INTERNATIONAL INSTITUTE OF
INFORMATION TECHNOLOGY

HYDERABAD

CSE578:
Computer Vision

Professor : Anoop Namboodiri
Notes By : Sai Manaswini Reddy I

2021

RECOGNITION

Image matching :

Detect if the images belong to same object or not :

We primarily look at some unique parts (need to be unique enough, so that we can recognise these in the other images) of the image, if they match then we decide that they are the same. We look at the similarity, we need to teach this to computers. (what objects to look at to match?). We extract some local features and try to match.

Steps to local feature matching :

1. Feature detection
    a. Which parts of the image are good to match? Which can be discarded?
2. Feature description
    a. Extract features out of the points that can be matched
3. Feature matching

Invariant local features :

Find features that are invariant to transformations (We want the features that do not change much when viewed from different directions/poses/illumination,...)

- geometric invariance: translation, rotation, scale
- photometric invariance: brightness, exposure, ...

feature descriptor and feature detector should be invariant to these kinds of changes to detect those same set of points.

Advantages of local features :

- Locality : features are local, so robust to occlusion and clutter. Other clutters do not affect this particular part.
- Distinctiveness : can differentiate a large set of objects. Local descriptors are rich, have high discriminative power.
- Quantity : hundreds or thousands in a single image. Local features are available in plenty, so we can extract a large number of features from different parts. This helps in lending robustness and applicability in a large set of scenarios.
- Efficiency : real-time performance achievable. We can compute locally.
- Generality : exploit different types of features in different situations. Theøre, a large set of applications

Note :

So it is important to have a descriptor of an image based on local features.

Applications :

- 3D reconstruction
- Motion tracking
- Object recognition
- Indexing and database retrieval
- Robot navigation
- Image alignment (e.g., mosaic

Feature Detector:

To decide which arts are good to match.

- *Repeatable* : Same set of local points in a different illumination or view point.
- *Efficient* : so, we use it in different applications
- *Invariant* : should be invariant
- *Localizing* : a point detected in an image then it should be the same in a slightly different but same image.

Features we choose are :

- Point/patch,
- Edge/curve :
- Region
- Want Uniqueness
    - Look for image regions that are unusual
    - Lead to unambiguous matches in other images.

Edges :

discontinuity in intensity/color along a line or a curve

Edges can be due to :

1. Surface normal (difference in normal implies change in reflected intensity)
2. Depth (change in object - background/foreground)
3. Illumination (shadow and no shadow)
4. Surface color

Edge detection :

Difference of gaussian :

After gaussian blurring, smooth regions do not change much but sharp edges show a significant change.

Edges = Original Image - Blurred image

Uniqueness :

Local measures of uniqueness :

1. Choose a small window.
2. Derive an objective metric to decide the uniqueness.
3. Decide if the window is unique using the objective metric.
4. Slide the window.

Amount of change in the content of the window :

Flat Region :

No change in all directions.

We need the windows that exhibit a *big change* when *moved slightly*(localised) in *any direction* to be sufficiently unique.

Feature Detection - Math :

1. Choose a window W
2. Shift the window by (u, v).
3. Compare each pixel and after by summing up the squared differences.
4. This defines an SSD “error” of $E(u, v)$:

$$E(u,v) = \sum_{(x,y) \in W} [\, I(x+u, y+v) - I(x,y) \,]^2$$

5. E(u, v) is the measure of change. E(u,v) should be high enough for a small (u,v)

   a. Since (u, v) are small we can use taylor series on it.

$$I(x+u, y+v) = I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + (\textit{higher order terms} \text{ can be ignored})$$

   b. *higher order terms* (can be ignored) since u and v are small.

$$I(x+u,y+v) \approx I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v$$
$$\approx I(x,y) + \begin{bmatrix} I_x & I_y \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix} \quad \text{where } I_x = \frac{\partial I}{\partial x}$$

6. Substituting the value of $I(x+u, y+v)$ back in error :

$$E(u,v) = \sum_{(x,y) \in W} [\, I(x+u, y+v) - I(x,y) \,]^2$$
$$\approx \sum_{(x,y) \in W} \left[I(x,y) + \begin{bmatrix} I_x & I_y \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix} - I(x,y) \right]^2$$
$$\approx \sum_{(x,y) \in W} \left[\begin{bmatrix} I_x & I_y \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix} \right]^2$$

We know that $A^2 = A'A$. therefore,

$$E(u,v) = \sum_{(x,y) \in W} \begin{bmatrix} u & v \end{bmatrix} \left[\begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \right]^2$$

7. Here, we want Error quantity to be high with a small change in any direction.
   a. Move the centre of the gray window in any direction along the unit circle. Information about change is encoded in H. Information about direction is encoded in [u v].
   b. Direction of maximum change in intensity is derived using eigen decomposition (remember maximization of L2 - norm problem)
8. Eigenvalues and eigenvectors of H define shifts with the smallest and largest change (E value):

   a. x_+ = direction of largest increase in E.
   b. λ_+ = amount of increase in direction x_+
   c. x_- = direction of smallest increase in E.
   d. λ_- = amount of increase in direction x_-
9. We want the region such that even x_- is high.
10. If in a particular region there are multiple points above the threshold, then we pick the point where it is the highest (local maximum)
11. This region is unique.

Harris operator :

λ_- is a variant of the Harris operator in feature detection.

$$f = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2} = \frac{determinant\ (H)}{trace\ (H)}$$

- If λ_1 and λ_2 are large then f is large.
- $\lambda_1 * \lambda_2$ influences more than $\lambda_1 + \lambda_2$. So, if one of them is small $\lambda_1 * \lambda_2$ is small ; f is small.
- The trace is the sum of the diagonals, i.e., trace(H) = $h_{11} + h_{22}$
- Very similar to λ_- but less expensive (need not compute square root)
- Called the “Harris Corner Detector” or “Harris Operator”.

- Lots of other detectors, this is one of the most popular

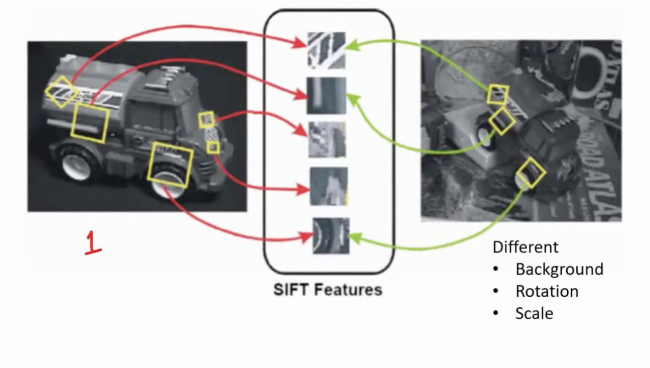
Even though these are a bit different, they end up having the same maxima.

Drawbacks :

If the images were scaled, they do not give such a good result. The SIFT method is able to produce good results in this case.

SIFT :

Scale invariant feature transform.



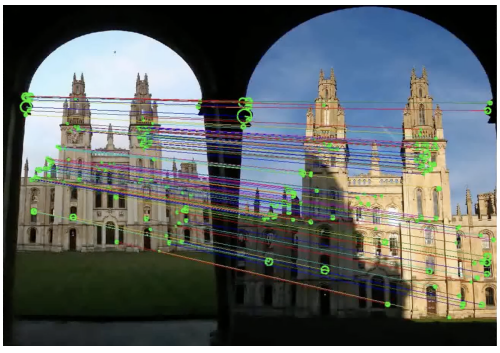
We apply a mathematical transform to the features we obtained. These are feature vectors. We match these feature vectors that we obtained.

Feature Descriptor :

Representing the selected points.

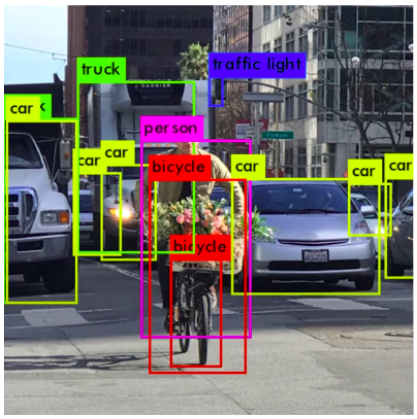
- Discriminative : descriptors should be sufficiently rich to describe two different points differently.
- Efficient : so, we use it in different applications
- Invariant : should be invariant
- Capture local geometry and appearance : it should Capture local geometry and appearance and variations in that region.

Image matching example :



Object detection :

We also want to know about the class the object belongs to. In general case many objects overlap with each other :



Template matching and sliding window :

Image

Template

We head a template (a small image that depicts one object that we actually wanted to detect) which shows the object in the exact same position where we actually expect to see it in the test image. We match using the sliding window approach. For every position we evaluate how much do the pixels in the image and template correlate. Placing the template in the exact position where the object is gives "*high correlation*".

When another object overlaps this object this results in a low correlation even though the image is the same.

Problems :

1. Occlusions : we need to see the whole object
2. This works to detect a given instance of an object, not a class of objects (cannot detect if pose or appearance changes).
3. In real life, objects have an unknown position, scale and aspect ratio, the search space is searched inefficiently with a sliding window (high computational complexity).

Appearance and shape changes

Pose change

Solution :

So, we select some patches and try to match with these. But all of these may not be important, we need to find these important patches to match to reduce computational complexity. For this we extract the features using the feature detection techniques.

Viola-Jones detector :

This is a feature extraction and classification method.

1. Learning multiple weak learners to build a strong classifier.
2. i.e., make many small decisions and combine them for a stronger final decision.

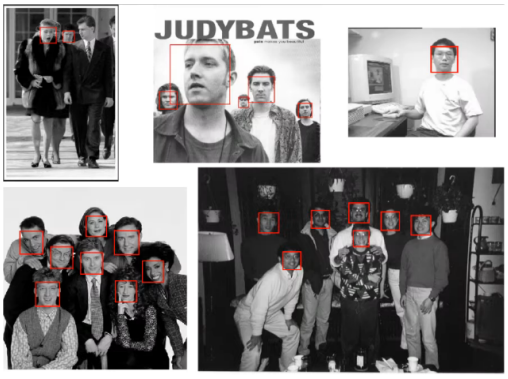
Eyes are darker compared to the rest. This pattern fits pretty well.

Steps :

1. Select your Haar-like features
    a. Hand crafted features

2. Integral image for fast feature evaluation
    a. I can evaluate which parts of the image have highest cross-correlation with my feature (template)
3. AdaBoost to find weak learner
    a. I cannot possibly evaluate all features at the test for all image locations
    b. Learn the best set of weak learners
    c. Our final classifier is the linear combination of all weak learners.

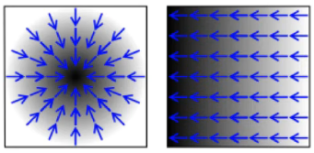
Example :



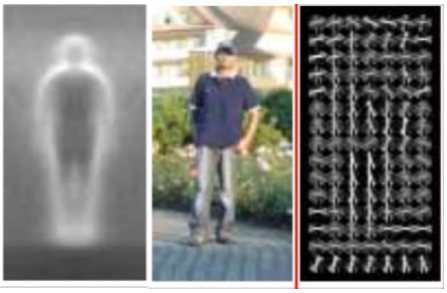
Histogram of Oriented Gradients detector (HOG) :

For a better detector, we can either improve features or classification. A new set of features by histogram oriented gradients, we take the average gradient image over training samples to represent the shape information. In HOG, feature vector is formed by

Gradient Magnitude + direction

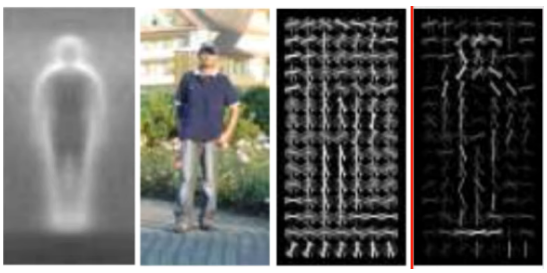


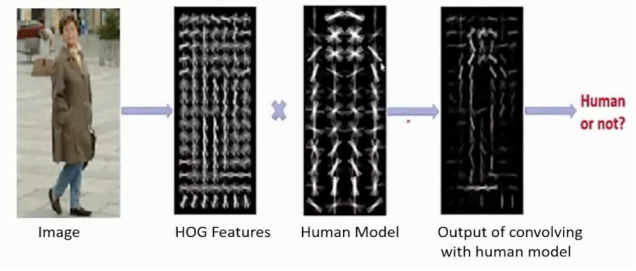
Feature Extraction :



1. Take an image
2. Compute gradients in dense grids
3. Create a histogram based on gradient detection.

Feature Classification :

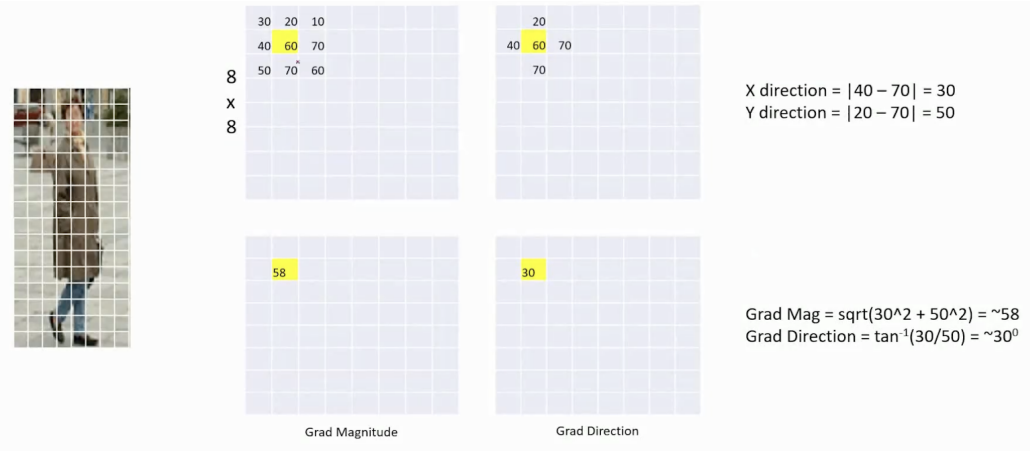




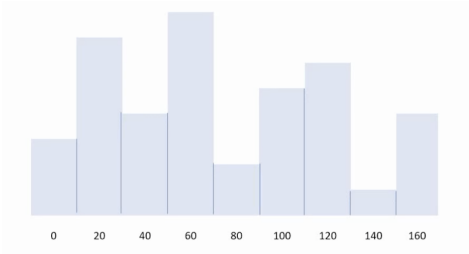
1. Choose your training set of images that contain the object we want to detect.
2. Choose a set of images that do not contain that object
3. Extract HOG features on both sets.
4. Train an SVM classifier on the two sets to detect whether a feature vector represents the object of interest or not (0/1 classification - Binary)

HOG Feature Vector Calculation :

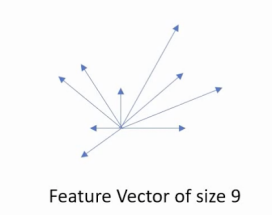
Take an 8 × 8 pixel matrix and calculate Gradient Magnitude and direction :



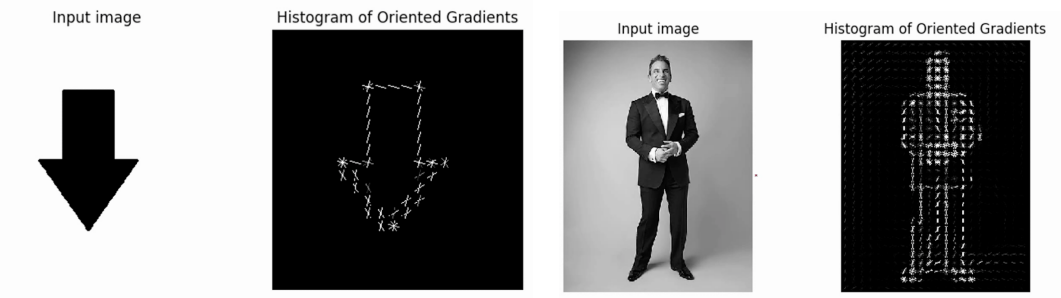
Form a histogram for gradient magnitude of gradient magnitudes based on gradient angle :



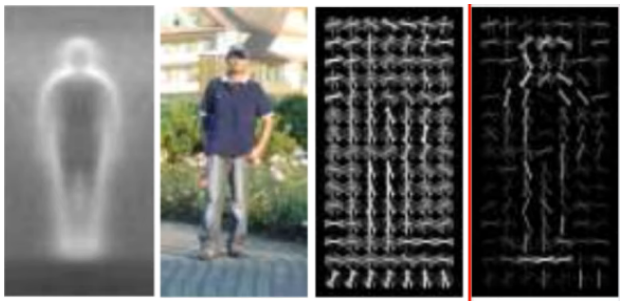
Add all the gradient magnitudes corresponding to each bin in the histogram. Form an array. 0-180 -> 20^0 bins => 9 vectors in a feature vector. Pictorial representation of each array is



Example :

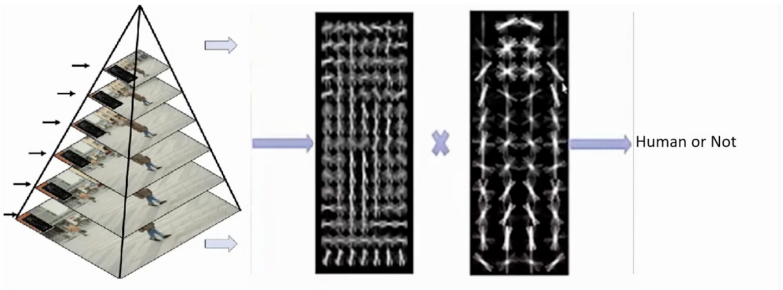


We notice the shape of the person in the HOG features weighted by SVM classifier.



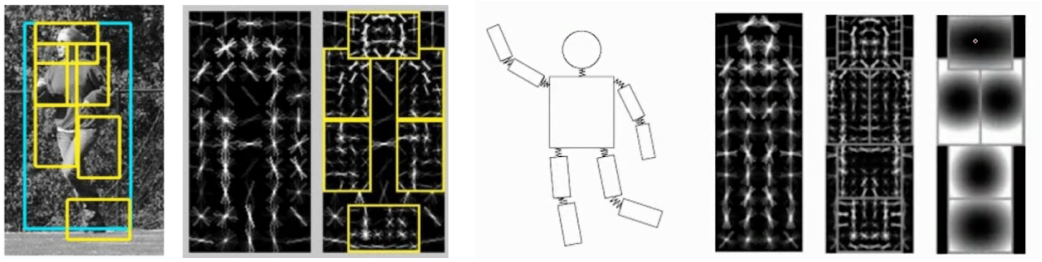
Object Detection using HOG+SVM :

For this we use sliding window technique. Use the sliding window technique for the image. Verify if the object in the patch is human or not by extracting HOG features of the patch and pass it to the classifier. But the window size may not capture the complete object. For this we use an image pyramid.

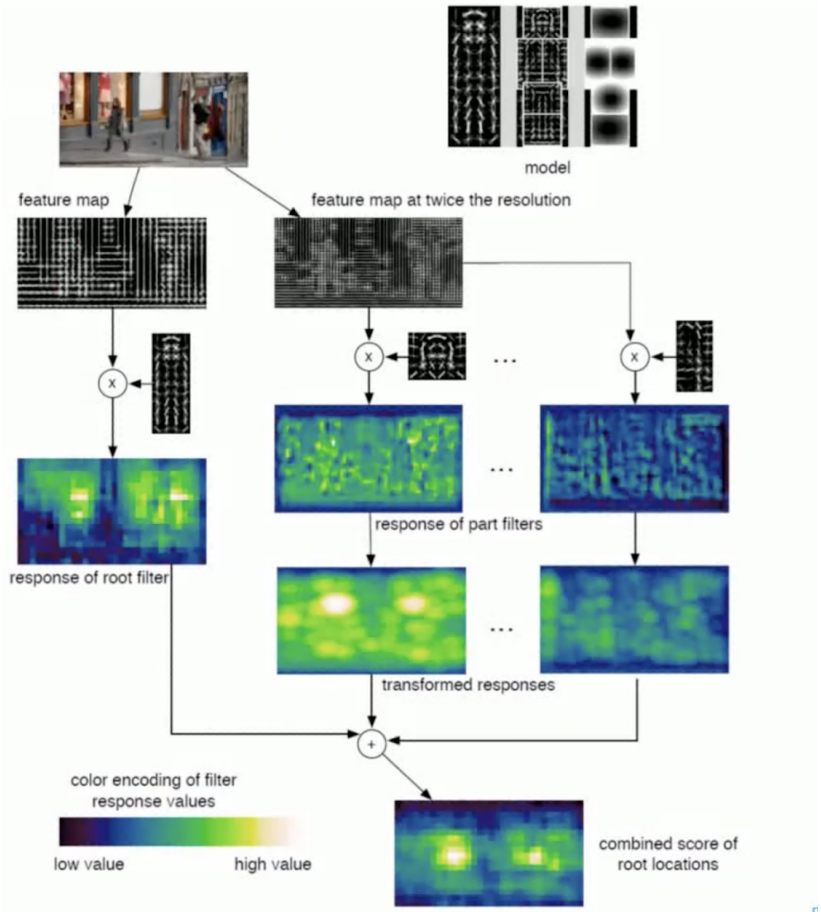


Deformable part model :

This is also based on HOG features, but based on body part detection i.e., it is robust to different body poses. They combine all the scores obtained for separate parts and classify. They use penalty score to check if the body part is in the ideal place where it has to be.



Procedure :

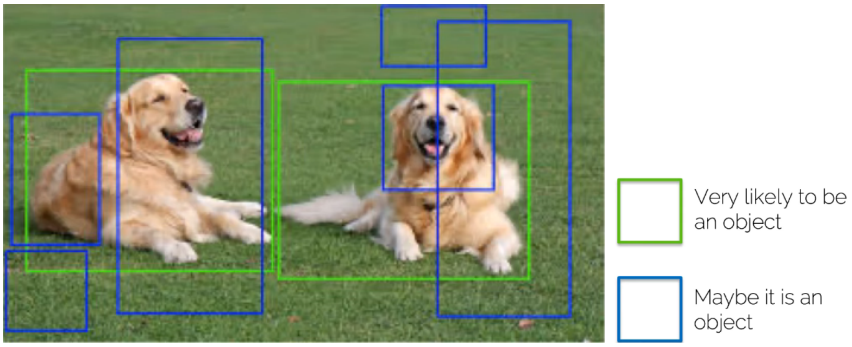


Points to be noted :

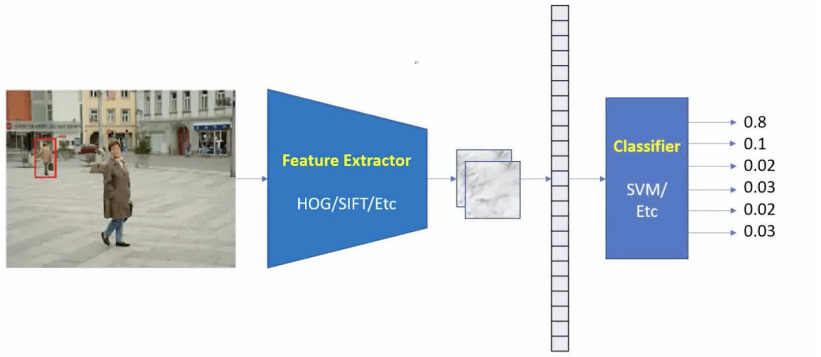
1. SIFT doesn't need a sliding window, because sift works by finding interest points. (Feature extraction is of two types : key/interest points & dense sampling)
2. HOG fails to detect when the object is at a different angle. We can overcome this by training SVM for different poses.
3. SIFT is a sparse feature detector whereas HOG is a dense feature detector (sliding window)

General object detection :

We need a generic, *class-agnostic* objectness measure : how likely it is for an image region to contain an object. Using this measure, yields a number of candidate *object proposals* or *regions of interest* (ROI) where to focus and use a classifier on top of it. (differentiate if it's an object or which class of objects it belongs to).



Object Detection Pipeline :



Note :

Many boxes try to explain an object. We need a method to keep only the "best" boxes. We do this using " non-max suppression".

Non-Maximum Suppression :

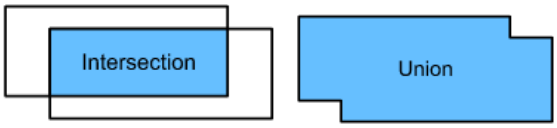
Algorithm :

1. Start with an anchor box i
2. For another box j, that overlaps i. If they overlap more than a threshold(hyperparameter).
3. Discard box i if its score is less than j (confidence score depends on the task).

Region overlap :

We measure overlap with the intersection over union (IoU) or Jaccard index.

$$J(A,B) = \frac{|A \cup B|}{|A \cap B|}$$



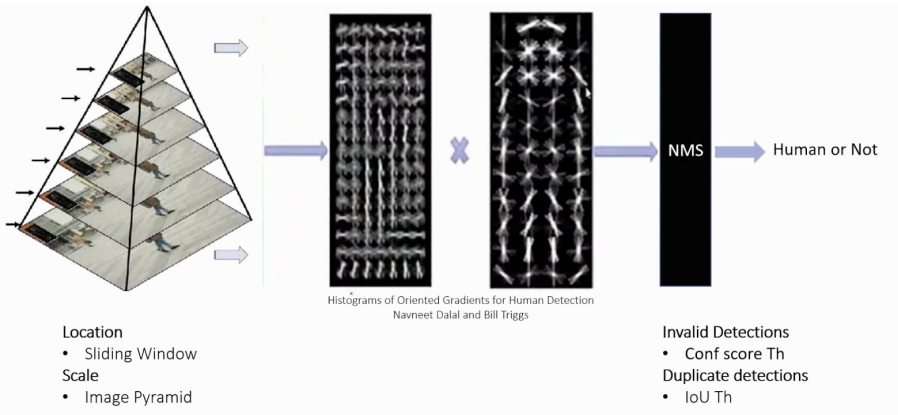
J(A,B) ≈ If intersection and union are almost the same, else if the jaccard index is low most likely they might not depict the same object.

Threshold :

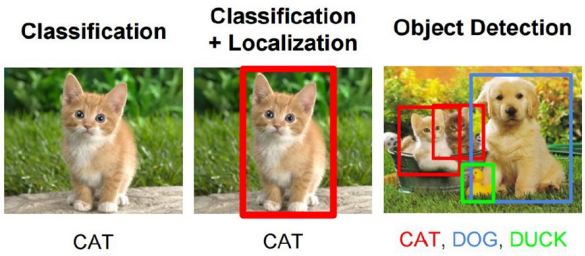
Consider a very crowded scene (heavy overlap between the objects), if two objects overlap with J 0.7 and if the threshold is set to 0.6 we might not at all represent these two objects separately.

Low threshold leads to False positives (high sensitivity and low specificity), high threshold leads to False negatives (lo sensitivity and high specificity) .

NMS in Objection Detection using HOG and SVM :



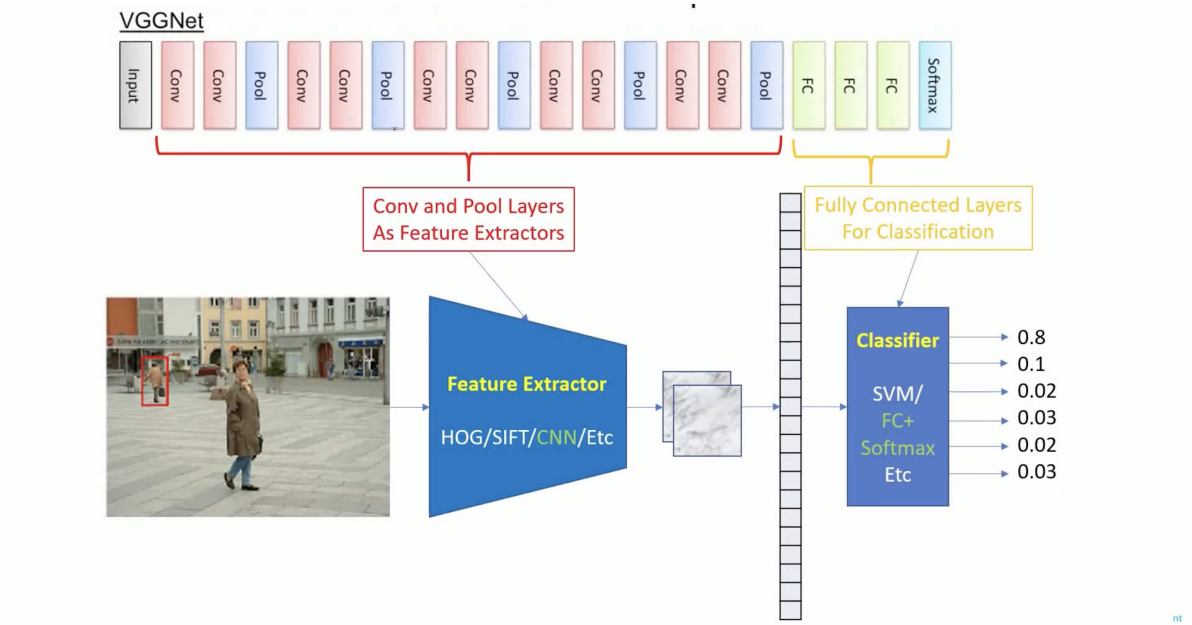
Introduction to CNN based Object Detection :



Note : Go through the CNN.pdf in this repo

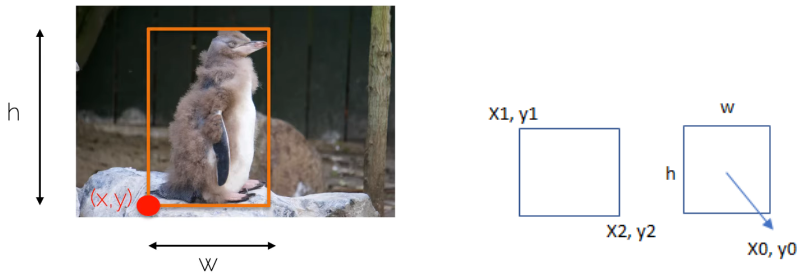
Classification :

Given an image with one object, we tell what kind of object it is. This is the case of classification.



Localization :

Draw the bounding box around the object. We take an image and find all the interesting objects in it. We represent an object by *bounding box* and *the class*. We want to represent the penguin using a rectangle (*bounding box)* that fits tightly the object inside.



Bounding box is represented by four values (x,y,w,h) where (x,y) is the bottom left corner in pixel space and h is height w is the width of the bounding box. For Localization, to get the bounding box, we need 4 outputs per class.

Bounding Box regression :

We just need to modify the last Fully Connected layer of the network to output 4 values for each class instead of 1.

We extract features using CNN and using fully connected layers we extract the bounding box coordinates. We here use another set of box coordinates that predicts the class scores for the object found on his image.

We train bounding box coordinates using L2 loss (regression head) and class scores using softmax loss (classification head). Here, we are interested in the absolute values of the coordinates and not the probabilities, we would not be needing the Softmax layer at the end. we need to change the way the loss is calculated. In case of classification, we use log loss, but here we need to use any of the Regression Loss functions like L1 or L2 loss. With these changes, we need to train the network to output correct values for these coordinates. But, this does not mean that we can throw away the Classification layer. We still need it to know what kind of object it is. So, by combining the results of both Classifier and BBox Regressor, we can infer the type of the object and its location.

Note :

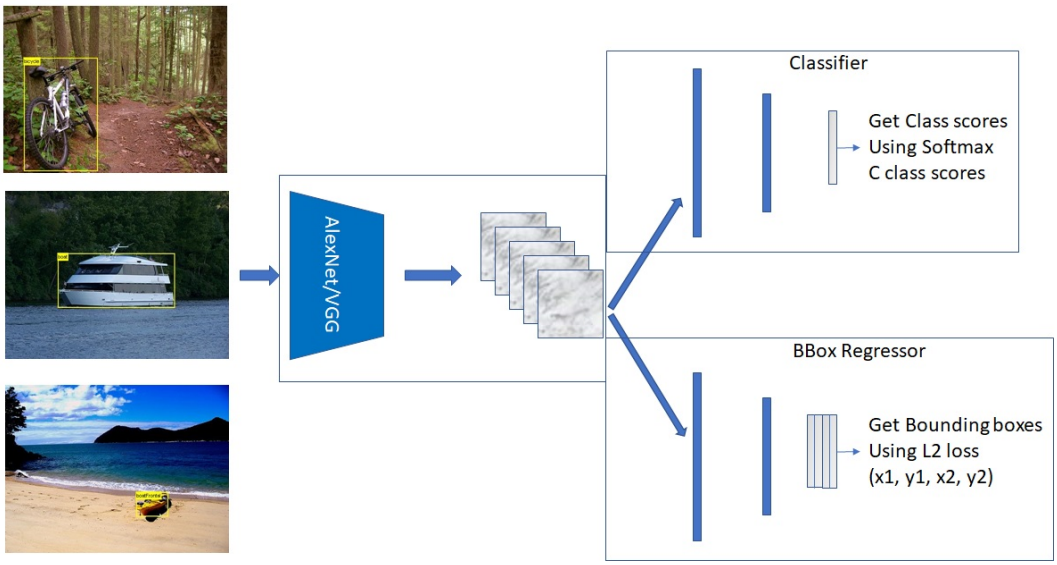
1. It was typical to train the classification head first, freeze the layers.
2. Then train the regression head
3. At test time, we use both!

We can train multiple heads at a time by balancing different losses.

Note:

1. The Convolution Networks like AlexNet, VGGNet etc need input images to be of fixed size like 224x224.

Final localization network :

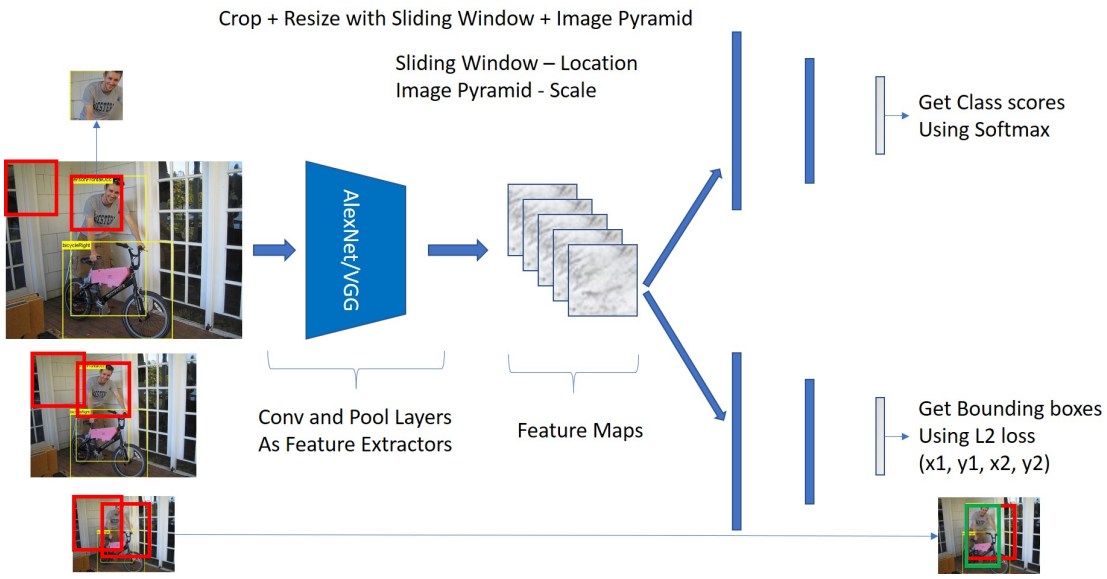


Note :

Irrespective of the object/present or not, our network blindly does all the calculations. But we use confidence scores to know if the classification makes sense or not.

Object Detection :

If there are more than one object in the image, we have to locate and identify all of them. This is Object Detection. If there are multiple objects we use the sliding window for all possible locations and feed it to the localization network. If they are of different sizes, we perform some preprocessing in the input side. We use an image pyramid.



Sliding window :

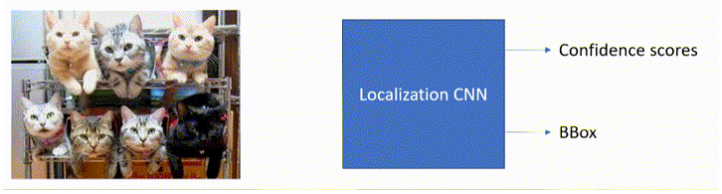
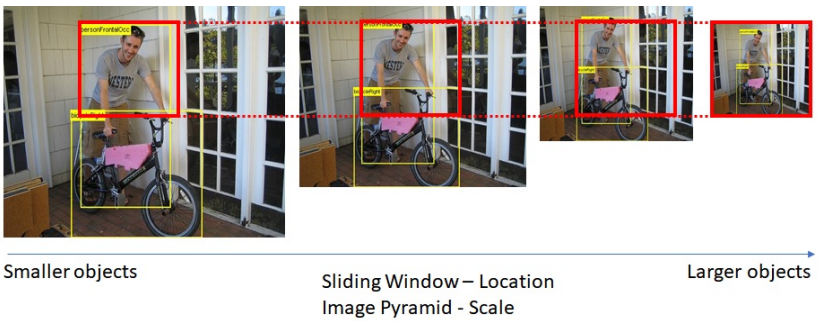
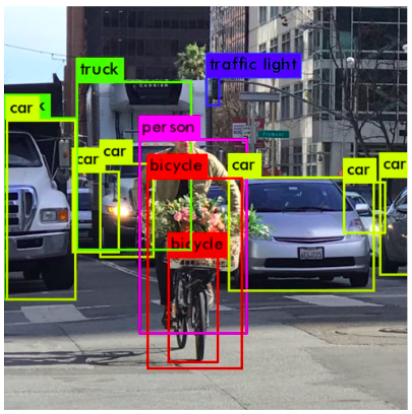


Image pyramid :

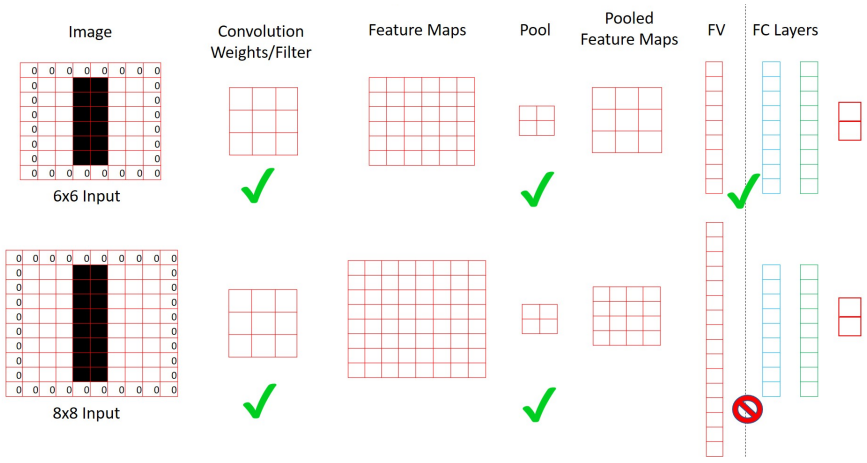


Example :

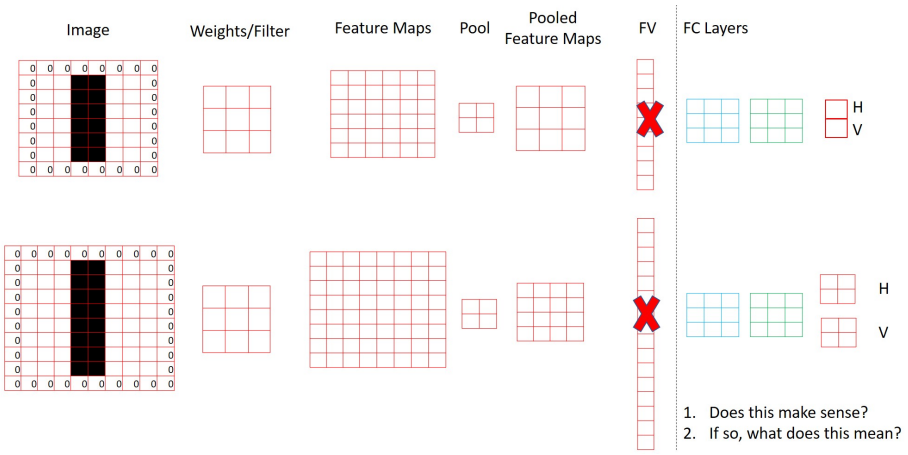


ConvNet's input Constraints :

using the Sliding window technique and taking the crop of the image at different locations, I am ending up with too many inputs to my Localization network. Because the ConvNets expect fixed sized image inputs. Convolution and pooling operations can be done on inputs of any size. Restriction is from the Fully Connected layer operations are done using 'Dot Product' of vectors. And when you are taking dot products, the size of input and size of the filter should be the same. Else, the operation fails.

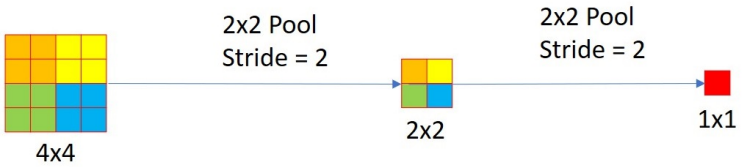


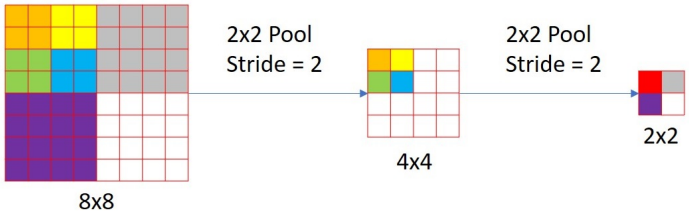
we can implement the FC layer as a Convolution operation instead of dot product. Since convolution operations have no size constraint, this will remove the fixed size restriction. Instead, what you can do is, take the Pool layer output without flattening. This will be in the form of a mxn matrix. Take the same filter used for FC operation and represent it as a matrix of mxn dimension. Now, if you convolve the Pool layer output Feature Map with this filter matrix, you will get the same scalar output as that of the dot product.



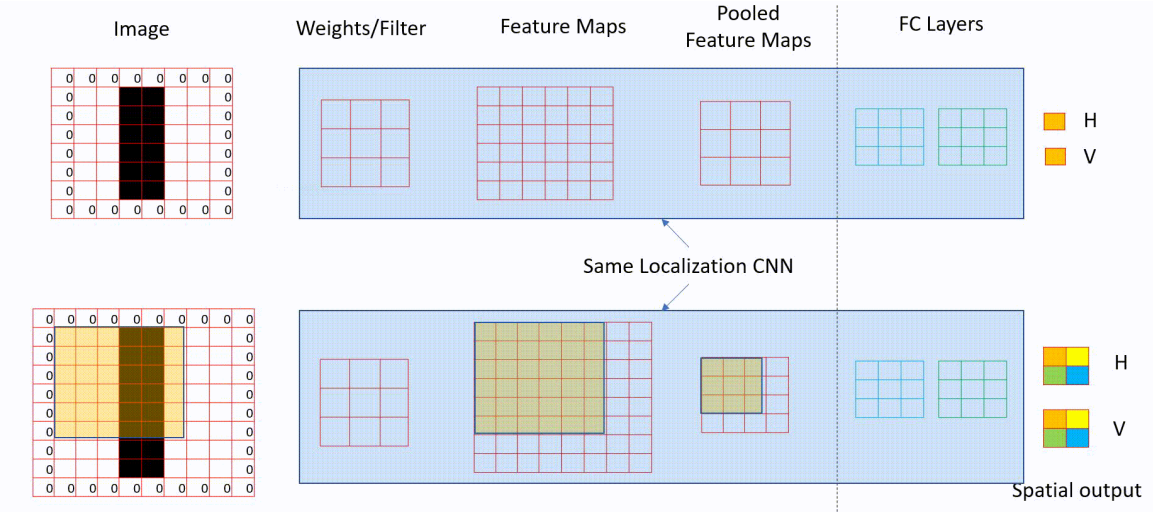
Receptive field :

If we get a 1x1 output from a 4x4 patch of the image, the receptive field of the network is 4x4. That is, each pixel in the output encodes information from a 4x4 patch of the image. If we run a 8x8 image through the same network, we get a 2x2 output.





Every value in the output encodes information from some 4x4 patch of the image. This 2x2 output is Spatial Output. Receptive Field of the network is 6x6 as can be seen as :

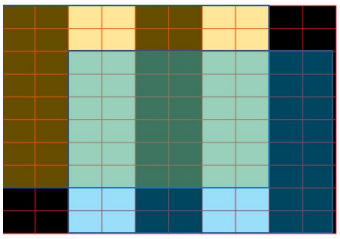


This represents computations on different portions of the image. With Spatial Outputs, we can detect different objects at different locations of the image. Below figure shows a 2x3 Spatial Output for a sample image. For 2x3 it can predict up to 6 objects.

ConvNets sliding window efficiency :

Not only is the Convolution Operation in the FC layers convenient, it is also efficient, since we are using 'Sliding Window' technique. This is because, using Sliding Window technique avoids repeated computations, which we would have incurred if we had taken the image crops.

the convolution operation is applied to the entire image at once, this avoids repeated computations in the overlap region.



Overfeat :

Sliding Window + Box Regression + Classification

FC as Conv + image pyramid + Box Regression + Classification

Sliding window :

Use many sliding window locations (FCa s conv) and multiple scales (to detect, small and large objects at a time).

Procedure :

1. Convolution operation for the entire image at once.
2. This gives a set of outputs that give a high target.
3. Combine them for final detection (in overfeat we use greedy method).

We use NMS in current object detectors.

Dealing with any image size :

Fully connected layers prevent us from dealing with any image size, they expect a fixed size as an input. By converting the FC layers into convolution operation, we have removed the fixed input size constraint.

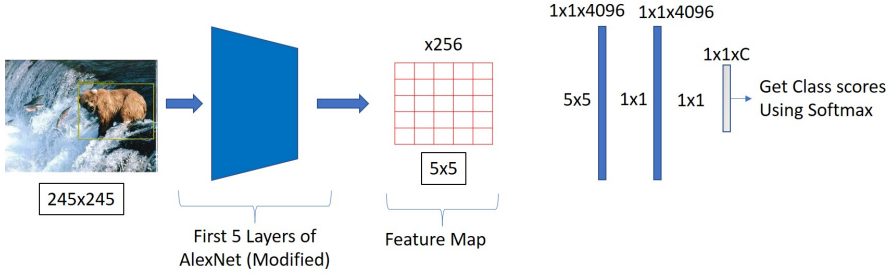
We can use the same localization network, without using the Sliding Window crops at different locations. Since there is no input size constraint, we can use the Image pyramids which results in the Spatial Output (gives object detections at different locations without image crops).

Multiple objects :

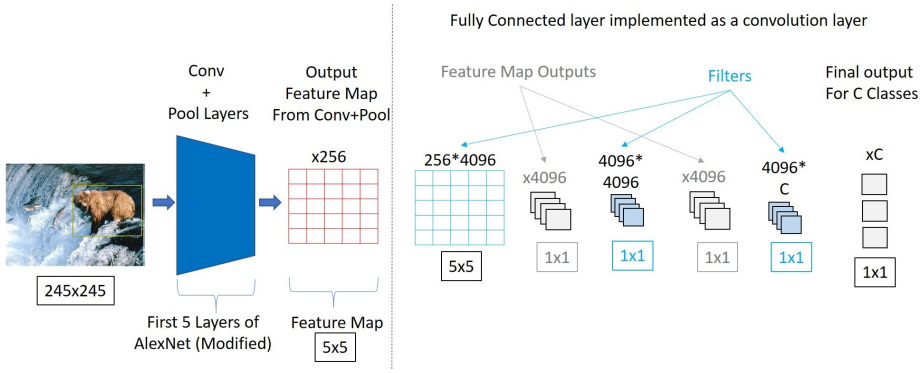
Due to spatial output this gives us multiple predictions. We avoid repetitions by controlling threshold in NMS

Classification :

This is the Overfeat Classification Network. It uses a modified AlexNet architecture for Convolution operations.The first FC layer has a 5x5 filter and the 2nd and 3rd have 1x1 filters. The depth at both layers are 4096 each.



to get a Feature Map output of depth 4096, at the 1st FC layer, we need 256*4096 filters. And each filter is of size 5x5. 256 - Since, the depth of AlexNet Conv layers is 256. at the 2nd FC layer, we need to have 4096 filters, each of size 1x1



Note :

In Overfeat, C = #classes + 1 (extra 1 is for background class i.e., no object is present)

1 X 1 filter advantages :

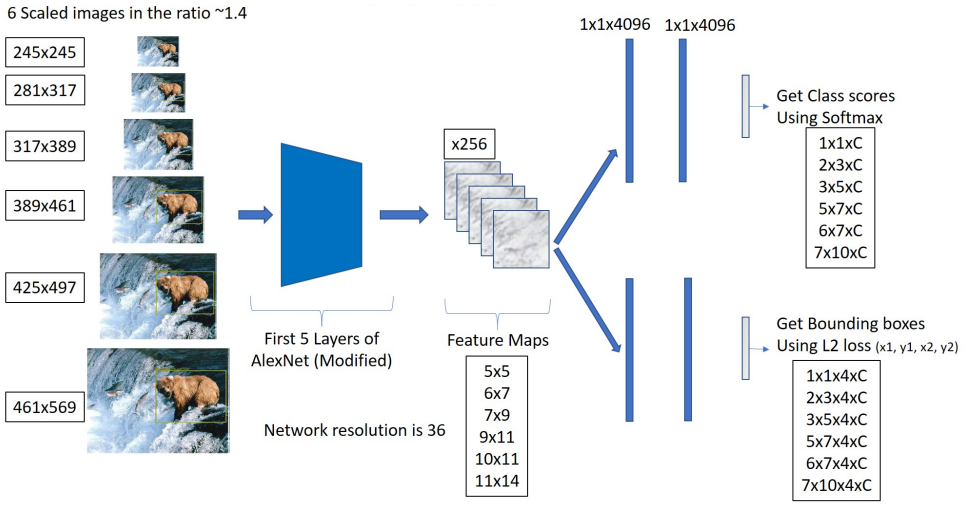
Input and output size is the same for 1x1 convolutions. the advantage of 1x1 convolutions becomes significant when we take larger images. FC as 1x1 convolutions - Model size remains the same irrespective of the image size.

Effective stride :

Effective Stride basically tells you, by how many pixels you are shifting your focus at the input side if you move by 1 pixel in the Spatial Output

Ideally your Effective Stride should be as low as possible, to ensure you are scanning all possible areas of the image. Effective Stride of Overfeat network is 36. Overfeat uses a 3x3 pool with stride of 3 and 0 padding in the last pooling layer. We can reduce effective stride to improve accuracy.

Detection Network :



Post processing :

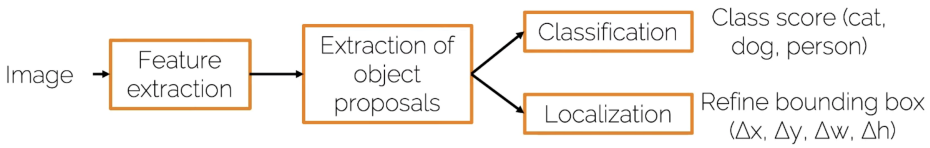
Use a threshold to discard the detections with low confidence scores. But this may result in multiple detections. This can be avoided by using NMS or by greedy method.

Types :

Object detectors are classified into two types :

- One stage object detector
- Two stage object detector

Two stage object detectors :



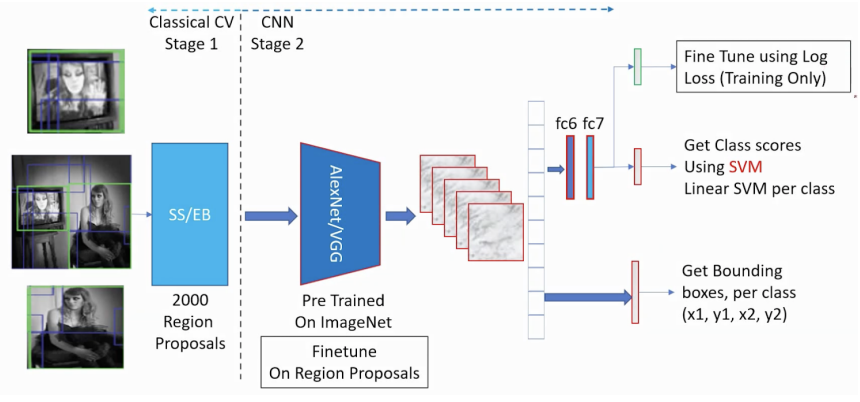
They first extract the features, then they extract object proposals (extract interesting regions) . Classification and Localization is performed only on these.

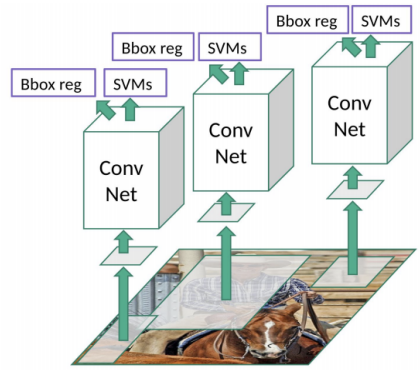
- R-CNN, Fast R-CNN, Faster R-CNN
- SPP-Net, R-FCN, FPN

It is expensive to try all positions, scales and aspect ratios. So, we work only on the interesting subset of boxes known as region proposals and work only on these. This is what happens in two stage object proposals. We can extract these using multi scale saliency (works based on FFT), color contrast, edge density detection, super pixel straddling (keep combining regions until the entire region becomes a single region, we get many levels of super pixels), selective search, edge boxes,....

Refer : https://youtu.be/1MthLfi8HnU

R-CNN :





Procedure :

1. Extract region proposals from the image (~2k) (eliminates false positives)
2. Warp them to a fixed size.
3. Apply CNN to extract features.
4. Further use bounding box regression to refine the bounding box locations obtained from region proposals using CNN.
5. Classify these objects using a classifier

Initially bounding box regressor was not added on the top, because we already obtained them from region proposals

Training scheme :

- Pre-train CNN on ImageNet.
    - ImageNet is finetuned on region proposals because warping distorts the aspect ratio of the image.
    - For fine tuning Softmax classifier (log loss)
    - This updates weights in both FC layers and Convolution layers
- Finetune the CNN on the number of classes the detector is aiming to classify (softmax loss)
    - Initially a softmax classifier was used.
    - For fine tuning region proposal networks.
- Train a linear Support Vector Machine classifier to classify image regions. One SVM per class! (hinge loss)
- Train the bounding box regressor (L2 loss)
    - So, we don't need FC6 and FC7 layers. Accuracy did not change without these.
    - After fine tuning on Region proposals FC6 did play an important role. (weights are being updated in the FC layers due to finetuning)

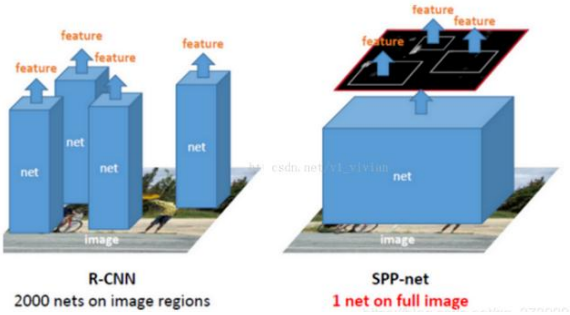
PROS :

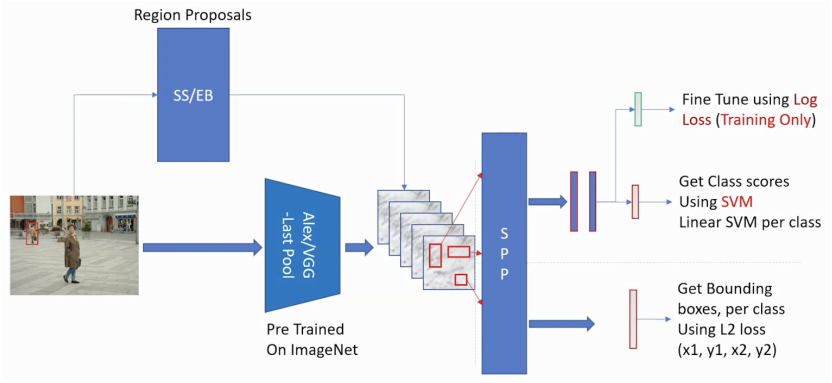
- The pipeline of proposals, feature extraction and SVM classification is well-known and tested. Only features are changed (CNN is used instead of HOG for feature extraction).
- CNN summarizes each proposal into a 4096 vector (much more compact representation compared to HOG)
- Leverage transfer learning : the CNN can be pre-trained for image classification with C classes. One needs only to change the FC layers to deal with Z classes.

CONS:

- Slow! 47Seconds/image with VGG16 backbone. One considers around 2000 proposals per image, they need to be warped and forwarded through CNN. Repeated computations for the overlapped regions between region proposals.
- Training is also slow and complex.
- The object proposal algorithm is fixed. Feature extraction and SVM classifier are trained separately → not exploiting learning to its full potential. So, we cannot improve the object proposal algorithm using CNN.
- Accuracy is low due to the region proposals are being warped and it gets difficult to learn

SPP -Net :





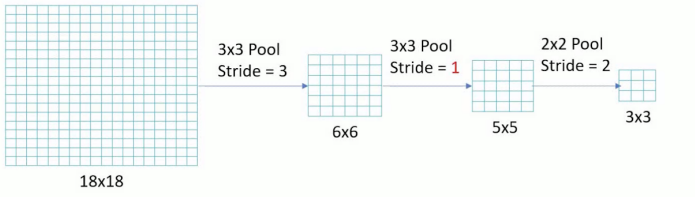
Instead of using multiple CNNs for each region proposal, we use one ConvNet which analyses the whole image. On top of that we extract proposals and compute features from those. This avoids repeated computations for the overlapping regions.

The neural net that extracted features is frozen.

Translate RoI proposals to the feature map

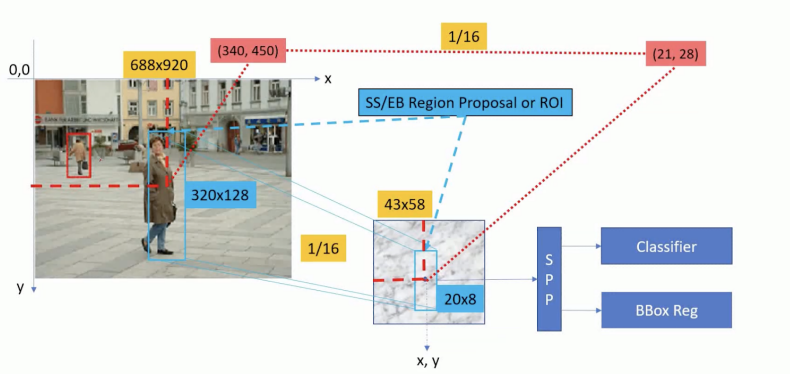
RoI Projection :

We use subsampling to find RoI projection of RoI



For this 18x18 image is sampled down finally to 3x3. So the subsampling ratio of height and width is 1/6. Shifting one pixel in the output is just the same as shifting 6 in the input.

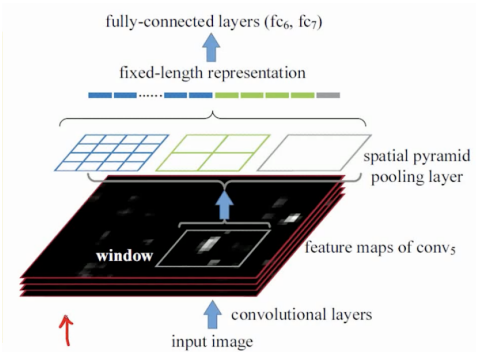
Example :



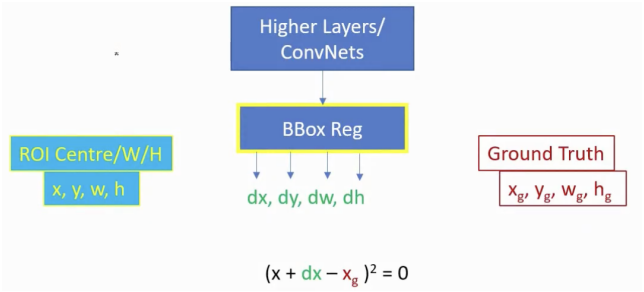
SP Pooling :

Apply all the levels of Spatial Pyramid pooling on the top of the region proposal feature map.

If the region of Interest is very small then it is difficult to place a large grid. For this, we enhance the image size. This has to be done to all the 256 feature maps. (instead of applying SPP on entire feature map, we are only doing it on small portion)



Bounding Box Regression training :



We use L2 Loss to train BBox Reg. dx, dy, dw and dh are the outputs of the bounding box regressor.

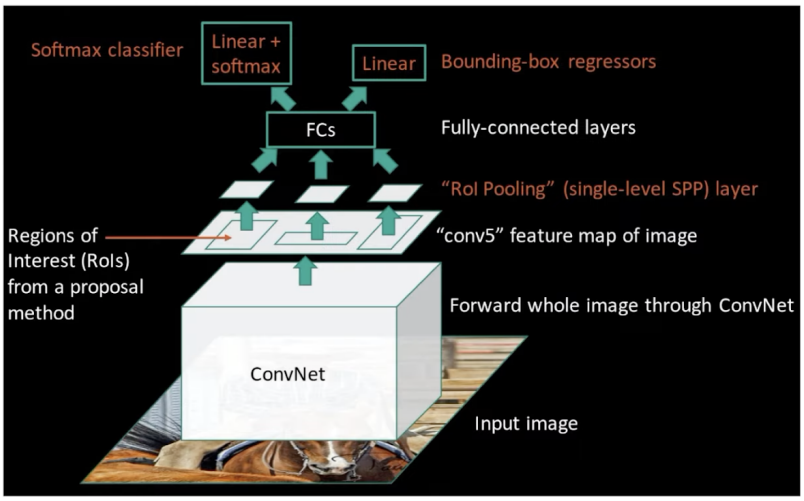
Pros and Cons :

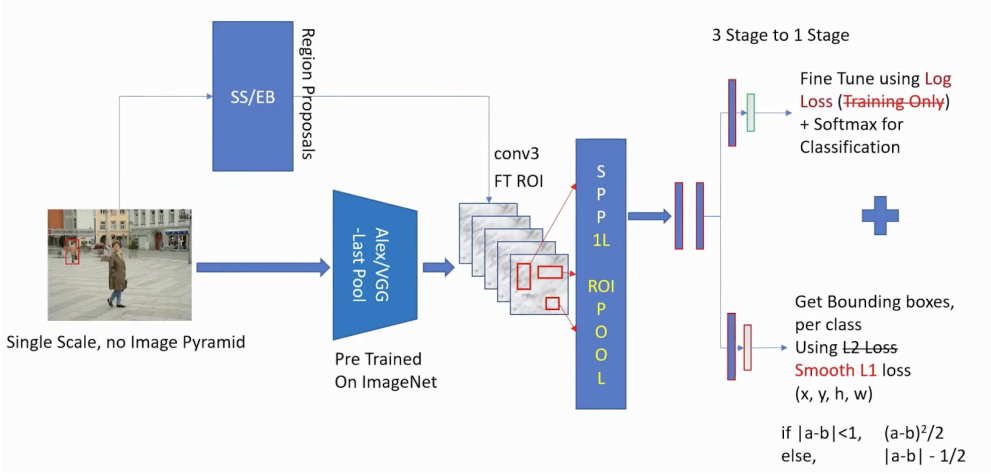
- It solved the R-CNN problem of being slow at test time

It still has some problems inherited from R-CNN:

- Training is still slow (a bit faster than R-CNN) - Training scheme is still complex
- Still no end-to-end training.
    - Neural net is frozen
    - No back propagation.

Fast R-CNN :





1. Only one forward pass instead of multiple forward passses like in R-CNN.
2. Extract Region Proposals
3. Project the region proposals on to the feature map.
4. Pool the RoI proposals from the feature map (Warping them into a fixed size is done here).
5. Use both the losses and back propagate.

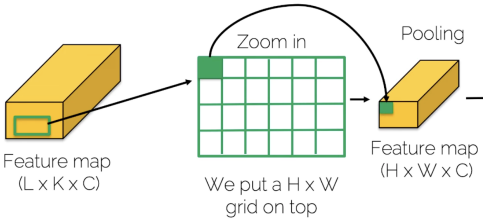
Note :

1. Fine tuning of RoI proposals is only done from Conv3 layer (complex shapes are learnt from layer 3).
2. Two more FC layers are added before the classifier and BBox Reg unlike SPP Net.

RoI Pooling :

Unlike SPP Net this only has one level of pooling.

Apply all the levels of Spatial Pyramid pooling on the top of the region proposal feature map.



We place a H X W grid on the region of interest and do the max pooling on each block of the grid.

Back propagation :

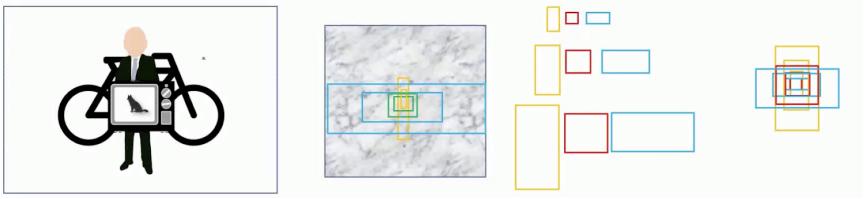
We pass on the gradient to the maximum location. We back propagate by combining both classification loss and BBox reg loss and finetune the network.

Region of interest from the feature map is sent into the classification layer. Model is fine tuned. Now, BBox Reg is fine tuned (back propagation is a 2 step process here, in RCNN,SPPNet it is a 3 step process). But back propagating by combining both losses improved accuracy.

Update :

1. Has to give <2000 proposals
2. As fast as SS or better.
3. As accurate as SS or better.
4. Should be able to propose overlapping RoIs with different aspect ratios and scales.

Results by SS :



Note :

We found out that at any point there can be objects that can be of different aspect ratios and scales.

Solutions :

Fast RCNN + image pyramid + Sliding window :

From SPP Net and fast RCNN we know that the image pyramid slows down the network.

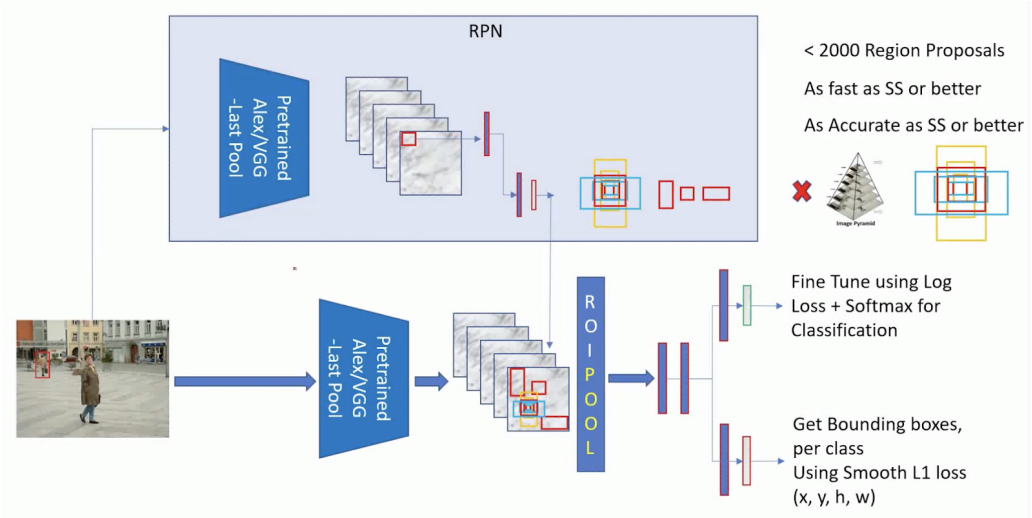
Fast RCNN + Feature Pyramid :

This works on the principle that at every location is possible to get 3*n (n is for different scales - usually 3) RoI proposals. We use 9 different sliding windows to pool the features. This results in way too many RoI proposals and ends up being time consuming.

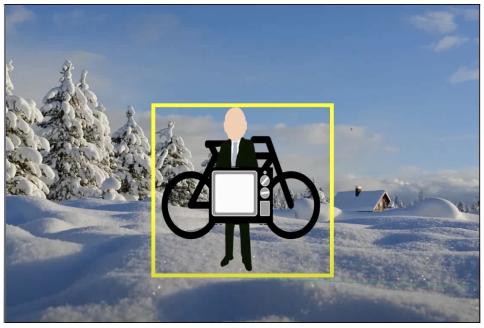
Fast RCNN + Neural Network :

We can use Overfeat (without classifier & NMS). This gives less #RoI proposals and satisfies all our conditions to replace SS. but Overfeat uses Image pyramid. Instead we can just use CNN bounding box regressors.

The bounding box regressor that we used in Fast RCNN is just to finetune with the reference we had (relative technique). Absolute BBox Reg can be used since we had no reference to start with (refer to BBox Reg in Localization). We can also use sliding window as reference and finetune (relative BBox Reg)



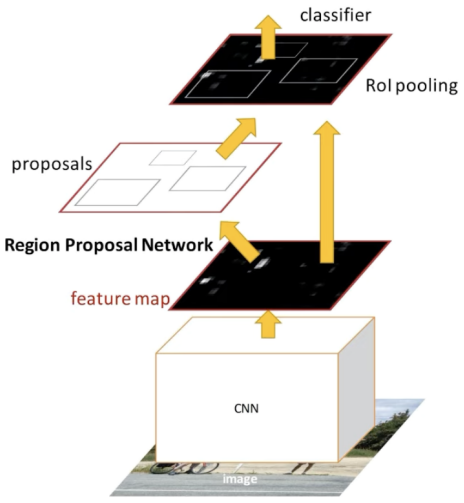
If the objects were not overlapping, either of these techniques could be used. But if the objects are overlapping, both these techniques would not work as it is. Since they will end up fitting the most dominating object in the foreground.



But this requires 3 different bounding boxes.

1. Absolute : all 3 reg converge to same object
2. Relative : implemented in faster RCNN

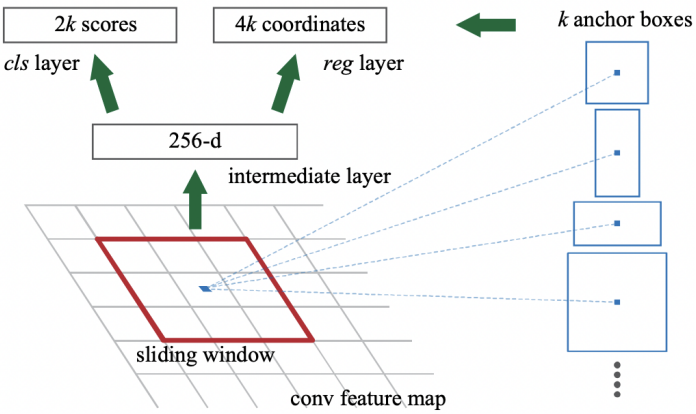
Faster R-CNN :



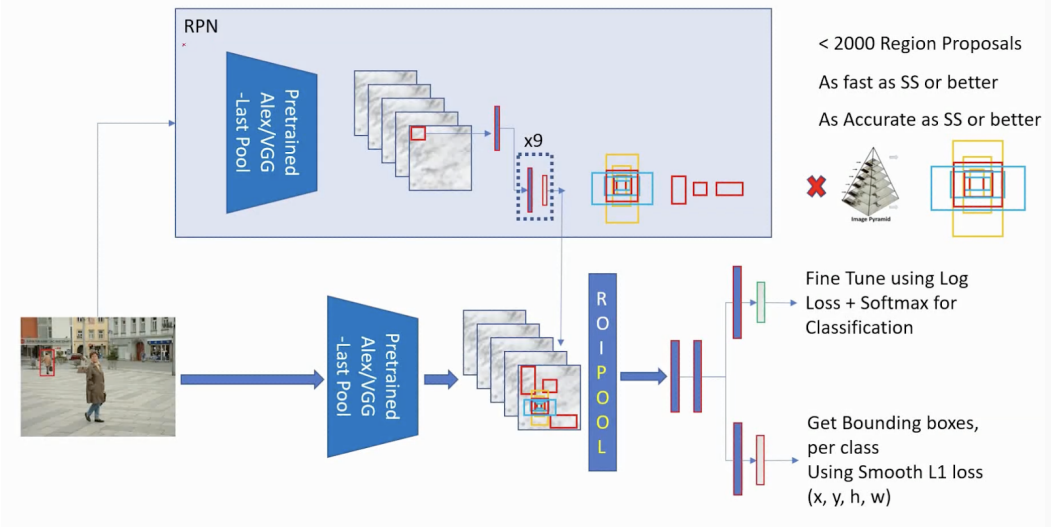
They made the proposal generation integrated with the rest of the pipeline. Region Proposal Network (RPN) trained to produce region proposals directly. After RPN, everything is like Fast R-CNN

Relative (BBox reg kinda)

Region Proposal Network :

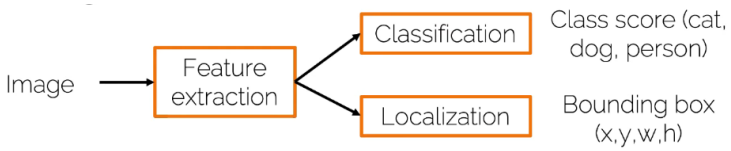


Anchor Boxes :



Reducing #RoI Proposals :

One stage object detectors :



Feature extraction + Classification + Localization

Feature extraction from the image is through CNN's. Localizing the objects with a bounding box and assigning a class happens in one stage.

Example :

- YOLO, SSD, RetinaNet
- CenterNet, CornerNet, ExtremeNet (point based detectors)

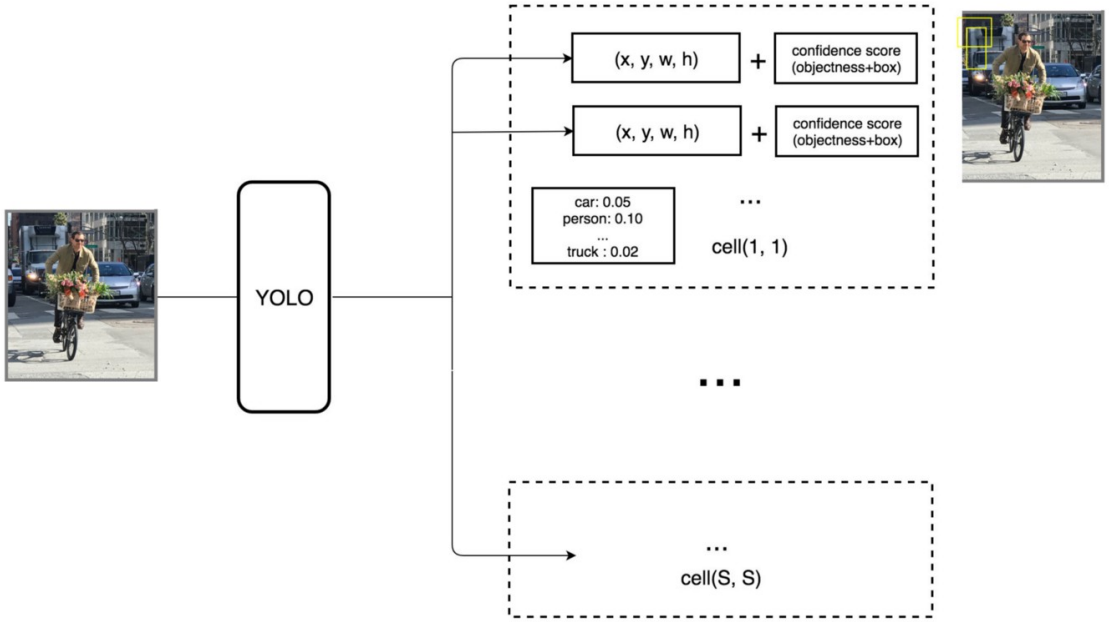
These are incredibly fast.

YOLO (You Only Look Once) :

YOLO V1 :

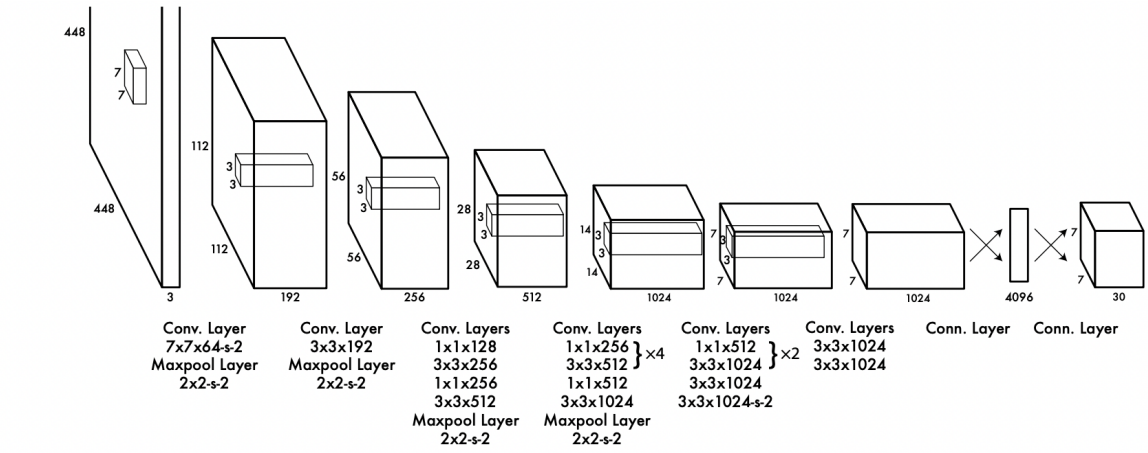
Recall sliding window object detection. To make it efficient, we will "slide our window" only on certain locations of the image by dividing the image using a grid. Classifier is used on these particular locations.

All of these models were first pre-trained as image classifiers before being adapted for the detection task.

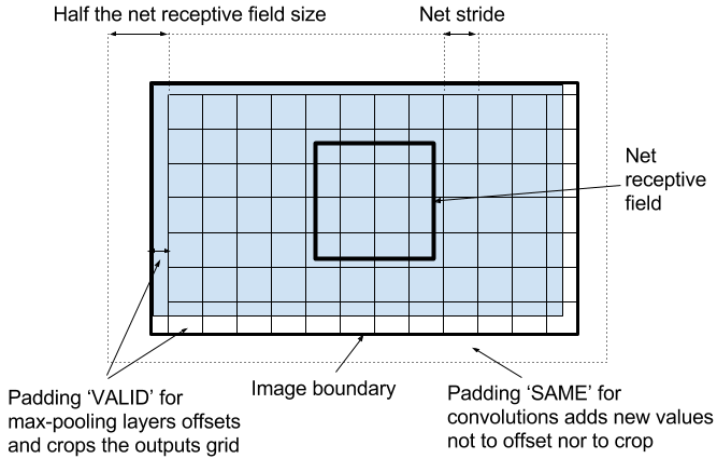


1. We divide our image in a grid and evaluate only on these locations.
2. The first iteration of the YOLO model directly predicts all four values which describe a bounding box.
    a. Predicts height and width as √height and √width because here they used normalised h and w (lies between 0 and 1) small deviations in large boxes matter less than in small boxes.
3. We place this box at the centre of each cell in the grid, and this will be our initial guess for that object.
4. We pass all these blocks of the image through the DarkNet.
    a. DarkNet-19 which follows the general design of a 3×3 filters, doubling the number of channels at each pooling step; 1×1 filters are also used to periodically compress the feature representation throughout the network.

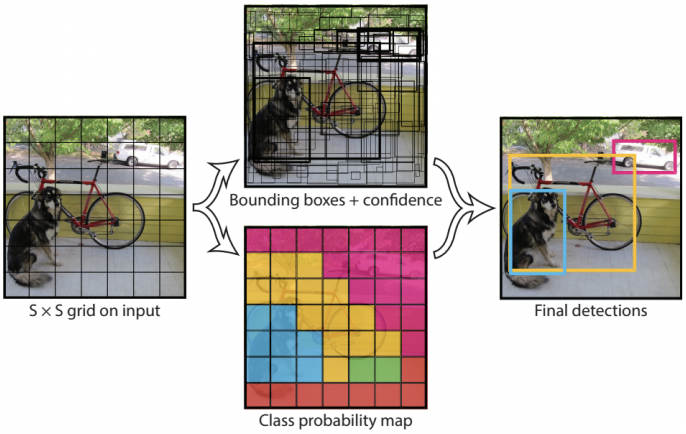
b. Has 3 × 3 filters and doubles the number of channels after every pooling step. Network also uses global average pooling to make predictions as well as 1 × 1 filters to compress the feature representation between 3 × 3 convolutions.



c. Each max-pooling layer divides the output size by 2, multiplies the network stride by 2, and shifts the position in the image corresponding to the net receptive field center by 1 pixel.



5. We classify and regress this box to better fit the object.
    a. Adapting the classification network for detection simply consists of removing the last few layers of the network and adding a convolutional layer with B(5+C) filters to produce the N×N×B bounding box predictions.
6. For each grid,
    a. it predicts B boundary boxes and each box has one box confidence score,
        i. Class Confidence Score = Box Confidence Score × Conditional CLass Probability
        ii. $p(\text{box cont obj}) \,.\, IoU = p(\text{object} \in \text{class}_i) \times p(\text{object} \in \text{class}_i \text{ if object is present})$
        iii. $P_r(\text{class}_i) \,.\, IoU = (P_r(\text{object}) \,.\, IoU) \times (P_r(\text{class}_i \mid \text{object}))$
        iv. It measures the confidence on both the classification and the localization (where an object is located).
    b. it detects one object only regardless of the number of boxes B,
    c. it predicts C conditional class probabilities (one per class for the likeliness of the object class).



Loss Function :

Total loss = Classification loss + Localization loss + Confidence loss

Classification loss :

If an object is detected, the classification loss at each cell is the squared error of the class conditional probabilities ($p_i(c)$) for each class:

$$\sum_{i=0}^{s^2} \left(\mathrm{I}_i^{\mathrm{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p_i}(c) \right)^2 \right)$$

$$\mathrm{I}_i^{\mathrm{obj}} = 1 \text{ if obj is in cell i else } 0$$

Localization loss :

The localization loss measures the errors in the predicted boundary box locations and sizes. We only count the box responsible for detecting the object. Yolo predicted √height and √width to not weigh absolute errors in large boxes and small boxes equally. To put more emphasis on the boundary box accuracy, we multiply the loss by λcoord (default: 5).

$$\sum_{i=0}^{s^2} \sum_{j=0}^{B} \left(\mathrm{I}_i^{\mathrm{obj}} \left[\left(\mathrm{x}_i - \hat{\mathrm{x}}_i \right)^2 + \left(\mathrm{y}_i - \hat{\mathrm{y}}_i \right)^2 + \left(\sqrt{\mathrm{w}_i} - \sqrt{\hat{\mathrm{w}}_i} \right)^2 + \left(\sqrt{\mathrm{h}_i} - \sqrt{\hat{\mathrm{h}}_i} \right)^2 \right] \right)$$

$$\mathrm{I}_i^{\mathrm{obj}} = 1 \text{ if obj is in cell i else } 0$$

Confidence loss :

If an object is detected in the box, the confidence loss (measuring the objectness of the box) is:

$$\sum_{i=0}^{s^2} \sum_{j=0}^{B} \left(\mathrm{I}_i^{\mathrm{obj}} \left[\left(\mathrm{C}_i - \hat{\mathrm{C}}_i \right)^2 \right] \right) + \lambda_{\mathrm{noobj}} \sum_{i=0}^{s^2} \sum_{j=0}^{B} \left(\mathrm{I}_i^{\mathrm{noobj}} \left[\left(\mathrm{C}_i - \hat{\mathrm{C}}_i \right)^2 \right] \right)$$

$$\mathrm{I}_i^{\mathrm{obj}} = 1 \text{ if obj is in cell i else } 0 \text{ , } \mathrm{I}_i^{\mathrm{noobj}} = 1 - \mathrm{I}_i^{\mathrm{obj}}$$

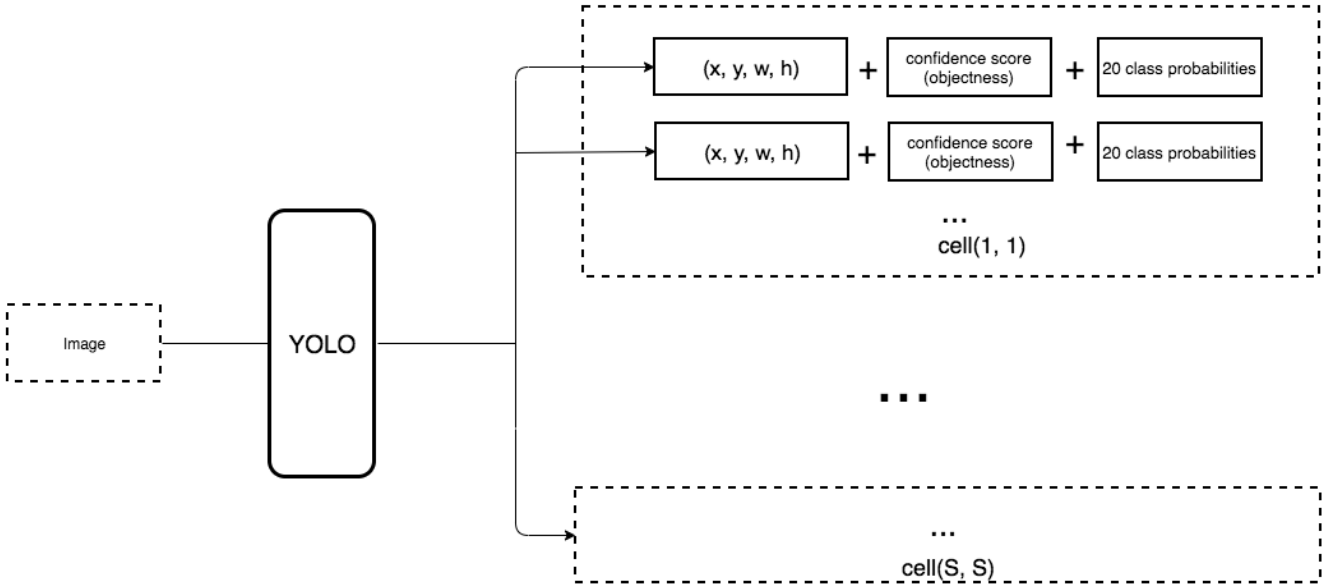
Pros :

- Fast. Good for real-time processing.
- Predictions (object locations and classes) are made from one single network. Can be trained end-to-end to improve accuracy.
- YOLO is more generalized. It outperforms other methods when generalizing from natural images to other domains like artwork.
- Region proposal methods limit the classifier to the specific region. YOLO accesses the whole image in predicting boundaries. With the additional context, YOLO demonstrates fewer false positives in background areas.
- YOLO detects one object per grid cell. It enforces spatial diversity in making predictions.

Cons :

1. It can only detect one object per grid block.
2. compared to Fast R-CNN shows that YOLO makes a significant number of localization errors.
3. YOLO has relatively low recall compared to region proposal-based methods

YOLO V2 :



Higher Resolution :

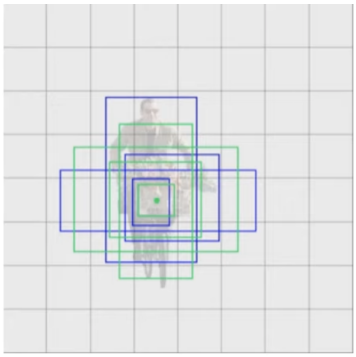
In the second iteration of the YOLO model, experimentally it was found that using higher resolution images at the end of classification pre-training improved the detection performance.

Batch Normalization :

Increase in MAP is observed when batch normalization (regularization technique, decreases the shift in unit value in the hidden layer which stabilizes neural networks) is added to convolutional layers. With batch normalization we can remove dropout from the model without overfitting.

Anchor boxes :

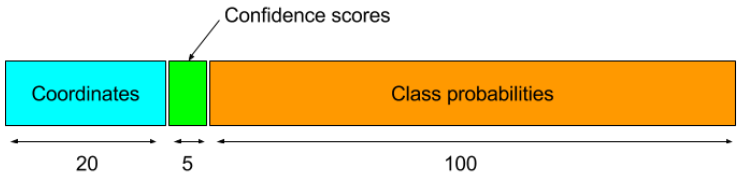
1. For each grid location we use n anchor boxes (with different aspect ratios and scales)
2. For each grid location predict for these n anchor boxes.
3. Similar to RPN we use Convolutions to go from depth C to the required depth (25*n) (3×3 convolution)
4. it predicts n boundary boxes and each box has one box confidence score



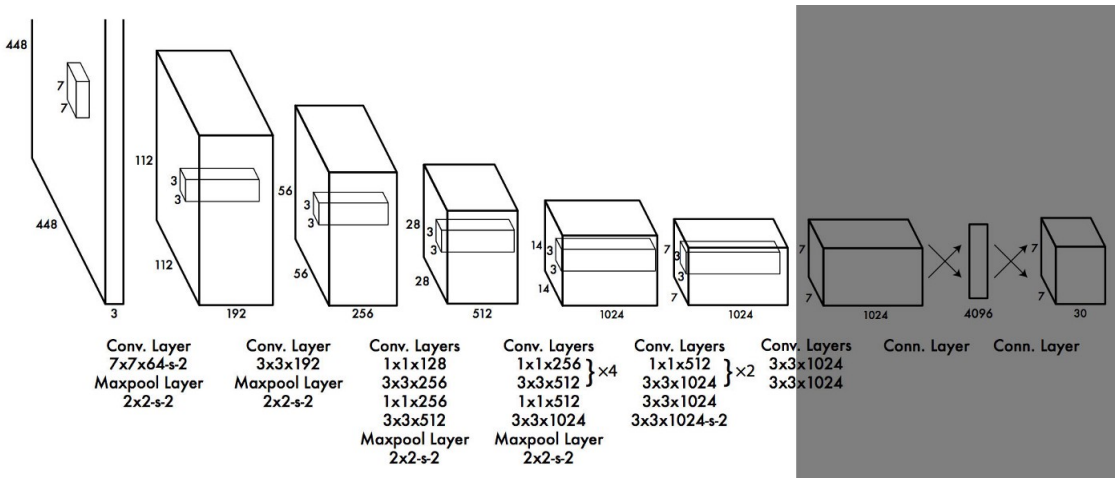
5. We then perform : (25 n because we want 25 outputs for every box)
    a. Object/Non-object classification (1 - yes/no)
    b. C class predictions (20 - 20 object classes are in the dataset they used)

i. it predicts C conditional class probabilities (one per class for the likeliness of the object class)

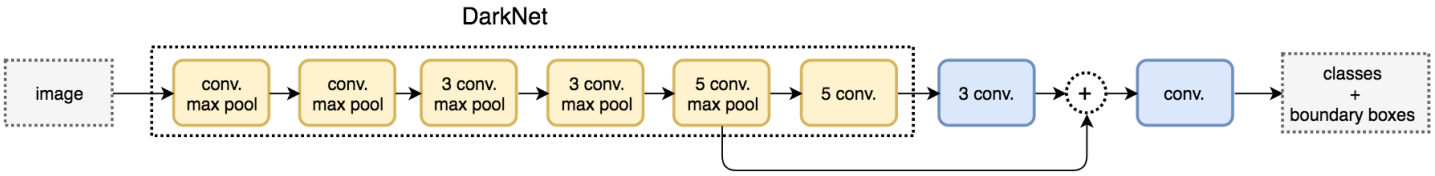
c. Anchor relative box regression (4 - x,y,h,w)



Changes in network :



We move the class prediction from the cell level to the boundary box level. Now, each prediction includes 4 parameters for the boundary box, 1 box confidence score (objectness) and 20 class probabilities. i.e. 5 boundary boxes with 25 parameters: 125 parameters per grid cell. Same as YOLO, the objectness prediction still predicts the IOU of the ground truth and the proposed box.



Dimension Clusters :

We encounter two issues with anchor boxes when using them with YOLO. The first is that the box dimensions are hand picked. Good priors can give good predictions.

We run k-means clustering on the training set bounding boxes to automatically find good priors that are independent of the box size. Distance Metric :

$$d(box, centroid) = 1 - IOU(box, centroid)$$

Direct location prediction :

Now, when we use anchor boxes with YOLO we encounter a second issue : model instability, especially during early iterations. Most of the instability comes from predicting the (x, y) locations for the box. In RPN the network predicts values t_x and t_y and

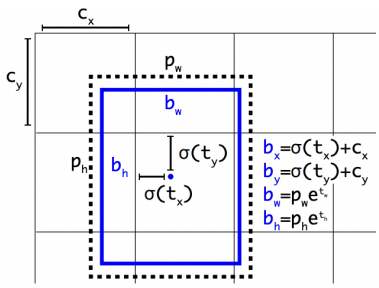
$$(x, y) = (((t_x * w_a) - x_a) , ((t_y * h_a) - y_a))$$

This is unconstrained since the shift of box for t_x = +1 and -1 gives the same shift.

Instead of predicting offsets we predict location coordinates relative to the location of the grid cell. This bounds the ground truth to fall between 0 and 1 using a logistic activation to constrain the predictions.

Here, the model predicts 5 values for each bounding box and 20 class probabilities :

$$
\begin{aligned}
b_x &= \sigma(t_x) + c_x \\
b_y &= \sigma(t_y) + c_y \\
b_w &= p_w e^{t_w} \\
b_h &= p_h e^{t_h} \\
Pr(\text{object}) * IOU(b, \text{object}) &= \sigma(t_o)
\end{aligned}
$$



Fine-Grained Features :

To detect small objects YOLO V2 adds a passthrough layer that concatenates the higher resolution features (26 × 26) with the low resolution features by stacking adjacent features into different channels instead of spatial locations, similar to the identity mappings in ResNet. This turns the 26 × 26 × 512 feature map into a 13 × 13 × 2048 feature map, which can be concatenated with the original features. Detector runs on top of this expanded feature map so that it has access to fine grained features.

Multi-Scale Training :

Unlike YOLO V1 which uses images of constant size, YOLO V2 changes the network every few iterations. Every 10 batches our network randomly chooses a new image dimension size to force the network to learn to predict well across a variety of input dimensions. The network runs faster at smaller sizes so YOLO V2 offers an easy tradeoff between speed and accuracy.

// read the hierarchical classification part.

YOLO V3 :

// look into this later

SSD (Single Shot Detector) :

SSD performs classification in different scales. In object detection, convolution is to reduce spatial size.

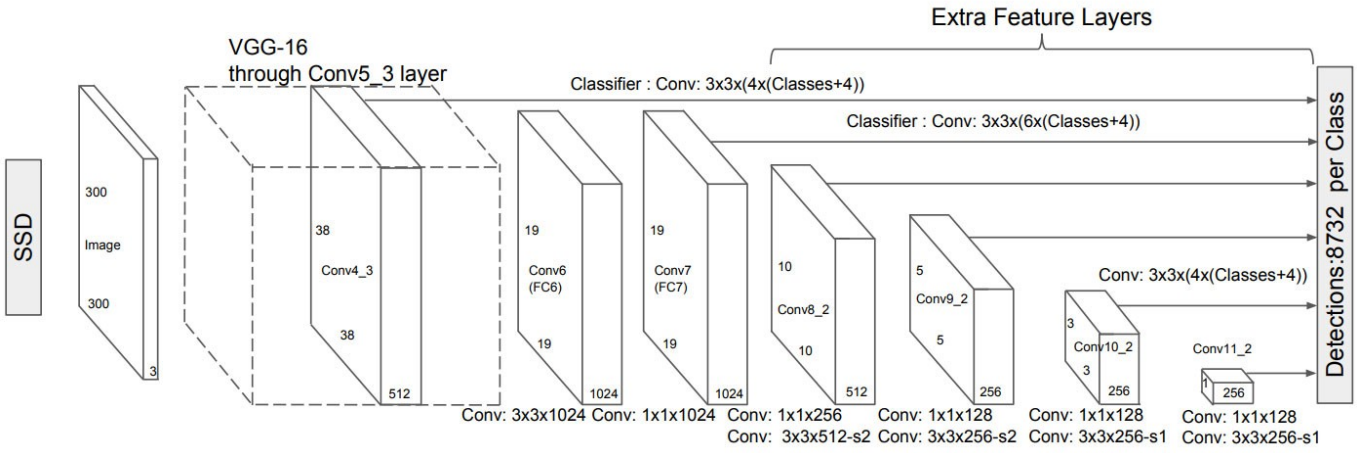


Image -> Feature Map -> Actual detection in form of refined anchors

This results in a 7 × 7 feature map. We might lose the information about the small objects due to the convolution operation.. With the 7 × 7 feature map, spatial resolution is difficult. So, object detection is done at multiple scales. Deeper layer helps in detecting bigger objects and shallower layers help in detecting smaller objects. The SSD network is just the same as YOLO with a few extra classification layers on the top.

Pros :

- Very fast
- End to end trainable and fully convolutional (with out fully connected layers, any image size works)
- SSD detects more objects than YOLO

Cons :

- Performance is not so good as two stage object detectors
- Difficulty with small objects

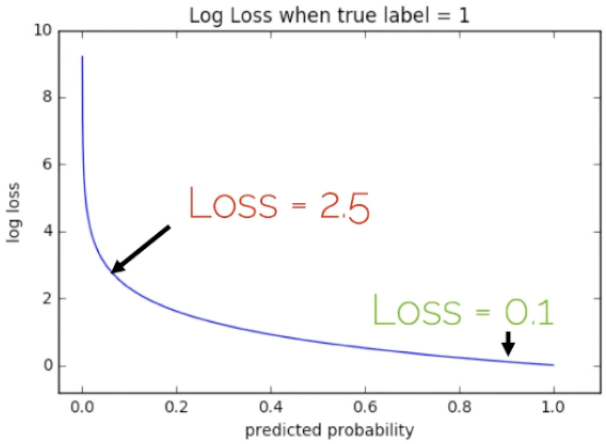
Problem with One Stage Object Detectors :

- Many locations need to be analyzed (100 K) densely covering the image → Foreground - Background imbalance.
  - Many negative samples
  - Few positive samples, rich information for learning
- Hard negative mining can improve, but not sufficient.

Conclusion : solution was RetinaNet

RetinaNet :

- Update the loss function to perform as good as two stage detectors.
- Cross entropy loss



From the graph :

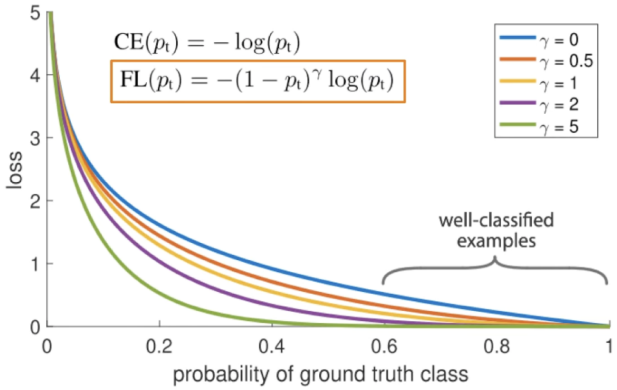
For a very hard example predicted loss is very high (2.5) and for a well classified example loss is low (0.1). Consider a case where there are very low number of hard samples and high well classified examples

100 hard + 100000 easy = 100*2.5 + 100000*0.1 = 250 + 10000

Because of the huge class imbalance we have a lot of easy examples. Even loss caused by each easy example is low, overall loss caused by them is high. So the easy examples are driving the training

So, they proposed focal loss to overcome this huge class imbalance

Focal Loss :



1. When ɣ = 0 Focal loss is Cross entropy loss.
2. When ɣ = 1, easy examples go down weighted
    a. as ɣ goes up easy samples go extremely down weighted
3. When ɣ = 2, p_t = 0.9 FL is 100 times lower than CEL
    a. 100 examples with focal loss have the same weight as one sample with CE loss

RetinaNet :

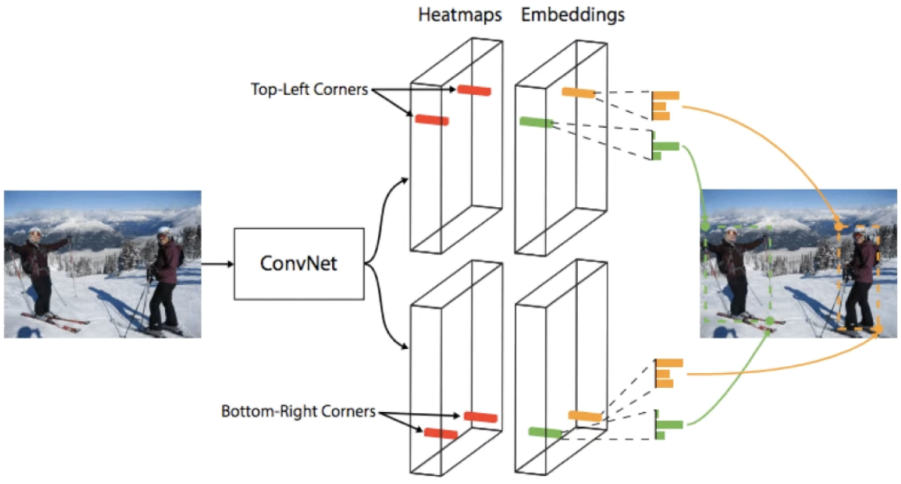
- ➔ Proposed focal loss
- ➔ Powerful Feature Extraction - ResNet
- ➔ Multi-Scale prediction like in SSD
- ➔ 9 anchor boxes per level, each one with a
    - Classification target
    - Regression target (like YOLO, RCNN)

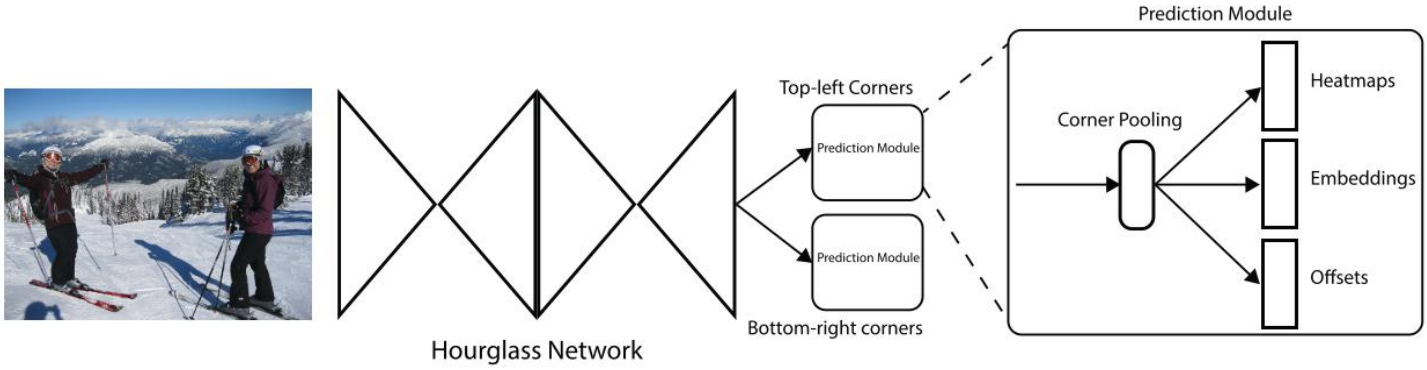
One Stage Point-Based Detectors :

Anchor boxes limit the objects that we can detect. So, we get rid of this using point based. They do not learn priorities for anchor boxes.

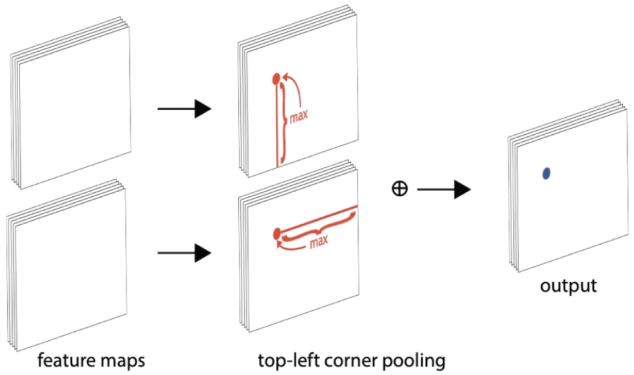
CornerNet :

CornerNet predicts corners and extracts boxes from them.





1. This uses CNN to extract the top left and bottom right corners of the bounding box.
    a. Corner net use a hourglass network
    b. We predict corners at a lower resolution and regress the offset (bounding box correction as we have seen in many other methods

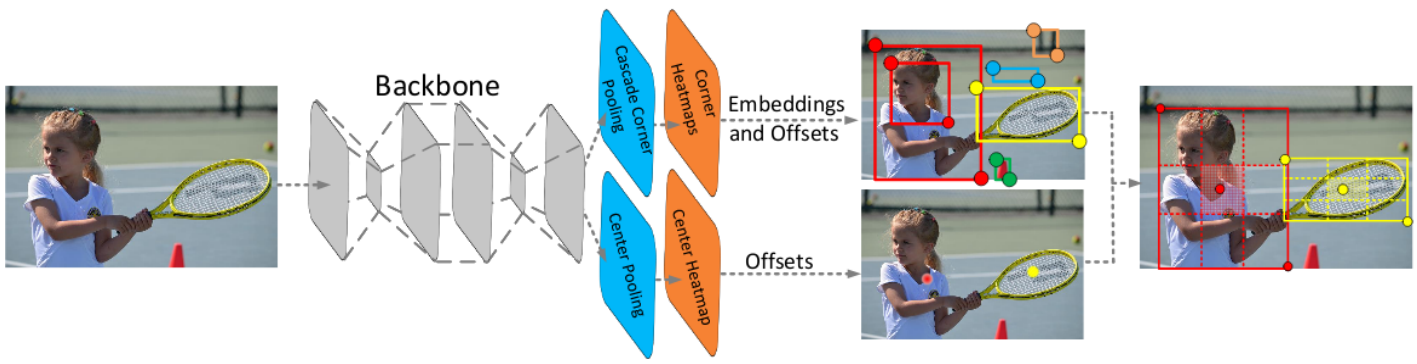


2. It is very difficult and ambiguous to predict the corners precisely so we do max pooling on both the directions
3. One head to predict heat maps for top left, another for bottom right
    a. Depicts probability of finding bottom left in several positions of feature map
    b. Similarly the bottom right heat map
4. Embeddings give probability map for each corner type
    a. Embeddings represents signature and identifier for corners
5. Match top left and bottom right using embeddings similarity
6. Predicts the class on the top

Cons :

1. Many incorrect bounding boxes (especially small) (too many false positives)
2. Hypothesis : it is hard to infer the class of the bounding box if the network is focused on boundaries

CentreNet :



To overcome the classification problem in CornerNet we focus on the centre to infer its class. We use corners as proposals, centre to verify the class of the object and filter the outliers.

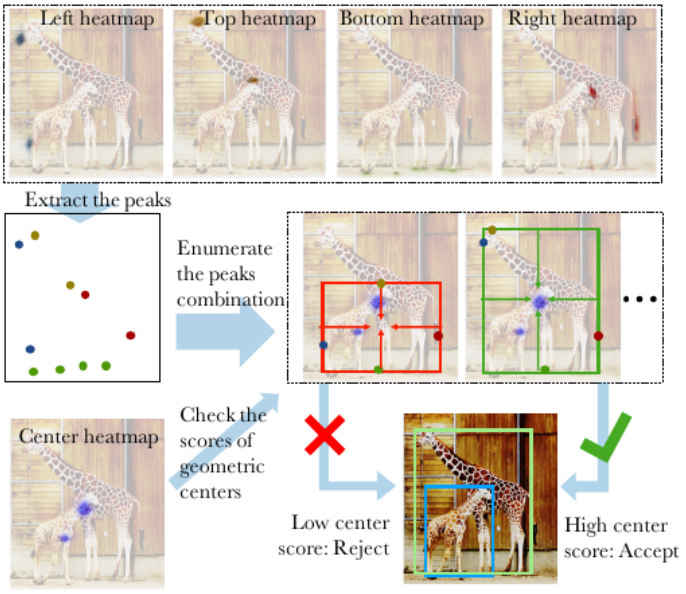
ExtremeNet :

Problem with bounding boxes :

The corners that we predict might be a part of the background, it is difficult for neural network to detect these type of corners

In this image red points are a part of the background. It is difficult for CNN to predict these red corners. So, we present objects by their extreme points instead of corners because for CNN it is easy to find extreme points that are present on the object rather than the point that is completely in the background.

Here we do not predict embeddings for computation, instead use four different heat maps for top, left, bottom and right extreme points. These express the probability of certain locations of the image being top, left, bottom and right corners.



1. We will have all the peaks and to predict the final extreme box, we enumerate these peak combinations by centre heatmap instead of embeddings.
2. If centre of the obtained box does not correspond to any centre peak then we reject that particular box
3. Else we accept it

Note :

1. All these finding of extreme points is to find ground truth.
2. They are commonly used for annotation
3. This is also useful in finding plausible mask object by extreme cut

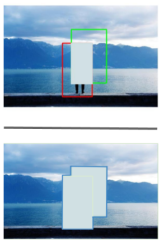
Object Detection Evaluation :

Referable link :
https://manalelaidouni.github.io/manalelaidouni.github.io/Evaluating-Object-Detection-Models-Guide-to-Performance-Metrics.html

to determine how many objects were detected correctly and how many false positives were generated (will be discussed below), we use the Intersection over Union (IoU) metric.

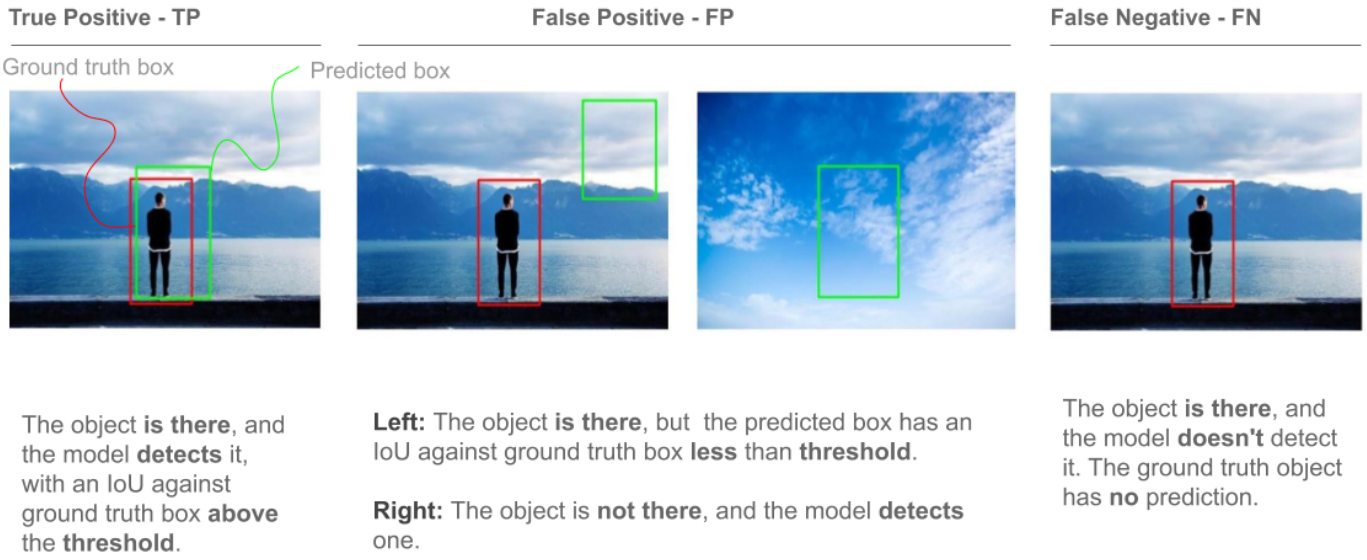
Intersection over Union (IoU) :

$$IoU^{truth}_{pred} = \frac{truth \cap pred}{truth \cup pred} =$$



Predictions: TP - FP - FN :

we set a threshold for converting those real-valued scores into classifications, where IoU values above this threshold are considered positive predictions and those below are considered to be false predictions. More precisely, the predictions are classified into True Positives (TP), False Negatives (FN), and False Positives (FP).



Note :

We do not include TN (true negatives) because they describe empty boxes are correctly detected as “non-object”. the model would evidently identify thousands of empty boxes which adds little to no value to our algorithm.

		Predicted +	Predicted -	
Actual	+	TP	FN	P = TP+FN
	-	FP	TN	N = FP+TN

How predictions work:

- When multiple boxes detect the same object, the box with the highest IoU is considered TP, while the remaining boxes are considered FP.
- If the object is present and the predicted box has an IoU < threshold with ground truth box, The prediction is considered FP. More importantly, because no box detected it properly, the class object receives FN, .
- If the object is not in the image, yet the model detects one then the prediction is considered FP.
- Recall and precision are then computed for each class by applying the below-mentioned formulas, where predictions of TP, FP and FN are accumulated.
- IoU threshold depends on Dataset

Accuracy :

The accuracy result can be very misleading when dealing with class imbalanced data, where the number of instances is not the same for each class, as It puts more weight on learning the majority classes than the minority classes.

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

In detection datasets, class distribution is considerably non-uniform.

Precision :

Precision (lies between 0 and 1) is the probability of the predicted bounding boxes matching actual ground truth boxes, also referred to as the positive predictive value.

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{\textit{True object detections}}{\textit{Total detections}}$$

We may consider applying hard negative mining to improve low precision (i.e. include negative examples in training) since the model suffers from high false positives.

Recall :

Recall is the true positive rate, also referred to as *sensitivity*, measures the probability of *ground truth objects* being correctly detected.

$$\text{Precision} = \frac{TP}{TP+FN} = \frac{\textit{True object detections}}{\textit{All ground truth boxes}}$$

Interpretations :

- High recall but low precision implies that all ground truth objects have been detected, but most detections are incorrect (many false positives).
- Low recall but High precision implies that all predicted boxes are correct, but most ground truth objects have been missed (many false negatives).
- High precision and high recall, the ideal detector has most ground truth objects detected correctly.

Note that we can evaluate the performance of the model as a whole, as well as evaluating its performance on each category label, computing class-specific evaluation metrics.

Precision - Recall :

Trade-off between precision and recall depends on the confidence threshold. The object detector predicts bounding boxes, each associated with a confidence score. The confidence score is used to assess the probability of the object class appearing in the bounding box.

Accordingly, we set a threshold to turn these confidence probabilities into classifications, where detections with a confidence score above the predetermined threshold are considered true positives (TP), while the ones below the threshold are considered false positives (FP).

When choosing a high confidence threshold, the model becomes robust to positive examples (i.e. boxes containing an object), hence there will be less positive predictions. As a result, false negatives increase and false positives decrease, this reduces recall (the denominator increases in the recall formula) and improves precision (denominator decreases in the precision formula).

Similarly, further lowering the threshold causes the precision to decrease and recall to increase. Therefore, the confidence threshold is a tunable parameter where by adjusting it we can define TP detections from FP ones, controlling precision and recall, thereby determining the model's performance.

In order to reduce false positive prediction, consider increasing recall by increasing the confidence threshold.

Precision - Recall Curve :

Precision and recall both do not include TN so, we use a new metric precision-recall curve to evaluate that includes both precision and recall.

when comparing multiple models, the one with the highest curve in the precision x recall plot is considered the better performer. Nevertheless, due to the trade-off between recall and precision, the curve can be noisy and have a particular saw-tooth shape that makes it difficult to estimate the performance of the model and compare different models with their precision x recall curves crossing each other.

Therefore, we estimate the area under the curve using a numerical value called Average Precision.

Average Precision :

Average precision (AP) serves as a measure to evaluate the performance of object detectors, it is a single number metric that encapsulates both precision and recall and summarizes the Precision-Recall curve by averaging precision across recall values from 0 to 1.

11 point interpolated average precision :

AP averages precision at a set of 11 spaced recall points (0, 0.1, 0.2, .. , 1) where we interpolate the corresponding precision for a certain recall value r by taking the maximum precision whose recall value $\bar{r} > r$.

In other words, take the maximum precision point where its corresponding recall value is to the right of r. In this case, Precision is interpolated at 11 recall levels, hence the name 11-point interpolated average precision. This is particularly useful as it reduces the variations in the precision - recall curve. This interpolated precision is referred to as