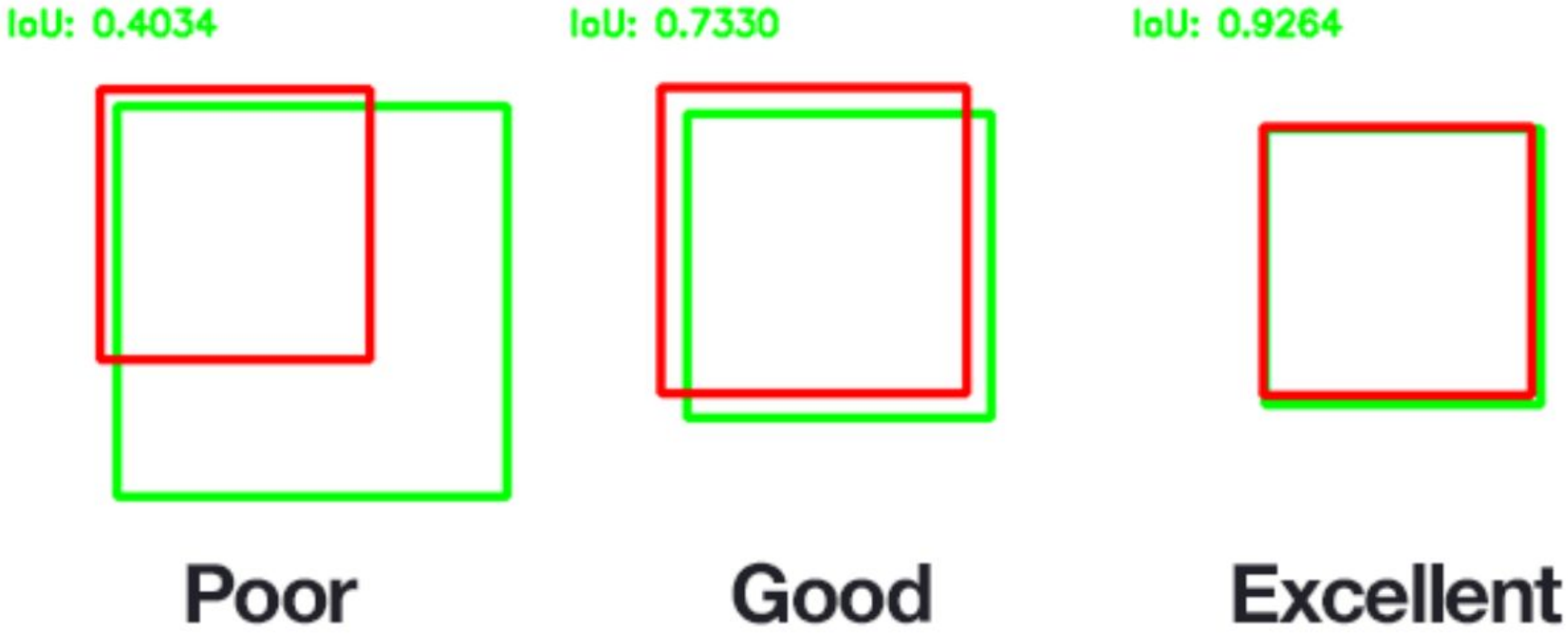
Measuring performance with IoU



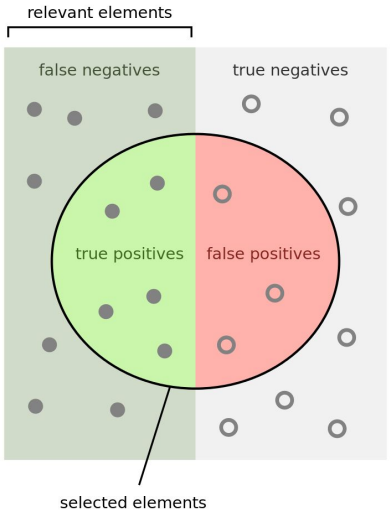
$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

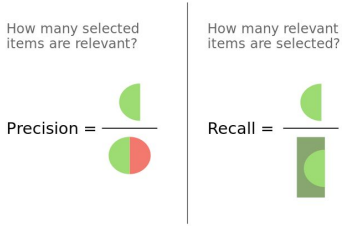


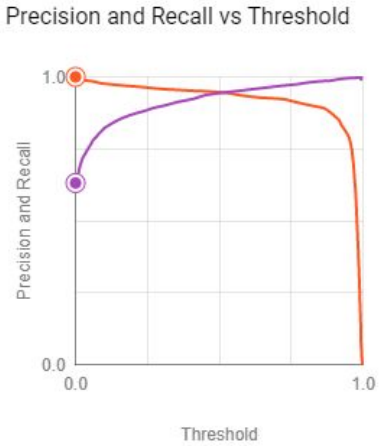
Measuring performance with IoU



Precision, recall, and f1 metrics



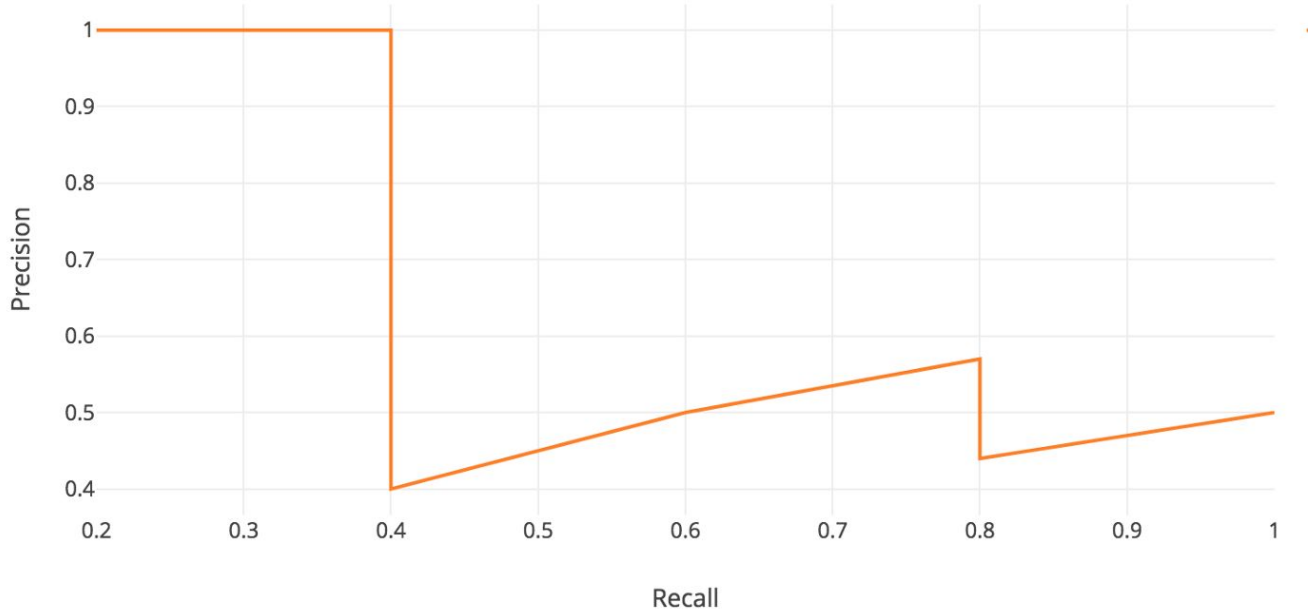




$$F_1 = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

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)



Understanding mean Average Precision (mAP)

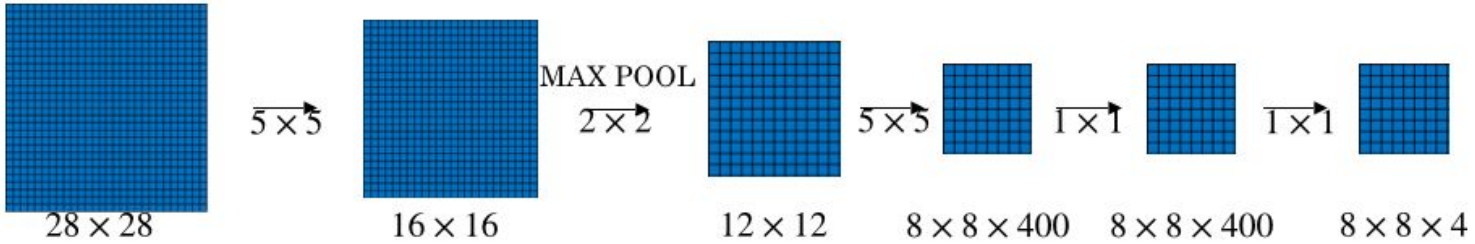
Model	Train	Test	mAP	FLOPS	FPS
SSD300	COCO trainval	test-dev	41.2	-	46
SSD500	COCO trainval	test-dev	46.5	-	19
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40
Tiny YOLO	COCO trainval	-	-	7.07 Bn	200
SSD321	COCO trainval	test-dev	45.4	-	16
DSSD321	COCO trainval	test-dev	46.1	-	12
R-FCN	COCO trainval	test-dev	51.9	-	12
SSD513	COCO trainval	test-dev	50.4	-	8
DSSD513	COCO trainval	test-dev	53.3	-	6
FPN FRCN	COCO trainval	test-dev	59.1	-	6
Retinanet-50-500	COCO trainval	test-dev	50.9	-	14
Retinanet-101-500	COCO trainval	test-dev	53.1	-	11
Retinanet-101-800	COCO trainval	test-dev	57.5	-	5
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35
YOLOv3-416	COCO trainval	test-dev	57.9	140.69 Bn	20

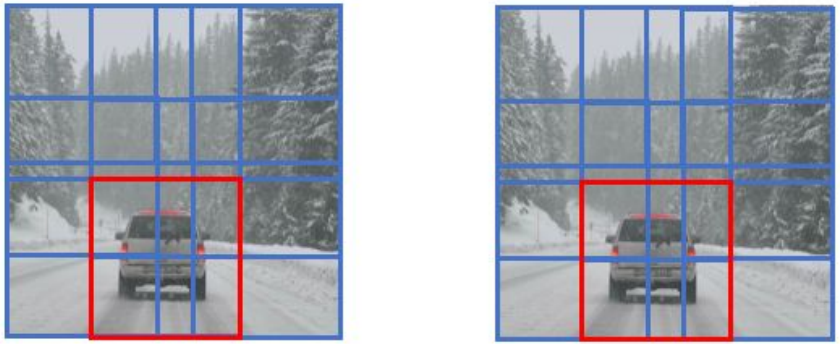
https://pjreddie.com/darknet/yolo/

Mean average precision

- Exercise
- Solution

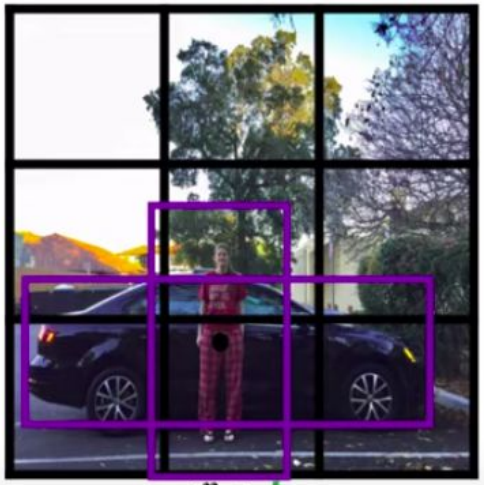
Sliding windows





Andrew Ng

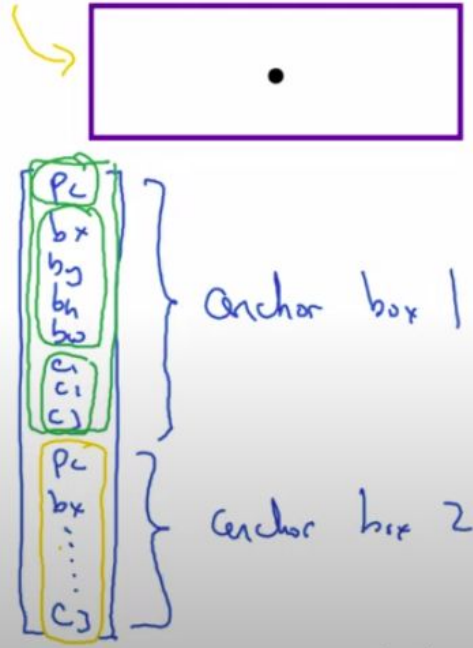
Overlapping objects:



$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Anchor box 1:

Anchor box 2:

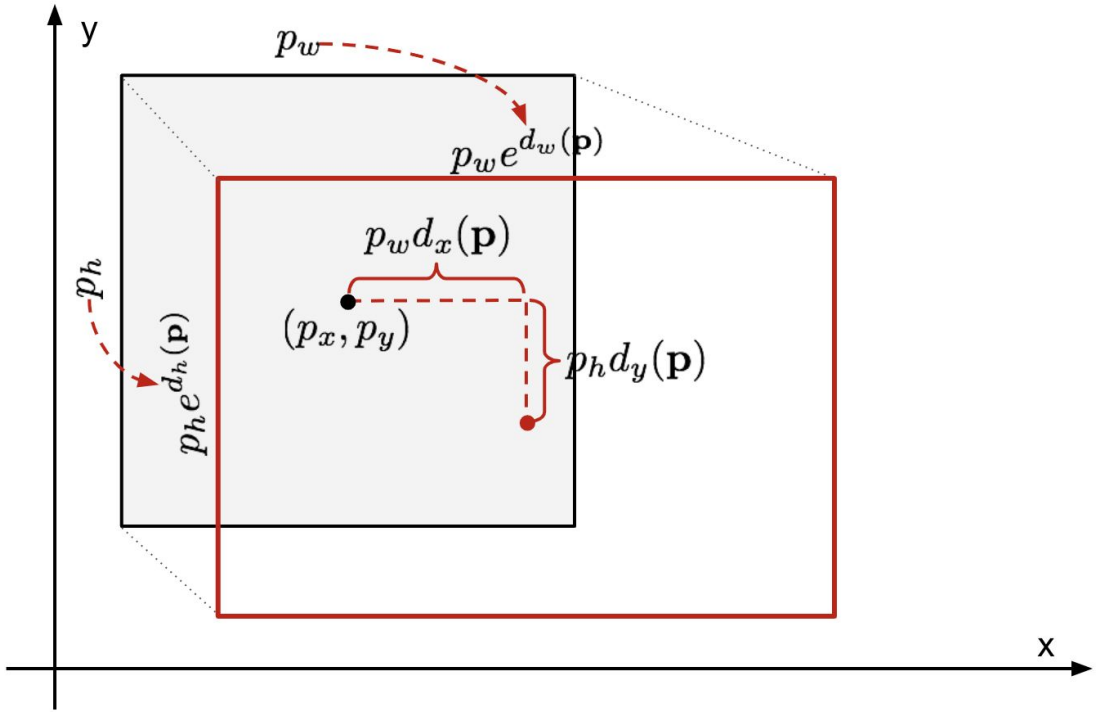


Redmon et al., 2015, You Only Look Once: Unified real-time object detection]

Andrew Ng

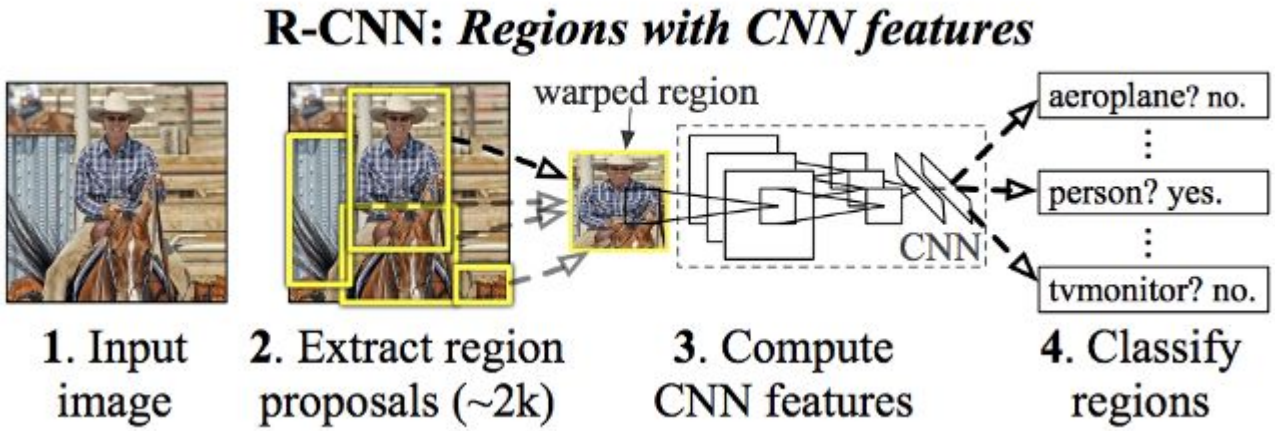
https://www.youtube.com/watch?v=RTlwl2bv0Tg

Object detection as bounding box regression (1st step)



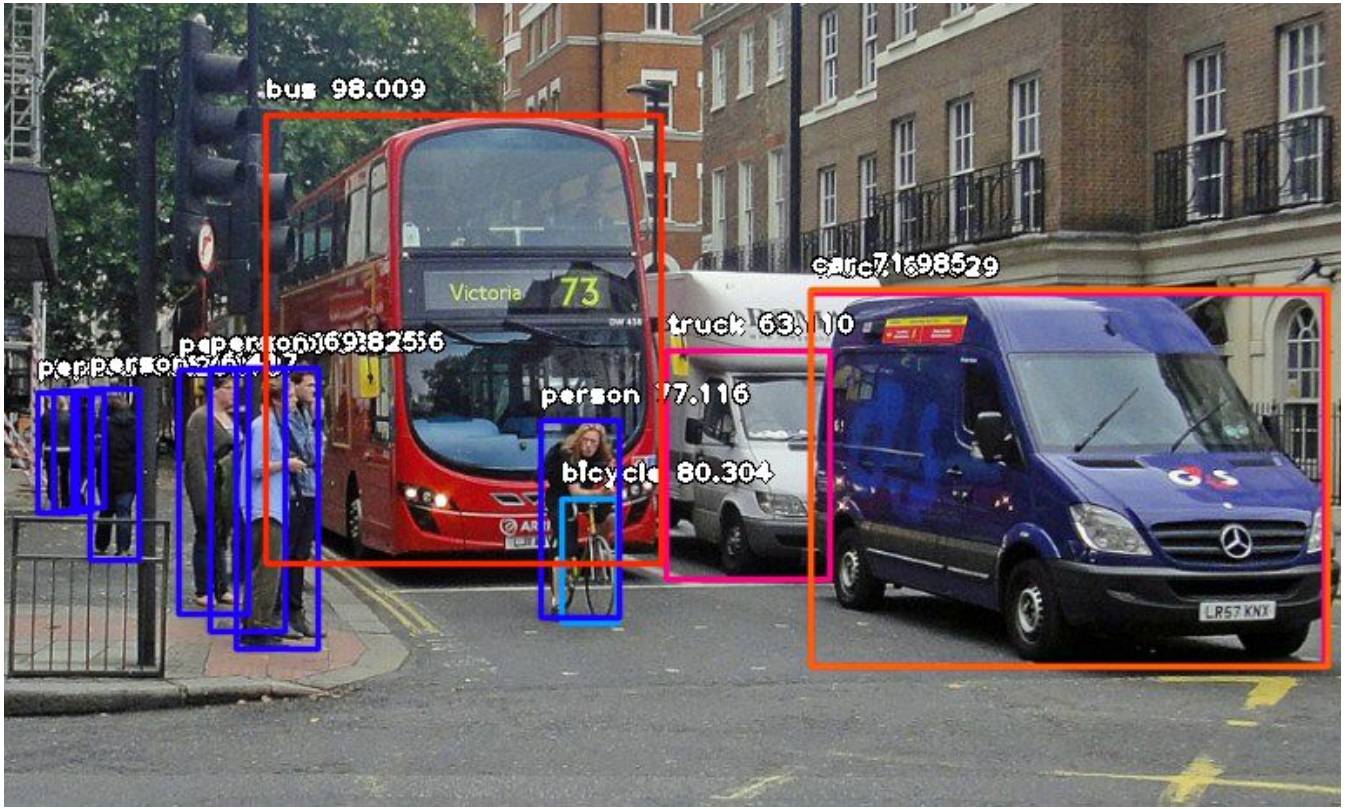
‘Regression’ means predicting a continuous value

Object detection as classification (2nd step)



After step 1, a classification algorithm (SVM, another neural network, etc.) will output a class label REF

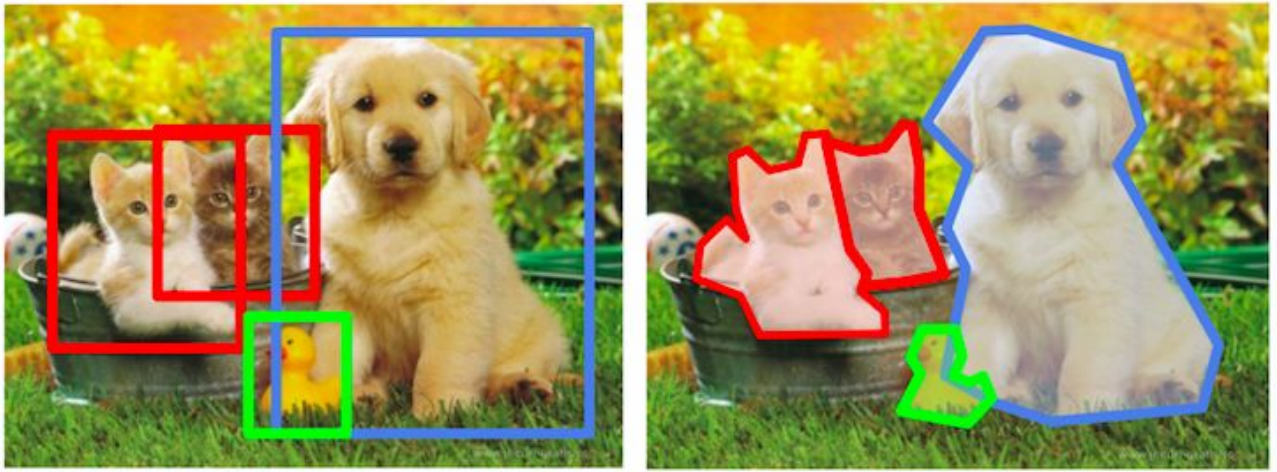
Object detection as bounding box localization



Object detection as bounding box localization

Class	Score	Location
Apple	0.92	[18, 21, 57, 63]
Banana	0.88	[100, 30, 180, 150]
Strawberry	0.87	[7, 82, 89, 163]
~~Banana~~	~~0.23~~	~~[42, 66, 57, 83]~~
~~Apple~~	~~0.11~~	~~[6, 42, 31, 58]~~

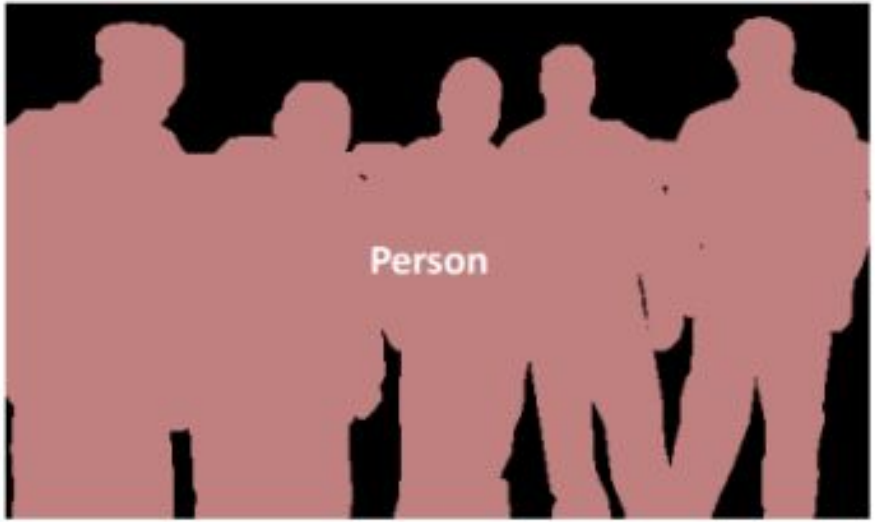
Instance segmentation requires object detection



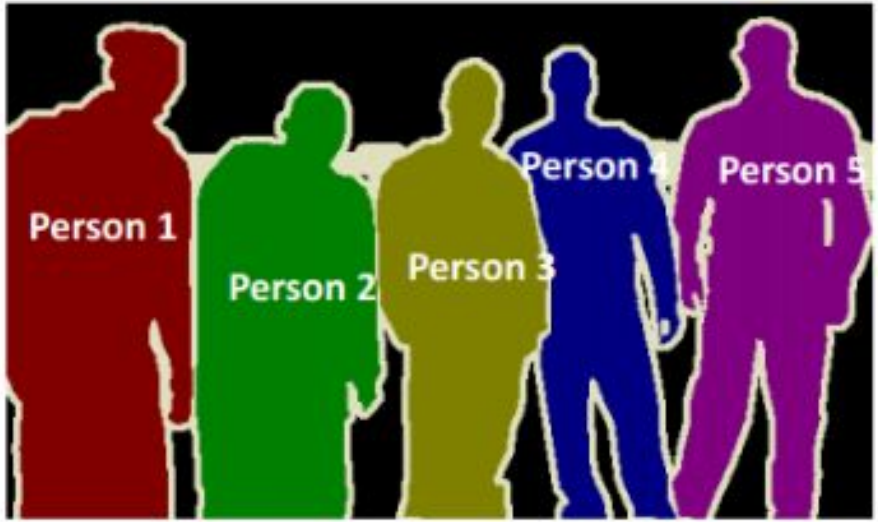
CAT, DOG, DUCK CAT, DOG, DUCK

Multiple objects

Segmentation as pixel-wise localization

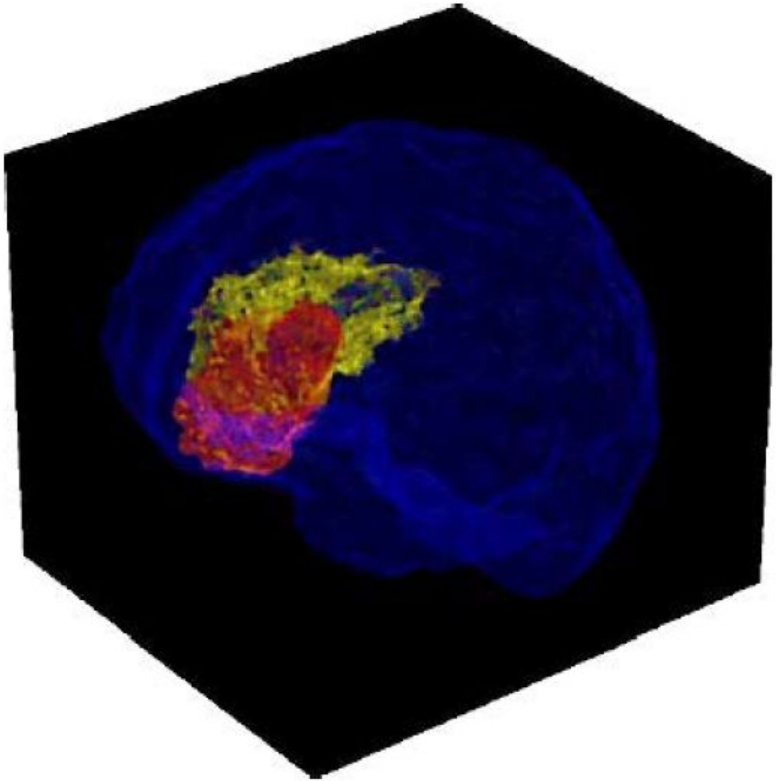


Semantic Segmentation



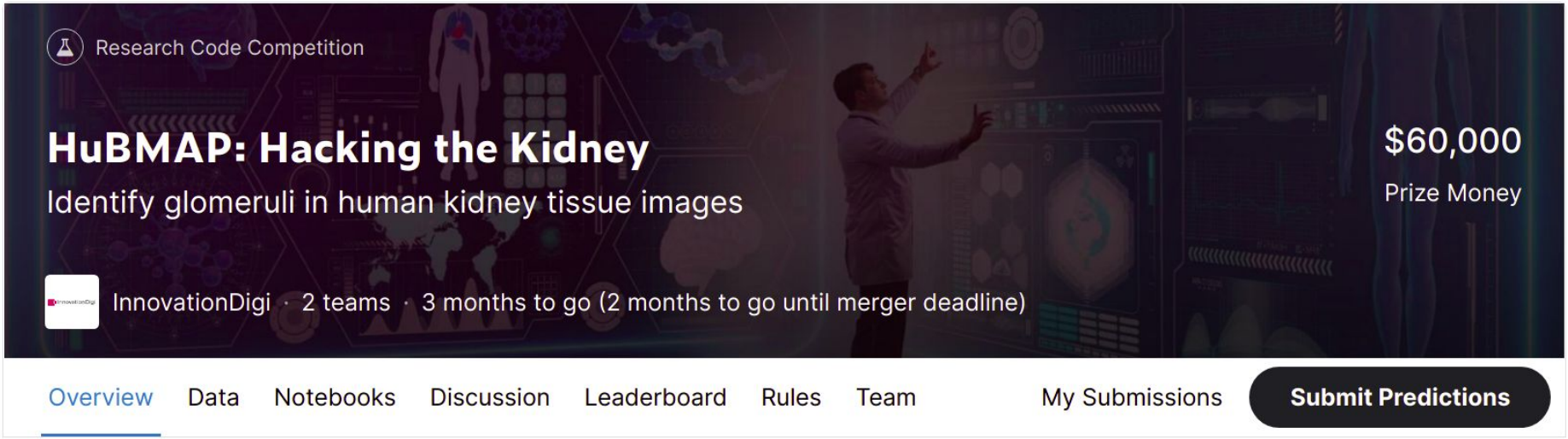
Instance Segmentation

Segmentation as pixel-wise localization

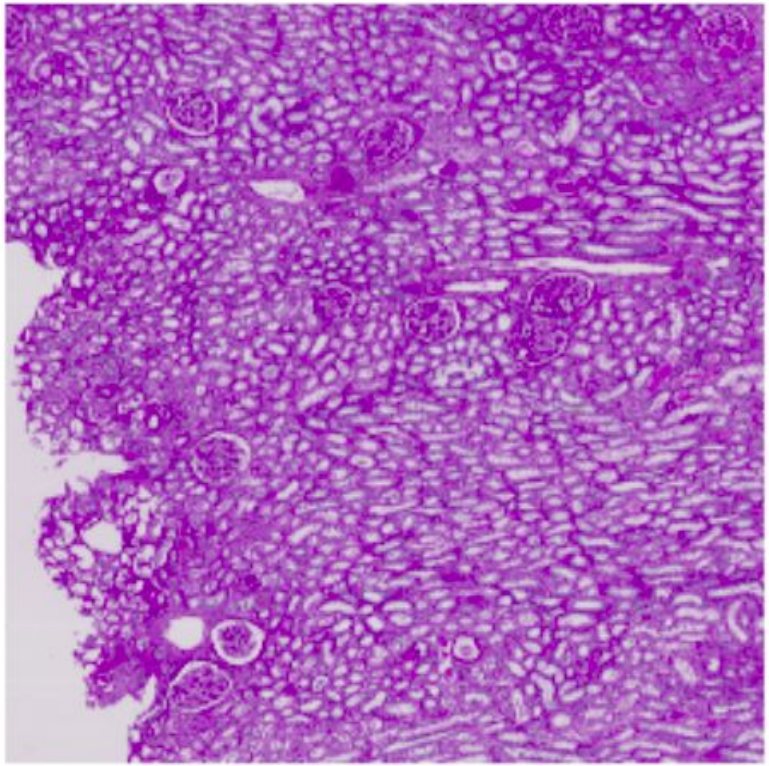


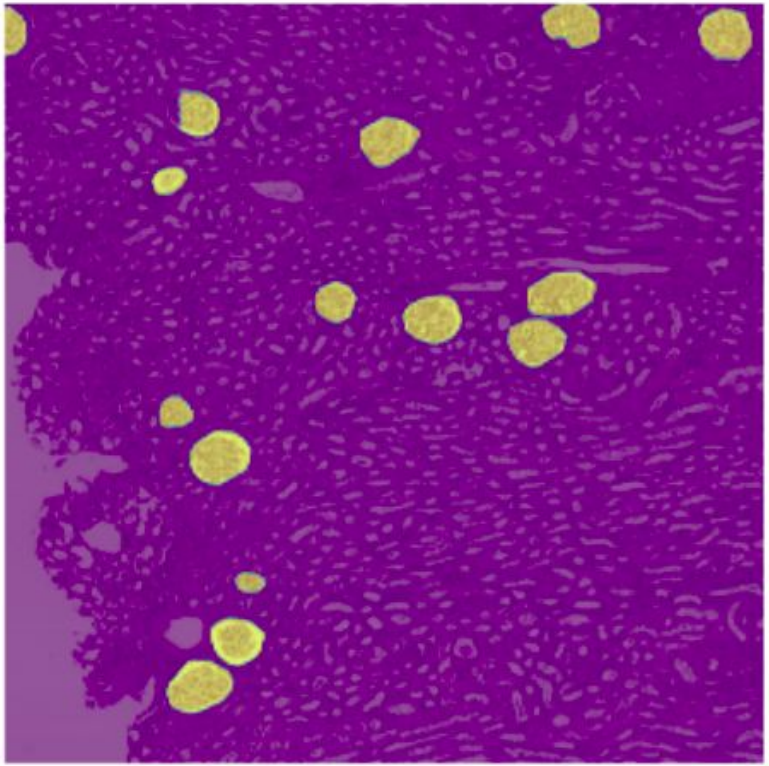
Link

Segmentation as pixel-wise localization

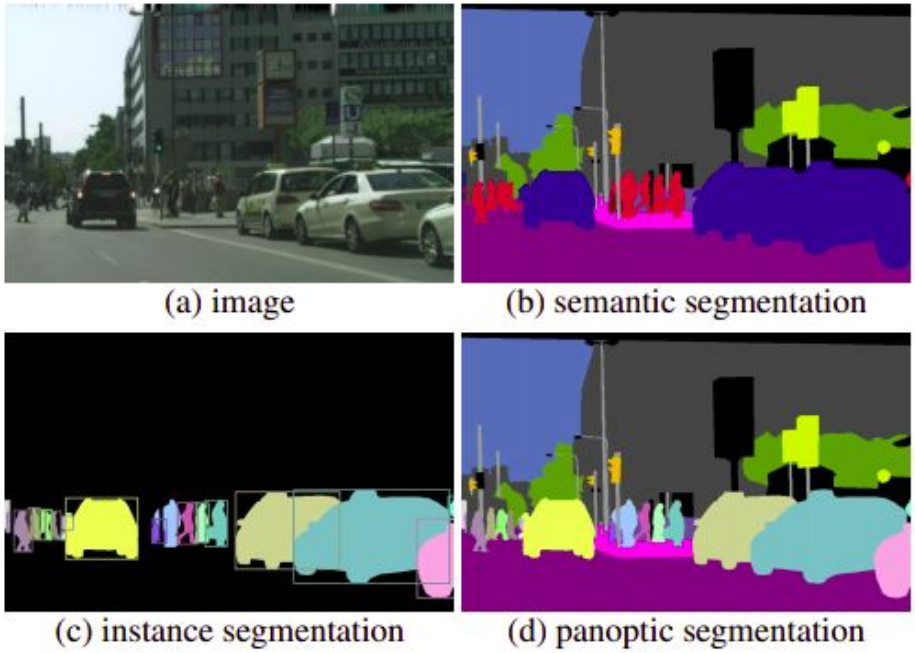


Segmentation as pixel-wise localization





Panoptic segmentation



https://arxiv.org/pdf/1801.00868.pdf

Explore it in the detectron2 inference notebook

Two main families of object detection methods

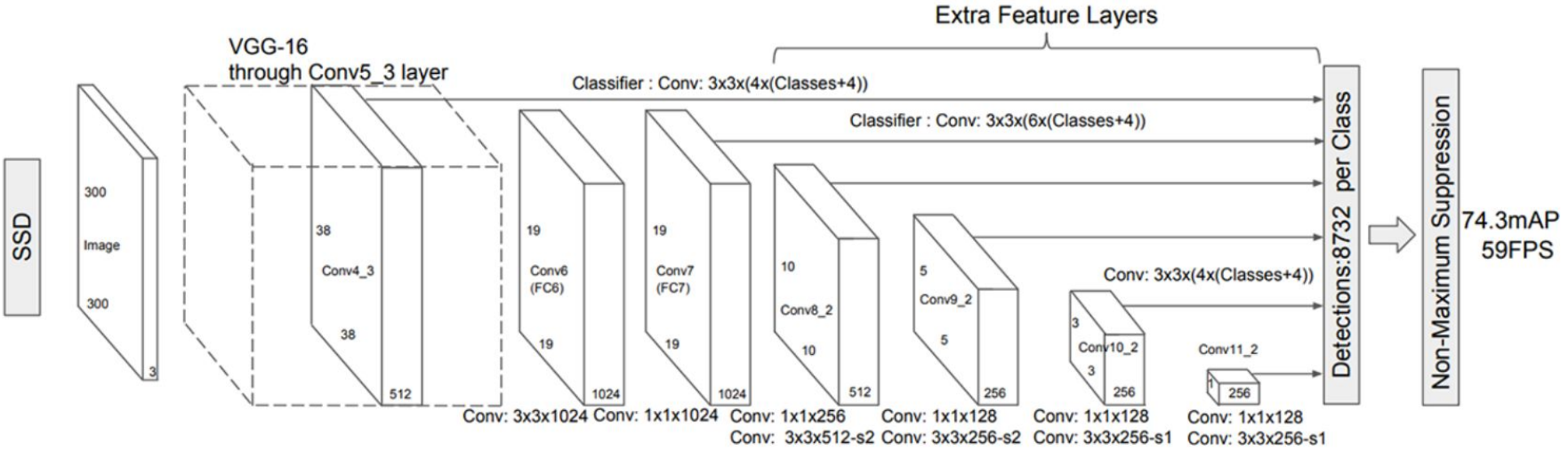
Two stage methods (R-CNN family)

- 1st stage: use a region proposal method to select possible object areas
- 2nd stage: classify proposed regions
- **Two stage methods are accurate, but slow**

Single stage methods (YOLO, Retinanet, and SSD):

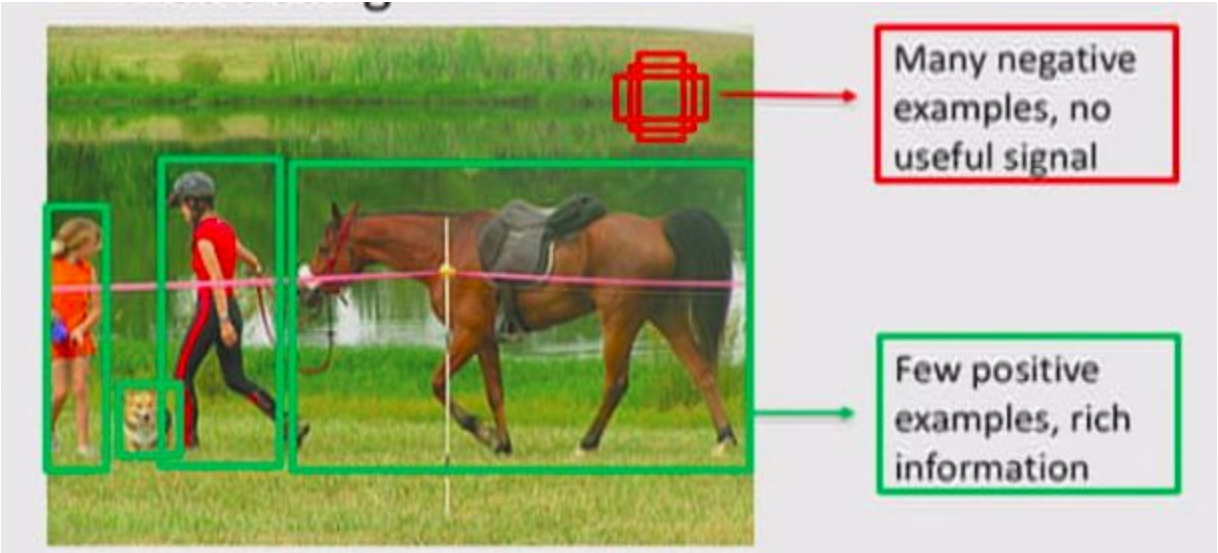
- No region proposal
- **Single stage methods are fast, but less accurate**

Single Shot Detectors



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

Retinanet's solution to the class imbalance problem



RetinaNet: how Focal Loss fixes Single-Shot Detection

‘You only look once’ (YOLO)

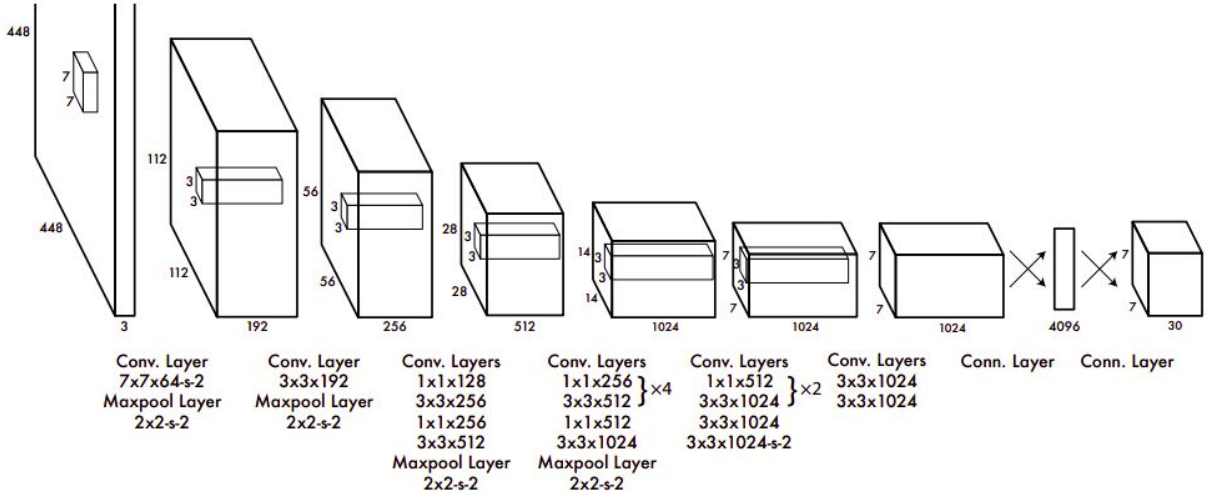
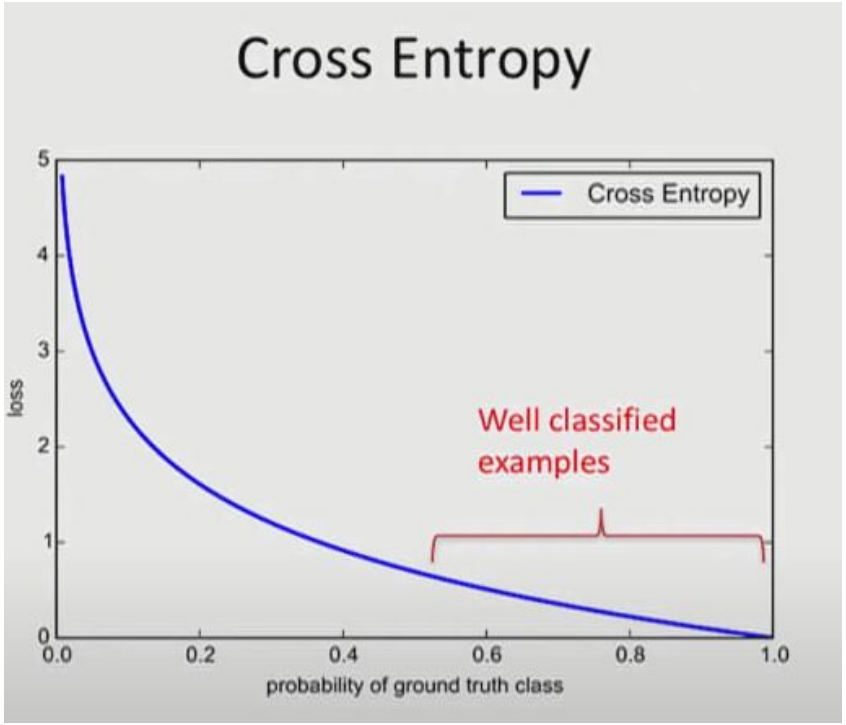


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

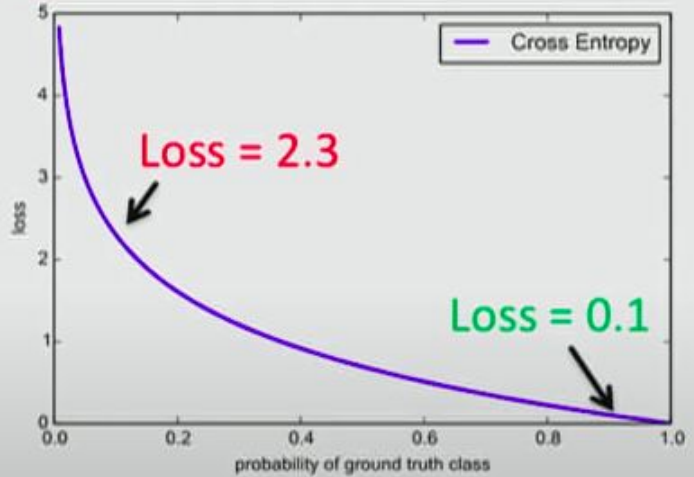
Cross entropy at different confidence levels



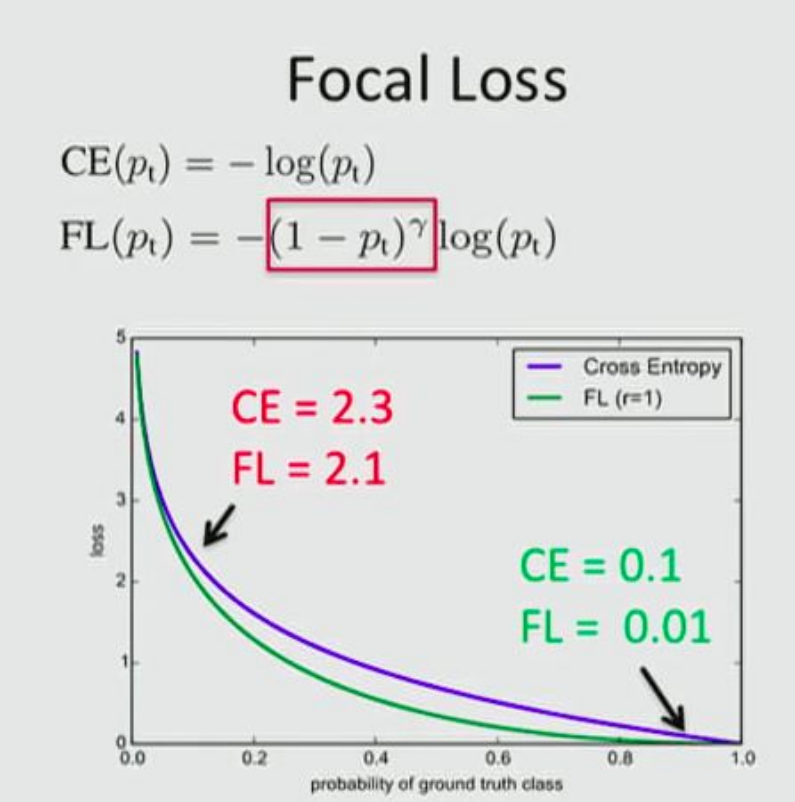
Andrew Ng's explanation of cross entropy loss

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 γ

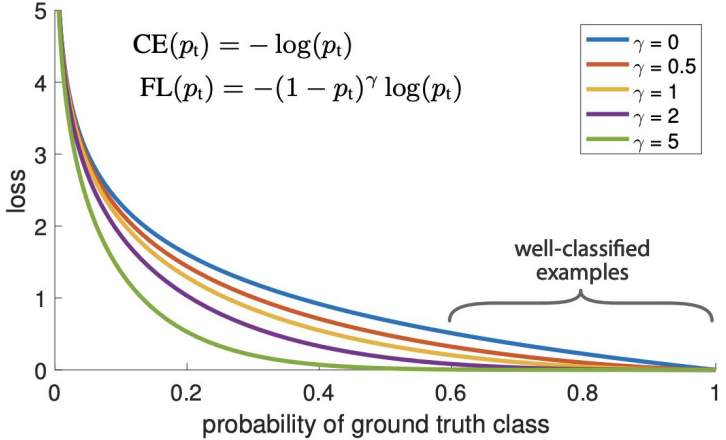


Figure 1. We propose a novel loss we term the *Focal Loss* that adds a factor $(1 - p_t)^{\gamma}$ to the standard cross entropy criterion. Setting $\gamma > 0$ reduces the relative loss for well-classified examples ($p_t > .5$), putting more focus on hard, misclassified examples. As our experiments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

γ = 0 regular cross entropy (no focus on infrequent samples)
γ = 2 show

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.

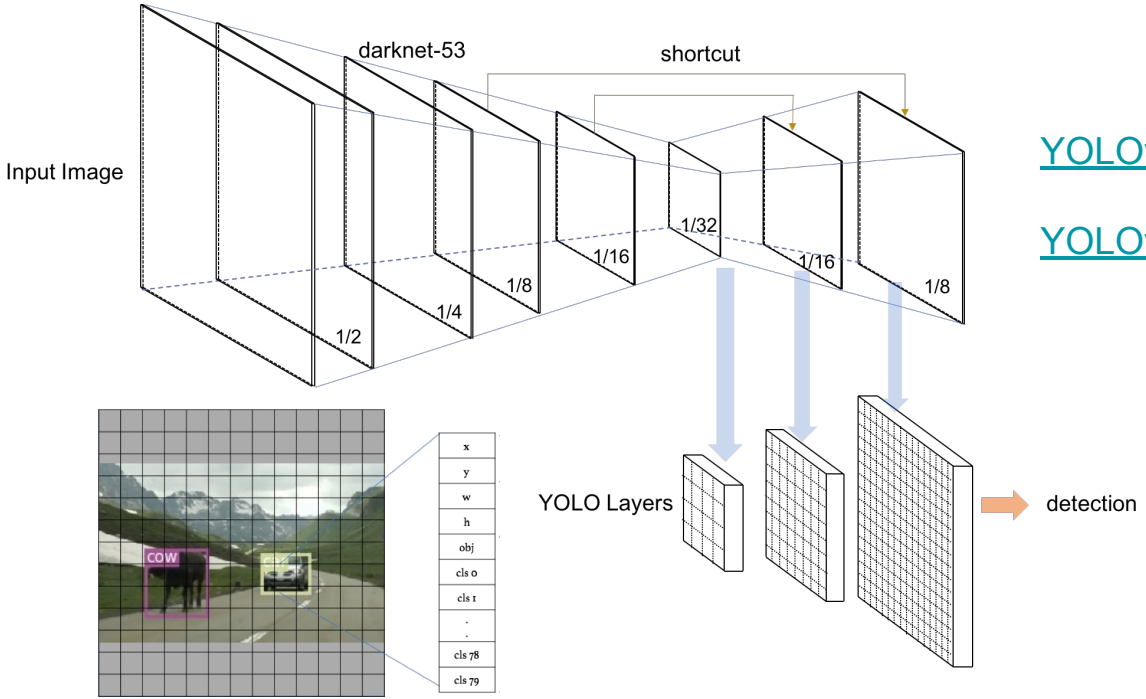
Anchor box x, y offset predictions. We tried using the normal anchor box prediction mechanism where you predict the x, y offset as a multiple of the box width or height using a linear activation. We found this formulation decreased model stability and didn't work very well.

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

‘You only look once’ (YOLO)



[YOLOv1 from scratch in PyTorch](#) (2 hours)

[YOLOv3 from scratch in PyTorch](#) (2 hours)

Two stage detectors: R-CNN variants

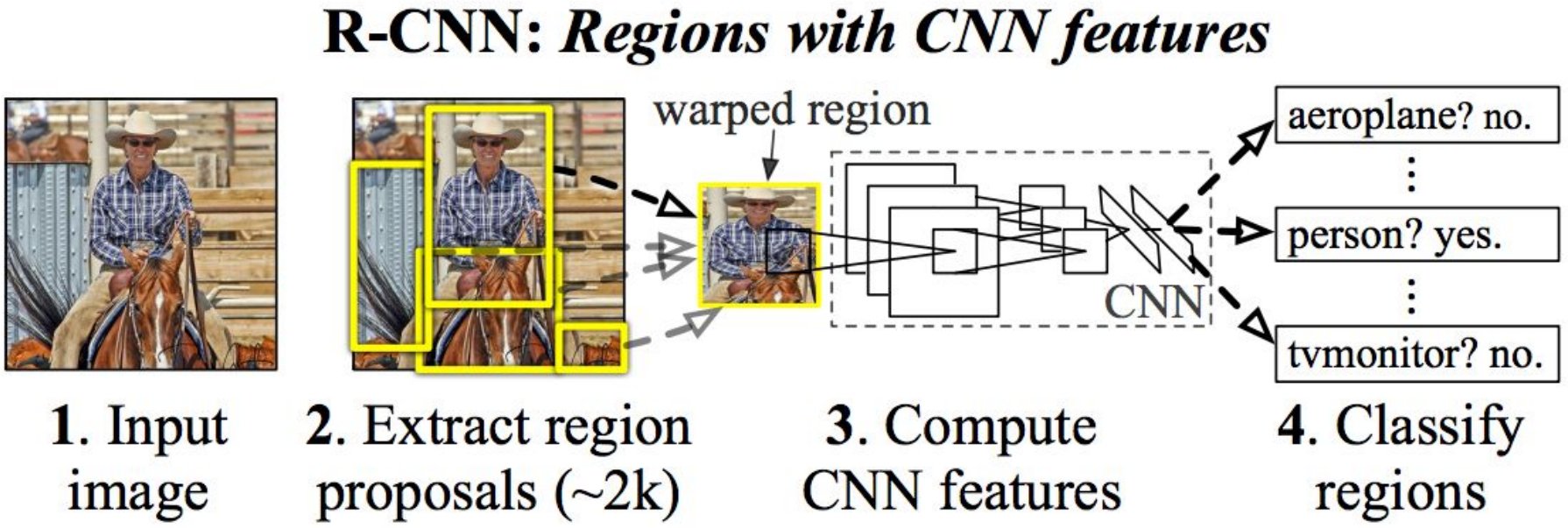
- **R-CNN**: Selective Search is run on the image, output segments from Selective Search are used for feature extraction and classification using a pre-trained CNN (two networks, separate training)
- **Fast R-CNN**: Uses the Selective Search algorithm to obtain region proposals, adding the Region of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

  (SVM)

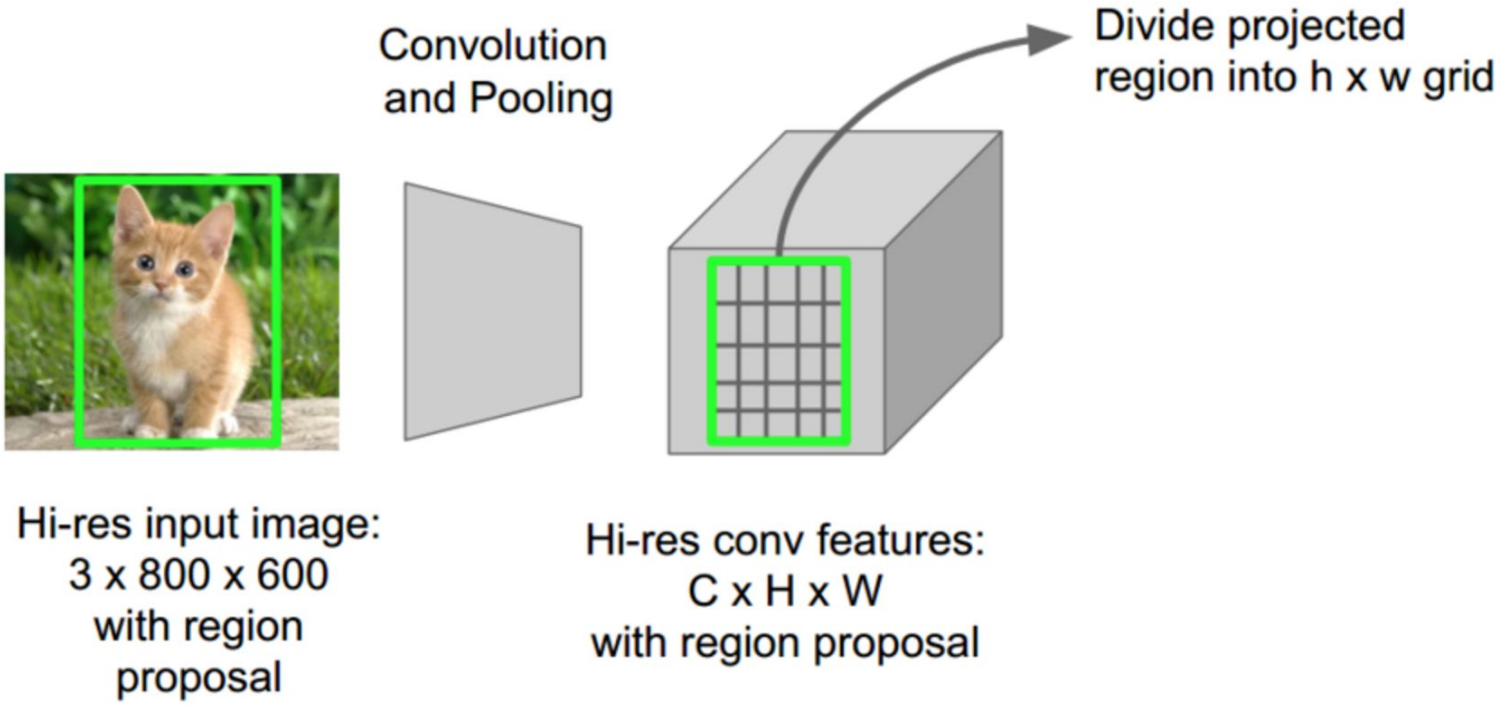
R-CNN


R-CNN: *Regions with CNN features*
warped region
CNN
aeroplane? no.
person? yes.
tvmonitor? no.
1. Input image
2. Extract region proposals (~2k)
3. Compute CNN features
4. Classify regions

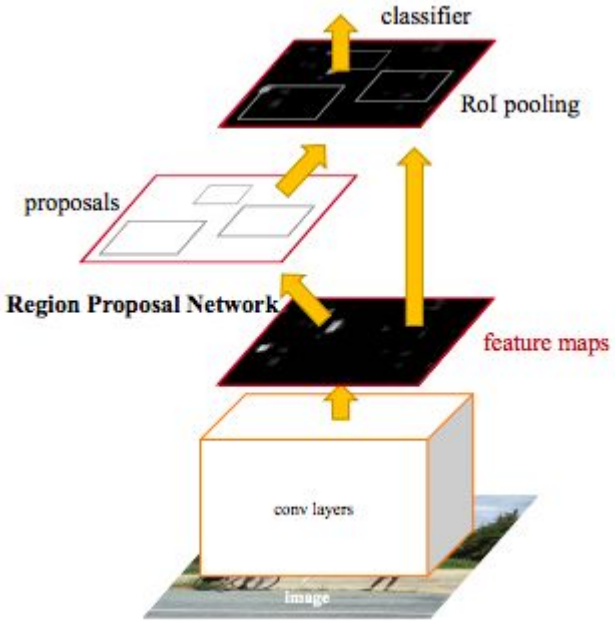
R-CNN

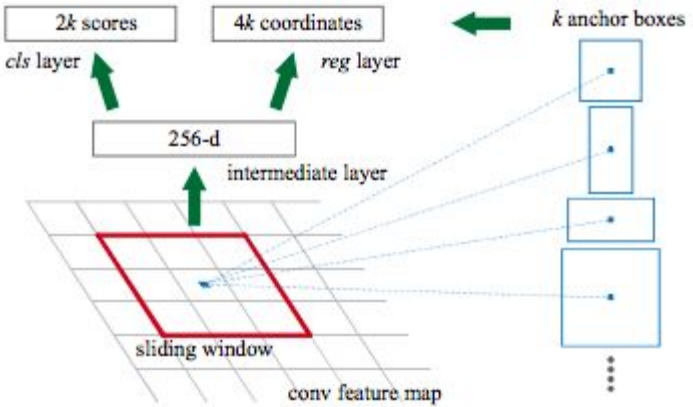
Region Proposals

Fast R-CNN


Convolution
and Pooling
Divide projected
region into h x w grid
Hi-res input image:
3 x 800 x 600
with region
proposal
Hi-res conv features:
C x H x W
with region proposal

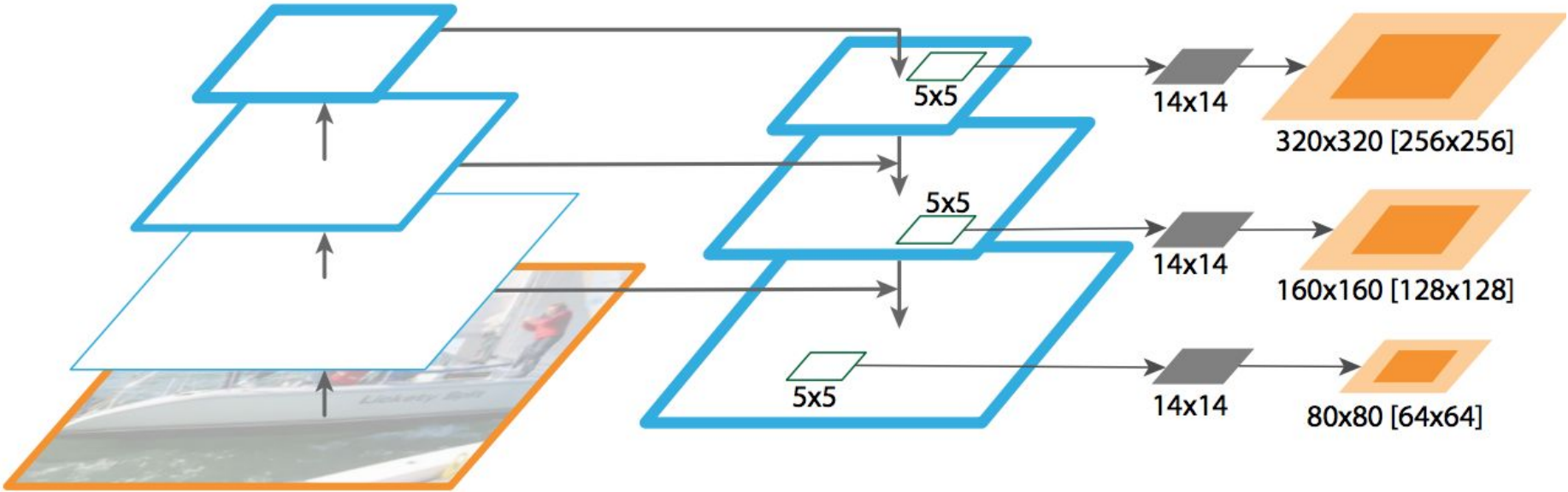
Region Proposal Networks - Faster R-CNN



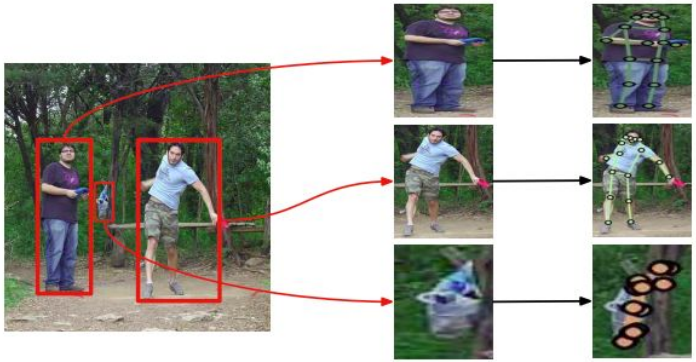


[Faster R-CNN breakdown]

Region Proposal Networks - Faster R-CNN



Keypoint detection



(1) Person detection + Crop (2) Pose estimation

Figure 1: Overview of our two stage cascade model. In the first stage, we employ a Faster-RCNN person detector to produce a bounding box around each candidate person instance. In the second stage, we apply a pose estimator to the image crop extracted around each candidate person instance in order to localize its keypoints and re-score the corresponding proposal.

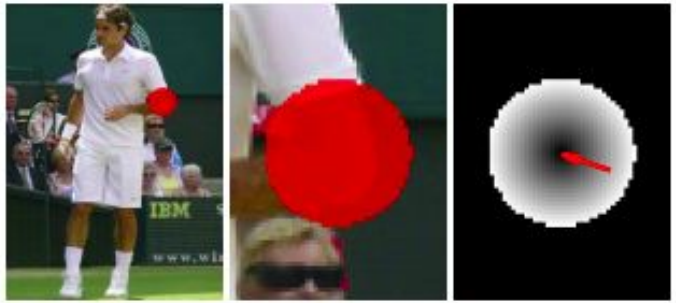


Figure 2: Network target outputs. *Left & Middle*: Heatmap target for the left-elbow keypoint (red indicates heatmap of 1). *Right*: Offset field L2 magnitude (shown in grayscale) and 2-D offset vector shown in red).

[Towards Accurate Multi-Person Pose Estimation in the Wild]

Figure 1: Overview of our two stage cascade model. In the first stage, we employ a Faster-RCNN person detector to produce a bounding box around each candidate person instance. In the second stage, we apply a pose estimator to the image crop extracted around each candidate person instance in order to localize its keypoints and re-score the corresponding proposal.

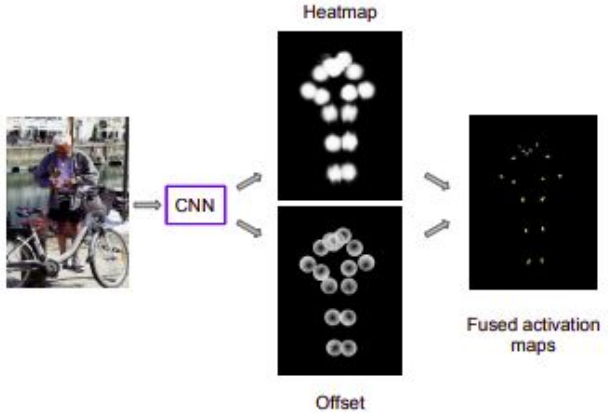
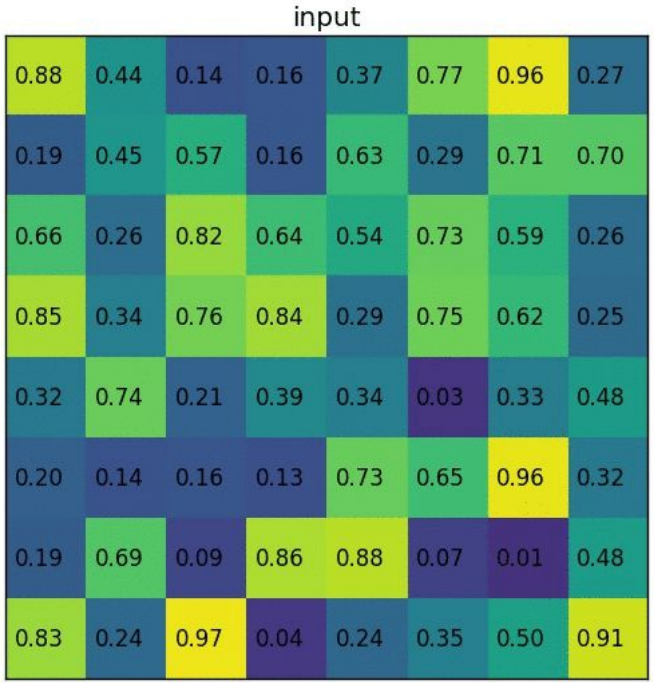


Figure 3: Our fully convolutional network predicts two targets: (1) Disk-shaped heatmaps around each keypoint and (2) magnitude of the offset fields towards the exact keypoint position within the disk. Aggregating them in a weighted voting process results in highly localized activation maps. The figure shows the heatmaps and the pointwise magnitude of the offset field on a validation image. Note that in this illustration we super-impose the channels from the different keypoints.

https://openaccess.thecvf.com/content_cvpr_2017/papers/Papandreou_Towards_Accurate_Multi-Person_CVPR_2017_paper.pdf

ROI Pooling- Faster R-CNN



Faster R-CNN

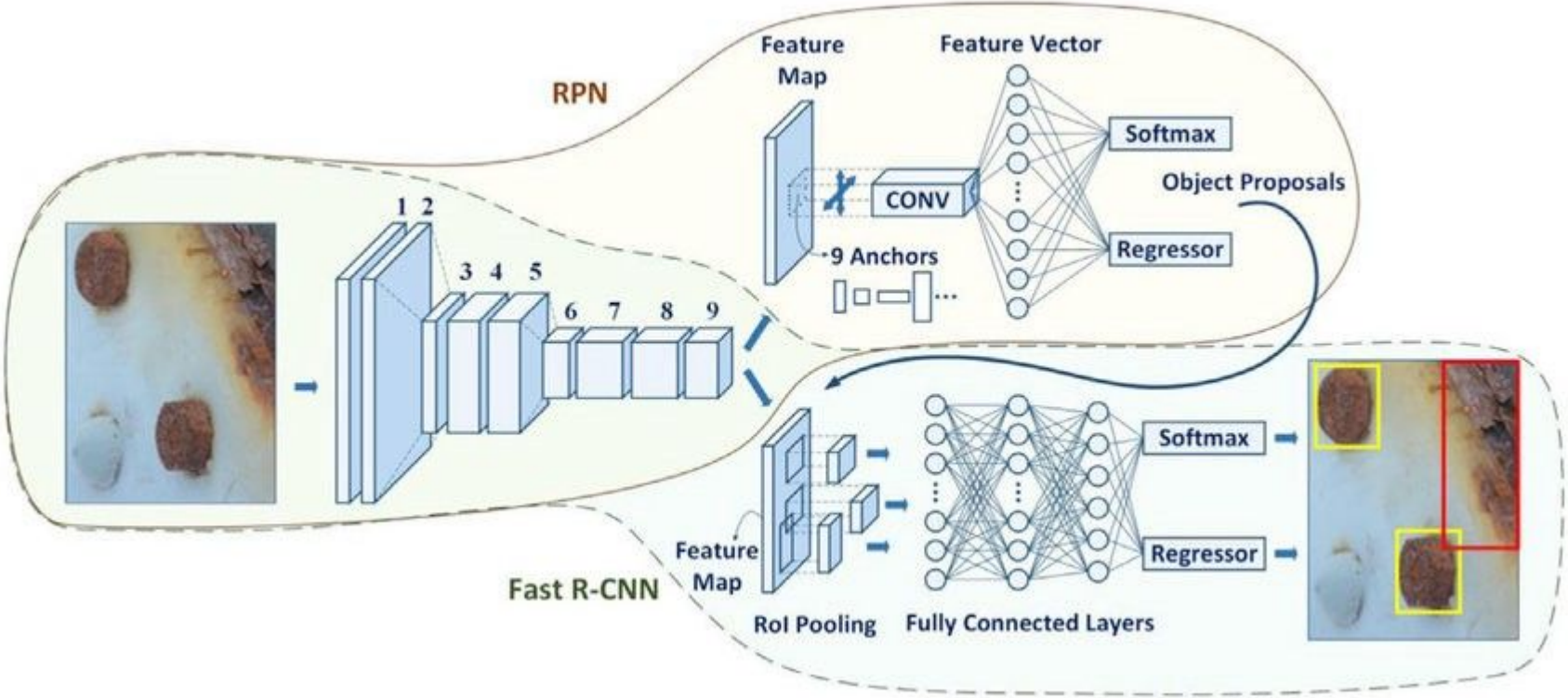

RPN
Feature Map
Feature Vector
Softmax
Object Proposals
CONV
Regressor
9 Anchors
1 2
3 4 5
6 7 8 9
Feature Map
Softmax
Regressor
Fast R-CNN
RoI Pooling
Fully Connected Layers

Image pyramids

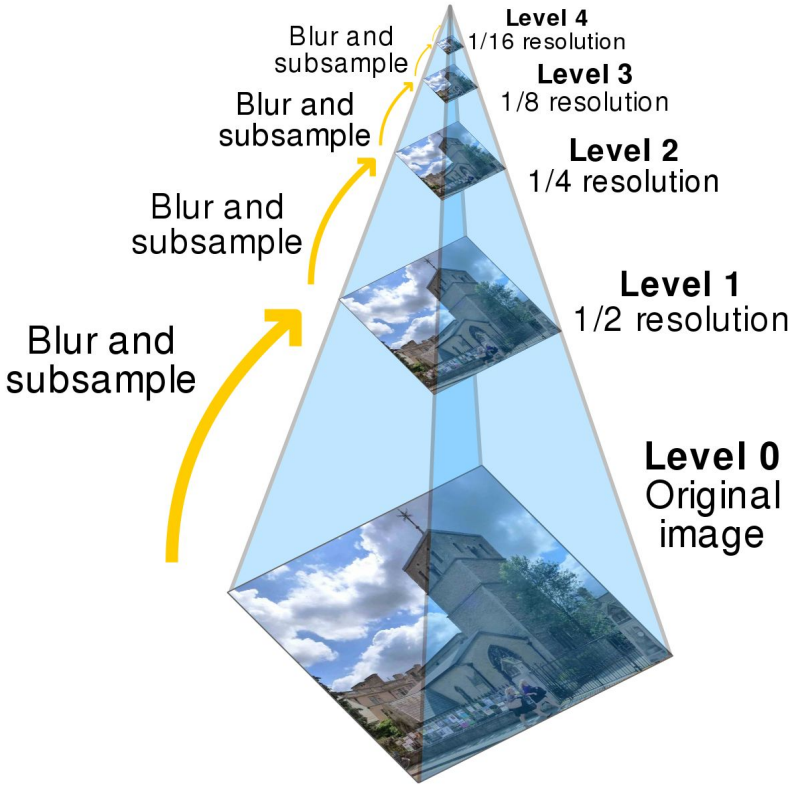


Image Pyramids in Feature Proposal Networks (FPNs)

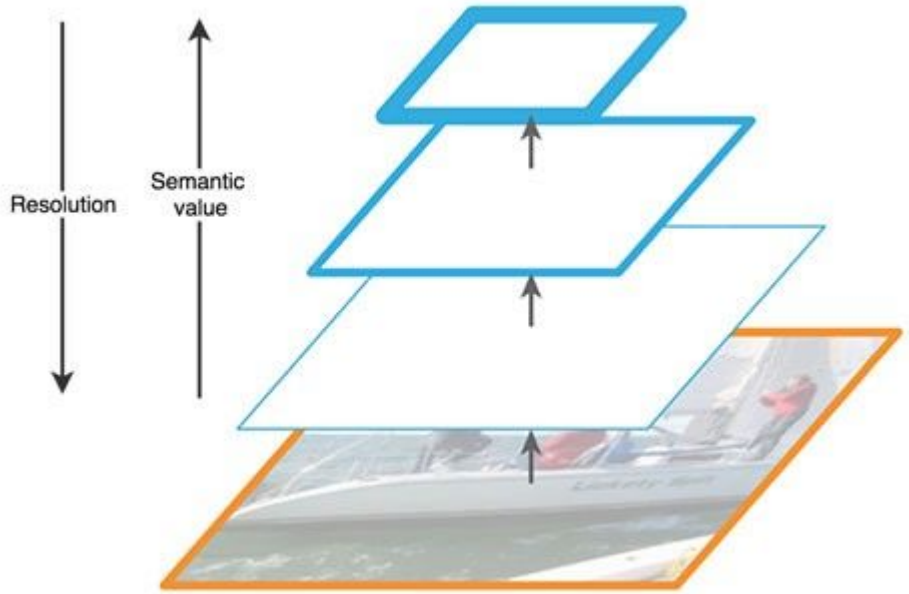
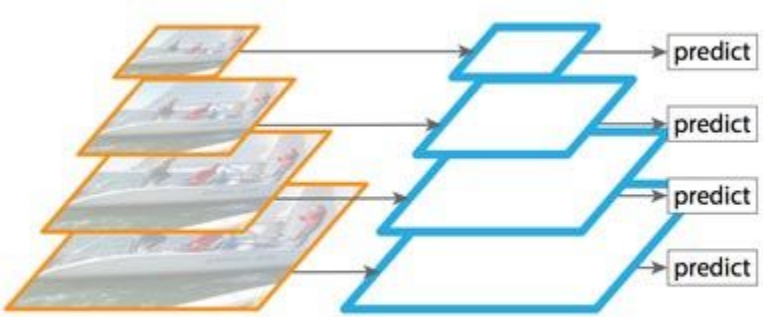
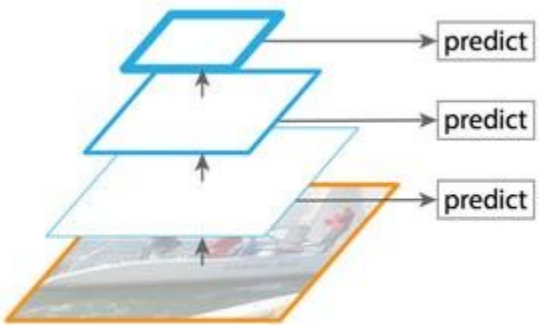


Image Pyramids in Feature Proposal Networks (FPNs)

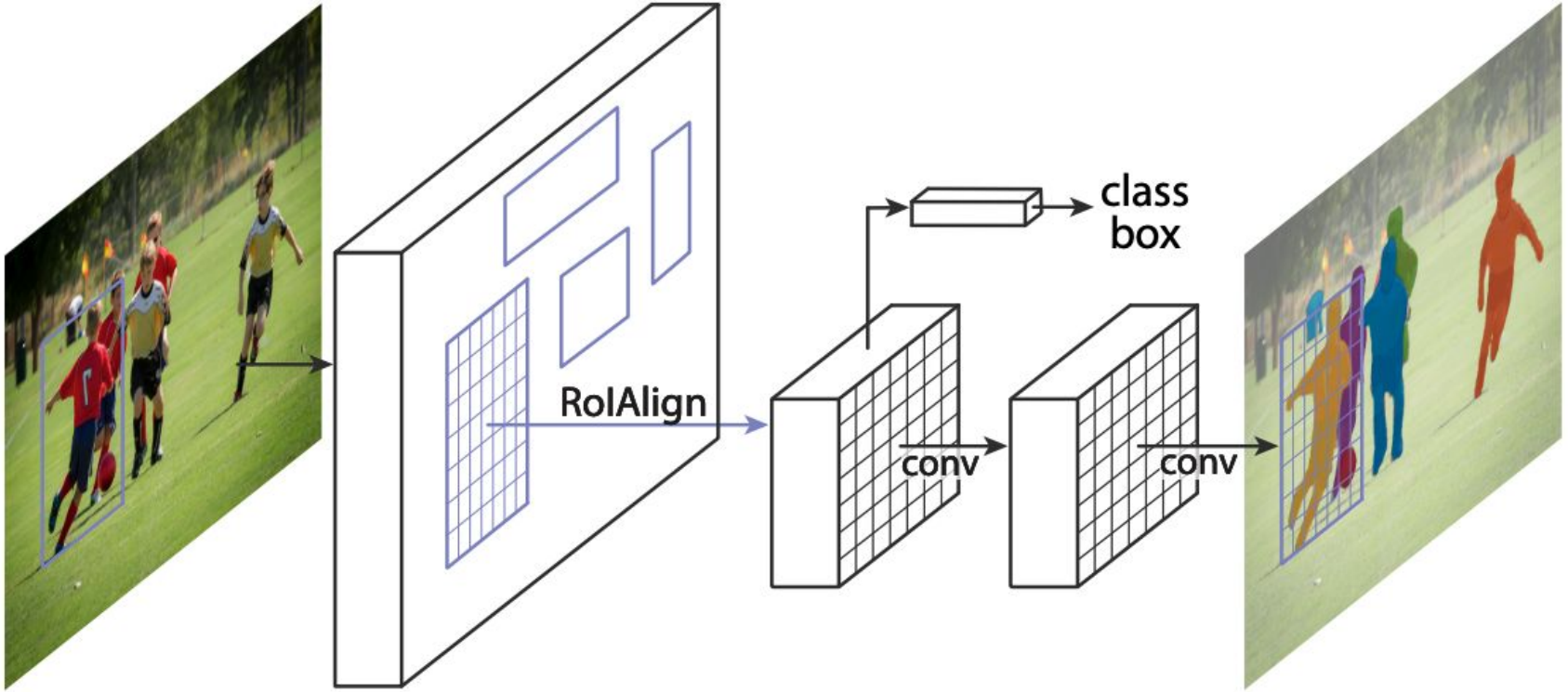


Pyramid of images



Pyramid of feature maps

Mask R-CNN



Use an API - avoid manual implementation

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.

Popular Implementations

Tensorflow’s object detection API

Matterport’s Mask-RCNN

Facebook’s Detectron (PyTorch)

The COCO dataset

http://cocodataset.org/#explore

The Google Open Images Dataset

Google Open Source

PROJECTS COMMUNITY DOCS BLOG

Open Images Dataset

g.co/dataset/open-images

A dataset of ~9 million varied images with rich annotations

The images are very diverse and often contain complex scenes with several objects (8.4 per image on average). It contains image-level labels annotations, object bounding boxes, object segmentations, visual relationships, localized narratives, and more

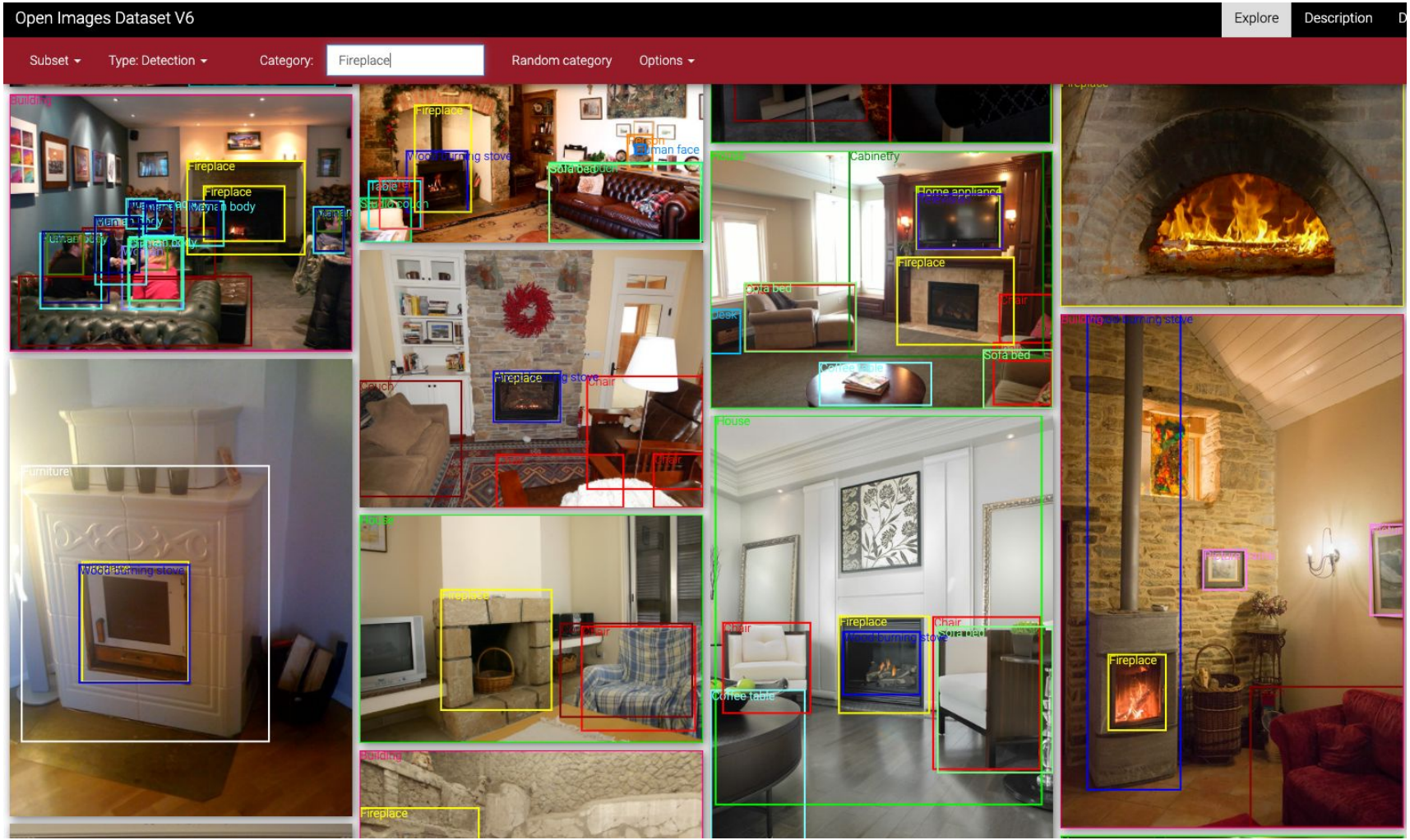
CATEGORIES

databases

samples

graphics-video-audio

machine-learning


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

Models trained on the Open Images Dataset

Open Images-trained models

Model name	Speed (ms)	Open Images mAP@0.5[^2]	Outputs
faster_rcnn_inception_resnet_v2_atrous_oidv2	727	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_oidv2	347		Boxes
facessd_mobilenet_v2_quantized_open_image_v4 [^3]	20	73 (faces)	Boxes

Model name	Speed (ms)	Open Images mAP@0.5[^4]	Outputs
faster_rcnn_inception_resnet_v2_atrous_oidv4	425	54	Boxes
ssd_mobilenetv2_oidv4	89	36	Boxes
ssd_resnet_101_fpn_oidv4	237	38	Boxes

Using pretrained models

Snagging Parking Spaces with Mask R-CNN and Python

Using Deep Learning to Solve Minor Annoyances

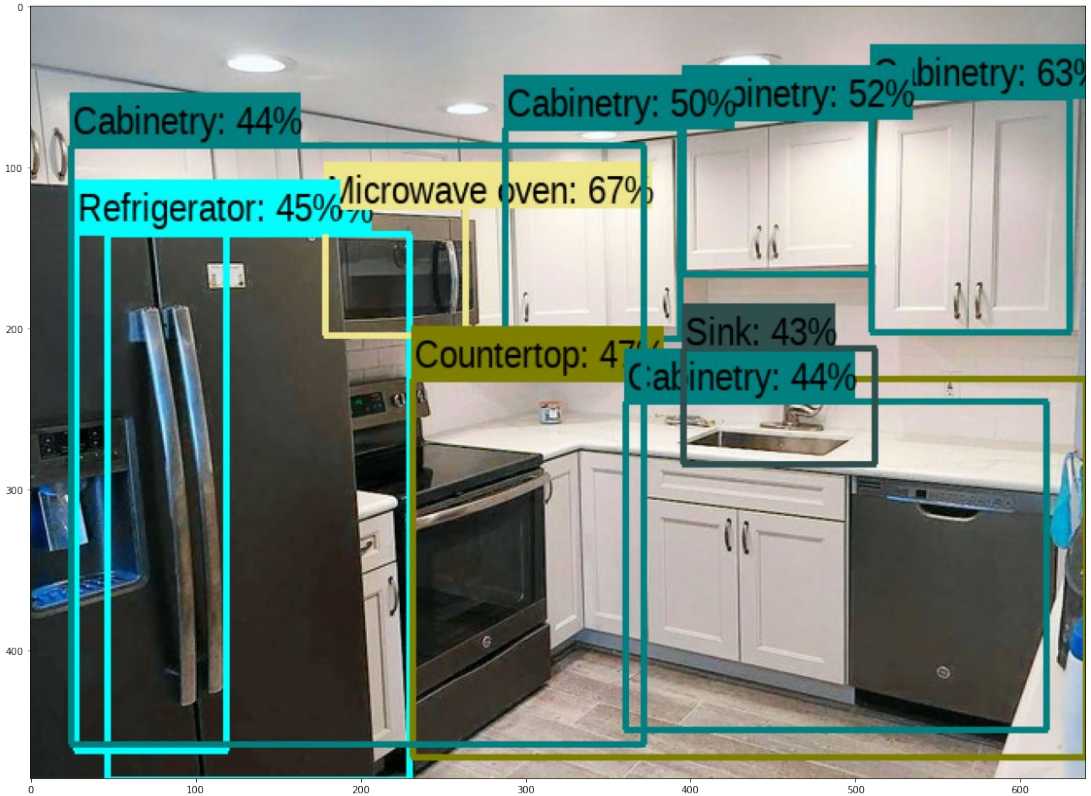
Adam Geitgey Jan 21, 2019 · 13 min read

[Using a Mask R-CNN pretrained on COCO to catch parking spaces]

Input Image

Mask R-CNN pretrained on COCO (inventory image)

SSD pretrained on Google Open Images

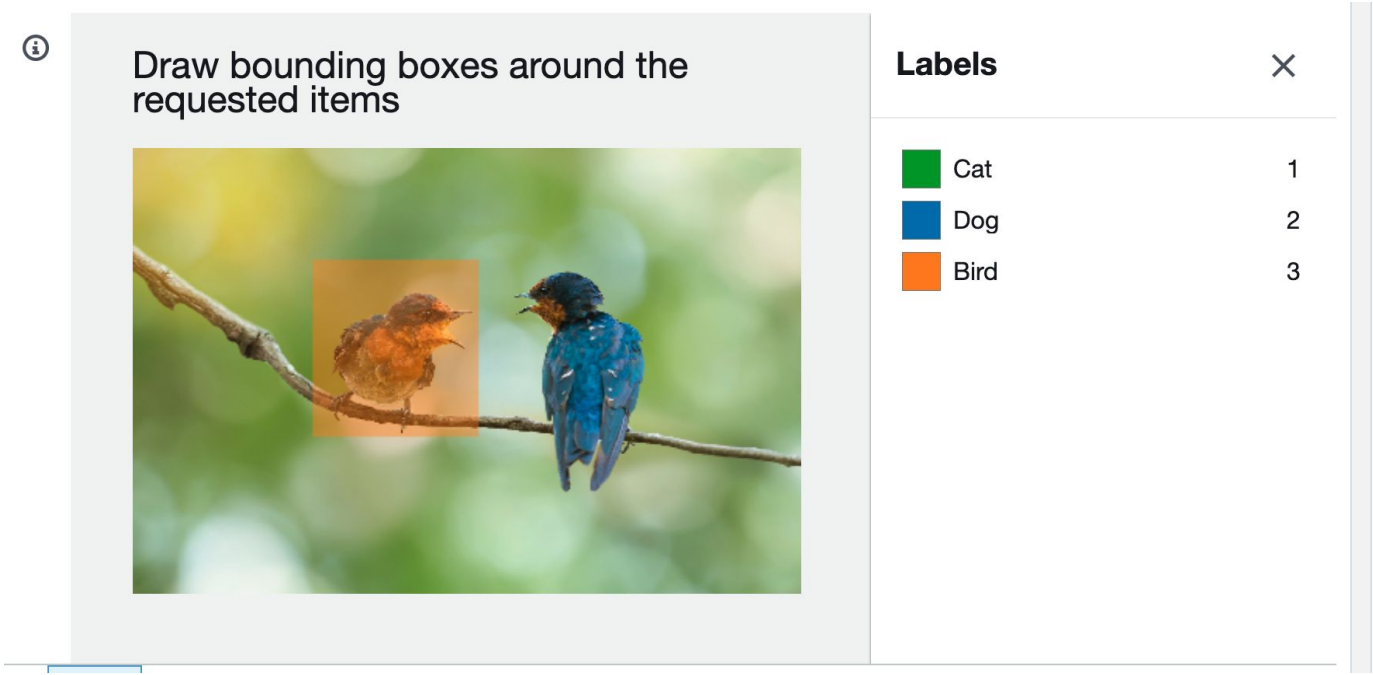


Input image

Mask R-CNN trained on COCO

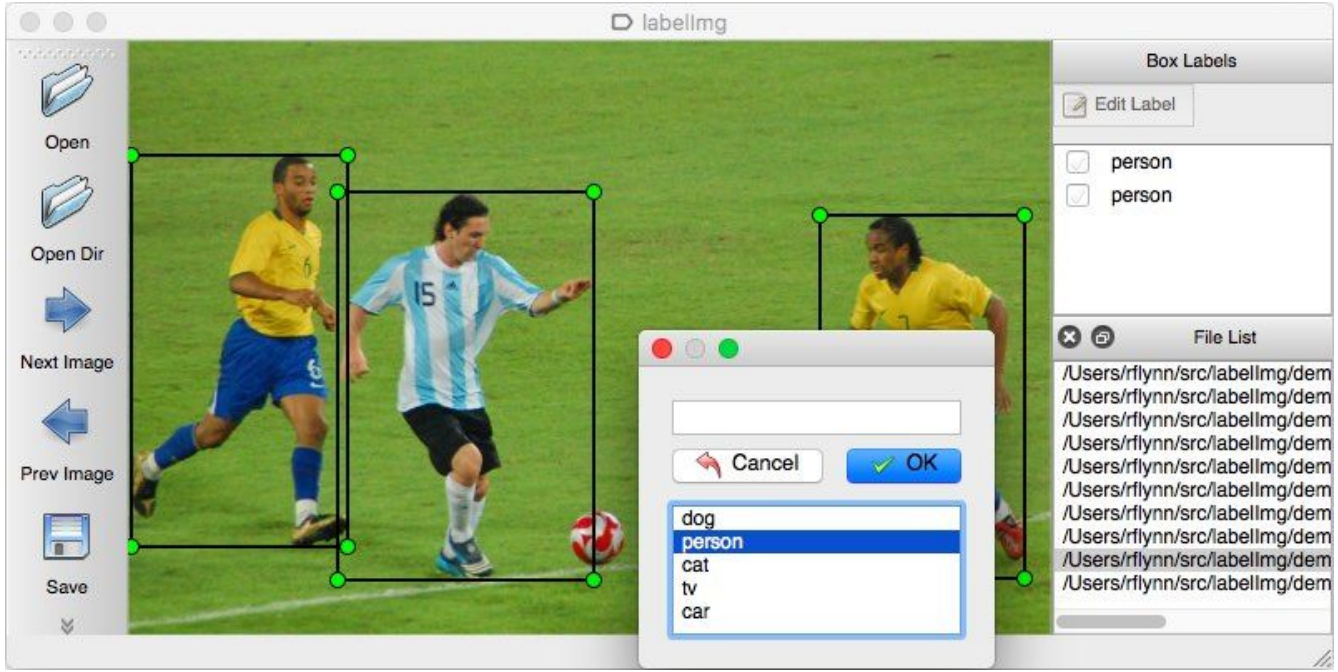
SSD trained on Open Images Dataset

Creating training and test sets



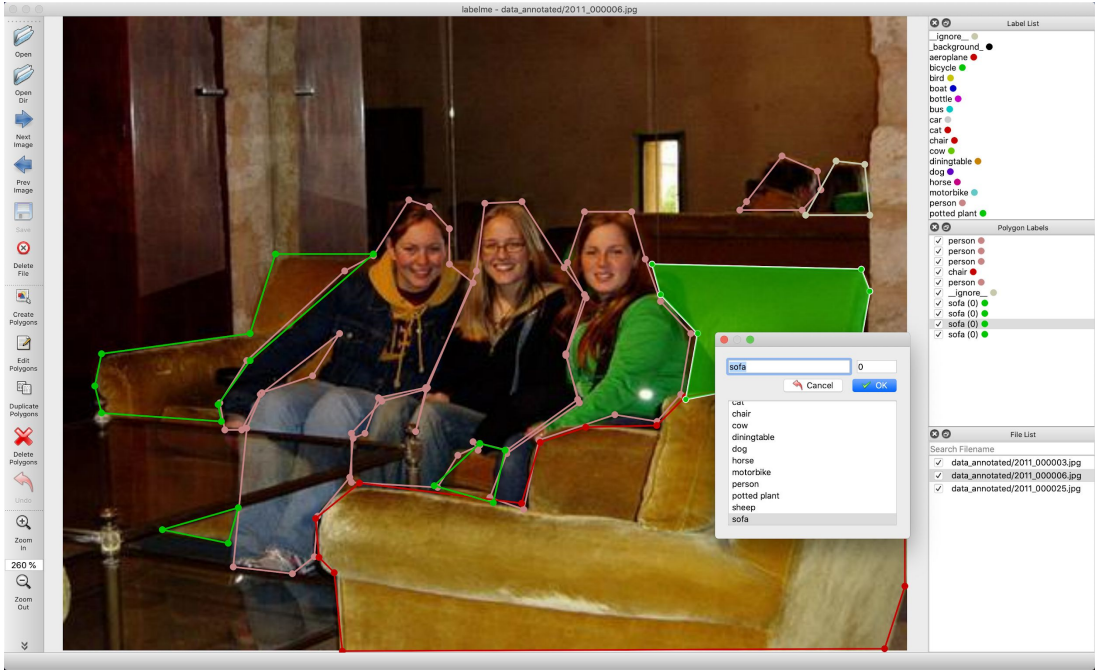
https://requester.mturk.com/

Creating training and test sets

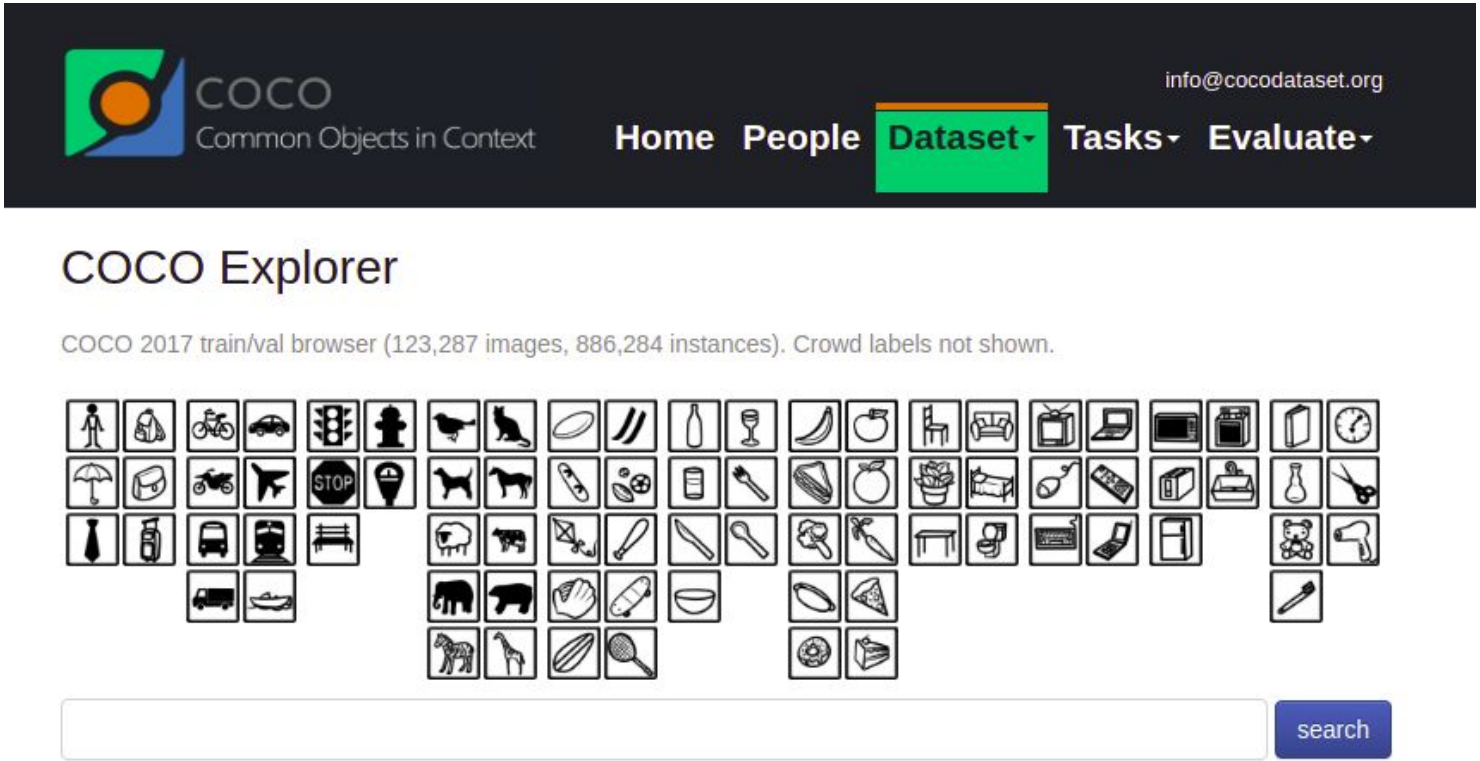


https://github.com/tzutalin/labelImg

Creating training and test sets



[labelme: Image Polygonal Annotation with Python]



https://cocodataset.org/#explore

Notebooks for Tensorflow implementations

Mask R-CNN

YOLO

RetinaNet

SSD and Faster RCNN comparison on Open Images Dataset v4

Faster R-CNN trained on Open Images Dataset v4

Using a TF-hub object detection model

Detectron2

detectron2/MODEL_ZOO.md at master · facebookresearch/detectron2 · GitHub

Recommended reading: digging into Detectron2

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

- For Faster/Mask R-CNN, we provide baselines based on **3 different backbone combinations**:
    - **FPN**: Use a ResNet+FPN backbone with standard conv and FC heads for mask and box prediction, respectively. It obtains the best speed/accuracy tradeoff, but the other two are still useful for research.
    - **C4**: Use a ResNet conv4 backbone with conv5 head. The original baseline in the Faster R-CNN paper.
    - **DC5** (Dilated-C5): Use a ResNet conv5 backbone with dilations in conv5, and standard conv and FC heads for mask and box prediction, respectively. This is used by the Deformable ConvNet paper.

https://github.com/facebookresearch/detectron2/blob/master/MODEL_ZOO.md

Model Output Format

When in training mode, the builtin models output a `dict[str->ScalarTensor]` with all the losses.

When in inference mode, the builtin models output a `list[dict]`, one dict for each image. Based on the tasks the model is doing, each dict may contain the following fields:

- "instances": Instances object with the following fields:
  - "pred_boxes": Boxes object storing N boxes, one for each detected instance.
  - "scores": `Tensor`, a vector of N confidence scores.
  - "pred_classes": `Tensor`, a vector of N labels in range [0, num_categories).
  - "pred_masks": a `Tensor` of shape (N, H, W), masks for each detected instance.
  - "pred_keypoints": a `Tensor` of shape (N, num_keypoint, 3). Each row in the last dimension is (x, y, score). Confidence scores are larger than 0.
- "sem_seg": `Tensor` of (num_categories, H, W), the semantic segmentation prediction.
- "proposals": Instances object with the following fields:
  - "proposal_boxes": Boxes object storing N boxes.
  - "objectness_logits": a torch vector of N confidence scores.
- "panoptic_seg": A tuple of `(pred: Tensor, segments_info: Optional[list[dict]])`. The `pred` tensor has shape (H, W), containing the segment id of each pixel.
  - If `segments_info` exists, each dict describes one segment id in `pred` and has the following fields:
    - "id": the segment id
    - "isthing": whether the segment is a thing or stuff
    - "category_id": the category id of this segment.

    If a pixel's id does not exist in `segments_info`, it is considered to be void label defined in Panoptic Segmentation.
  - If `segments_info` is None, all pixel values in `pred` must be ≥ -1. Pixels with value -1 are assigned void labels. Otherwise, the category id of each pixel is obtained by `category_id = pixel // metadata.label_divisor`.

https://detectron2.readthedocs.io/tutorials/models.html#model-output-format

Detectron exercises

Inference (Exercise)
Inference (Solution) - here change the input image with a change to wget

Training (Exercise) - this one is broken, sorry
Training (Solution)

Case study

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

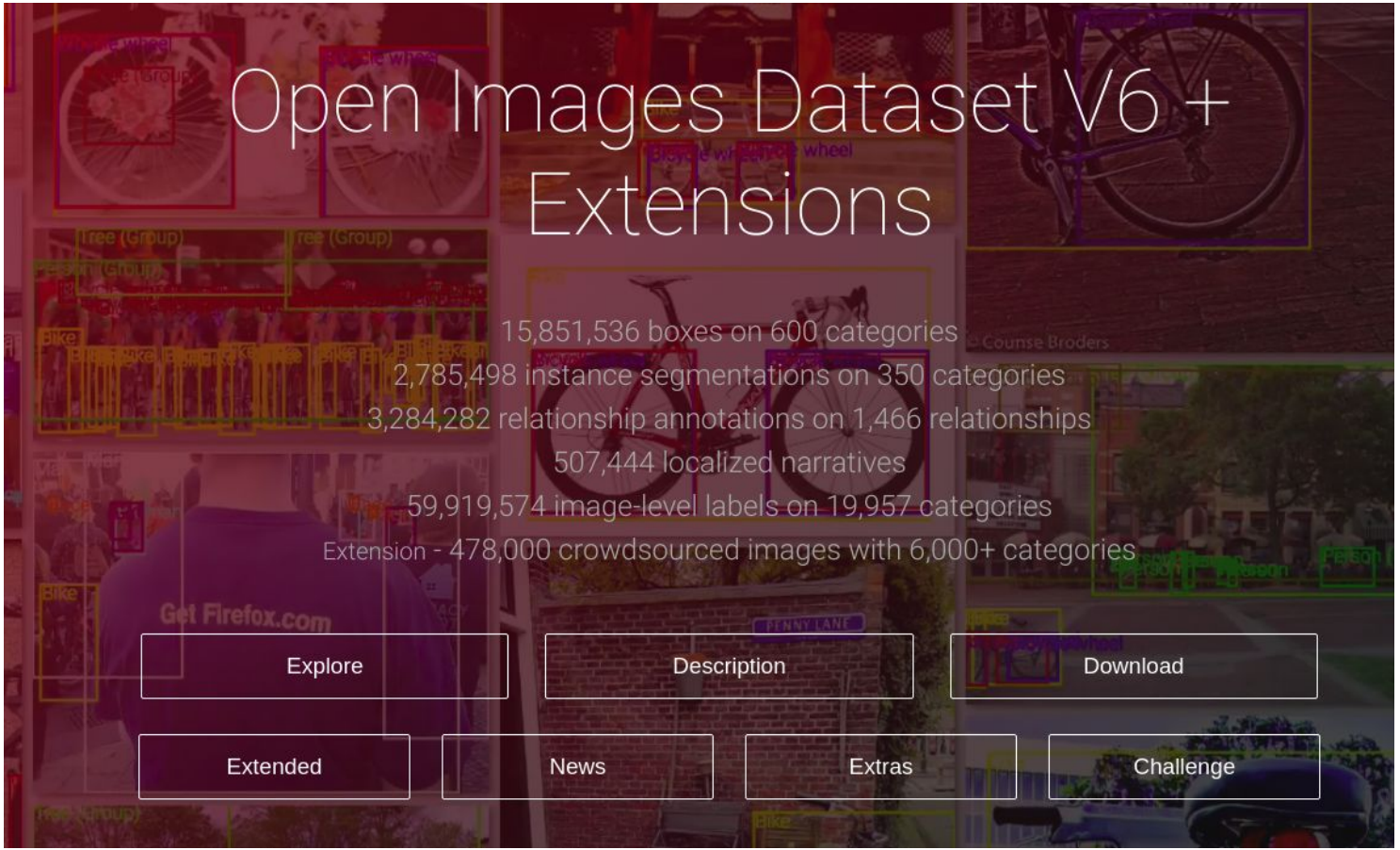
Build highly customized AI technologies into home-sharing products and help our guests belong anywhere.

Shijing Yao Follow

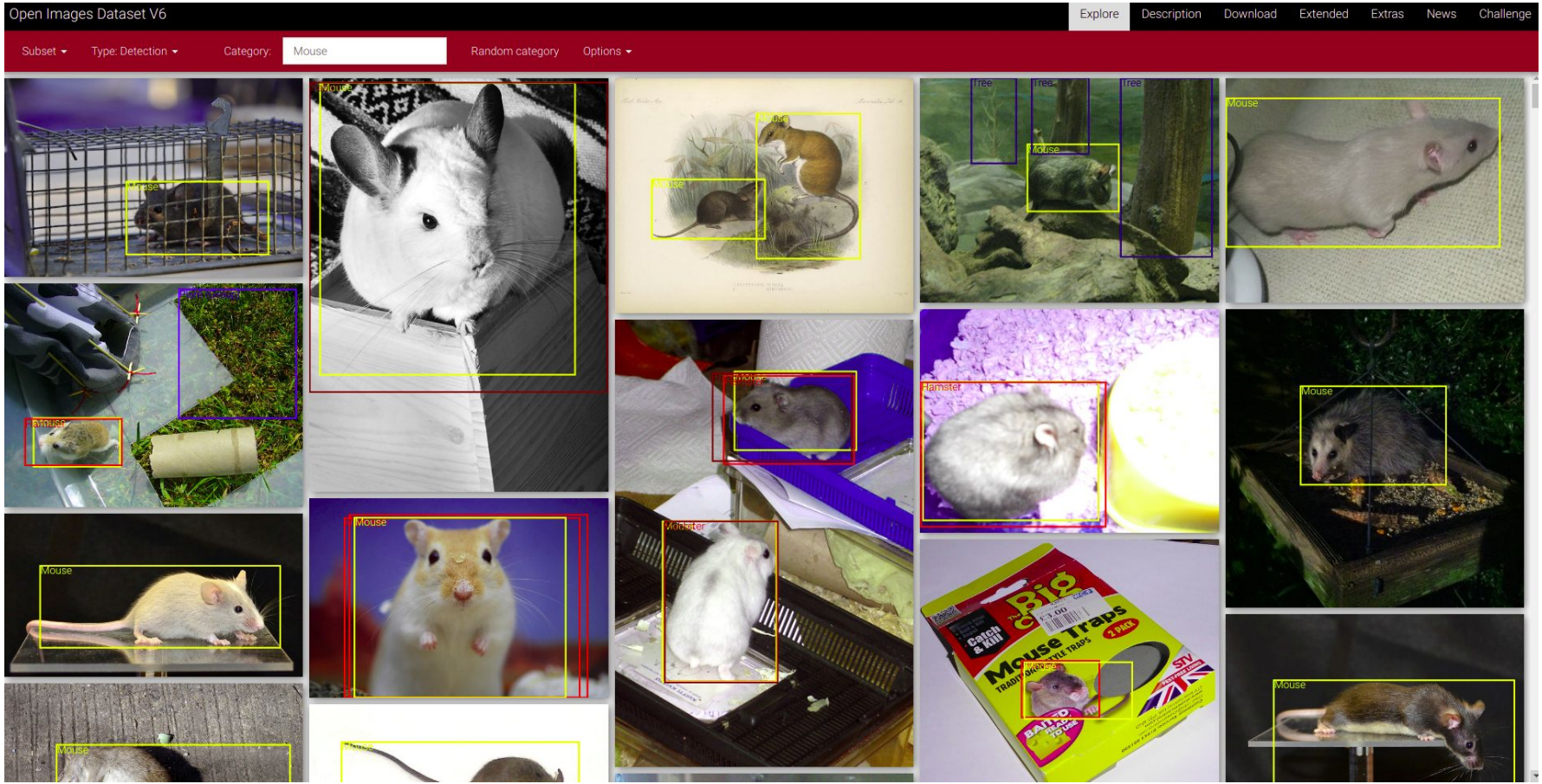
Jul 16, 2019 · 15 min read ★

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

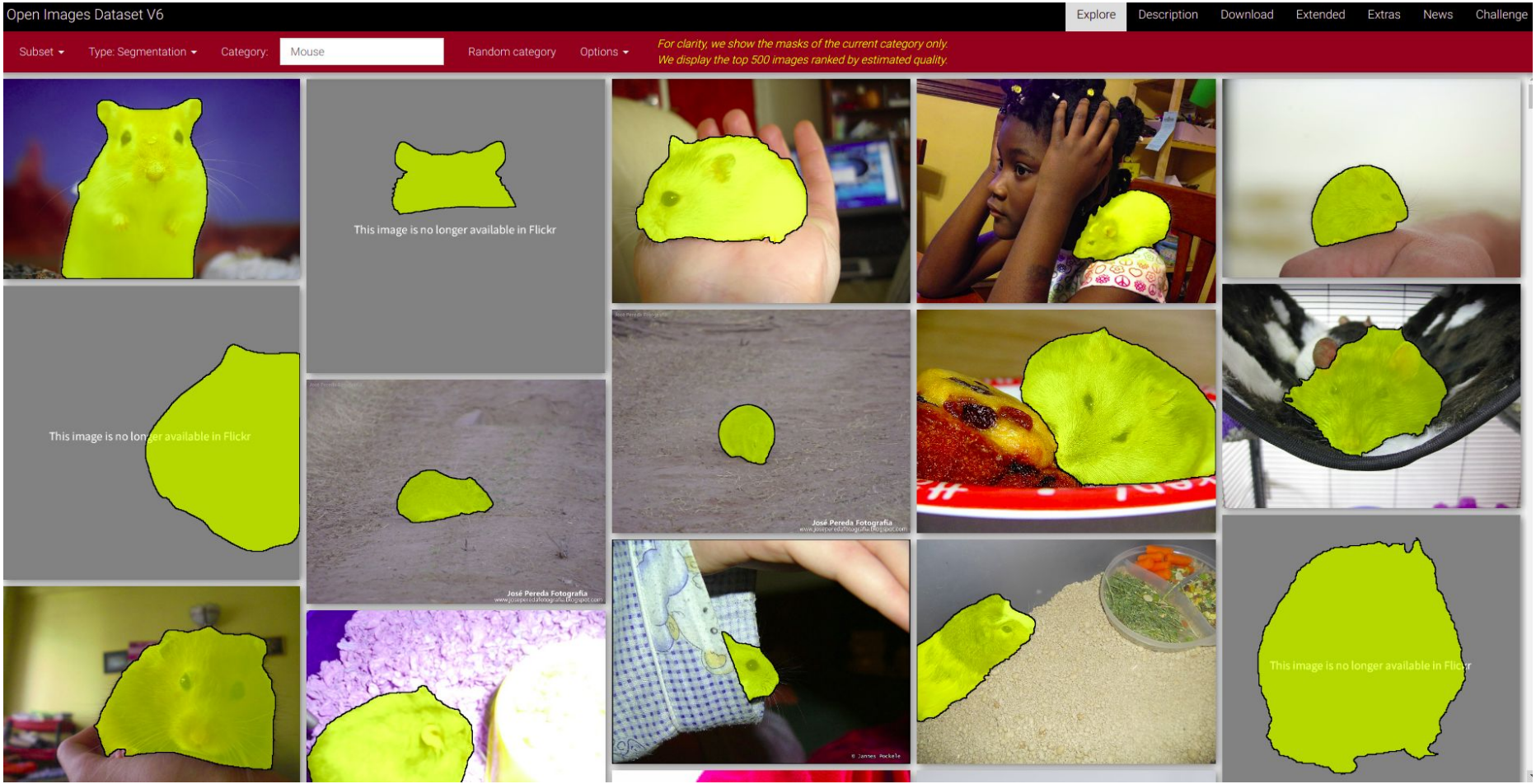

Refrigerator - 0.96
Microwave oven - 0.85
Gas stove - 0.61
Oven - 0.61
Kitchen & dining room table - 0.55



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



https://storage.googleapis.com/openimages/web/visualizer/index.html?set=train&type=detection&c=%2Fm%2F04rmv



https://storage.googleapis.com/openimages/web/visualizer/index.html?set=train&type=segmentation&r=false&c=%2Fm%2F03g8mr

Case study

Replicating Airbnb's Amenity Detection with Detectron2

Ingredients: 1 x Detectron2, 38,188 x Open Images, 1 x GPU. Model training time: 18-hours. Human time: 127(ish)-hours.

Daniel Bourke Apr 27 · 24 min read ★

Replicating Airbnb's Amenity Detection with Detectron2

File Edit View Insert Runtime Tools Help Changes will not be saved

Share

+ Code + Text Copy to Drive Connect Editing

Replicating Airbnb's Amenity Detection with Detectron2

This notebook goes through a small tutorial on how you can replicate Airbnb's article: Amenity Detection and Beyond — New Frontiers of Computer Vision at Airbnb.

In other words, using computer vision to discover amenities in pictures of rooms.

Colab notebook

Review questions

- What is a common issue with segmenting small images with a U-net?
- What is the point of using FastAI? What is it? What does it offer (other than a *ton* of bugs)?
- What is perceptual hashing? How does it differ from cryptographic hashing?
- How do we compute the Hamming distance for two hashes?
- What problems are we solving when using embeddings from a pretrained network? Is this transfer learning?
- Which inputs do we need to train an image segmentation model from scratch?
- Which inputs do we need to train an object detector from scratch?
- How is the ‘error rate’ metric for classification defined? Should we use it for imbalanced datasets?
- How do we compute the confusion matrix for a segmentation mask? How do we compute it for a bounding box?

Review questions

- What’s the recall of a classifier that only outputs ‘1’ (positive class)?
- What’s an “hypercolumn”?
- Does the F2 measure weigh precision and recall equally?
- How do deep learning object detection methods reduce the search space of detection regions? How does object detection exploit transfer learning of image classification in Imagenet?

Review questions

- What's the appeal of using Detectron2? Do we need to write a Pytorch model to use it for inference or training with Detectron?
- How does panoptic segmentation combine instance and semantic segmentation? Which method produces the 'stuff'? Which method produces the 'things'?
- Suppose that you wish to detect objects in video and you only have a modest embedded processor, would you solve this problem using YOLO or using Faster RCNN? How would you argue in favor of using either method?
- Is semantic segmentation more computationally costly than instance segmentation? Why?

References

- Stanford’s cs231n lecture on Object Detection and Segmentation
- PyImageSearch tutorial on Mask R-CNN
- Training Faster R-CNN on the Google Open Images Dataset
- Focal loss for Single Shot Detectors
- Training an object detector on Google Colab