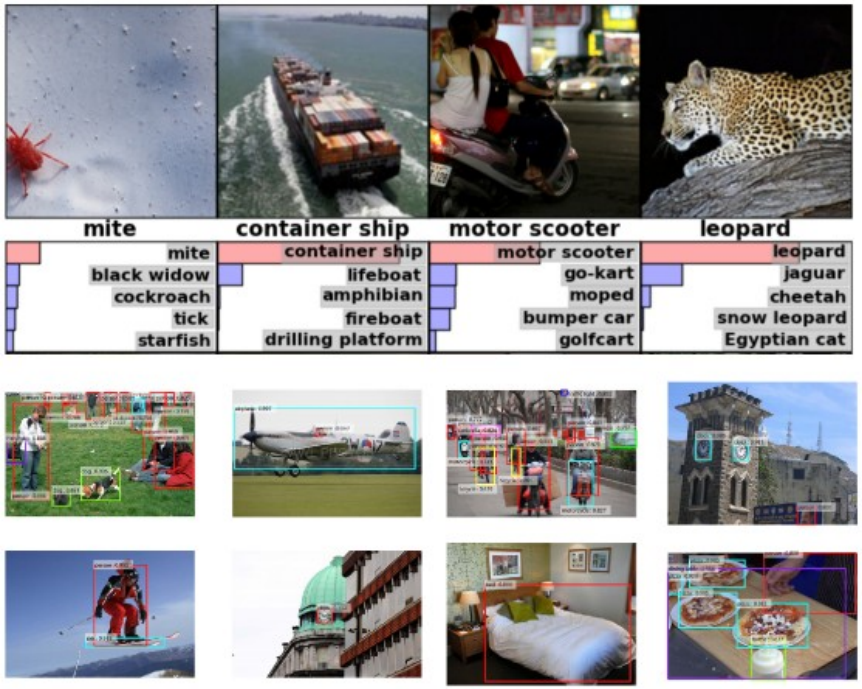
Convolutional Neural Networks
Applications to Object Detection

Vahid Tarokh

ECE685D, Fall 2025

Classification versus detection

- Very related, but different problems.
- Many aspects will be shared between the two problems
- In detection algorithms, we try to draw a bounding box around the object of interest to locate it within the image.
- There could be many bounding boxes representing different objects of interest within the image and you would not know how many beforehand.

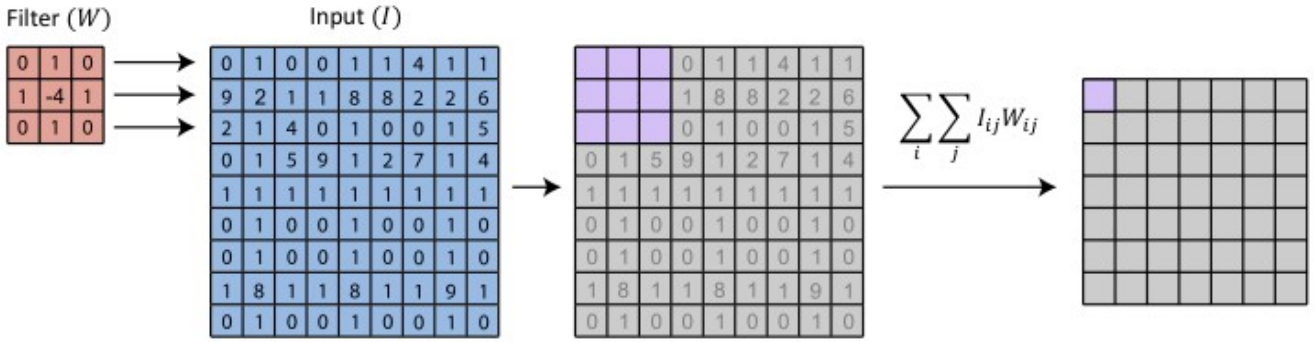


Pieces of a Deep Algorithm

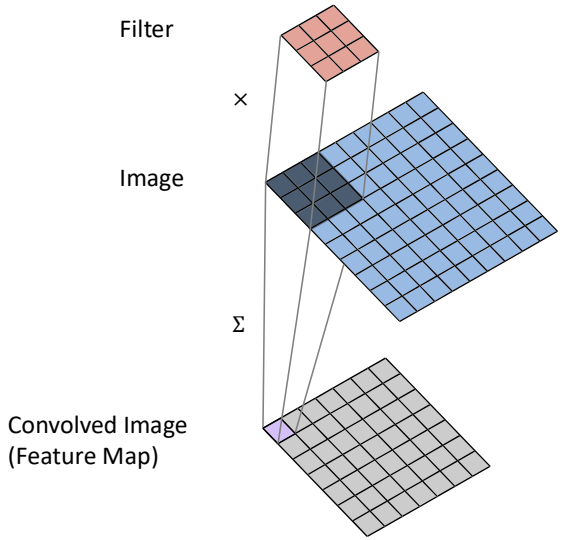
Object detection is very related to image classification. We use similar types of algorithms and related data types.

	Image Classification	Object Detection
Model	CNN Feature Extractor + Softmax Classifier	SSD, Faster R-CNN
Data	Imagenet, CIFAR-10, e.g.	COCO, PASCAL VOC
Scalar loss function (Objective)	CE loss + Regularization	SSD Loss, Faster R-CNN Loss, (both very complicated)
Optimization Algorithm and Hyperparameters	Stochastic Gradient Descent (SGD), Adam, Momentum, etc.	

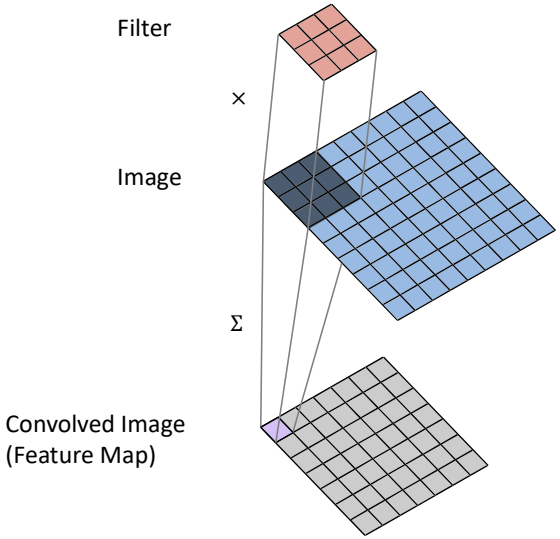
Review: 2D Convolution

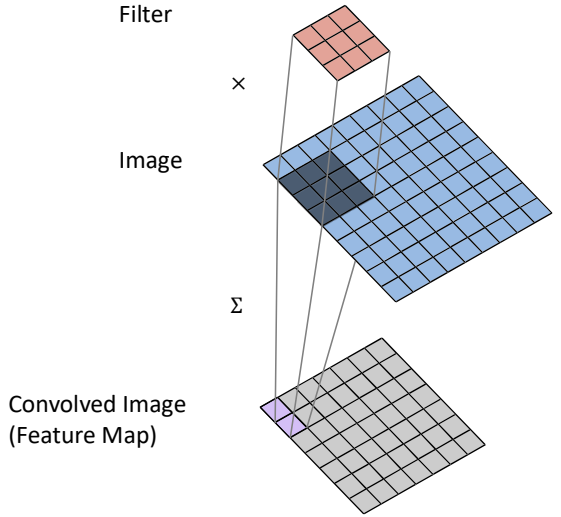


Review: 2D Convolution



Review: 2D Convolution



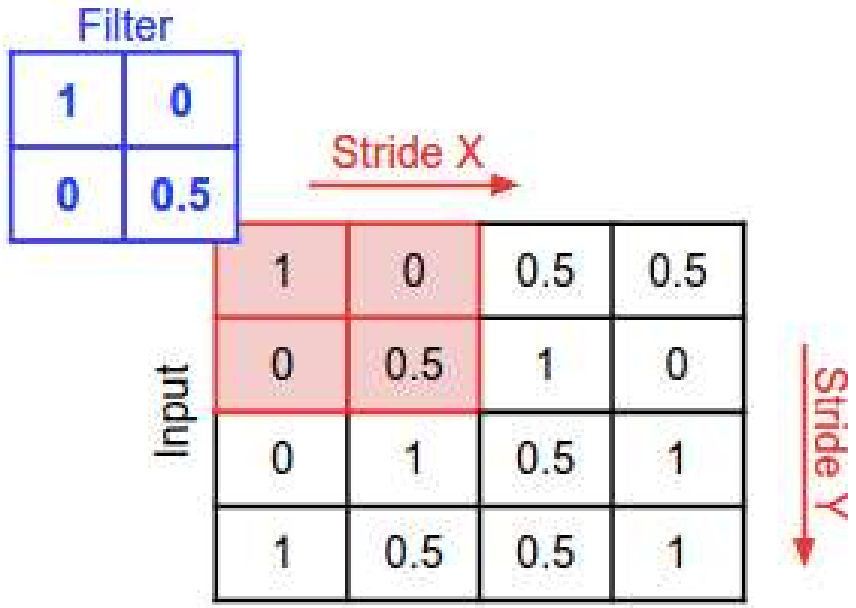


CNN Convolution Parameters

CNN — Parameters

- **Filters**: Represents the amount of filters in a CL.
- **Kernel Size**: Defines the dimensions of the filters.
- **Stride**: Sets the size of the filter shift step.
- **Padding**: defines whether or not there is entry zeroing, influencing the output dimensions:

CNN Convolution Parameters



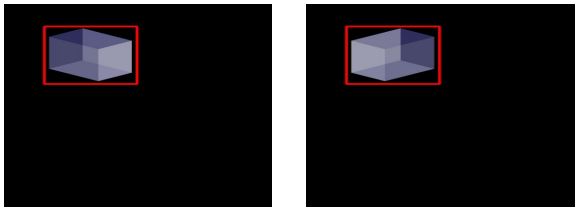
Review of Useful Properties

Recall that we have some important properties that are useful in object detection

- **Translation Invariance**

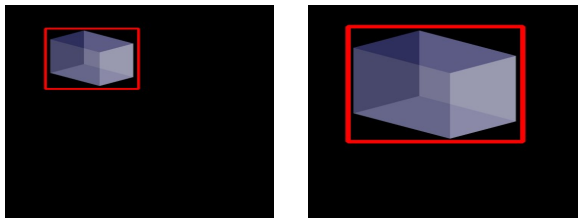
This is addressed with convolutional features.

- **Rotation Invariance**



We handle this by data augmentation and other techniques in the dataset.

- **Scale Invariance**



We handle by data diversity and rescaling feature maps.

Some are built into the algorithm, and some come from the structure of the dataset

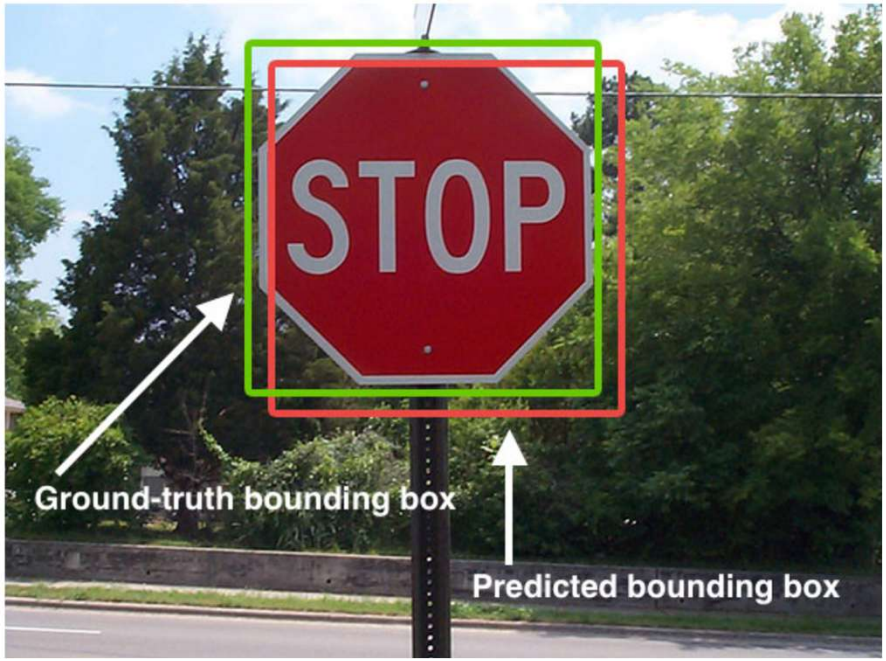
We need a metric to evaluate our model

Classification – did we choose the correct class?

Object detection – is our object correct?

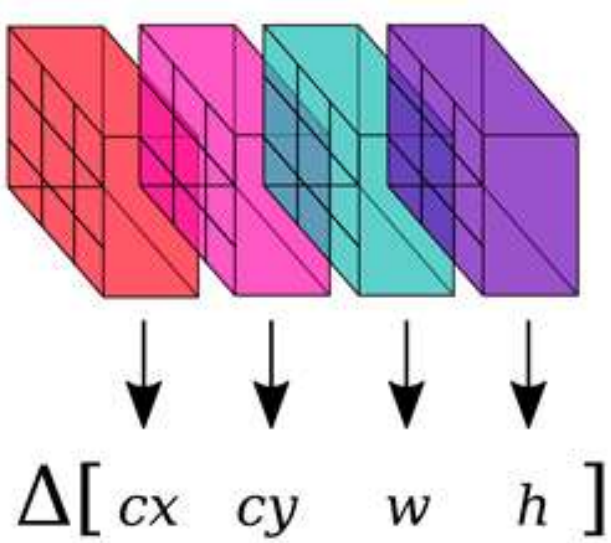
How do we determine if we got the correct location?

We want the bounding box to be good enough.



Detection and Regression

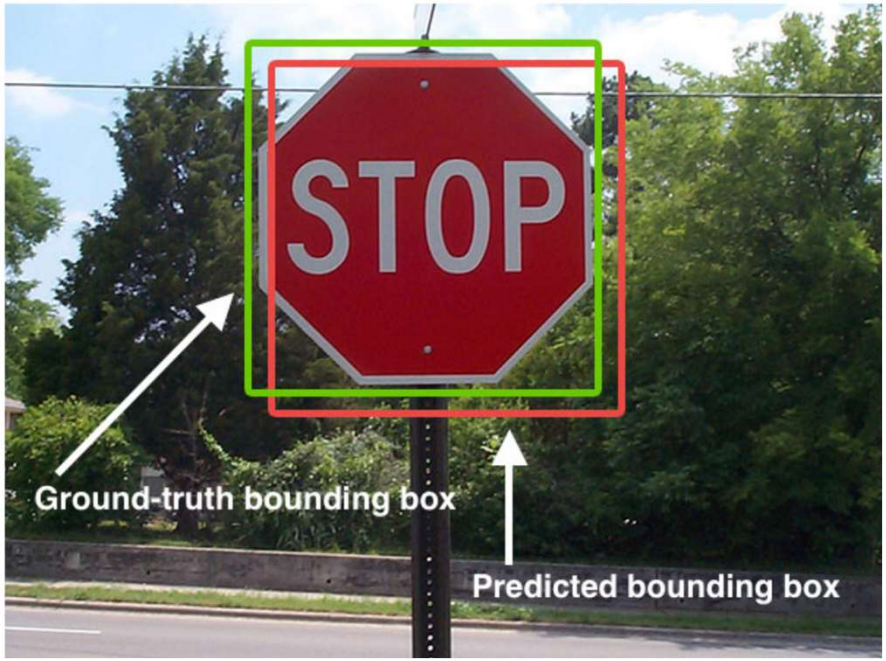
- Finding the bounding box can be thought of as a "**regression problem**".
- Given the image, we must find the center coordinates, width, and height of the box.
- This means we need to output a 4-dimensional vector corresponding to the box for each object of interest.
- We need to understand what is in the box too.
- Need a measure of how good the box is.



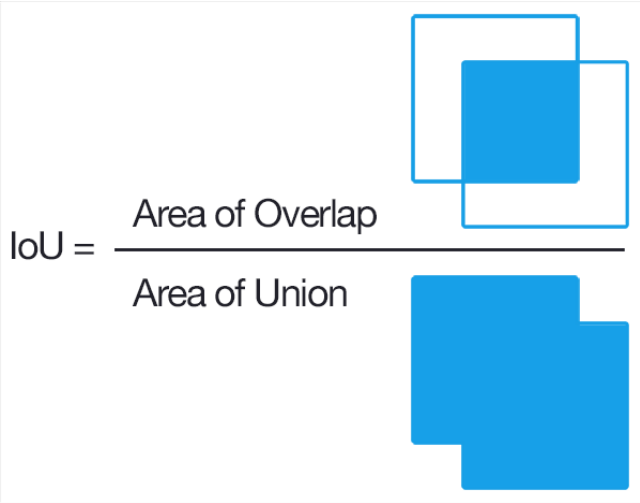
Intersection-over-Union

- The **intersection-over-union** metric is one way of defining this.
- Compute the **Intersection-the** common area covered by the ground-truth bounding box and the predicted bounding box.
- Compute the **Union-the** total area covered by either the ground-truth bounding box and predicted bounding box.

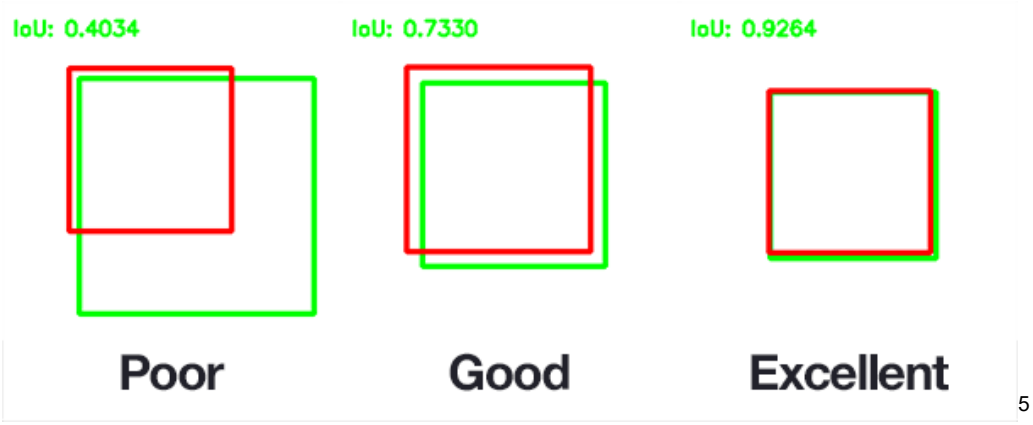
$$\text{IoU} = \frac{\text{Intersection}}{\text{Union}}$$



Intersection-over-Union

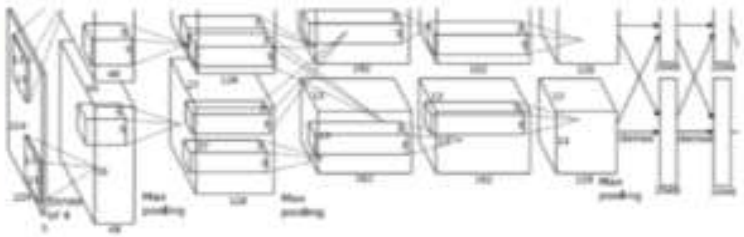


Intersection-over-Union



From Wikipedia

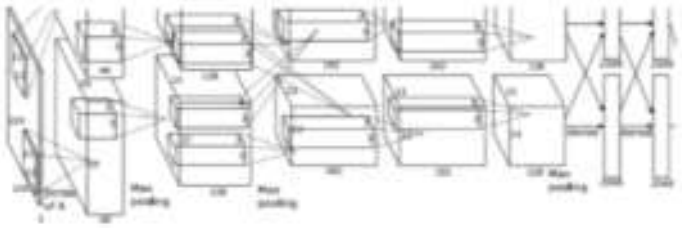
The idea



CAT: (x, y, w, h)

- Images are fed through a convolutional feature extractor.
- Bounding box regression and classification occur in one stage relative to a set of anchor boxes.
- This is a new step that we must figure out.

The idea



DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
....

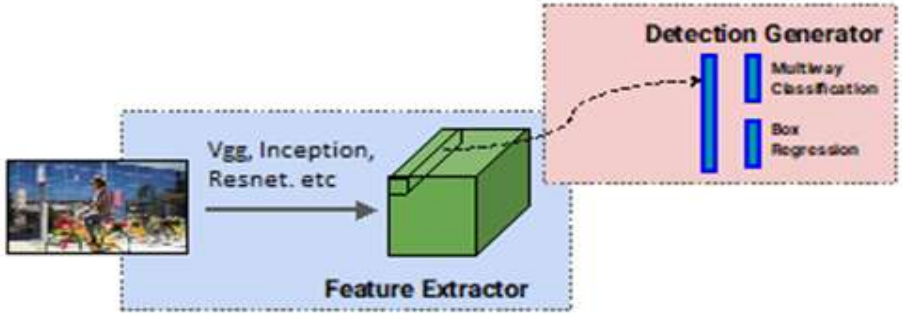
We might not necessarily draw just one bounding box in an object detection case, there could be many bounding boxes representing different objects of interest within the image and **we would not know how many beforehand**.

The Major Barrier to Overcome

- The major reason why you cannot proceed with this problem by building a standard convolutional network followed by a fully connected layer is that
- The length of the output layer is variable — not constant, this is because the number of occurrences of the objects of interest is not fixed.
- A naive approach to solve this problem would be to take different regions of interest from an image and use a CNN to classify the presence of the object within that region.
- The problem with this approach is that the objects of interest might have different spatial locations within the image and different aspect ratios. Hence, we would have to select a huge number of regions and this could computationally blow up.
- Therefore, algorithms like R-CNN, YOLO etc., have been developed to find these occurrences and find them fast.

Object Detection Algorithm

- Images are fed through a convolutional feature extractor.
- Bounding box regression and classification occur in one stage relative to a set of anchor boxes.
- This is a new step that we have to figure out.



FROM CONVOLUTIONAL FEATURES TO OBJECT DETECTION

R-CNN

- To bypass the problem of selecting a huge number of regions, Girshick proposed a method where a selective search is used to extract just 2000 regions from the image (These regions are called ***region proposals***).

- Therefore, now, instead of trying to classify a huge number of regions, you can just work with 2000 regions.

- These 2000 region proposals are generated using the **selective search algorithm**:

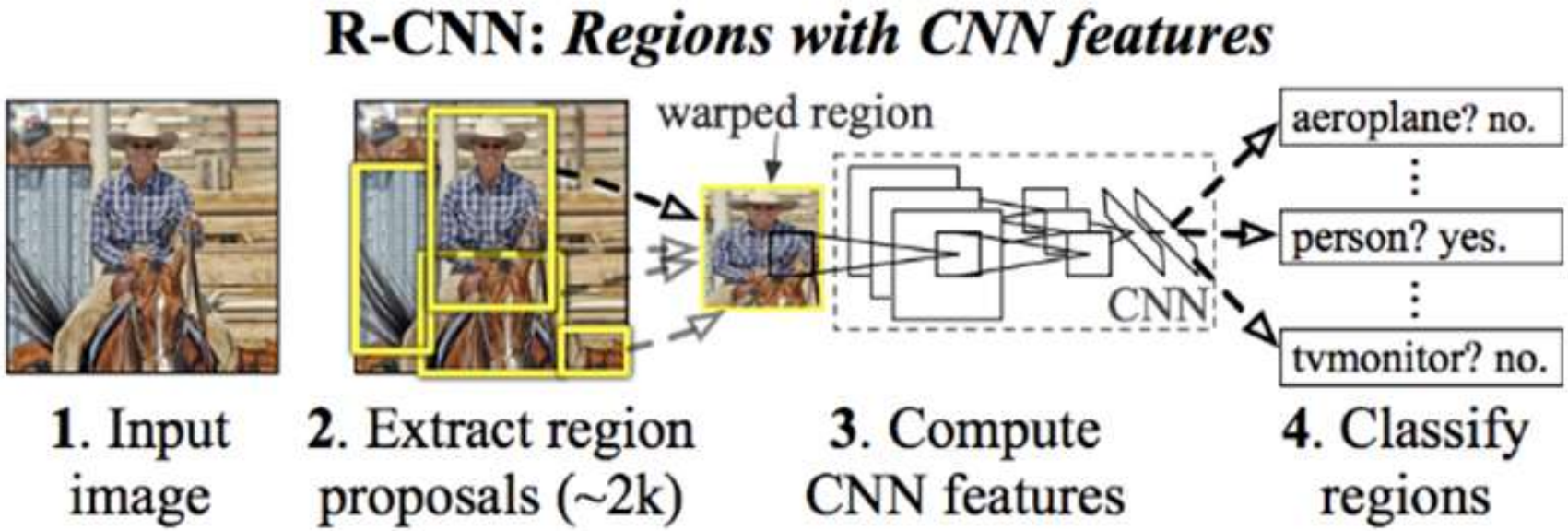
- Selective Search (High Level Description):
  1. Generate initial sub-segmentation, we generate many candidate regions
  2. Use greedy algorithm to recursively combine similar regions into larger ones
  3. Use the generated regions to produce the final candidate region proposals

Details are given in
https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013/UijlingsIJCV2013.pdf

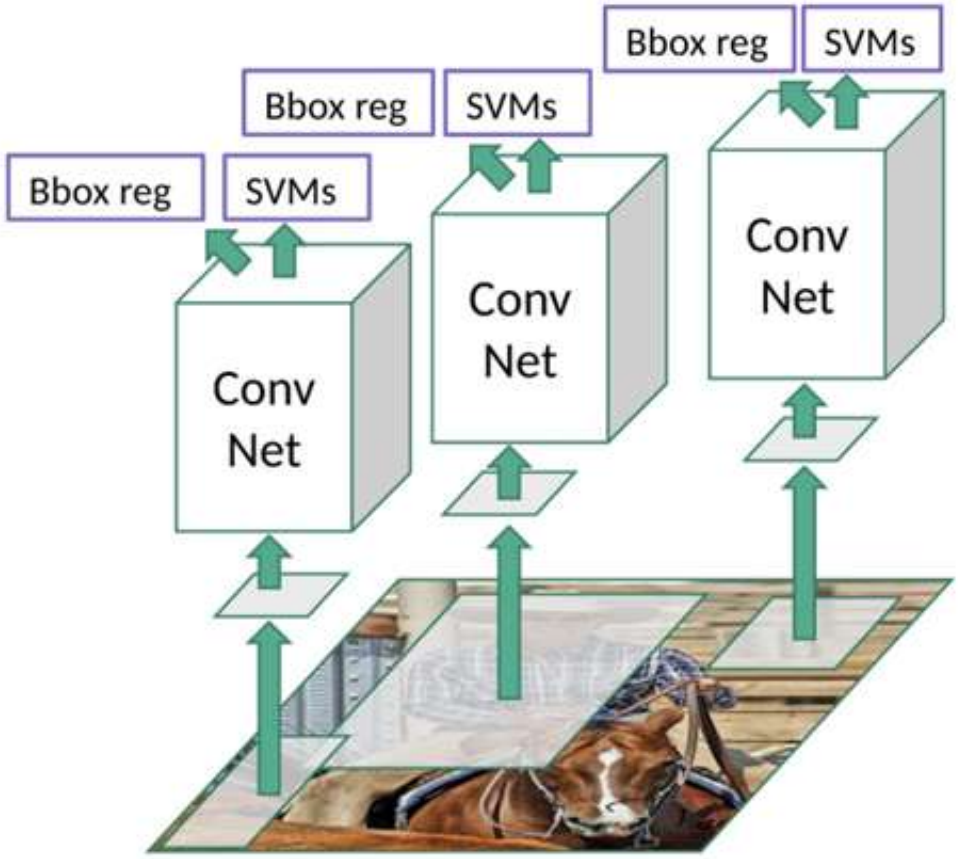
R-CNN

- These 2000 candidate region proposals are warped into a square and fed into a convolutional neural network that produces a 4096-dimensional feature vector as output.
- The CNN acts as a feature extractor and the output dense layer consists of the features extracted from the image and the extracted features are fed into a support vector machine (SVM) to classify the presence of the object within that candidate region proposal.
- In addition to predicting the presence of an object within the region proposals, the algorithm also predicts **four** values which are offset values to increase the precision of the bounding box.
- For example, given a region proposal, the algorithm would have predicted the presence of a person but the face of that person within that region proposal could've been cut in half.
- Therefore, the offset values help in adjusting the bounding box of the region proposal.

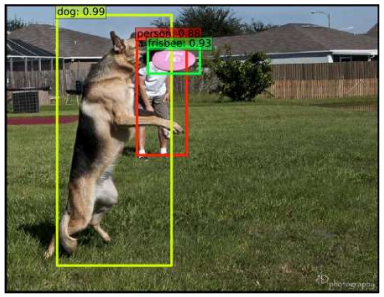
R-CNN

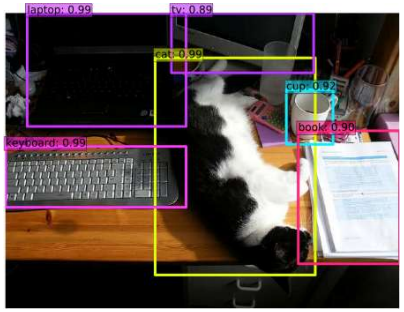

R-CNN: Regions with CNN features
warped region
CNN
aeroplane? no.
person? yes.
tvmonitor? no.
1. Input image
2. Extract region proposals (~2k)
3. Compute CNN features
4. Classify regions

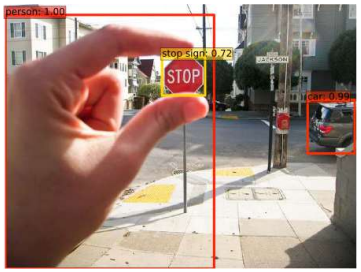
R-CNN

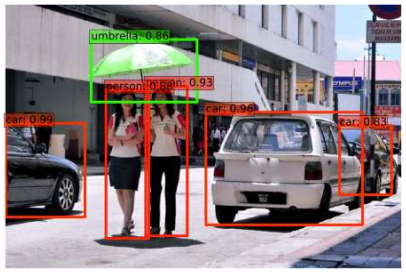

Bbox reg
SVMs
Bbox reg
SVMs
Bbox reg
SVMs
Conv Net
Conv Net
Conv Net

Sample Detections









R-CNN

- It still takes a huge amount of time to train the network as you would have to classify 2000 region proposals per image.

- It cannot be implemented real time as it takes around 47 seconds for each test image.

- The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.

FAST R-CNN

- The approach is similar to the R-CNN algorithm.
- Instead of feeding the region proposals to the CNN, we feed the input image to the CNN to generate a convolutional feature map.
- From the convolutional feature map, we identify the region of proposals and warp them into squares and by using a Region of Interest (RoI) pooling layer we reshape them into a fixed size so that it can be fed into a fully connected layer.

VGG16: Very Deep Convolutional Networks for Large-Scale Image Classification"

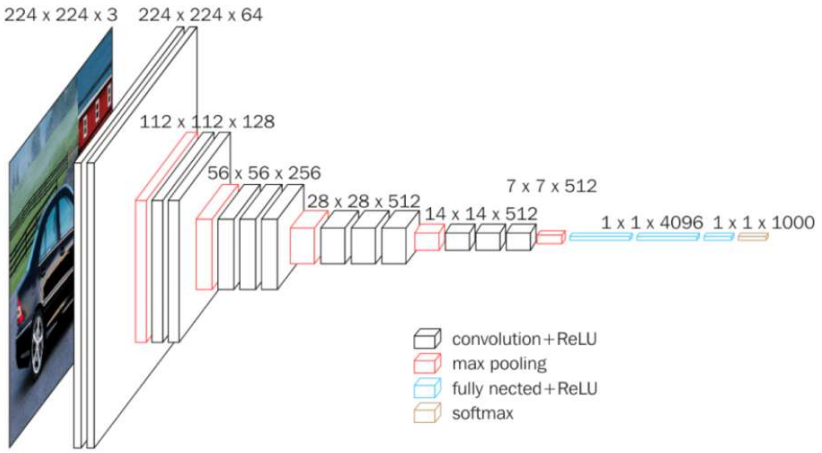
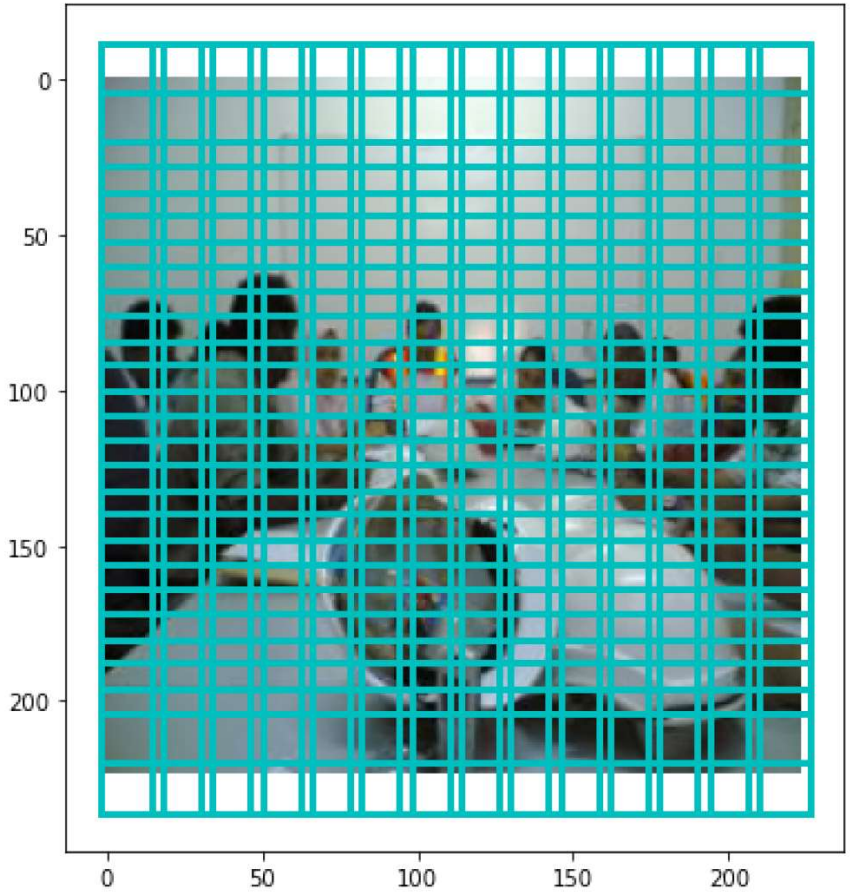


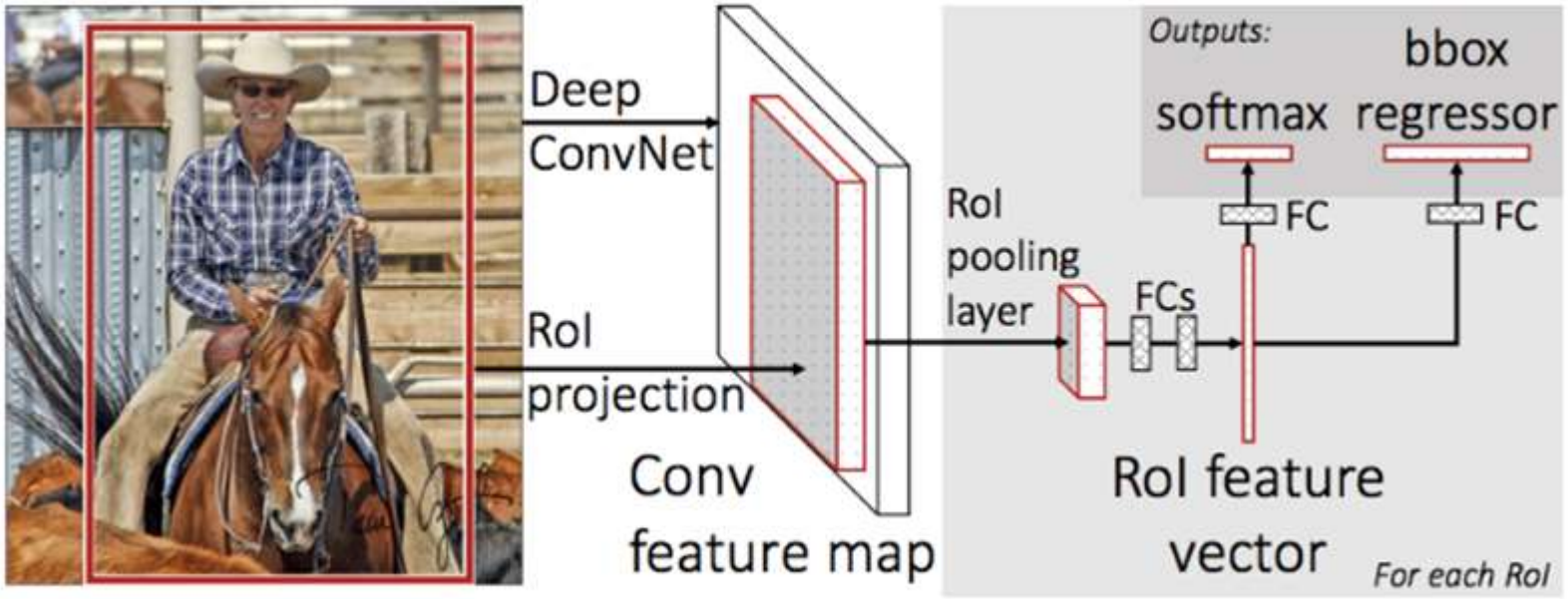
Image copyright Simonyan & Zisserman, 2015

- Input normalized to 224×224 pixels, 3 color channels.
- Last convolutional layer is 14×14 pixels, 512 channels. Call this $\vec{f}[m, n]$, where $\vec{f} \in \Re^{512}$, $0 \leq (m, n) \leq 13$.
- Output FCN trained for object recognition: 1000 different object types.

Last convolutional layer contains 196 features



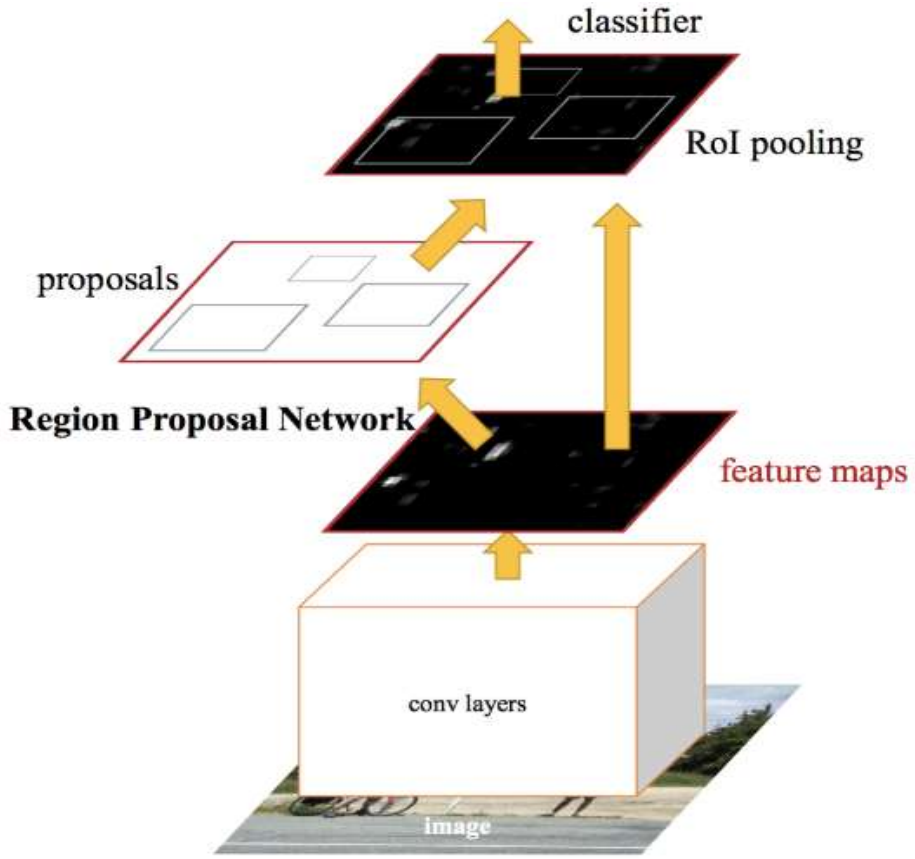
Fast R-CNN


Deep
ConvNet
RoI
projection
Conv
feature map
RoI
pooling
layer
FCs
Outputs:
softmax
bbox
regressor
FC
FC
RoI feature
vector
For each RoI

FASTER R-CNN

- Both R-CNN & Fast R-CNN use selective search to find out the region proposals.
- Selective search is a slow and time-consuming process affecting the performance of the network.
- Ren et al, came up with an object detection algorithm that eliminates the selective search algorithm and lets the network learn the region proposals.

Faster R-CNN


classifier
RoI pooling
proposals
Region Proposal Network
feature maps
conv layers
image

FASTER R-CNN

- Similar to Fast R-CNN, the image is provided as an input to a convolutional network which provides a convolutional feature map.

- Instead of using selective search algorithm on the feature map to identify the region proposals, a separate network is used to predict the region proposals.

- The predicted region proposals are then reshaped using a RoI pooling layer.

- This is then used to classify the image within the proposed region and predict the offset values for the bounding boxes.

VGG16: Very Deep Convolutional Networks for Large-Scale Image Classification"

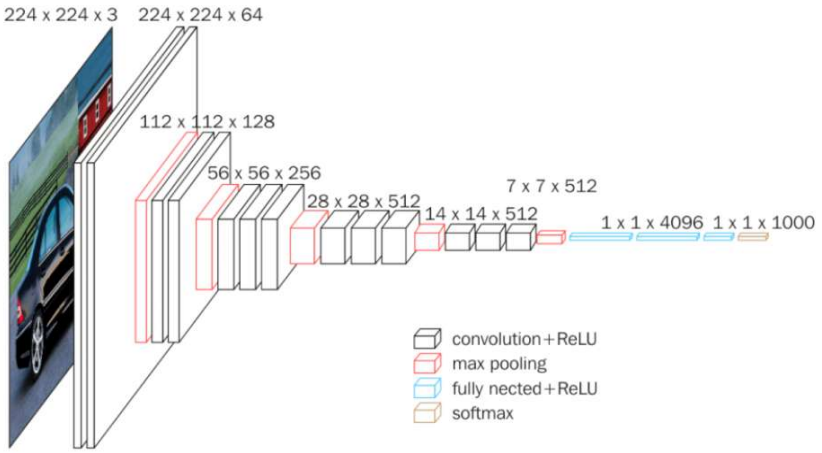


Image copyright Simonyan & Zisserman, 2015

- Input normalized to 224×224 pixels, 3 color channels.
- Last convolutional layer is 14×14 pixels, 512 channels. Call this $\vec{f}[m, n]$, where $\vec{f} \in \Re^{512}$, $0 \leq (m, n) \leq 13$.
- Output FCN trained for object recognition: 1000 different object types.

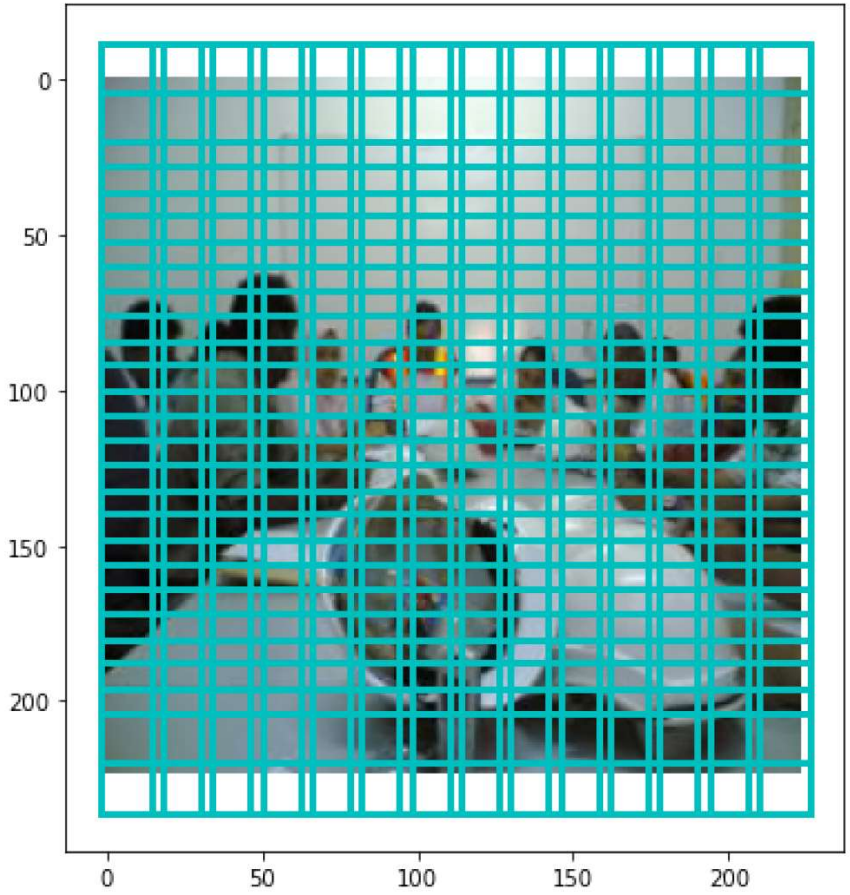
- Faster RCNN assumes that the original image is 1064×1064 pixels, which is then downsampled to the 224×224-pixel size required as input to VGG16.
- There are 4 layers of max pooling before the last conv layer, so each feature vector in the last conv layer represents

$$\left(2^4 \left(\frac{1064}{224}\right)\right) \times \left(2^4 \left(\frac{1064}{224}\right)\right) = 76 \times 76 \ \frac{\text{input pixels}}{\text{feature vector}}.$$

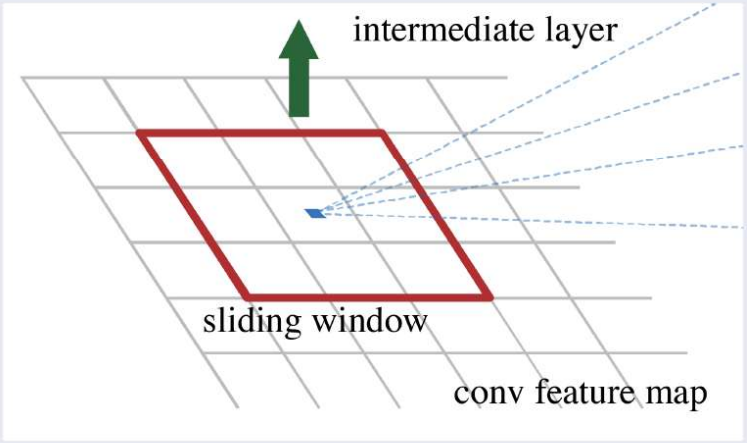
- The last conv layer contains

$$\left(\frac{224}{2^4}\right) \times \left(\frac{224}{2^4}\right) = 14 \times 14 = 196 \text{ feature vectors.}$$

Last convolutional layer contains 196 features



ROI $= 3 \times 3$ grid of VGG16 feature vectors



The region proposal network takes, as input, the concatenation of nine neighboring feature vectors from the VGG16 layer:

$$\vec{x}_{m,n} = \begin{bmatrix} \vec{f}[m-1, n-1] \\ \vec{f}[m-1, n] \\ \vdots \\ \vec{f}[m+1, n+1] \end{bmatrix}$$

Notice, we could think of this as another convolutional layer, but Ren et al. treat it as $14 \times 14 = 196$ different FCNs.

Features and Original Image

The $(m, n)^{\text{th}}$ feature vector, $\vec{f}_{m,n}$, covers a particular block of pixels in the input image:

$$(x_{ROI}, y_{ROI}, w_{ROI}, h_{ROI}) = (76n, 76m, 228, 228)$$

- Each $\vec{x}[m, n]$ covers 76×76 input pixels.
- Each $\vec{f}_{m,n}$ is $(3 \cdot 76) \times (3 \cdot 76) = 228 \times 228$.
- $m \rightarrow y$ is the vertical axis, $n \rightarrow x$ horizontal.

Suppose the nearest true object is in rectangle $(x_{REF}, y_{REF}, w_{REF}, h_{REF})$. We want to somehow encode the difference between where we are now $(x_{ROI}, y_{ROI}, w_{ROI}, h_{ROI})$ and where we want to be $(x_{REF}, y_{REF}, w_{REF}, h_{REF})$. Fast RCNN does this using the following target vector, $\vec{y}_r$, for the neural network:

$$\vec{y}_r = \begin{bmatrix} \frac{x_{REF} - x_{ROI}}{w_{ROI}} \\ \frac{y_{REF} - y_{ROI}}{h_{ROI}} \\ \ln\left(\frac{w_{REF}}{w_{ROI}}\right) \\ \ln\left(\frac{h_{REF}}{h_{ROI}}\right) \end{bmatrix}$$

The neural net is trained to find a $\hat{y}_r$ that is as close as possible to $\vec{y}_r$ (minimum MSE).

Training a bounding box regression network

The network is now trained with two different outputs, $\hat{y}_c$ and $\hat{y}_r$. The total loss is

$$\mathcal{L} = \mathcal{L}_c + \mathcal{L}_r$$

where $\mathcal{L}_c$ is BCE for the classifier output:

$$\mathcal{L}_c = -\frac{1}{n}\sum_{i=1}^{n}\left(y_{c,i}\ln\hat{y}_{c,i} + (1-y_{c,i})\ln(1-\hat{y}_{c,i})\right)$$

and $\mathcal{L}_r$ is zero if $y_c = 0$ (no object present), and MSE if $y_c = 1$:

$$\mathcal{L}_r = \frac{1}{2n}\sum_{i=1}^{n} y_{c,i}\|\vec{y}_{r,i} - \hat{y}_{r,i}\|^2$$

Summary of Approach

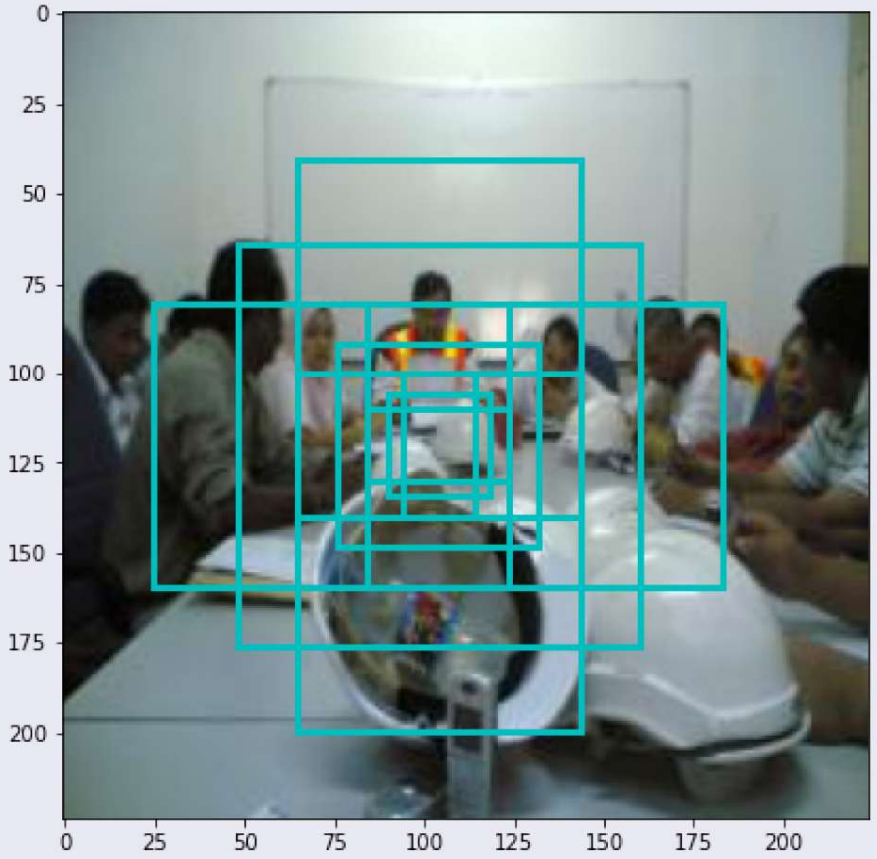
- An ROI network has a 4608d input, corresponding to a 3×3 grid of 512d feature vectors from the last conv layer of a VGG16 object recognizer.
- Faster-RCNN defines 9 different anchors centered on each ROI.
- W.r.t. each anchor, we define the classification target $y_c = 1$ if $IOU > 0.7$, otherwise $y_c = 0$.
- If $y_c = 1$, then we define a regression target $\vec{y}_r$, specifying how much the REF bbox differs from the anchor.

3 sizes, 3 aspect ratios

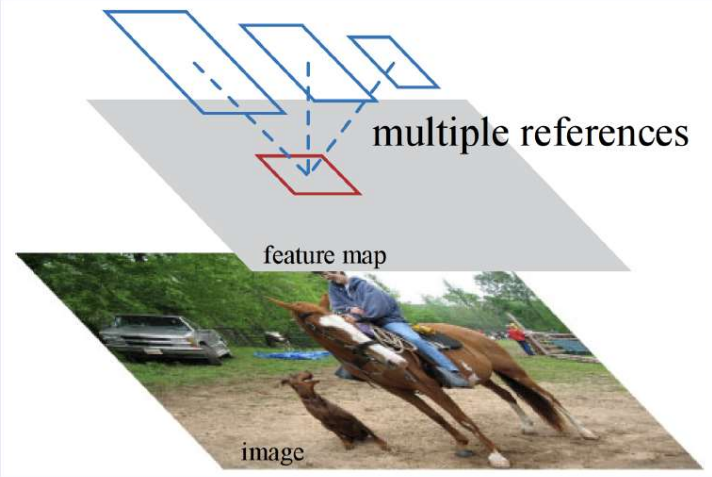
The Faster RCNN paper described 9 anchors per ROI:

- 3 different anchor sizes: 128×128, 256×256, and 512×512.
- 3 different aspect ratios: $1:2$, $1:1$, and $2:1$

9 anchors per ROI



"Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," Ren, He, Girshick & Sun, 2016



- Each candidate bounding box computes 9 different regression outputs, each of which is a 4-vector (x,y,w,h)
- The 9 different regression outputs from each bbox are w.r.t. 9 different "anchor" rectangles, each offset from the input ROI. Thus:

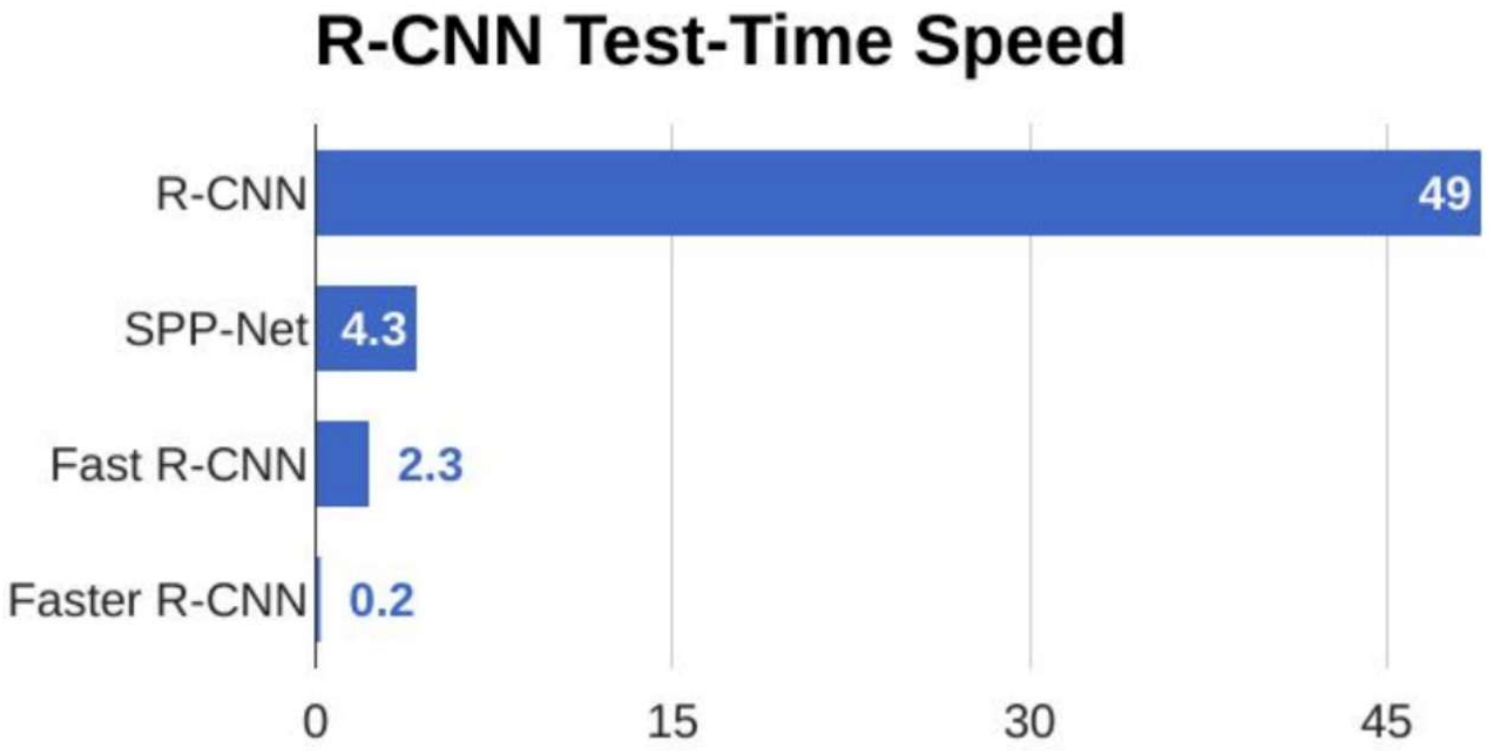
$$\text{anchor} = \text{ROI} + \text{known shift}$$

$$\text{object} = \text{anchor} + \text{regression}$$

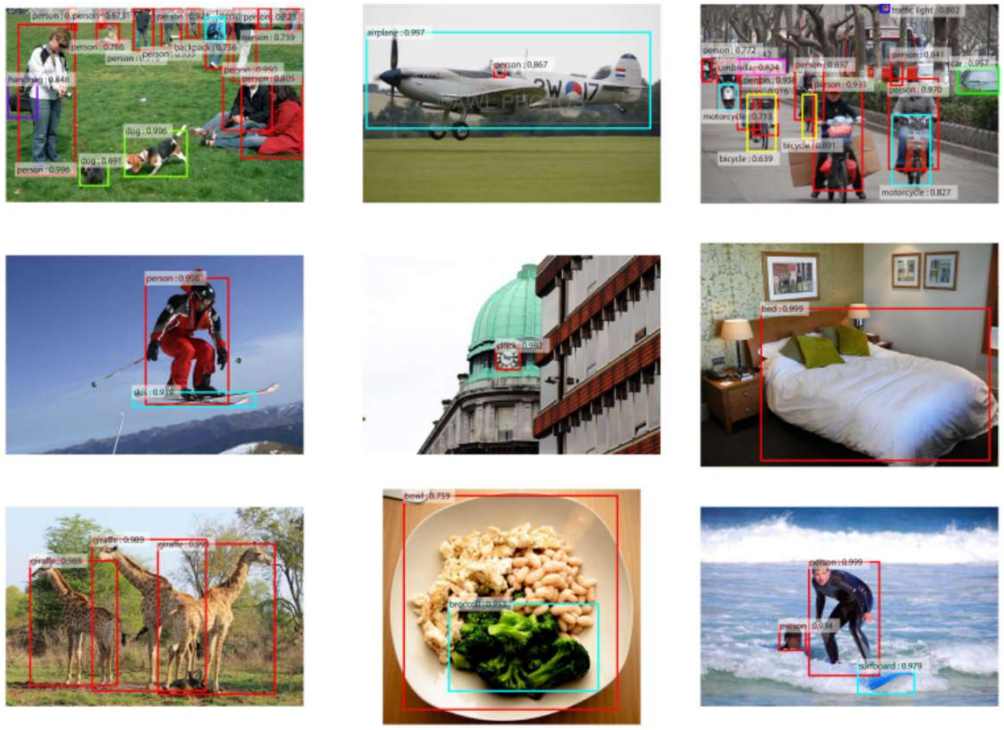
- The ROI is $(x_{ROI}, y_{ROI}, w_{ROI}, h_{ROI})$.
- The anchor is (x_a, y_a, w_a, h_a).
- The true object is located at $(x_{REF}, y_{REF}, w_{REF}, h_{REF})$.
- The regression target is:

$$\vec{y_r} = \begin{bmatrix} \frac{x_{REF} - x_a}{w_a} \\ \frac{y_{REF} - y_a}{h_a} \\ \ln\left(\frac{w_{REF}}{w_a}\right) \\ \ln\left(\frac{h_{REF}}{h_a}\right) \end{bmatrix}$$

Speeds


R-CNN Test-Time Speed
R-CNN
SPP-Net
Fast R-CNN
Faster R-CNN
49
4.3
2.3
0.2
0
15
30
45

Sample Detections



How can this be learned?

- Object detection adds *a lot* of complexity compared to a CNN
- Many practical details become important for effective training
- Please read the papers if you go to implement this yourself

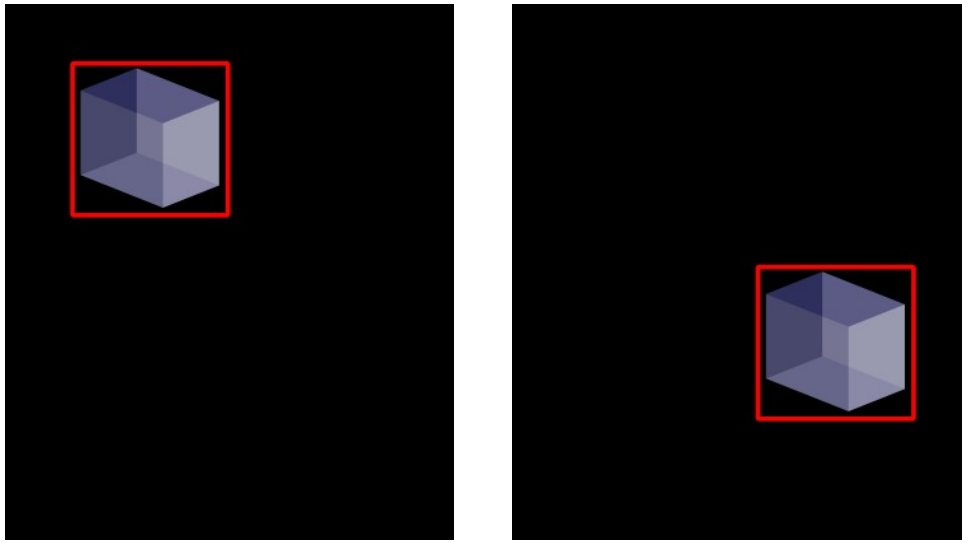
Let's go back to our goals:

We have some important properties that are useful in object detection

- Translation Invariance
- Scale Invariance
- Rotation Invariance

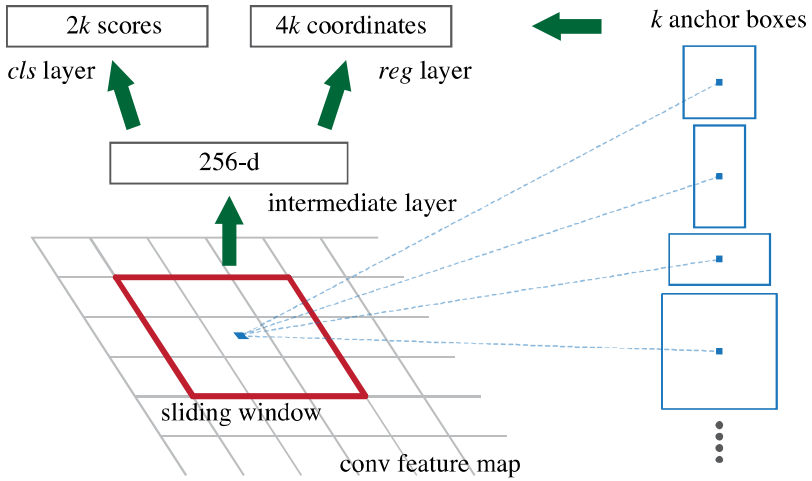
Some are built into the algorithm, and some come from the structure of the dataset

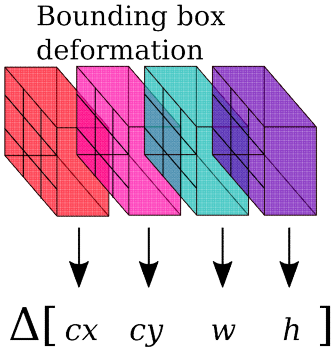
Translation Invariance



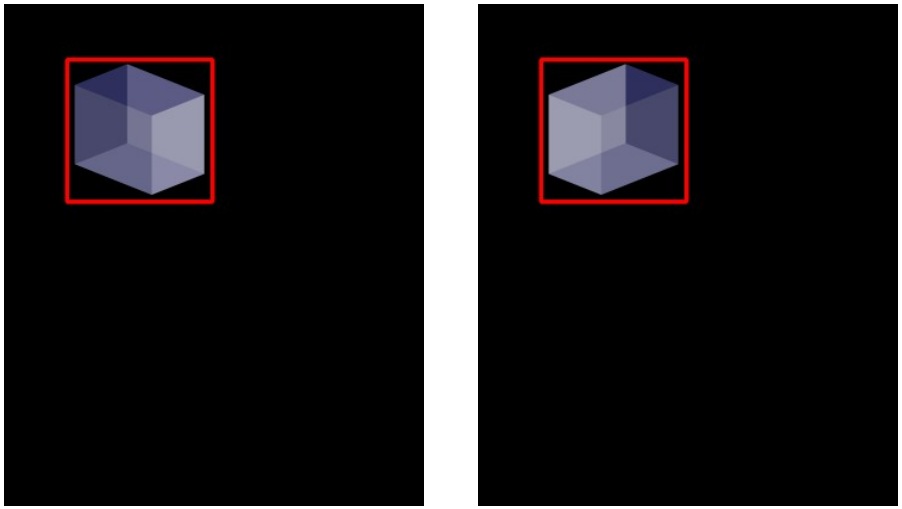
This is addressed with convolutional features…

Translation invariance comes from the sliding windows (same as conv.)



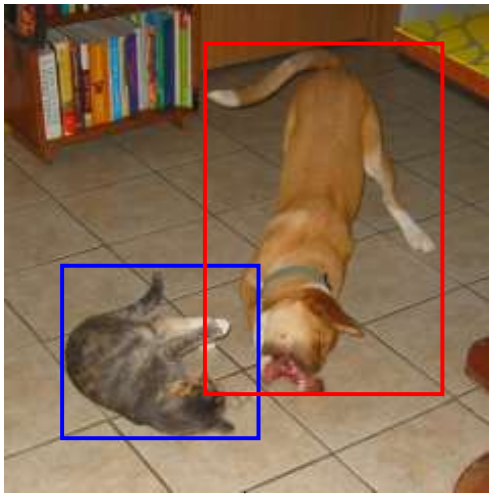


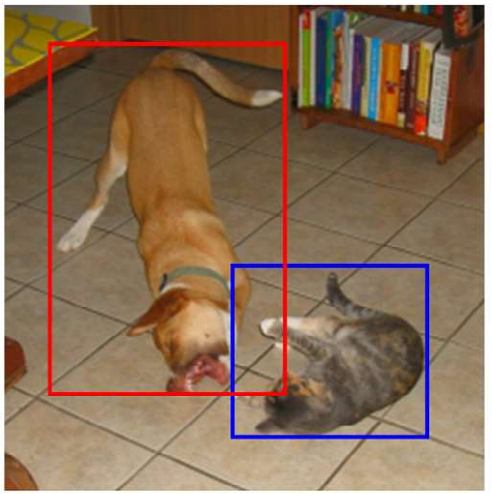
Rotation Invariance



handled by data augmentation and other techniques in the dataset.

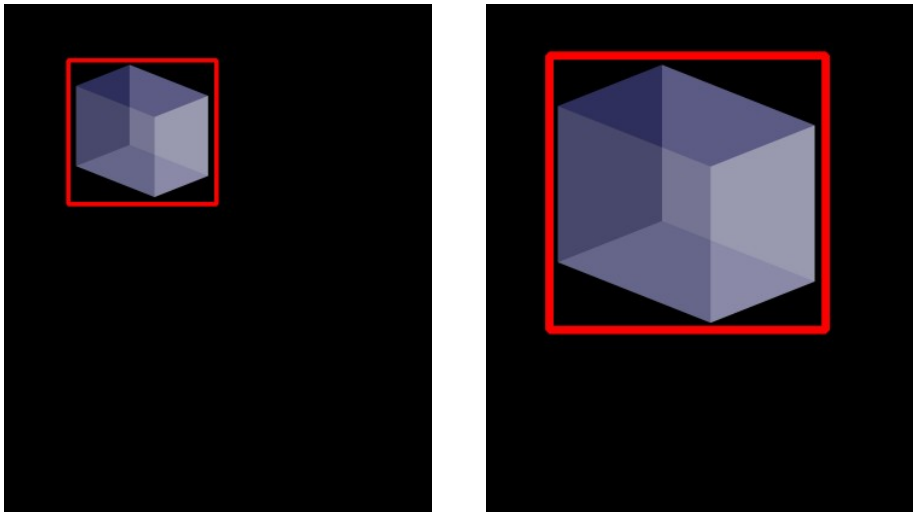
Data Augmentation





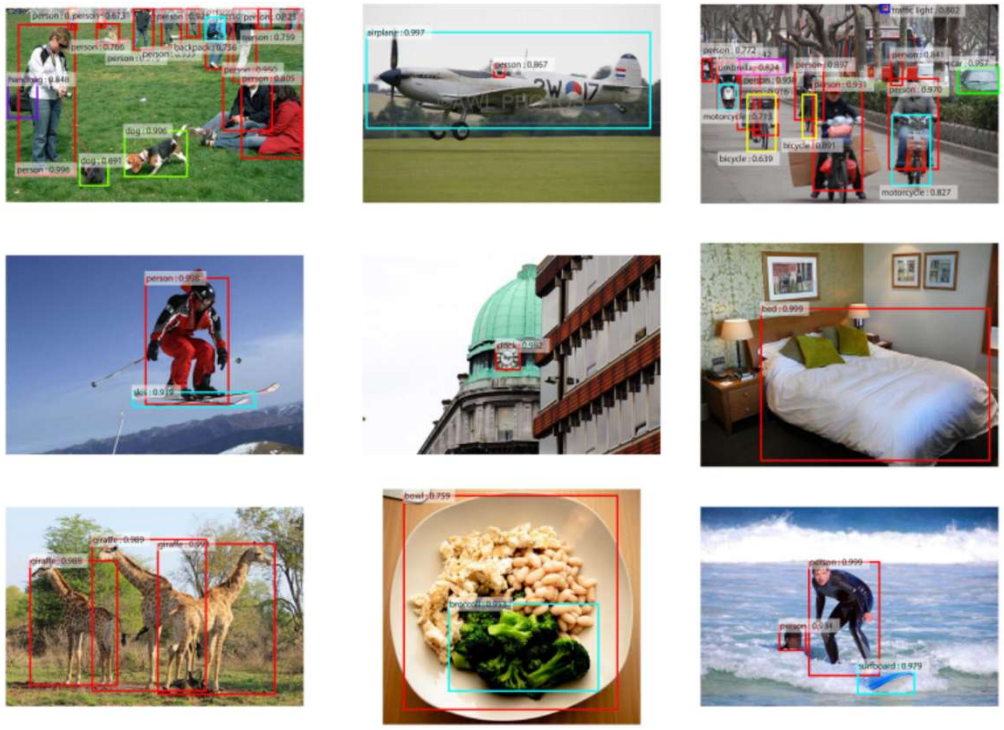
Feed in mirros, rotations, shifts, etc.

Scale Invariance



handled by data diversity and rescaling feature maps.

Sample Detections

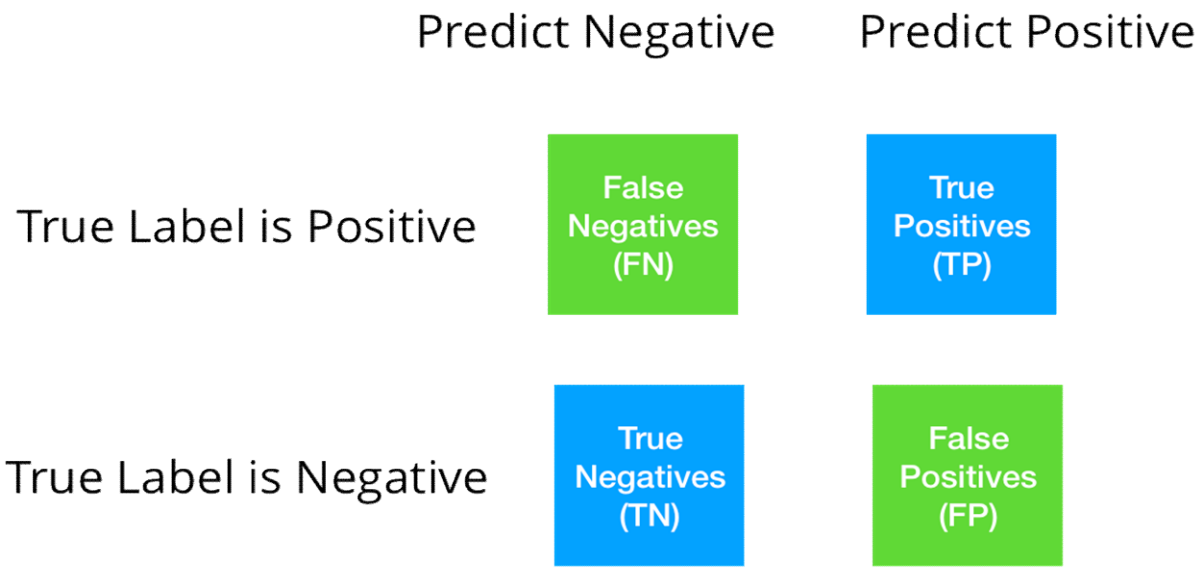


How do we measure performance

- We use mean average precision (mAP) over all our classes

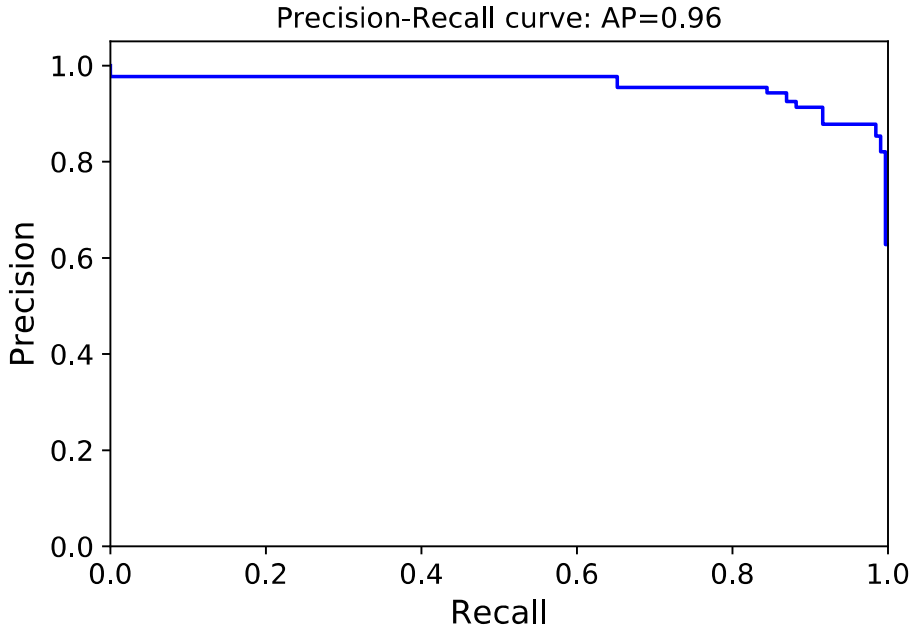
- Must use a precision-recall metric *because there are so many ways to get things wrong*

Types of Predictions



Precision is given by (TP)/(TP+FP).
Recall is given by (TP)/(TP+FN).

Precision-Recall



Average Precision is the integral of this curve.

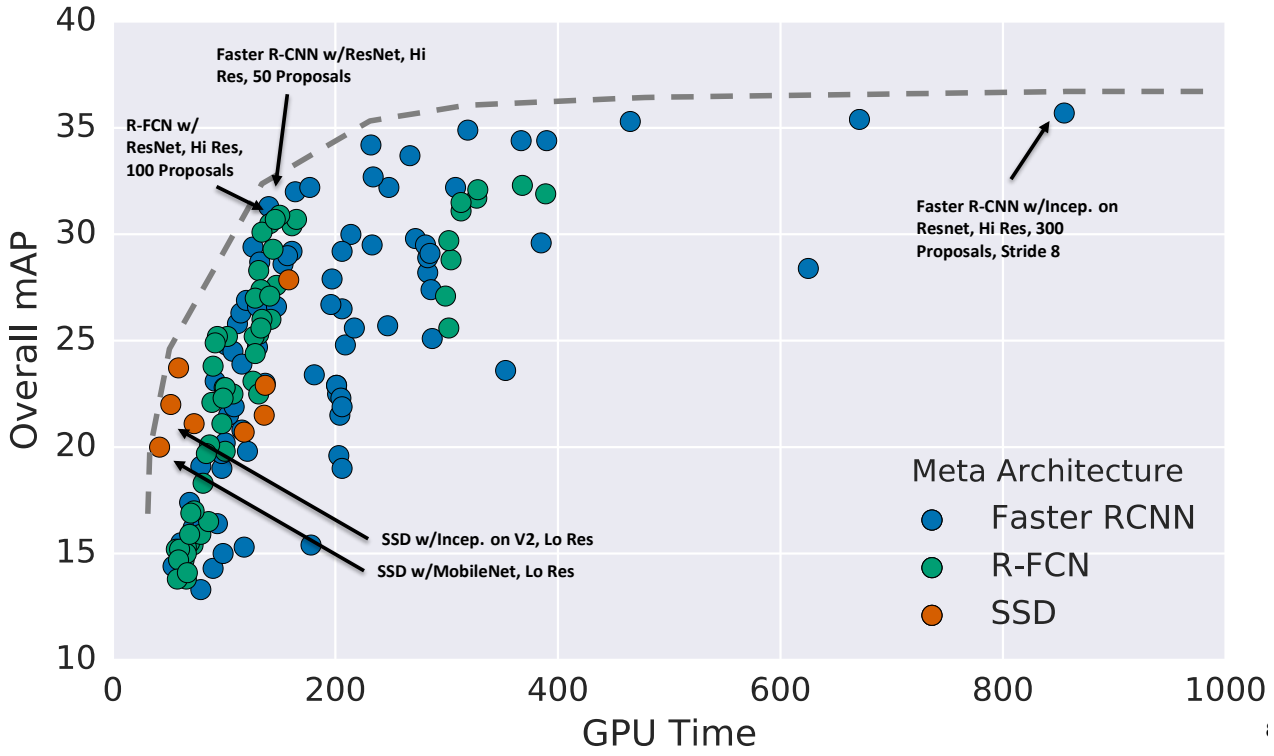
Mean Average Precision

- Average Precision (AP) isn't perfect. Nevertheless, it works well for a single class.
- When we have multiple classes, we can take the "Mean Average Precision," which is the average precision over multiple classes:
- $$mAP\ (or\ MAP) = \ mean\ (AP\ for\ each\ class).$$
- Other common metrics include:

$$F_1 = 2\ (Precision * Recall)/(Precision + Recall)$$

- MRR = mean reciprocal rank (useful when only one answer is appropriate out of many)

Performance Tradeoffs



Where is object detection going?

http://cocodataset.org/#detection-2018

Resources for training your own object detector

Object detection in PyTorch
https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection