Introduction to Deep Learning (IT3320E)

7 - Object detection with Deep Learning

Hung Son Nguyen

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Oct. 25, 2022

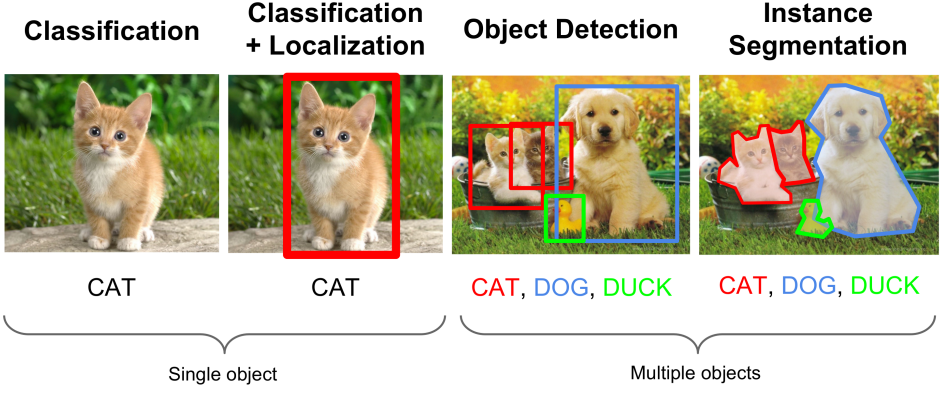
Agenda

1. Object Detection
2. Histogram of Oriented Gradients (HOG)
3. Region-based convolutional neural networks (R-CNNs)
   (do not use this method, deprecated)
4. Fast R-CNN (do not use this method, deprecated)
5. Faster R-CNN
6. YOLO
7. Single Shot Detector (SSD)

Section 1

Object Detection

- We know how we can use CNN for image classification tasks, where we assumed that there is only one main target object in the image.
- In many situations, there are multiple targets in the image, and we not only want to classify them, but also want to obtain their positions in the image.



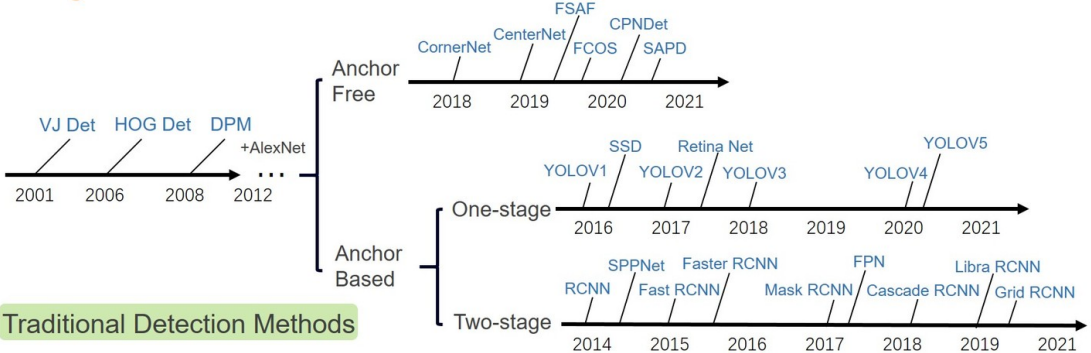
- In computer vision, we refer to such tasks as object detection.

Image classification	Object detection
The goal is to predict the type or class of an object in an image. ■ Input: an image with a single object ■ Output: a class label (cat, dog, etc.) ■ Example output: class probability (for example, 84% cat)	The goal is to predict the location of objects in an image via bounding boxes and the classes of the located objects. ■ Input: an image with one or more objects ■ Output: one or more bounding boxes (defined by coordinates) and a class label for each bounding box ■ Example output for an image with two objects: – box1 coordinates (x, y, w, h) and class probability – box2 coordinates and class probability Note that the image coordinates (x, y, w, h) are as follows: (x and y) are the coordinates of the bounding-box center point, and (w and h) are the width and height of the box.

This lecture layout

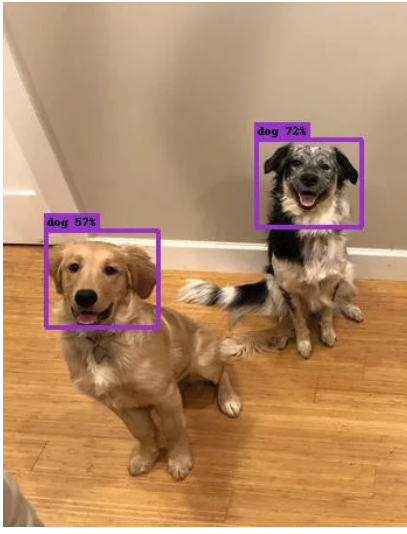
- Understanding image classification vs. object detection
- the general framework of the object detection algorithms.
- exploration of the most popular detection algorithms: the R-CNN family of networks, SSD, and the YOLO family of networks.
- Application in a real-world project to train an end-to-end object detector.

Object Detection Milestones



Data requirements

- Training a custom model needs labelled data.
- Labelled data in the context of object detection are images with corresponding bounding box coordinates and labels.
- That is, the bottom left and top right (x,y) coordinates + the class.
- The normalized bounding box coordinates for the dogs in the image are e.g. [0.1, 0.44, 0.34, 0.56] and [0.72, 0.57, 0.87, 0.77]



Question:

In order to do object detection on problem X, how many pictures do we need?

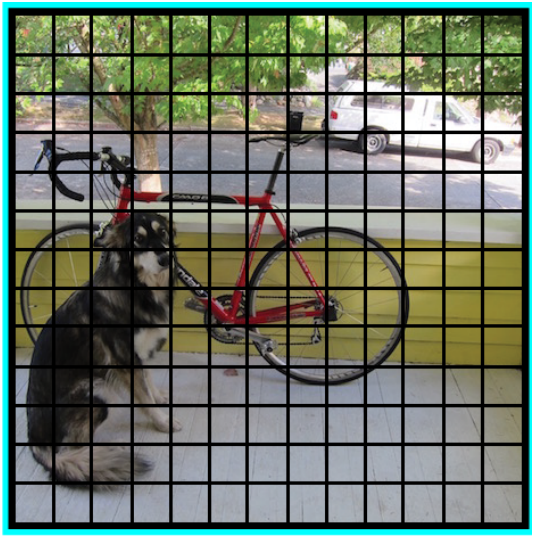
- Instead, it is more important to properly understand in which scenarios the model will be deployed.
- It is crucial that a large number (for example, > 100 and potentially >1000) representative images are available per class.
- Representative in this context means that they should corresponds with the range of scenarios in which the model will be used.

Typically, an object detection framework has four components:

1. **Region proposal**—First, a deep learning model or algorithm is used to generate a large set of bounding boxes spanning the full image (that is, an object localization component)
2. **Feature extraction and network predictions**—visual features are extracted for each of the bounding boxes. They are evaluated and it is determined whether and which objects are present in the boxes based on visual features (i.e. an object classification component)
3. **Non-maximum suppression (NMS)**—is a technique that aims at selecting the best bounding box out of a set of overlapping boxes.
4. **Evaluation metrics**—Similar to accuracy, precision, and recall metrics in image classification tasks, object detection systems have their own metrics to evaluate their detection performance. In this section, we will explain the most popular metrics, like mean average precision (mAP), precision-recall curve (PR curve), and intersection over union (IoU).

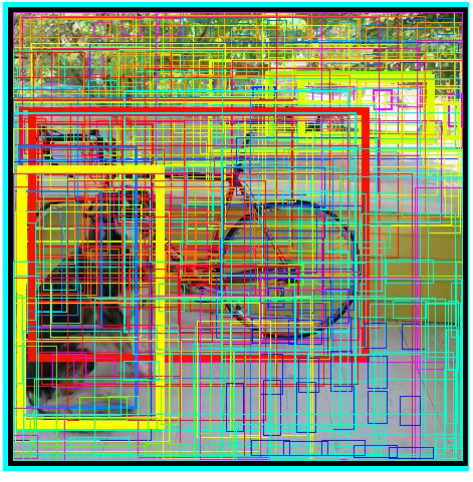
Region proposal: In this component, a DL model is used to generate regions of interest (RoIs) to be further processed by the system.

- These are regions that the network believes might contain an object;
- The output is a large number of bounding boxes, each of which has an objectness score.
- Boxes with large objectness scores are then passed along the network layers for further processing.



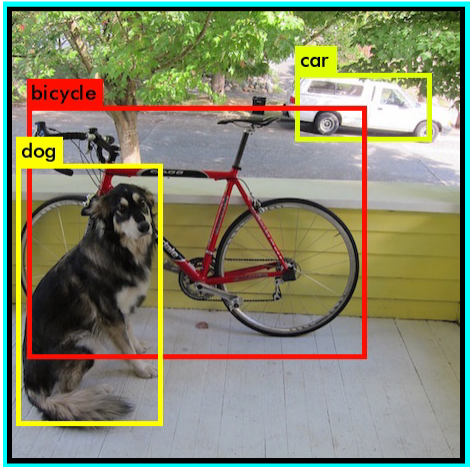
Feature extraction and network predictions: the network analyzes all the regions that have been identified as having a high likelihood of containing an object and makes two predictions for each region:

- Bounding-box prediction The coordinates that locate the box surrounding the object. The bounding box coordinates are represented as the tuple (x, y, w, h), where x and y are the coordinates of the center point of the bounding box and w and h are the width and height of the box.
- Class prediction: The classic softmax function that predicts the class probability for each object.



Non-maximum suppression (NMS):

- In this step, the model has likely found multiple bounding boxes for the same object.
- NMS helps avoid repeated detection of the same instance by combining overlapping boxes into a single bounding box for each object.
- The Intersection over Union (IoU) metric is essentially a method used usually to quantify the percent overlap between the two Bounding Boxes.

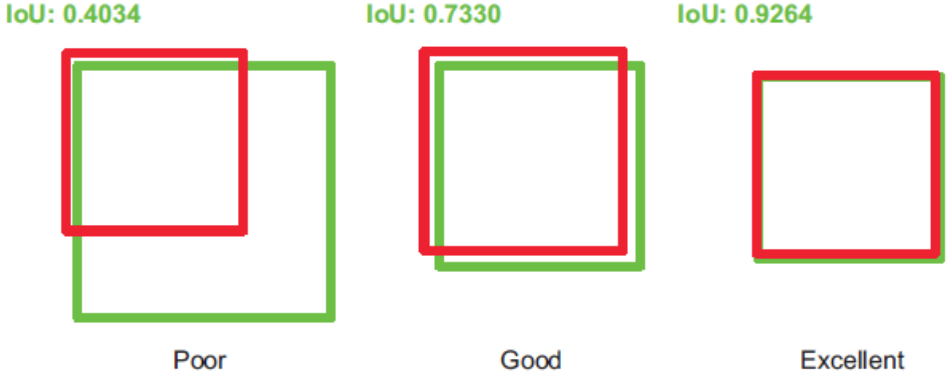


Predicted person bounding box

Ground truth person bounding box

$$\text{Score} = \frac{\text{Area of overlap}}{\text{Area of union}}$$

$$IOU = \frac{Area\ of\ Intersection\ of\ two\ boxes}{Area\ of\ Union\ of\ two\ boxes}$$



Input: We get a list P of prediction BBoxes of the form $(x1, y1, x2, y2, c)$, where $(x1, y1)$ and $(x2, y2)$ are the ends of the BBox and c is the predicted confidence score of the model. We also get overlap threshold IoU thresh_iou.

Output: We return a list keep of filtered prediction BBoxes.

- **Stage 1:** Select the prediction S with highest confidence score and remove it from P and add it to the final prediction list **keep**. (**keep** is empty initially).
- **Stage 2**: Now compare this prediction S with all the predictions present in P. Calculate the IoU of this prediction S with every other predictions in P. If the IoU is greater than the threshold thresh_iou for any prediction T present in P, remove prediction T from P.
- **Stage 3:** If there are still predictions left in P, then go to Stage 1 again, else return the list keep containing the filtered predictions.

- INTERSECTION OVER UNION (IOU): (above). If the IoU value is above this threshold, the prediction is considered a True Positive (TP), and if it is below the threshold, it is considered a False Positive (FP).
- PRECISION-RECALL CURVE (PR CURVE): With the TP and FP defined, we can now calculate the precision and recall of our detection for a given class across the testing dataset.

$$\text{Recall} = \frac{TP}{TP + FN} \quad \text{Precision} = \frac{TP}{TP + FP}$$

Now that we have the PR curve, we can calculate the average precision (AP) by calculating the area under the curve (AUC). Finally, the mAP for object detection is the average of the AP calculated for all the classes. It is also important to note that some research papers use AP and mAP interchangeably

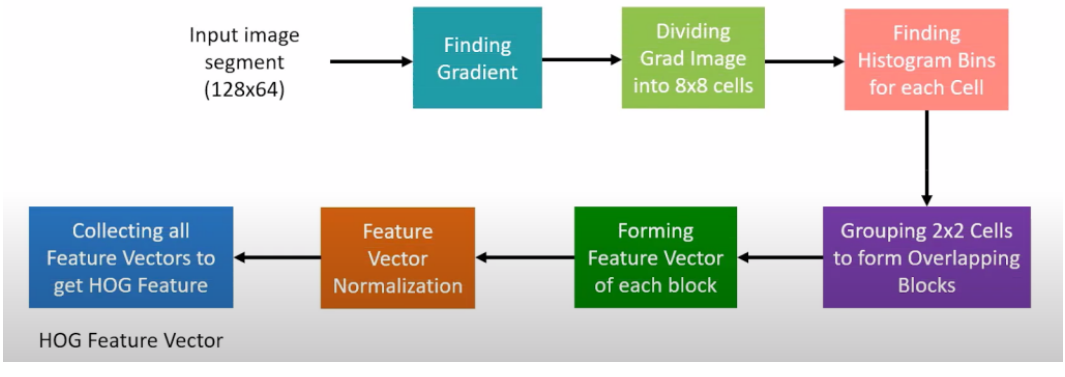
- MEAN AVERAGE PRECISION (MAP) TO MEASURE NETWORK PRECISION: The most common evaluation metric used in object recognition tasks is mean average precision (mAP). It is a percentage from 0 to 100, and higher values are typically better, but its value is different from the accuracy metric used in classification.
  To recap, the mAP is calculated as follows:
  1. Get each bounding box's associated objectness score (probability of the box containing an object).
  2. Calculate precision and recall.
  3. Compute the PR curve for each class by varying the score threshold.
  4. Calculate the AP: the area under the PR curve. In this step, the AP is computed for each class.
  5. Calculate the mAP: the average AP over all the different classes.

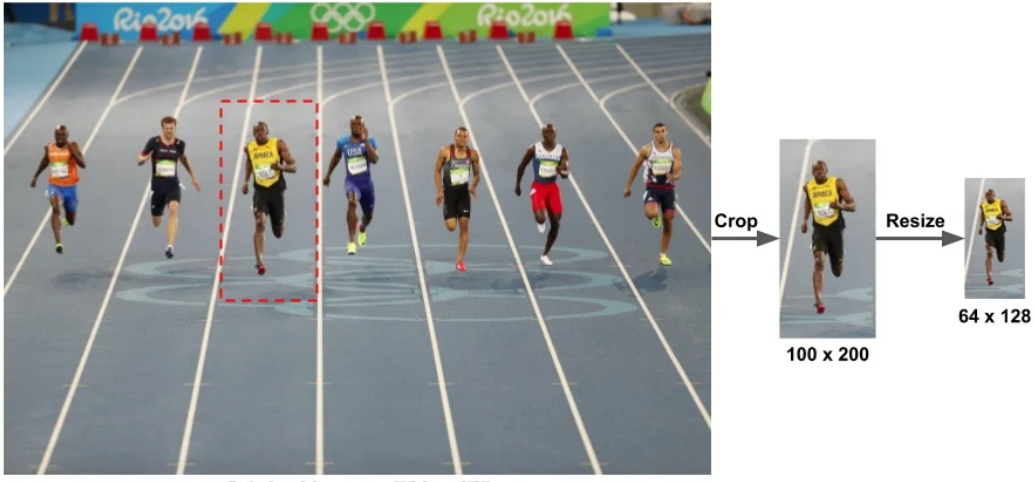
- FRAMES PER SECOND (FPS) TO MEASURE DETECTION SPEED: The most common metric used to measure detection speed is the number of frames per second (FPS). For example, Faster R-CNN operates at only 7 FPS, whereas SSD operates at 59 FPS.
  In benchmarking experiments, you will see the authors of a paper state their network results as: “Network X achieves mAP of Y% at Z FPS,” where X is the network name, Y is the mAP percentage, and Z is the FPS.

Section 2

Histogram of Oriented Gradients (HOG)

- The Histogram of Oriented Gradients is one of the oldest methods of object detection. It was first introduced in 1986.
- Despite some developments in the upcoming decade, the approach did not gain a lot of popularity until 2005 when it started being used in many tasks related to computer vision.
- HOG uses a feature extractor to identify objects in an image.





HOG feature descriptor is calculated on a 64×128 patch of an image. An image may be of any size. The only constraint for patches: aspect ratio = 1:2.

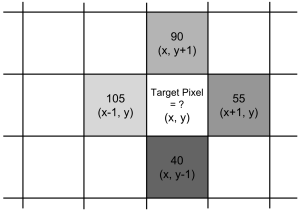
Suppose $f(x, y)$ records the color of the pixel at location (x, y), the gradient vector of the pixel (x, y) is defined as follows:

$$\nabla f(x, y) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} f(x+1, y) - f(x-1, y) \\ f(x, y+1) - f(x, y-1) \end{bmatrix}$$

There are two important attributes of an image gradient:

- Magnitude is the L2-norm of the vector, $g = \sqrt{g_x^2 + g_y^2}$
- Direction is the slope angle of the gradient: $\theta = \arctan(g_y / g_x)$

For example:



$$\nabla f = \begin{bmatrix} f(x+1, y) - f(x-1, y) \\ f(x, y+1) - f(x, y-1) \end{bmatrix} = \begin{bmatrix} 55 - 105 \\ 90 - 40 \end{bmatrix} = \begin{bmatrix} -50 \\ 50 \end{bmatrix}$$

- The magnitude $= \sqrt{50^2 + (-50)^2} = 70.7107$
- the direction is: $\theta = \arctan(-50/50) = -45°$

Repeating the gradient computation process for every pixel iteratively is too slow. Instead, it can be well translated into applying a convolution operator on the entire image matrix, labeled as $\mathbf{A}$ using one of the specially designed convolutional kernels.

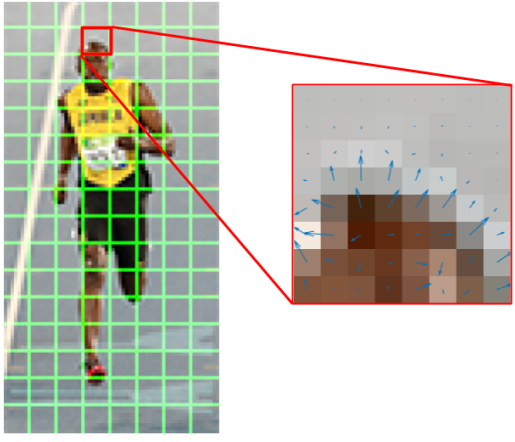
- Prewitt operator: Rather than only relying on four directly adjacent neighbors, the Prewitt operator utilizes eight surrounding pixels for smoother results.

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix} \circledast \mathbf{A} \text{ and } \mathbf{G}_y = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \circledast \mathbf{A}$$

- Sobel operator: To emphasize the impact of directly adjacent pixels more, they get assigned with higher weights.

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \circledast \mathbf{A} \text{ and } \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \circledast \mathbf{A}$$

Divide the image into many 8x8 pixel cells. In each cell, the magnitude values of these 64 cells are binned and cumulatively added into 9 buckets of unsigned direction (no sign, so 0-180 degree rather than 0-360 degree; this is a practical choice based on empirical experiments).



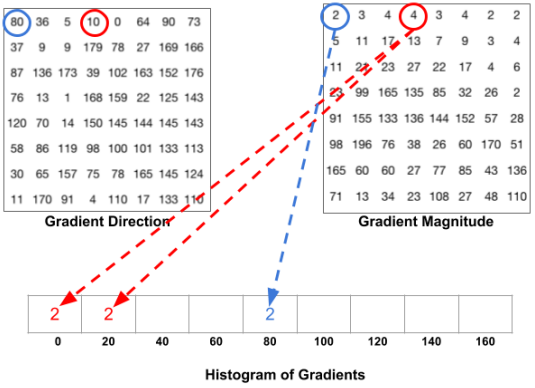
2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

Step 3 : Calculate Histogram of Gradients in 8×8 cells

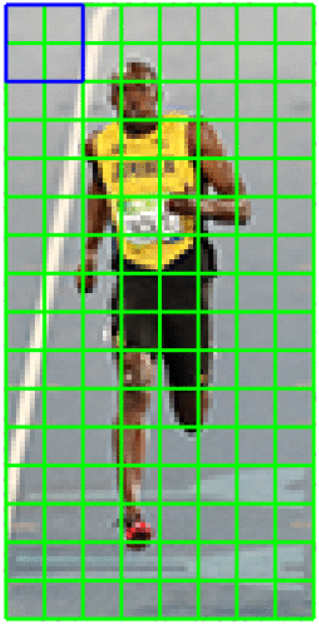


For better robustness:

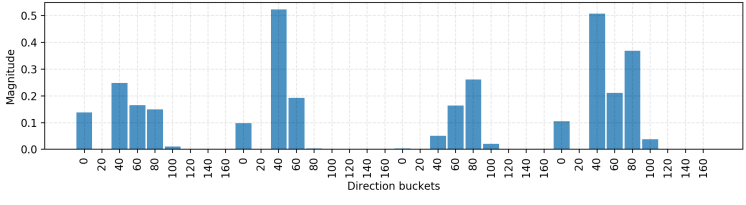
- if the direction of the gradient vector of a pixel lays between two buckets, its magnitude does not all go into the closer one but proportionally split between two.
- For example, if a pixel's gradient vector has magnitude 8 and degree 15, it is between two buckets for degree 0 and 20 and we would assign 2 to bucket 0 and 6 to bucket 20.

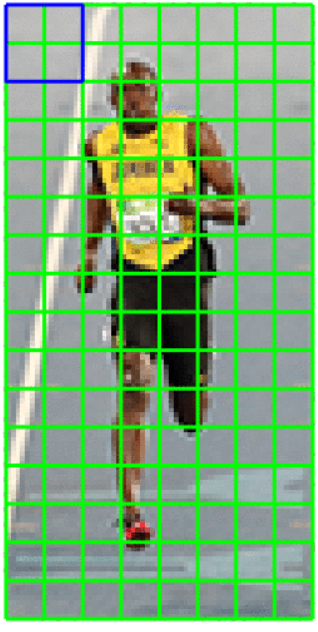
Normalization: example

- Let's say we have an RGB color vector [128, 64, 32].
- The length of this vector is $\sqrt{128^2 + 64^2 + 32^2} = 146.64$.
- This is also called the L2 norm of the vector. Dividing each element of this vector by 146.64 gives us a normalized vector [0.87, 0.43, 0.22].



- Then we slide a 2x2 cells (thus 16x16 pixels) block across the image.
- In each block region, 4 histograms of 4 cells are concatenated into one-dimensional vector of 36 values and then normalized to have an unit weight.
- The final HOG feature vector is the concatenation of all the block vectors. It can be fed into a classifier like SVM for learning object recognition tasks.



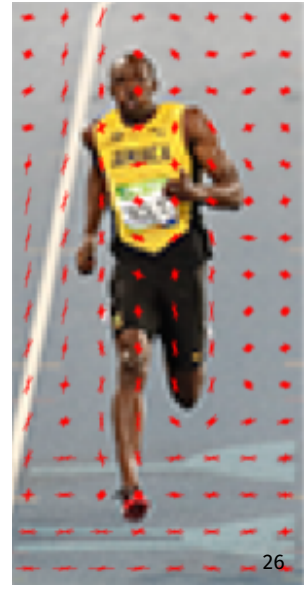


To calculate the final feature vector for the entire image patch, the 36×1 vectors are concatenated into one giant vector.

What is the size of this vector? Let us calculate

- How many positions of the 16×16 blocks do we have ? There are 7 horizontal and 15 vertical positions making a total of 7 x 15 = 105 positions.
- Each 16×16 block is represented by a 36×1 vector. So when we concatenate them all into one gaint vector we obtain a 36×105 = 3780 dimensional vector.

- HOG descriptor visualized on an image by plotting the 9×1 normalized histograms in the 8×8 cells.
- The HOG descriptor of an image patch is usually visualized by plotting the 9×1 normalized histograms in the 8×8 cells.
- See image on the side. You will notice that dominant direction of the histogram captures the shape of the person, especially around the torso and legs.



Achievements of HOG

- Creation of a feature descriptor useful for performing object detection.
- Ability to be combined with support vector machines (SVMs) to achieve high-accuracy object detection.
- Creation of a feature descriptor useful for performing object detection.

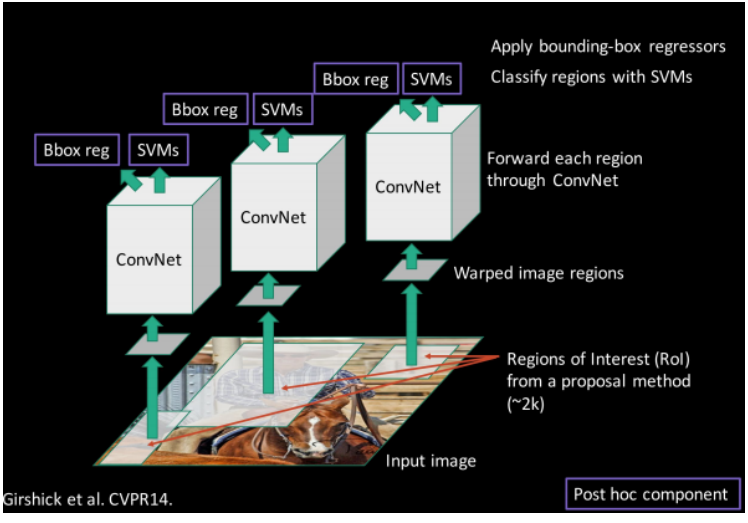
Points to consider

- **Limitations** – While (HOG) was quite revolutionary in the beginning stages of object detection, there were a lot of issues in this method. It's quite time-consuming for complex pixel computation in images, and ineffective in certain object detection scenarios with tighter spaces.
- **When to use HOG?** – HOG should often be used as the first method of object detection to test other algorithms and their respective performance. Regardless, HOG finds significant use in most object detection and facial landmark recognition with decent accuracy.
- **Example use cases** – pedestrian detection due to its smooth edges. Other general applications include object detection of specific objects.

Section 3

Region-based convolutional neural networks (R-CNNs) (do not use this method, deprecated)

So the idea for object detection is to use R-CNN.



R-CNN

First run a region proposal algorithm to obten regions of interets.

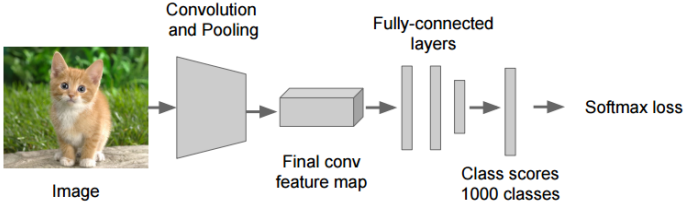
Then, warp this regions into a fixed size and run the ConvNet with regression head and classification head.

The regression head objective is to output an offset to correct ”slightly wrong” region proposals.

R-CNN problems:

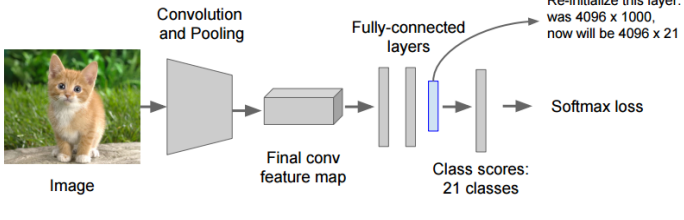
- Slow at test-time: need to run full forward pass of CNN for each region proposal
- SVMs and regressors are post-hoc: CNN features not updated in response to SVMs and regressors
- Complex multistage training pipeline

1.- **Train** (or download) a classification model (e.g. AlexNet).



Train

2.- **Fine-tune model for detection**. Instead of X classes you want Y classes + background. To do so, throw away the final fully-connected layer, add the new one and retrain using positive/negative regions from detection images.

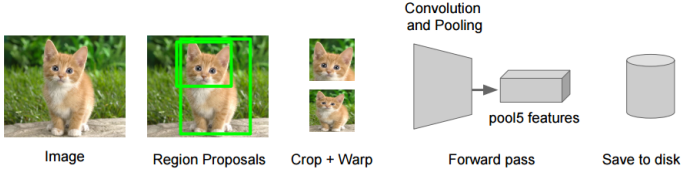


Fine-tune

3.- **Extract features**. Extract region proposals for all images. For each region: warp to CNN input size, run forward through CNN, save last pool features to disk.

4.- **Train classification head**. Train one binary SVM per class using as input the saved features from last step.

5- **Train regression head**. For each class, train a linear regression model to map from cached features to offset to GT boxes to make up for "slightly wrong" proposals of the region proposals.

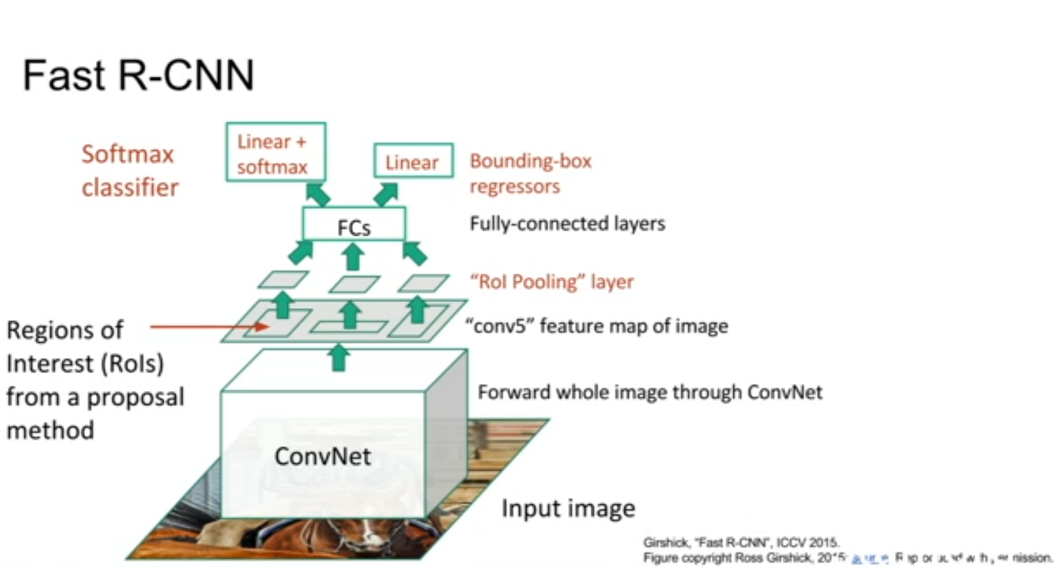


Extract features

Train regression head

Section 4

Fast R-CNN (do not use this method, deprecated)

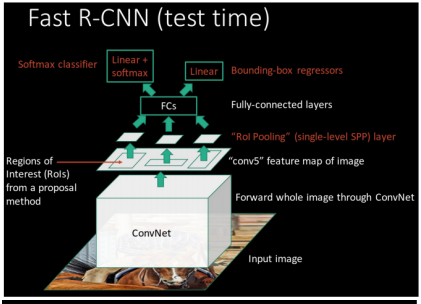

Fast R-CNN
Softmax classifier
Linear + softmax
Linear
Bounding-box regressors
FCs
Fully-connected layers
"RoI Pooling" layer
Regions of Interest (RoIs) from a proposal method
"conv5" feature map of image
Forward whole image through ConvNet
ConvNet
Input image
Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 20

Fast R-CNN training/testing time

A new proposal called Fast R-CNN appeared trying to solve R-CNN problems.

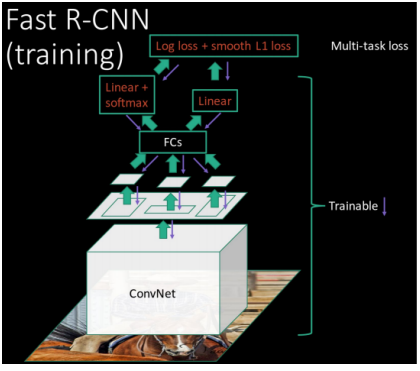
The idea is just to swap the order of extracting regions of interest and features.

Still, no sliding windows.



R-CNN Problem #1:
Slow at test-time due to independent forward passes of the CNN

Solution:
Share computation of convolutional layers between proposals for an image



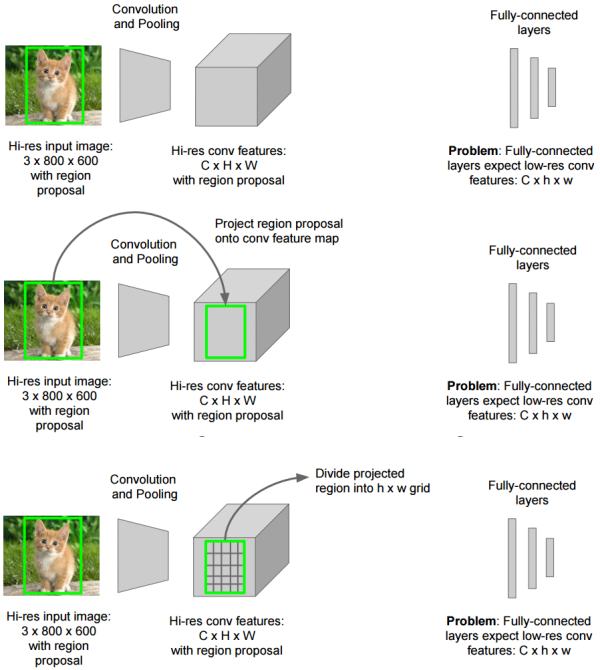
R-CNN Problem #2:
Post-hoc training: CNN not updated in response to final classifiers and regressors

R-CNN Problem #3:
Complex training pipeline

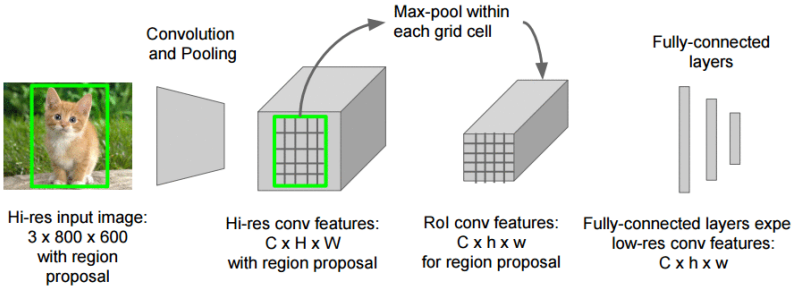
Solution:
Just train the whole system end-to-end all at once!

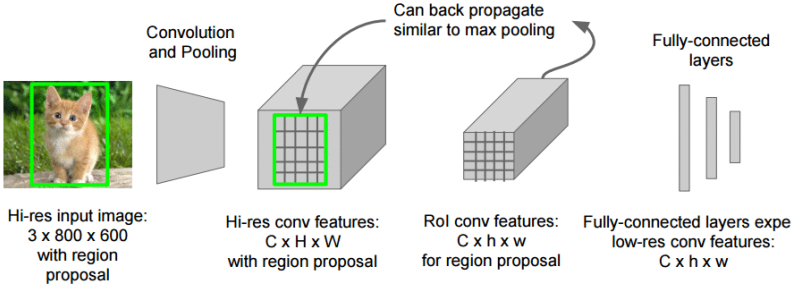
Slide credit: Ross Girschick

The only mystery here are region of interest pooling.



- The HxW image is processed by the convolution and pooling part and regions of interest are proposed (with an external algorithm).
- The problem now is that the FC layer is expecting a fixed size input.
- To solve this, given the region proposal, we project it onto the conv feature map, divide it into $h \times w$ grid and do max-pool within each grid cell.


Convolution
and Pooling
Max-pool within
each grid cell
Fully-connected
layers
Hi-res input image:
3 x 800 x 600
with region
proposal
Hi-res conv features:
C x H x W
with region proposal
RoI conv features:
C x h x w
for region proposal
Fully-connected layers expe
low-res conv features:
C x h x w


Convolution
and Pooling
Can back propagate
similar to max pooling
Fully-connected
layers
Hi-res input image:
3 x 800 x 600
with region
proposal
Hi-res conv features:
C x H x W
with region proposal
RoI conv features:
C x h x w
for region proposal
Fully-connected layers expe
low-res conv features:
C x h x w

With this simple strategy the FC layer will always receive the same input size. Moreover, we can backprop through this strategy with no problem because it only uses max-pool.

In terms of mAP (accuracy) is more or less the same The computational time improvement of Fast R-CNN w.r.t. R-CNN is impressive:

- x8.8 training time speed up
- x25 testing time speed up (50s to 2s)

However, the bottleneck is extracting the regions with a region proposals algorithm. Without it, the speed would be:

- x146 testing time speed up (47s to 0.32s)

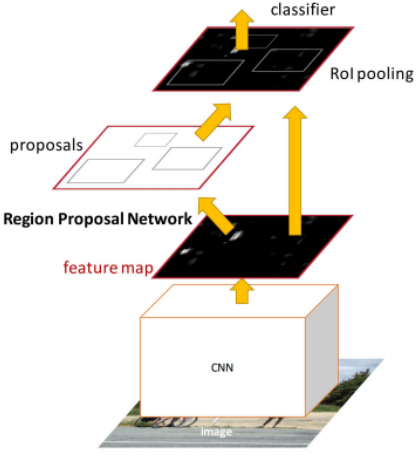
so why not extracting the ROIs with the CNN? this is what Faster R-CNN proposes

Section 5

Faster R-CNN

To solve the problem of region proposals algorithm botleneck in Fast R-CNN, Faster R-CNN proposes to extract the regions of interest with another network using the information of the last layer of the CNN net.

- Insert a Region Proposal Network (RPN) after the last convolutional layer.
- RPN trained to produce region proposals directly; no need for external region proposals!
- After RPN, use RoI Pooling and an upstream classifier and bbox regressor just like Fast R-CNN
- Still, the RoI pooling does not allow to introduce rotations into your regions of interests. To introduce rotations, a DeepMind paper proposes to do bilinear interpolation similar computer graphics.



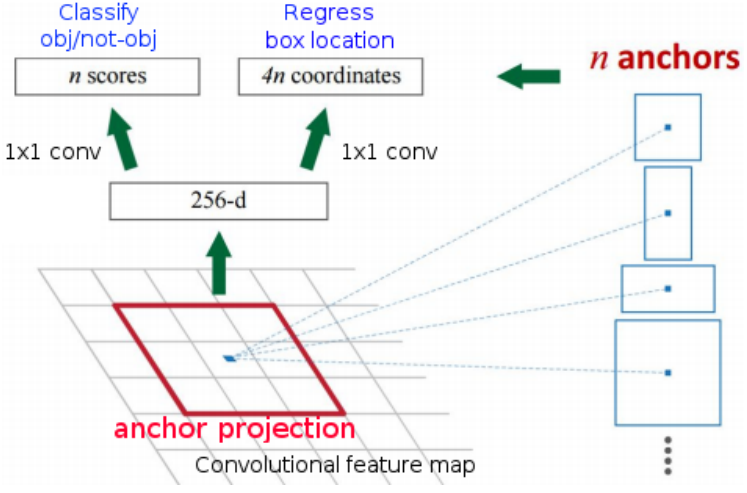
Build a small network for:

- Classifying object or no-object, and
- Regressing bbox locations

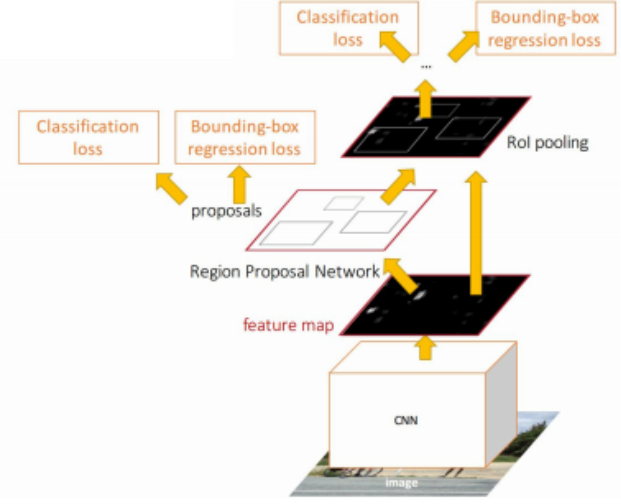
How does it work?

- Use N anchor boxes at each location.
- Anchors are translation invariant: use the same ones at every location.
- For all the feature map points, at the original input image apply all anchors to the point corresponding to the current feature map point.
- So we take as features the convolutional feature map region corresponding to the anchor region in the original image.
- For each of this anchor boxes it produces a score weather there is an object or not; and a finer localization with reference to the anchor (they are an offset to the anchor boxes)

We are learning different weights for each anchor.



The cool thing here is the region proposal network, how does it work?



How to train it?

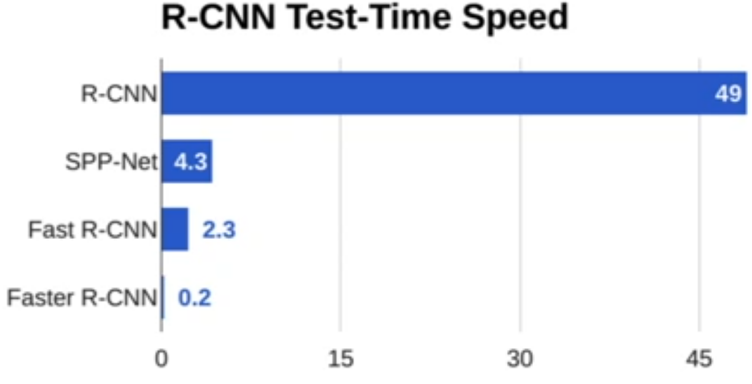
To train it, in some unpublished work, it trains the net as a net with 4 losses.

One network, four losses

- RPN classification (anchor good / bad)
- RPN regression (anchor to proposal)
- Fast R-CNN classification (over classes)
- Fast R-CNN regression (proposal to box)

Faster R-CNN:

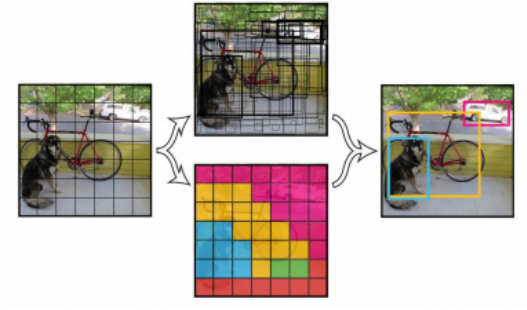
Make CNN do proposals!

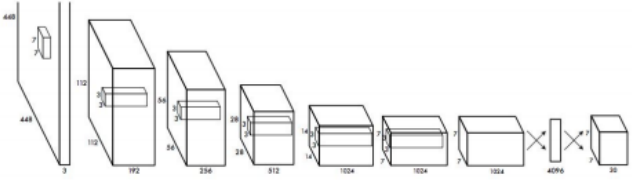


Section 6

YOLO

- YOLO is one of the most popular model architectures and algorithms for object detection.
- The YOLO model uses one of the best neural network archetypes to produce high accuracy and overall speed of processing.
- This speed and accuracy is the main reason for its popularity.

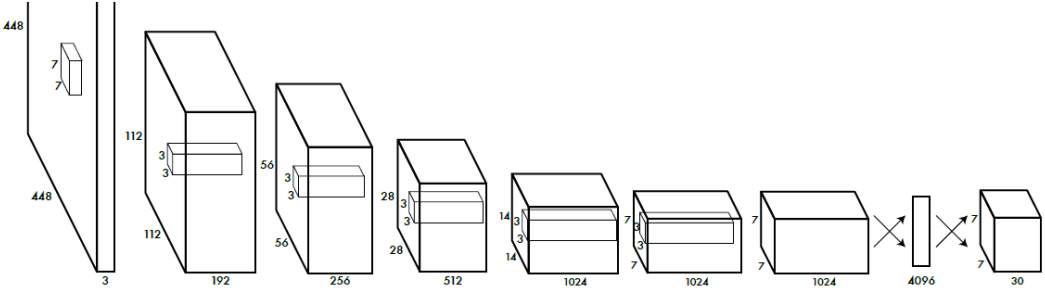




The YOLO architecture utilizes three primary terminologies to achieve its goal of object detection.

1. **residual blocks:** In the first architectural design, they have used 7×7 residual blocks to create grids in the particular image.
2. each of the central points for a particular prediction is considered for the creation of the bounding boxes. While the classification tasks work well for each grid, it's more complex to segregate the bounding boxes for each of the predictions that are made. Each of these grids acts as central points and a particular prediction for each of these grids is made accordingly.
3. Using the intersection of union (IOU) to calculate the best bounding boxes for the particular object detection task.

Working process of YOLO



Advantages of YOLO

1. The computation and processing speed of YOLO is quite high, especially in real-time compared to most of the other training methods and object detection algorithms.
2. Apart from the fast computing speed, the YOLO algorithm also manages to provide an overall high accuracy with the reduction of background errors seen in other methods.
3. The architecture of YOLO allows the model to learn and develop an understanding of numerous objects more efficiently.

Limitations of YOLO

- Failure to detect smaller objects in an image or video because of the lower recall rate.
- Cant't detect two objects that are extremely close to each other due to the limitations of bounding boxes.

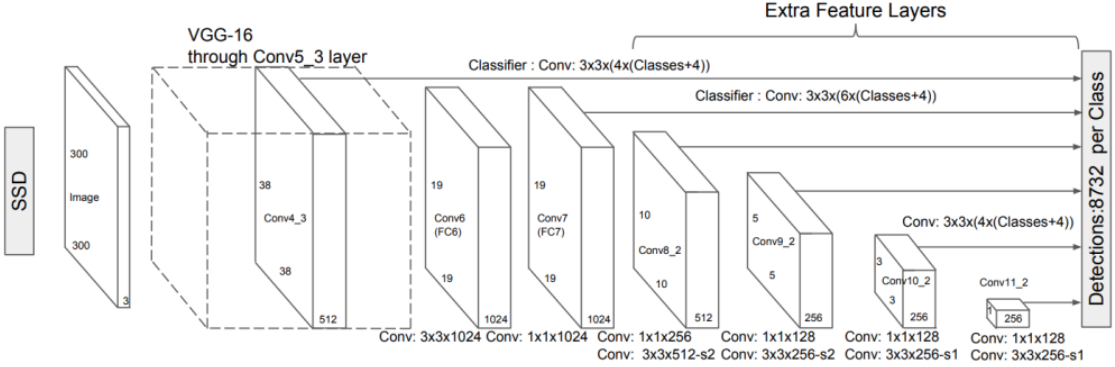
- The YOLO architecture is one of the most influential and successful object detection algorithms.
- With the introduction of the YOLO architecture in 2016, their consecutive versions YOLO v2 and YOLO v3 arrived in 2017 and 2018.
- While there was no new release in 2019, 2020 saw three quick releases: YOLO v4, YOLO v5, and PP-YOLO.
- Each of the newer versions of YOLO slightly improved on their previous ones.
- The tiny YOLO was also released to ensure that object detection could be supported on embedded devices.

1. **When To Use YOLO?** – While all the previously discussed methods perform quite well on images and sometimes video analysis for object detection, the YOLO architecture is one of the most preferred methods for performing object detection in real-time. It achieves high accuracy on most real-time processing tasks with a decent speed and frames per second depending on the device that you're running the program on.
2. **Example use cases** – Some popular use cases of the YOLO architecture apart from object detection on numerous objects include vehicle detection, animal detection, and person detection.

Section 7

Single Shot Detector (SSD)

- The single-shot detector for multi-box predictions is one of the fastest ways to achieve the real-time computation of object detection tasks.
- While the Faster R-CNN methodologies can achieve high accuracies of prediction, the overall process is quite time-consuming and it requires the real-time task to run at about 7 frames per second, which is far from desirable.
- The single-shot detector (SSD) solves this issue by improving the frames per second to almost five times more than the Faster R-CNN model.
- It removes the use of the region proposal network and instead makes use of multi-scale features and default boxes.


SSD
300
Image
300
3
VGG-16
through Conv5_3 layer
38
Conv4_3
38
512
Extra Feature Layers
Classifier : Conv: 3x3x(4x(Classes+4))
Classifier : Conv: 3x3x(6x(Classes+4))
Conv: 3x3x(4x(Classes+4))
19
Conv6
(FC6)
19
1024
Conv: 3x3x1024
19
Conv7
(FC7)
19
1024
Conv: 1x1x1024
10
Conv8_2
10
512
Conv: 1x1x256
Conv: 3x3x512-s2
5
Conv9_2
5
256
Conv: 1x1x128
Conv: 3x3x256-s2
3
Conv10_2
3
256
Conv: 1x1x128
Conv: 3x3x256-s1
1
Conv11_2
1
256
Conv: 1x1x128
Conv: 3x3x256-s1
Detections:8732 per Class

The single-shot multibox detector architecture can be broken down into mainly three components.

1. **feature extraction:** in this step all the crucial feature maps are selected. This architectural region consists of only fully convolutional layers and no other layers.
2. **detection heads:** the task is not to find the semantic meaning for the images. Instead, the primary goal is to create the most appropriate bounding maps for all the feature maps.
3. **error rate reducing:** final stage is to pass it through the non-maximum suppression layers for reducing the error rate caused by repeated bounding boxes.

Limitations of SSD

- The SSD, while boosting the performance significantly, suffers from decreasing the resolution of the images to a lower quality.
- The SSD architecture will typically perform worse than the Faster R-CNN for small-scale objects.

Points to consider

1. **When To Use SSD?** – it is often the preferred method, because SSD is a faster predictions on an image for detecting larger objects, where accuracy is not an extremely important concern. However, for more accurate predictions for smaller and precise objects, other methods must be considered.
2. **Example use cases** – The Single-shot detector can be trained and experimented on a multitude of datasets, such as PASCAL, VOC, COCO, and ILSVRC datasets. They can perform well on larger object detections like the detection of humans, tables, chairs, and other similar entities.