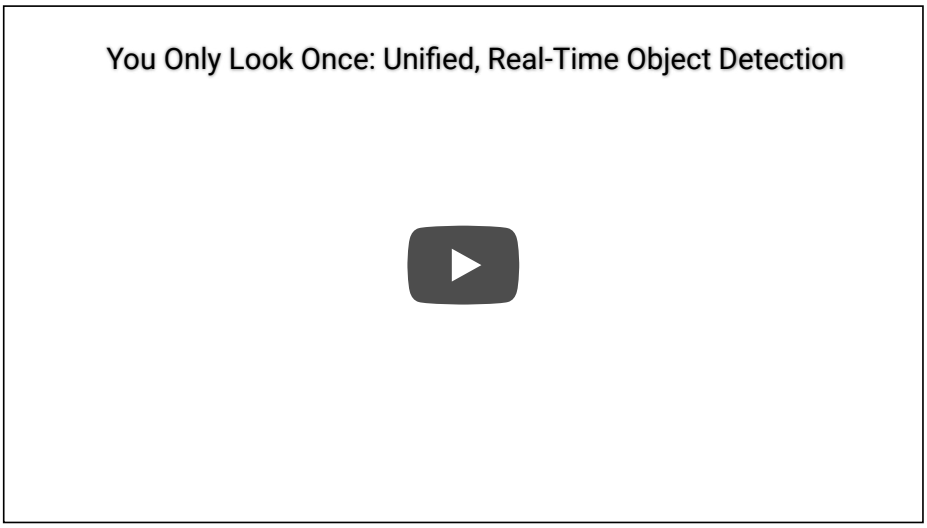
S12

Due No Due Date **Points** None **Available** after Apr 12 at 9:30am

Object Localization - YOLO



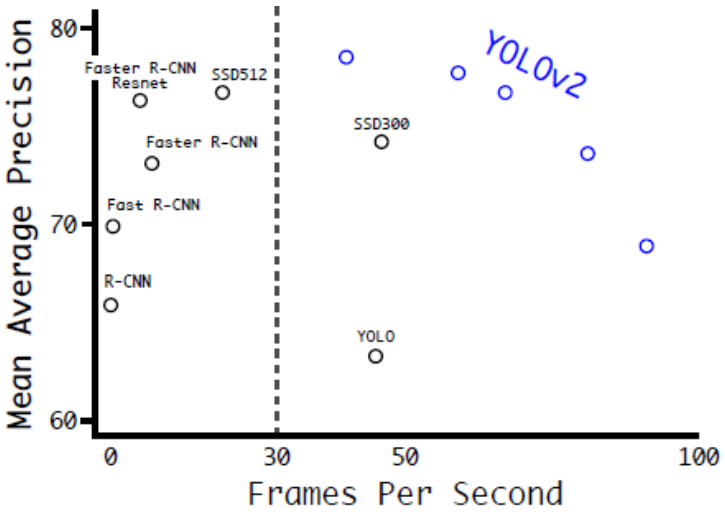
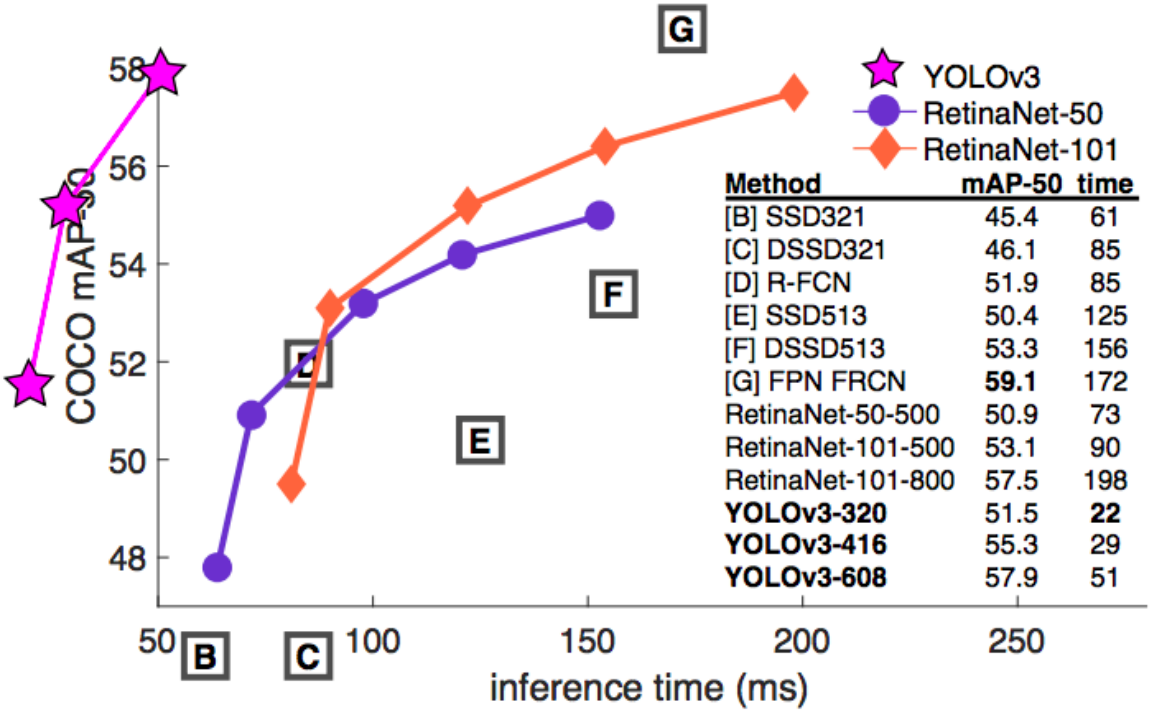
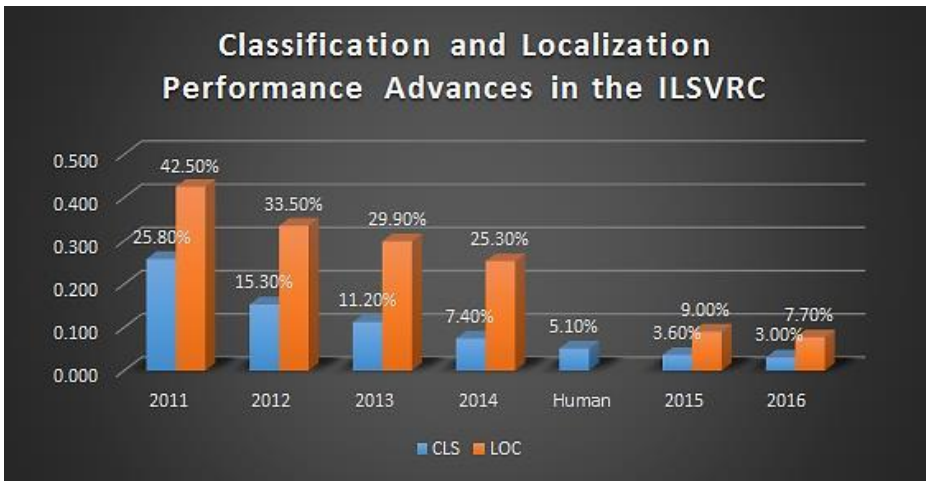


Figure 4: Accuracy and speed on VOC 2007.



Where are we today?



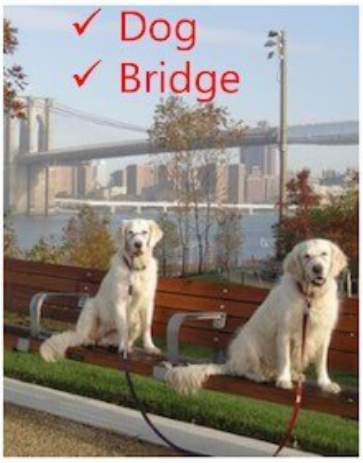
Let us observe how good ML has become is to see the ILSRVC results:

Year	Team	Layers	Contribution	Position
2010	NEC	Shallow	Fast Feature Extraction, Compression, SVM	First
2011	XRCE	Shallow	High dimensional image, SVM	First
2012	SuperVision	8	GPU based DCNN, Dropout	First
2013	Clarifai	8	Deconvolution visualization	First
2014	GoogLeNet	22	DCNN, 1x1 conv	First
2014	VGG	19	All 3x3 convs	Second
2015	MSRA	152	Ultra Deep, Residuals	First
2016	CUImage	269	Ensamble, Gated Bi-Directional CNN	First
2017	Momenta	152	Squeeze & excitation, feature recalibration	First
2018*	Facebook	264	Direct connection between any two layers	SOA*

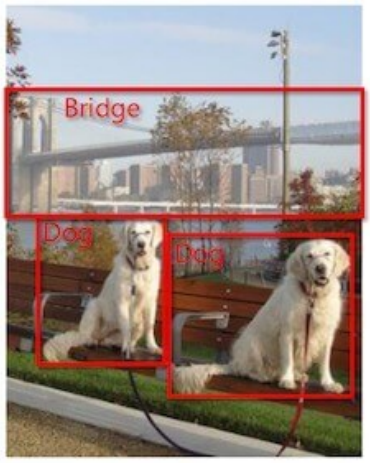
As we can see networks are getting deeper and more sophisticated.

Every year there is an addition of new technology which is making everything till then obsolete.

Recognition vs Detection



Classification, easy these days



Object detection, still a lot harder

In object recognition, our aim is to recognize what all is there in the image, for e.g. Dog and Bridge.

In object detection, however, we need to specify where exactly the dog(s) and bridge are.

Recognition is pretty easy these days, while detection is still a work in development.

Detection Approaches

Simplify Object Detection problem by:

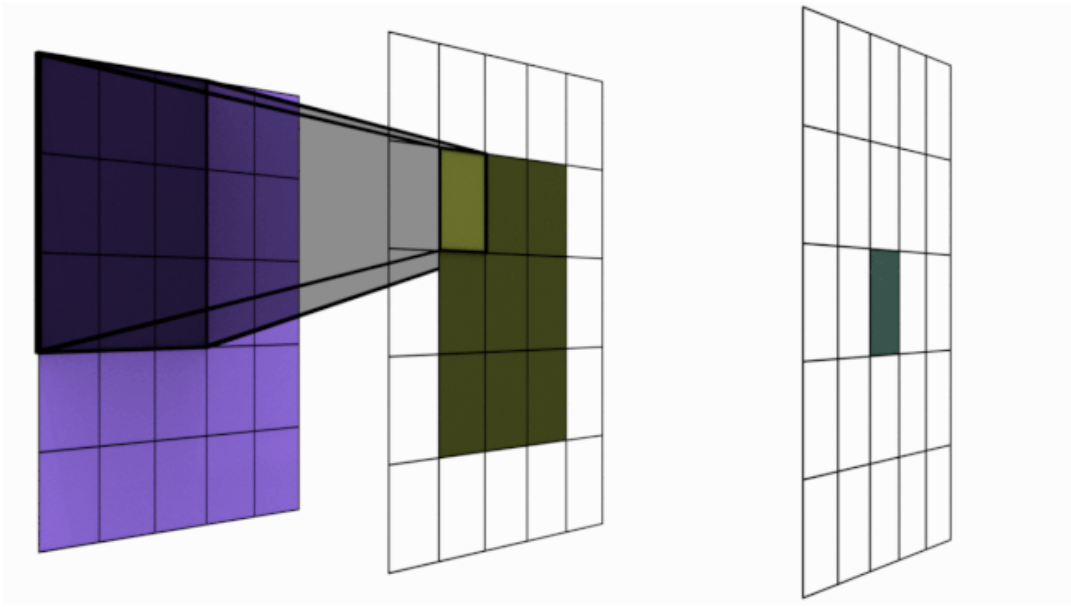
1. ignoring lower prediction values
2. predicting bounding boxes instead of the exact object mask
3. mixing different receptive field layers
4. but this is easier said than done.

There are two main approaches to driving detection algorithms, namely:

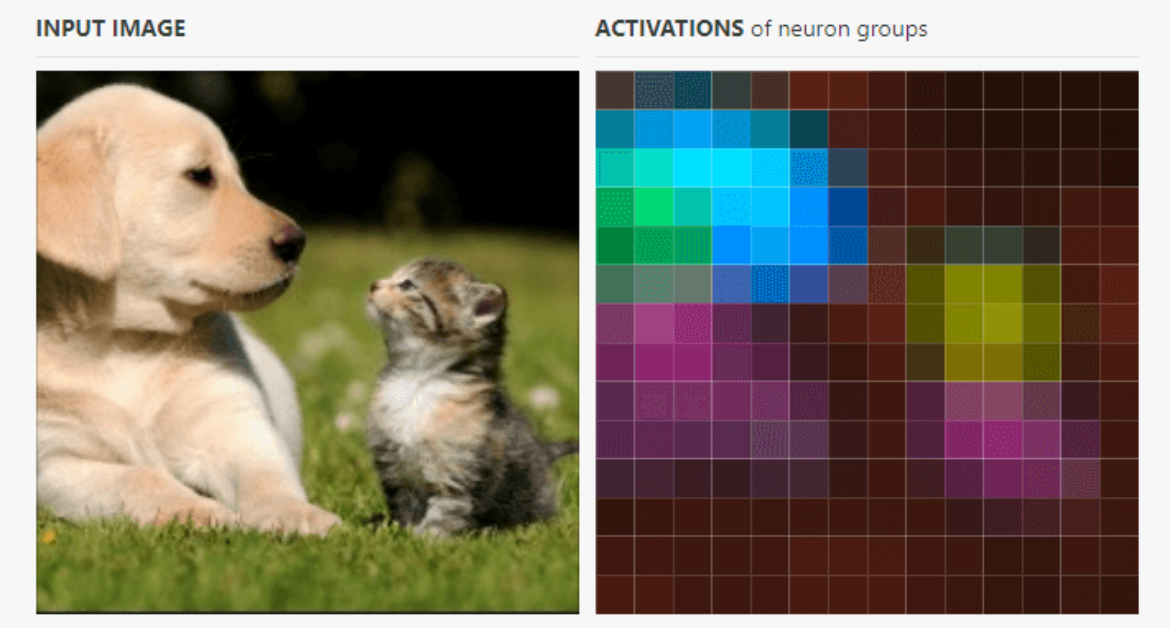
1. YOLO-like approach, where k-means extracted anchor boxes are used, and
2. SSD-like approach, where a fixed number of predefined bounding boxes are used.

Let's recap

We know this image:

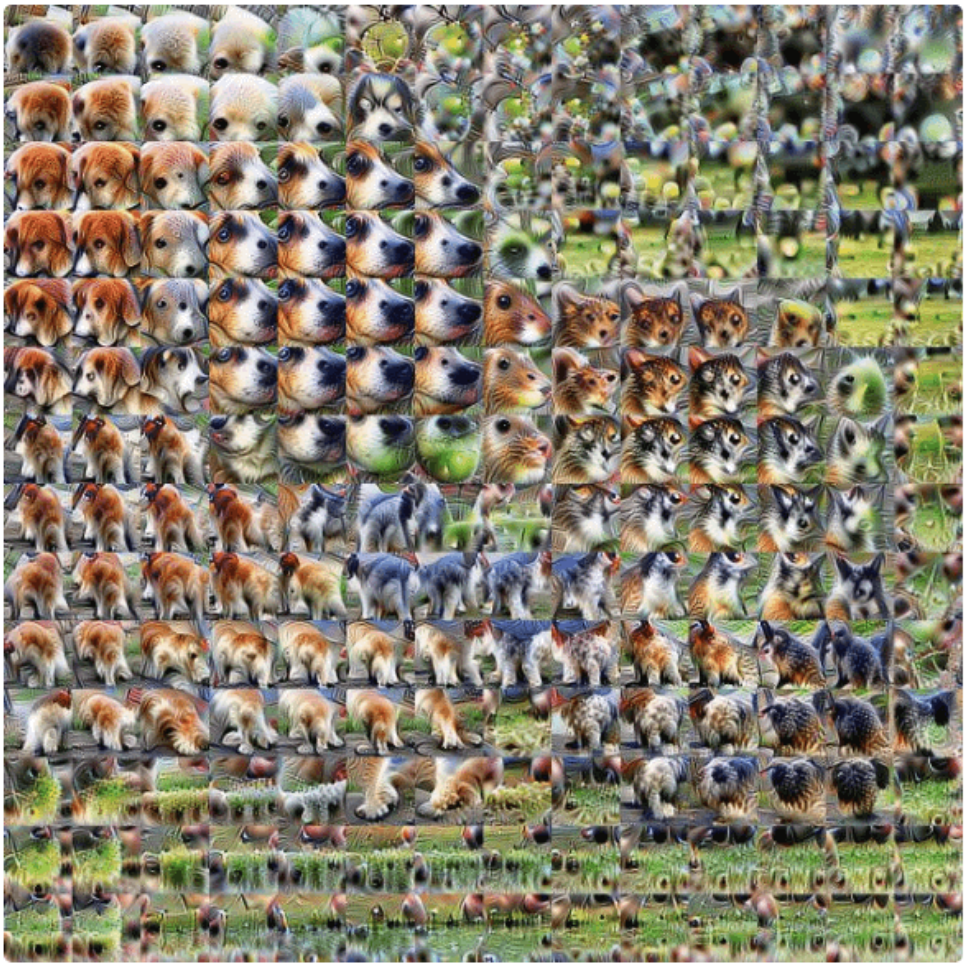


Later on

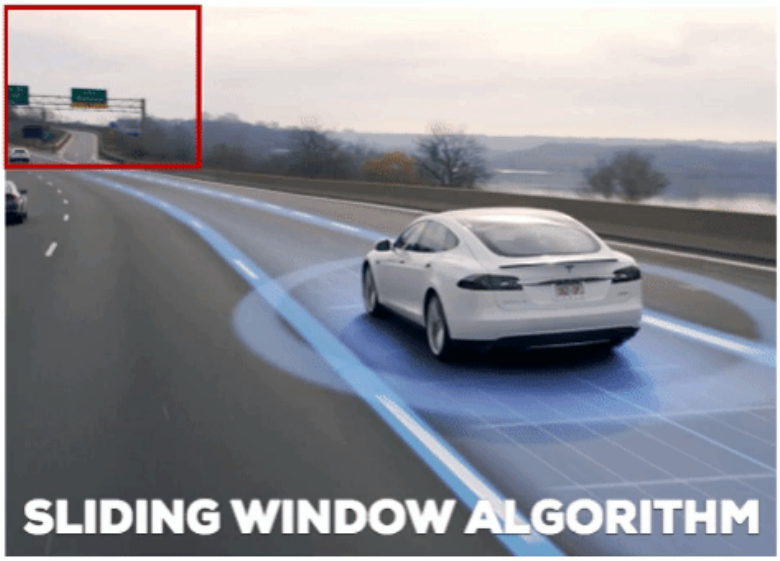


This is how prediction cells look like. You can relate the cells with the image.

THEN



How we used to find BBoxes



Why do you think this is bad? Because we need to answer a few hard questions.

What should be the size of this window? How do we handle big and small objects at the same time?

Do we need to slide it at each pixel, or we can make a jump? What is the perfect jump size?

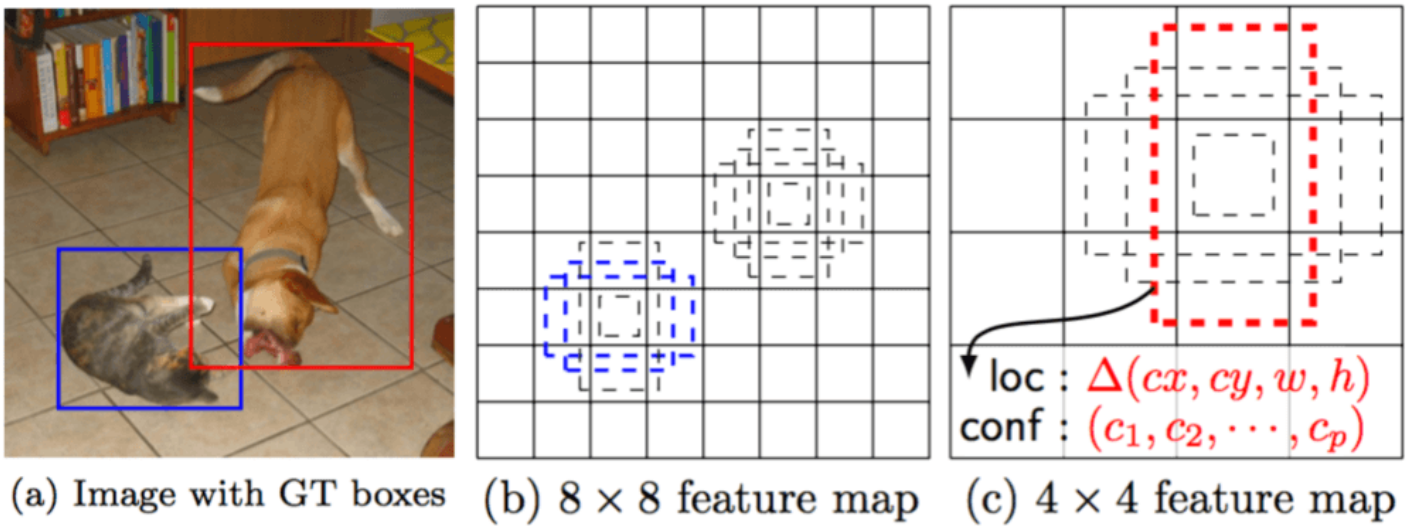
Sliding Window as a concept is simple but extremely compute-intensive.

Region Proposal

RCNN uses this, and it was very bad, so bad that RCNN took 20 seconds to go through 1 image (it was running AlexNet 2000 times for 2000 proposals)

Welcome Anchor Boxes

Faster RCNN, SSD and YOLOv2, all use Anchor Boxes.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Intuitively, we know that objects in an image should fit certain common aspect ratios and sizes.

For instance, we know that we want some rectangular boxes that resemble the shapes of humans.

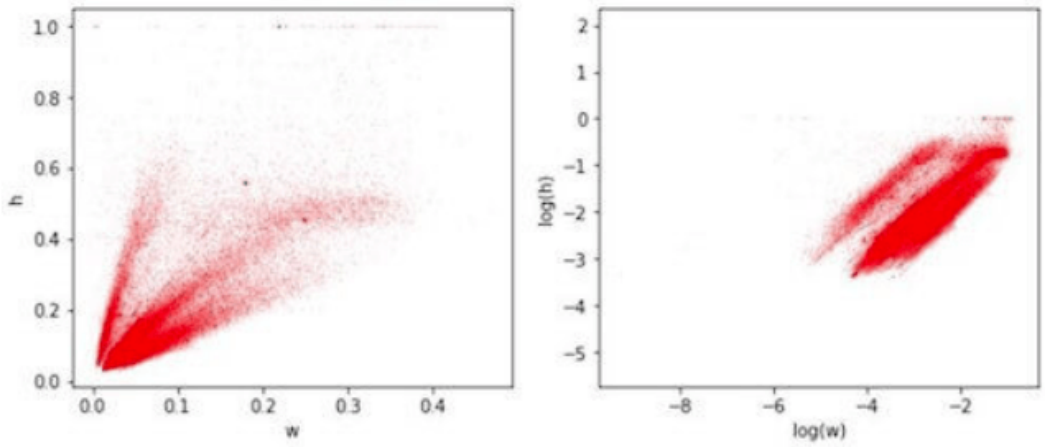
Likewise, we know we won't see many boxes that are very very thin. In such a way, we create k such common aspect ratios we call anchor boxes.

For each such anchor box, we output one bounding box and score per position in the image.

How to calculate Anchor Boxes?

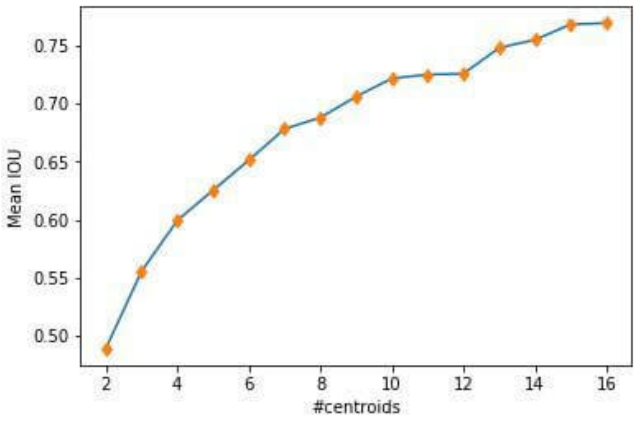
Let us see how to do it!

- Compute the center location, width and height of each bounding box, and normalize it by image dimensions (in the dataset)
- Plot the h and w for each box as shown in the right "h vs w graph"
- Use K-means clustering to compute cluster centers (centroids).

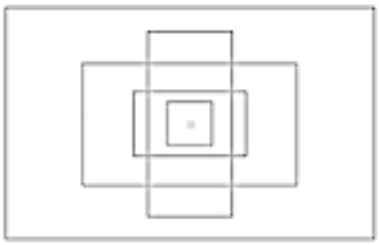


Compute the different numbers of clusters and compute the mean of maximum IOU between bounding boxes and individual anchors.

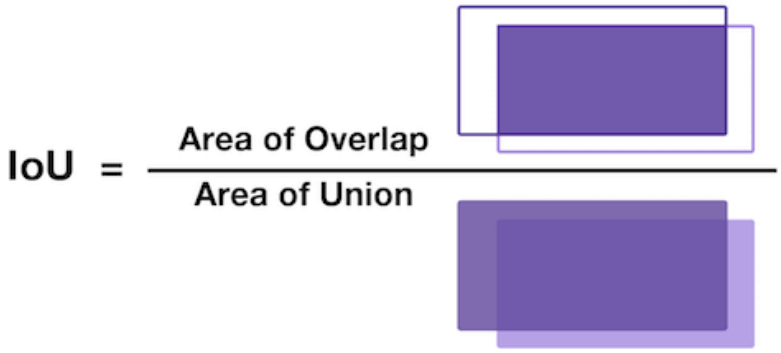
Plot centroids vs mean IOU.



Pick the top 5 anchor boxes (5 for YOLOv2 where IOU was above 65%)



Intersection Over Union



Intersection over Union is an evaluation metric used to measure the accuracy of an object detector on a particular dataset.

The Scary loss function

$$
\begin{aligned}
&\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
$$

We have 5 anchor boxes.

For each anchor box, we need Objectness-Confidence Score (where there is an object found?), 4 Coordinates (t_x, t_y, t_w and t_h), and 20 top classes. This can crudely be seen as 20 coordinates, 5 confidence scores, and 100 class probabilities as shown in the image on the right, so in total 125 filter of 1x1 size would be needed.

So we have a few things to worry about:

- x_i, y_i , which is the location of the centroid of the anchor box
- w_i, h_i , which is the width and height of the anchor box
- C_i, which is the Objectness, i.e. confidence score of whether there is an object or not, and
- $p_i(c)$, which is the classification loss.

All losses are mean squared errors, except classification loss, which uses cross-entropy function.

Understanding YoloV2 Loss Function

Let's look at that loss function again.

$$\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

Now, let's break the code in the image.

- We need to compute losses for each Anchor Box (5 in total)
  - $\sum^{B}$ represents this part.
- We need to do this for each of the 13x13 cells where S = 13
  - $\sum^{S2}$ represents this part.
  - 1_{ij}^{obj} is 1 when there is an object in the cell i, else 0.
- 1_{ij}^{noobj} is 1 when there is no object in the cell i, else 0. We need to do this to make sure we
  reduce confidence when there is no object as well.
- 1_{i}^{obj} is 1 when there is a particular class is predicted, else 0.
- λs are constants. λ is highest for coordinates in order to focus more on detection (remember,
  we have already trained the network for recognition!)
- We can also notice that w_i, h_i are under square-root. This is done to penalize the smaller
  bounding boxes as we need to adjust them more.

Check out this table:

var1	var2	(var1-var2)^2	(sqrtvar1-sqrtvar2)^2
0.0300	0.020	9.99e-05	0.001
0.0330	0.022	0.00012	0.0011
0.0693	0.046	0.000533	0.00233
0.2148	0.143	0.00512	0.00723
0.8808	0.587	0.0862	0.0296
4.4920	2.994	2.2421	0.1512

Annotations

Let's look at VGG Annotator: **http://www.robots.ox.ac.uk/~vgg/software/via/via_demo.html (http://www.robots.ox.ac.uk/~vgg/software/via/via_demo.html)**

The assignment

1. Assignment A:
    1. Download this **TINY IMAGENET (http://cs231n.stanford.edu/tiny-imagenet-200.zip)** dataset.
    2. Train ResNet18 on this dataset (70/30 split) for 50 Epochs. Target 50%+ Validation Accuracy.
    3. Submit Results. Of course, you are using your own package for everything. You can look at **this (https://github.com/sonugiri1043/Train_ResNet_On_Tiny_ImageNet/blob/master/Train_ResNet_On_Tiny_ImageNet.ipynb)** for reference.
2. Assignment B:
    1. Download 50 images of dogs.
    2. Use **this (http://www.robots.ox.ac.uk/~vgg/software/via/via_demo.html)** to annotate bounding boxes around the dogs.
    3. Download JSON file.
    4. Describe the contents of this JSON file in FULL details (you don't need to describe all 10 instances, anyone would work).
    5. Refer to this **tutorial (https://towardsdatascience.com/machine-learning-algorithms-part-9-k-means-example-in-python-f2ad05ed5203)**. Find out the best total numbers of clusters. Upload link to your Colab File uploaded to GitHub.

Questions in S12-Assignment-Solution:

1. What is your final accuracy?
2. Share the Github link to your ResNet-Tiny-ImageNet code. All the logs must be visible.
3. Describe the contents of the JSON file in detail. You need to explain each element in detail.

4. Share the link to your Github file where you have calculated the best K clusters for your 50 dog dataset.
5. Share the link to your 50 Dog Images Folder on GitHub
6. Share the link to your JSON file on GitHub

Session Video: