TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Object detection models for a Formula Student Driverless Car

Paolo Melissari

0287567

Index

Introduction 3

R-CNN 8

Fast R-CNN 11

Faster R-CNN 15

Single Shot Detector (*SSD*) 19

YOLO v3 24

YOLO v4 30

Final Considerations 37

References 38

1.Introduction

Formula SAE is a student design competition organized by SAE International.

The concept behind Formula SAE is that a fictional manufacturing company has contracted a student design team to develop a small Formula-style race car. The prototype race car is to be evaluated for its potential as a production item. The target marketing group for the race car is the non-professional weekend autocross racer. Each student team designs, builds and tests a prototype based on a series of rules, whose purpose is both ensuring on-track safety (the cars are driven by the students themselves) and promoting clever problem solving.

In 2019, University of Rome “Tor Vergata” took part in the Formula Student event at the Hockenheimring (Germany) with “Artemide” prototype, shown in the following pictures:

Figure 1. Artemide

Figure 2. Artemide during dynamic test

With the introduction of the Driverless category in 2017, all teams are getting ready for the construction of an autonomous vehicle.

In this category, in addition to "static" tests in which judges will evaluate the design choices, costs and business plan, the prototype must be able to complete the following "dynamic" tests:

- Acceleration, in which the vehicle must make a straight trajectory of 75 meters, remaining inside some blue and yellow cones, and then stop within a Stop Area delimited by orange cones, as shown in Figure 4:

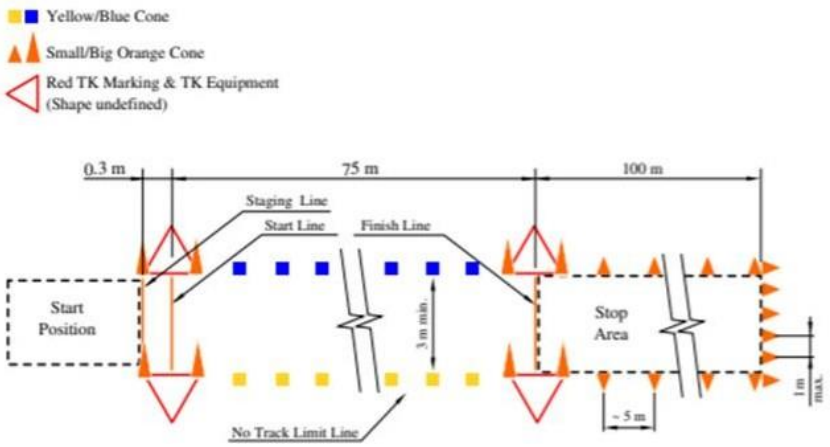


Figure 3. Acceleration Layout

- Skidpad, in which the vehicle must follow a trajectory similar to an 8 for two consecutive times, as shown in Figure 5:

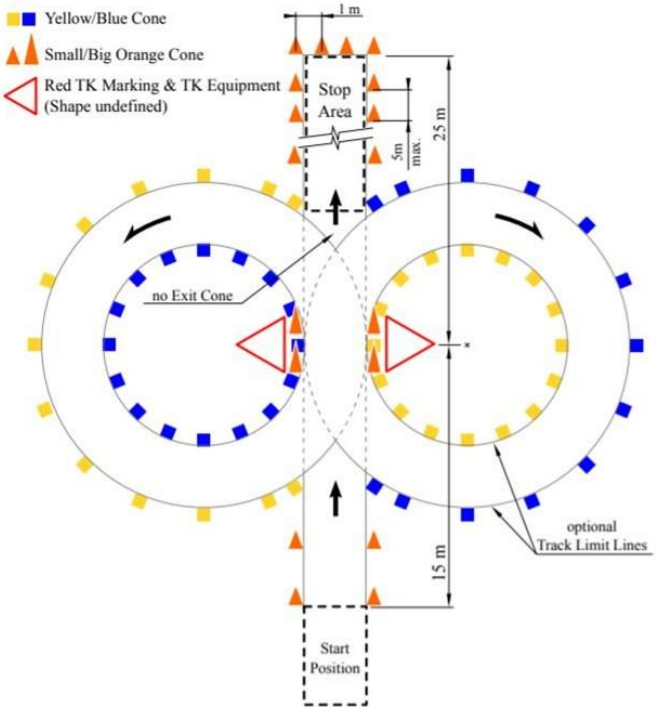


Figure 4. Skidpad layout

- Autocross and Trackdrive, in which the vehicle must run within a 500 meter track as shown in Figure 6. The Autocross consists of a single lap, while the TrackDrive consists of 10 laps (the layout of the track is not known before the event).

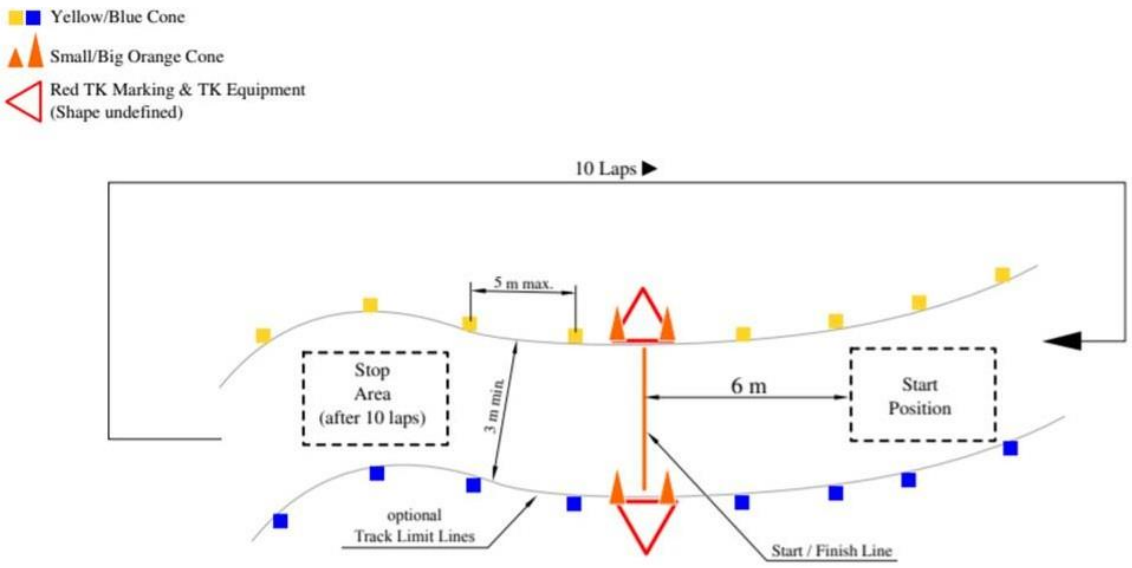


Figure 5. AutoCross and TrackDrive

The cones have the following shape

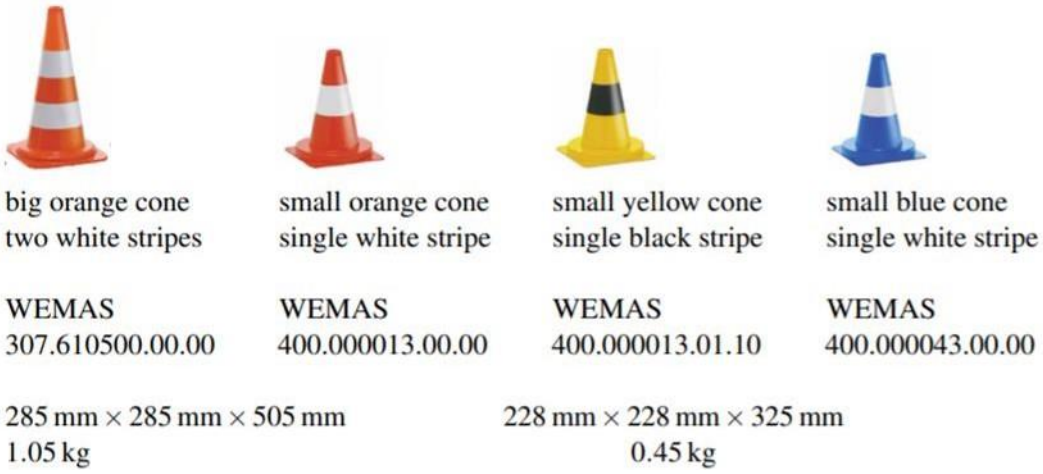


Figure 6. Cones

The first step needed to create a control system that allows the vehicle to move autonomously is to recognize the cones through images acquired by a video camera, distinguishing their size and color.

The goal of this study is therefore to analyze the different possible solutions for detecting the cones along the route, comparing them with each other and choosing the one that is most suitable for this task.

In particular, the goal is to choose the model that allows us to achieve good accuracy and that has a high speed.

The possible solutions identified are:

- R-CNN
- Fast-CNN
- Faster-CNN
- SSD
- YOLO v3
- YOLO v4

3. R-CNN (Region proposal Convolutional Neural Network)

In this approach, we select what are the regions that are likely to contain the cones (Region Of Interest, ROI) and then apply a CNN classification only on this region, drastically reducing the time needed to detect all cones in the frame then applying CNN to each crop of the image (varying size and shape).

These regions can be found using a region proposal method such as Selective Search, which generate 2000 ROI. Selective Search is a method for finding a large set of possible object locations in an image, and it works by clustering image pixels into segments and then performing hierarchical clustering to combine segments from the same object into object proposals:

1. It starts generating an initial sub-segmentation for the input image, and then recursively merge the smaller “similar” regions into the larger ones.
2. Then, it use the segmented region proposals to generate the candidate object locations on which will be applied the CNN classification.

This workflow is illustrated in Figure 7:

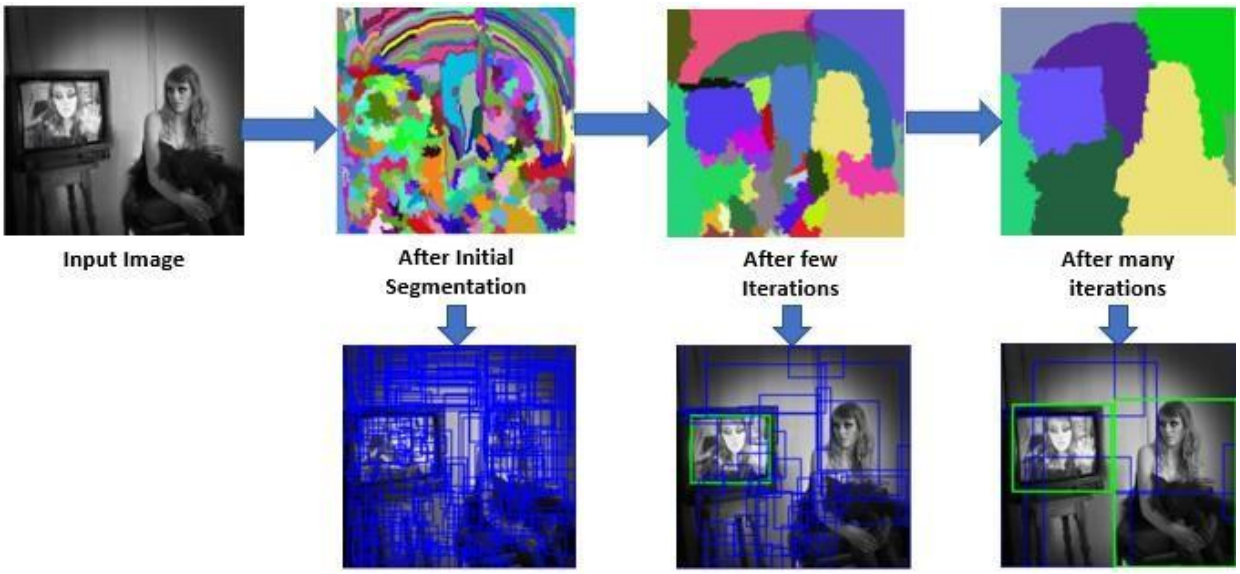


Figure 7. Selective Search

We have to define when a region could be considered “similar” to another region.

A possible approach is to define 4 type of similarity measure and then combine each other:

1. Color Similarity: for each region we generate an histogram for each color channels (R,G.B) present in the image. In the paper 25 bins are taken in histogram for each color channel, so we will have a vector of n = 75 bins for each region. We will find similarity using the equation:

$$s_{color}(r_i, r_j) = \sum_{k=1}^{n} \min(c_i^k, c_j^k)$$

with c_i^k = k-th value of histogram bin of region $r_i \; and \; r_j$ respectively

2. Texture Similarity: calculated using generated 8 2D-Gaussian derivatives of image and extracts histogram with 10 bins for each color channels. This gives us 10 x 8 x 3 = 240 dimensional vector for each region. We will find similarity using the equation:

$$s_{texture}(r_i, r_j) = \sum_{k=1}^{n} \min\,(t_i^k, t_j^k)$$

with t_i^k = k-th value of texture histogram bin of region $r_i \; and \; r_j$ respectively

3. Size Similarity: used in order to make smaller region merge easily, calculated comparing the size of two regions using the equation:

$$s_{size}(r_i, r_j) = 1 - \frac{(size(r_i) + size(r_j))}{size(img)}$$

with $size(r_i),\ size(r_j)\ and\ size(image)$ = size in pixels of regions r_i, r_j and image respectively

4. Fill Similarity: measures how well two regions fit with each other. If two region fit well into one another (for Example one region is present in another) then they should be merged, if two region does not even touch each other then they should not be merged. We will find similarity using the equation:

$$s_{fill}(r_i, r_j) = 1 - \frac{(size(BB_{ij}) - size(r_i) - size(r_j))}{size(img)}$$

with $size(BB_{ij})$ = size of bounding box around i and j

These similarity function can be combined as:

$$S(r_i, r_j) = a_1 * s_{color}(r_i, r_j) + a_2 * s_{texture}(r_i, r_j) + a_3 * s_{size}(r_i, r_j) + a_4 * s_{fill}(r_i, r_j)$$

Where a_i is either 0 or 1 depending upon we consider this similarity or not

We can now use this combined similarity function and merge the two most similar region at every iteration of pass 1.

Lastly, since our CNN expect an input of fixed size, we reshape the region proposals to the same size.

The CNN acts as a feature extractor and the output dense layer consists of the features extracted from the image. This features are fed into an SVM to classify the presence of the object within that candidate region proposal. In addition to this, the model also predicts four values which are offset values to increase the precision of the bounding box. For instance, given a region proposal, the

algorithm would have predicted the presence of a cone but the height of that cone within that region proposal could've been cut in half. Therefore, the offset values help in adjusting the bounding box of the region proposal.

Even though this approach is much faster than sliding window, the main problem is that it still has a very high detection time (about 47 s) , which makes it unsuitable for real-time applications.

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

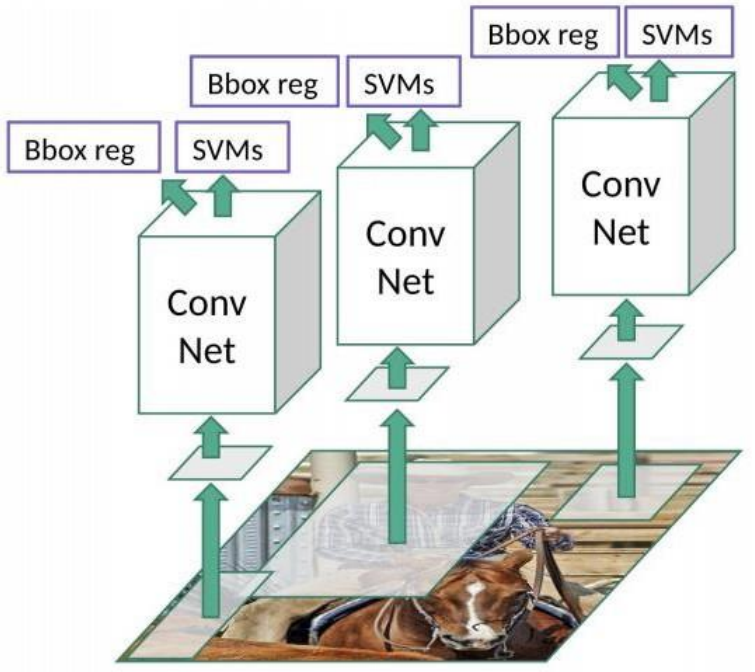


Figure 8. Region Proposal

4. Fast R-CNN

This approach is an evolution of R-CNN approach. Instead of feeding the region proposals to the CNN, we feed the input image to the CNN to generate a feature map. From this feature map, we identify the region of proposals, warp them into squares and we reshape them into a fixed size by using a RoI pooling layer, so that it can be fed into a fully connected layer. From the RoI feature vector, we use a softmax layer to predict the class of the proposed region and also the offset values for the bounding box, as shown in Figure 9:

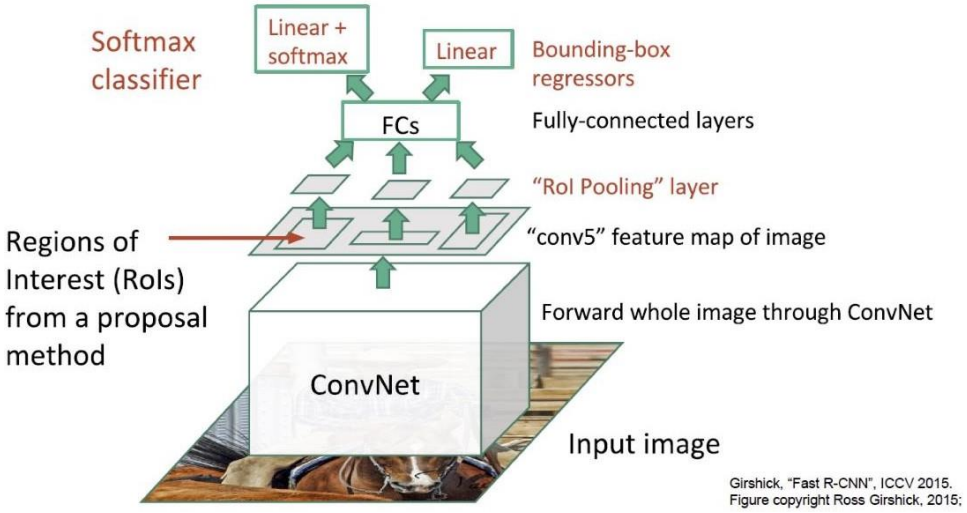


Figure 9. Fast CNN

ROI pooling produces the fixed-size feature maps from non-uniform inputs by doing a sort of max-pooling on the inputs. ROI pooling layer takes two inputs:

1. A feature map obtained from a Convolutional Neural Network after multiple convolutions and pooling layers.
2. 'N' proposals or Region of Interests from Region proposal network. Each proposal hasfive values, the first one indicating the index and the rest of the four are proposal coordinates. Generally, it represents the top-left and bottom-right corner of the proposal.

Once established the output dimension of this layer (that are the pooled width (pool _w) and the pooled height (pool_h)), every ROI (that has size of W x H) will be divided into pool_w * pool_h

blocks, each of dimension of $\frac{W}{pool\_w} * \frac{H}{pool\_h}$

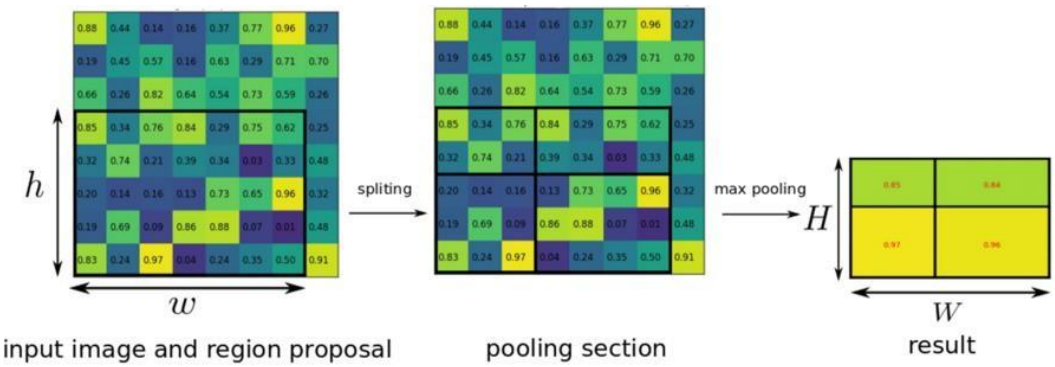


Figure 10 ROI Pooling

Suppose we got the region proposal from with size h*w (left) and we would like to have an output layer with size H*W (right). Then, the area of each pooling area (middle) will be = $\frac{h}{H} * \frac{w}{W}$.

In the example above we have an input ROI of 5*7 and an output size of 2*2, then each pooling area will have an area of 3*2, 3*3, 4*2 or 4*3 after rounding.

The pooling layer in its entirety is shown on Figure 12:

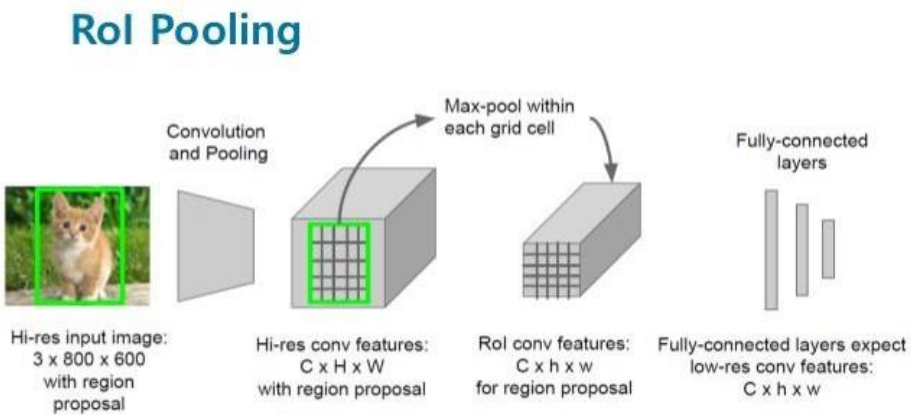


Figure 11 RoI Pooling Layer

We need to define the loss function to be used during training time. We will use a Multi-task loss function that combines Log Loss of classification as a classification function loss and Smooth L1 loss of bounding-box regression as a bounding-box function loss:

$$\mathcal{L}(p, u, t^u, v) = \mathcal{L}_{\text{cls}}(p, u) + 1[u \geq 1]\mathcal{L}_{\text{box}}(t^u, v)$$
$$\mathcal{L}_{\text{cls}}(p, u) = -\log p_u$$
$$\mathcal{L}_{\text{box}}(t^u, v) = \sum_{i \in \{x,y,w,h\}} L_1^{\text{smooth}}(t_i^u - v_i)$$

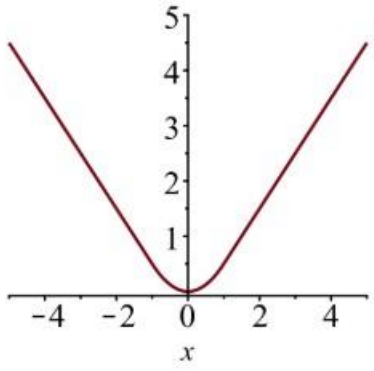
Symbol	Explanation
u	True class label, $u \in 0, 1, \ldots, K$; by convention, the catch-all background class has $u = 0$.
p	Discrete probability distribution (per RoI) over K + 1 classes: $p = (p_0, \ldots, p_K)$, computed by a softmax over the K + 1 outputs of a fully connected layer.
v	True bounding box $v = (v_x, v_y, v_w, v_h)$.
t^u	Predicted bounding box correction, $t^u = (t_x^u, t_y^u, t_w^u, t_h^u)$.

The indicator function is defined as:

$$1[u >= 1] = \begin{cases} 1 & \text{if } u \geq 1 \\ 0 & \text{otherwise} \end{cases}$$

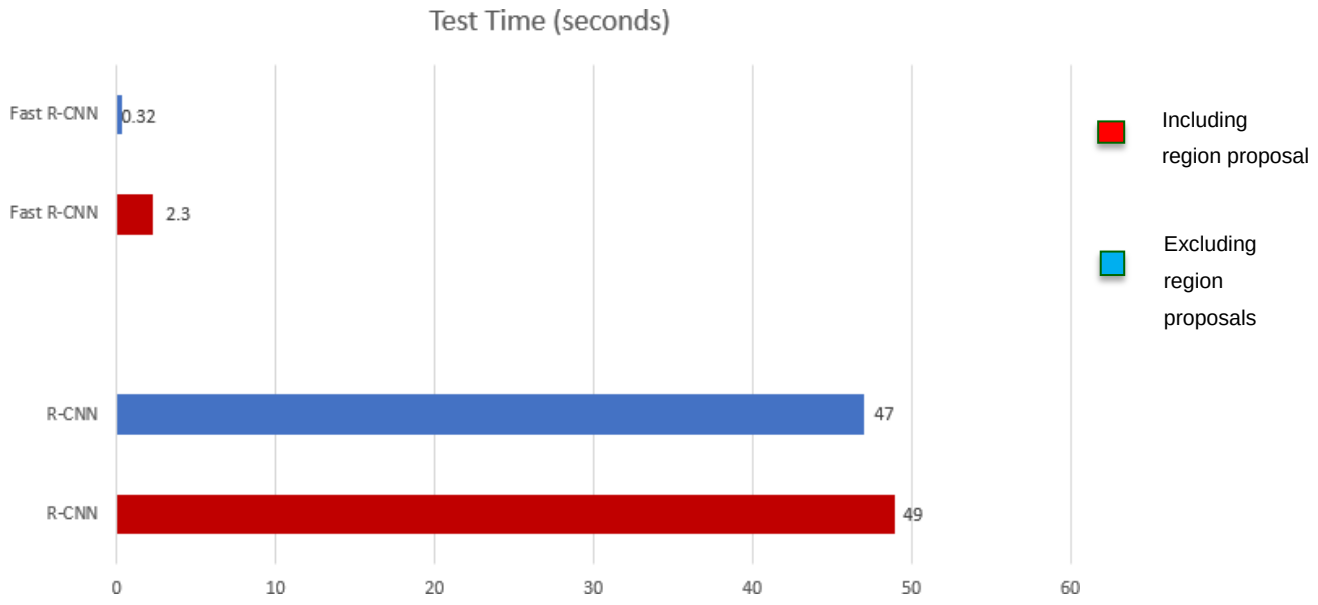
The bounding box loss function measure the difference between the predicted bounding box correction and the true bounding box using a smooth L1 loss, since this type of loss function is claimed to be less sensitive to outliers:

$$L_1^{\text{smooth}}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$



The following table shows the results obtained by applying these two models trained with VOC 2007 train set (the results are then shown in the next graph):

	R-CNN	**Fast R-CNN**
Training time	84 hours	9.5 hours
(Speedup)	1x	8.8x
Test time per image	47 seconds	0.32 seconds
(Speedup)	1x	146x
Test time per image with Selective Search	50 seconds	2 seconds
(Speedup)	1x	25x
mAP(tested with VOC 2007 test set)	66.0	66.9



From these results, we can see that with Fast R-CNN there is a huge training and test time improvement and a small improvement on mAP, but the use of Region Proposals constitutes a bottleneck.

5. Faster R-CNN

The idea in faster R-CNN is to make the network itself predict its own region proposals, instead of compute them using a fixed algorithm (since it was a bottleneck).

In this architecture, we take the input image and apply the CNN in order to generate a feature map and then, instead of using selective search algorithm on this feature map to identify the region proposals, a separate Region Proposal Network is used to predict them. The predicted region proposals are then reshaped using a RoI pooling layer which is then used to classify the image within the proposed region and predict the offset values for the bounding boxes.

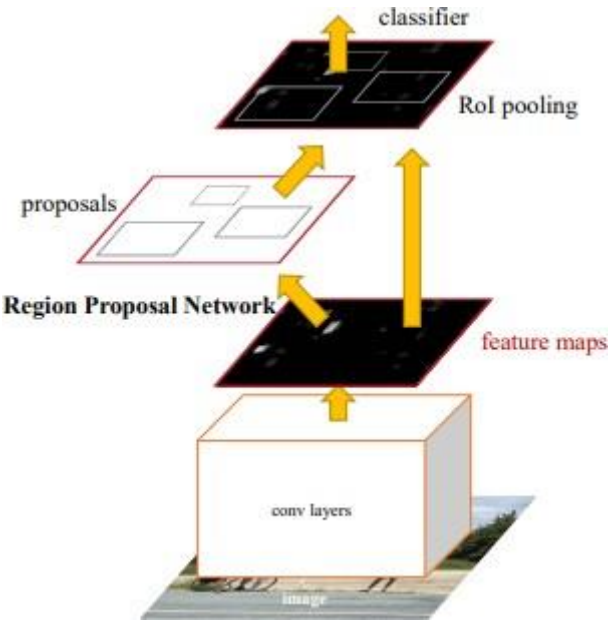


Figure 12 Faster R-CNN

The goal of RPN is to output a set of proposals, each of which has a score of its probability of being an object and also the class/label of the object.

The RPN that generates the proposals slide a small network over the output of the last layer of the feature map and the position of the sliding window provides localization information with reference to the image while the regression provides finer localization information.

One of the most important concepts of RPN are the Anchor Boxes which are responsible for providing a predefined set of bounding boxes of different sizes and ratios that are used for reference when first predicting object locations for the RPN. These boxes are defined to capture the scale and aspect ratio of specific object classes you want to detect and are typically chosen based on object sizes in the training dataset and they are typically centered at the sliding window.

The original implementation uses 3 scales (size) and 3 aspect ratios (width of image / height of image) , which means k=9 proposals are possible for each pixel. If the final feature map from feature extraction layer has width W and height H , then the total number of anchors generated will be W*H*k.

These proposals are given as input to 2 fully connected layers, one for bounding box regression and the other for box classification.

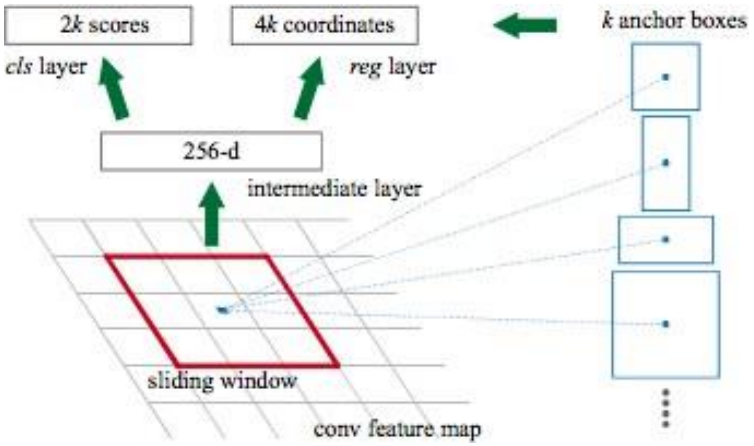


Figure 13. RPN architecture

Faster R-CNN is optimized for a multi-task loss function, similar to Fast R-CNN

We will use a slightly different Multi task loss function that combines Log Loss of classification and Smooth L1 loss of bounding-box regression:

$$\mathcal{L}(\{p_i\},\{t_i\}) = \frac{1}{N_{\text{cls}}}\sum_i \mathcal{L}_{\text{cls}}(p_i, p_i^*) + \frac{\lambda}{N_{\text{box}}}\sum_i p_i^* \cdot L_1^{\text{smooth}}(t_i - t_i^*)$$

Symbol	Explanation
p_i	Predicted probability of anchor i being an object.
p_i^*	Ground truth label (binary) of whether anchor i is an object.
t_i	Predicted four parameterized coordinates.
t_i^*	Ground truth coordinates.
N_{cls}	Normalization term, set to be mini-batch size (~256) in the paper.
N_{box}	Normalization term, set to the number of anchor locations (~2400) in the paper.
λ	A balancing parameter, set to be ~10 in the paper (so that both $\mathcal{L}_{\text{cls}}$ and $\mathcal{L}_{\text{box}}$ terms are roughly equally weighted).

The log loss function for classification can be defined starting from log loss function over two classes, as we can easily translate a multi-class classification into a binary classification by predicting a sample being a target object versus not.

$$\mathcal{L}_{\text{cls}}(p_i, p_i^*) = -p_i^* \log p_i - (1 - p_i^*) \log(1 - p_i)$$

The bounding box loss is the same as before.

In the end, we will have the following situation:

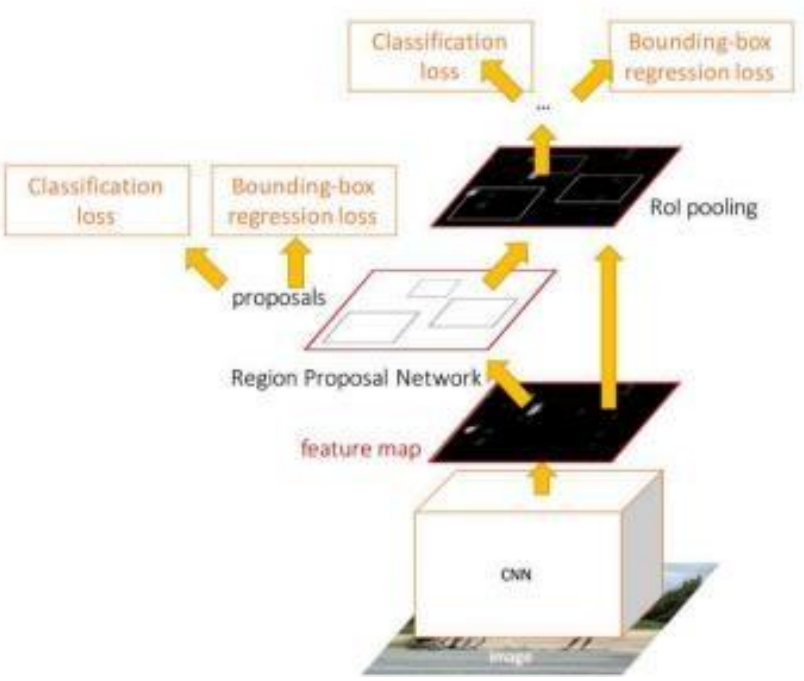


Figure 14. Training for Faster CNN

The following table shows the results obtained by applying Faster R-CNN trained with VOC 2007 train set, compared with previous results for R-CNN and Fast R-CNN:

	R-CNN	**Fast R-CNN**	**Faster R-CNN**
Test time per image (with proposals)	50 seconds	2 seconds	0.2 seconds
(Speedup)	1x	25x	250x
mAP (tested with VOC 2007 test set)	66.0	66.9	66.9

As you can see, the mAP remains the same but the test time has decreased a lot.

A summary of all the models of R-CNN family considered so far is shown below:

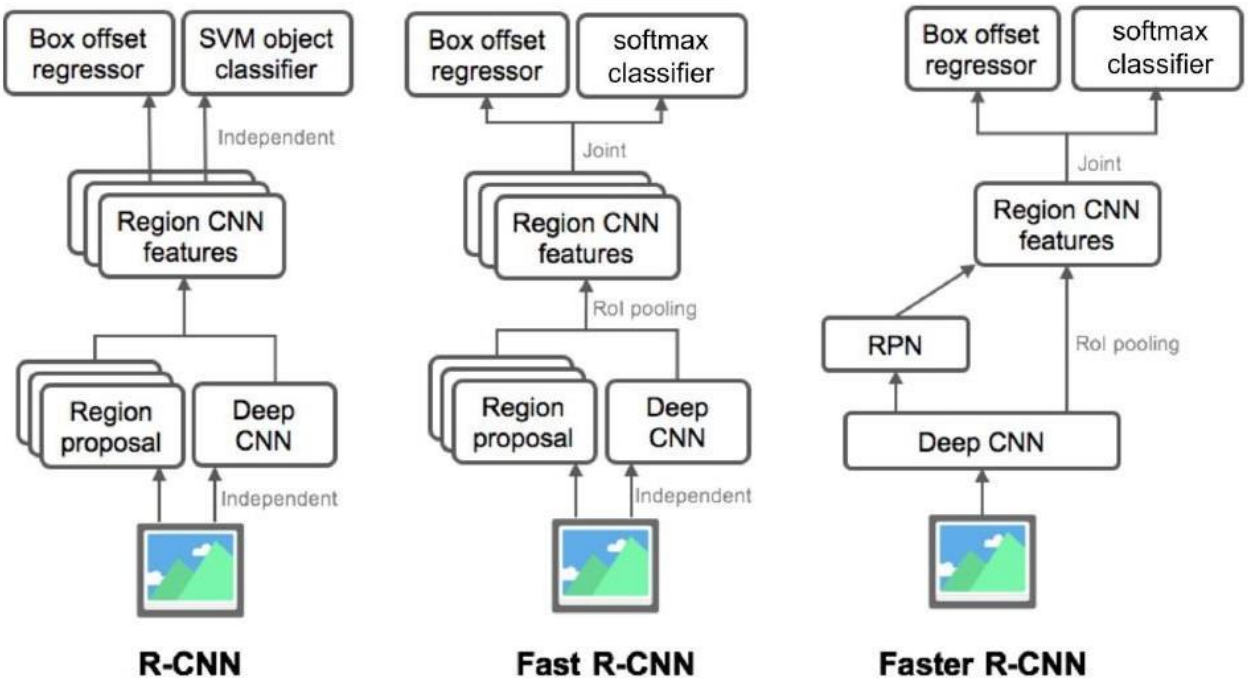


Figure 15. Summary of R-CNN family

Even though Faster R-CNN has good accuracy, the whole process runs at 7 frames per second, not the best for real time applications. So we need a faster model.

6. SSD (Single Shot Detector)

The R-CNN family is a multi-stage detector, a detector where the network first predict the object presence score of the bounding box and then pass this box through a classifier to predict the class probability.

There is a type of detector called ‘single-stage detector’ where the convolutional layers make both predictions directly in one shot, the Single Shot Detector (SSD) is one of this.

The SSD approach is based on a feed-forward convolutional network that produces a fixed-size collection of bounding boxes and the relative scores for the presence of an object in those boxes, followed by a non-maximum suppression step to produce the final detections. This approach speeds up the process by eliminating the region proposal network.

The architecture of the SSD model is composed of three main parts:

1. *Base network to extract feature maps*: a standard pretrained network used for high quality image classification and truncated before any classification layers (e.g. VGG16 network).
2. *Multi-scale extra feature layers*: a series of convolution filters added after the base network. These layers decrease in size progressively to allow predictions of detections at multiple scales.
3. *Non-maximum suppression* to eliminate overlapping boxes and keep only one box for each object detected.

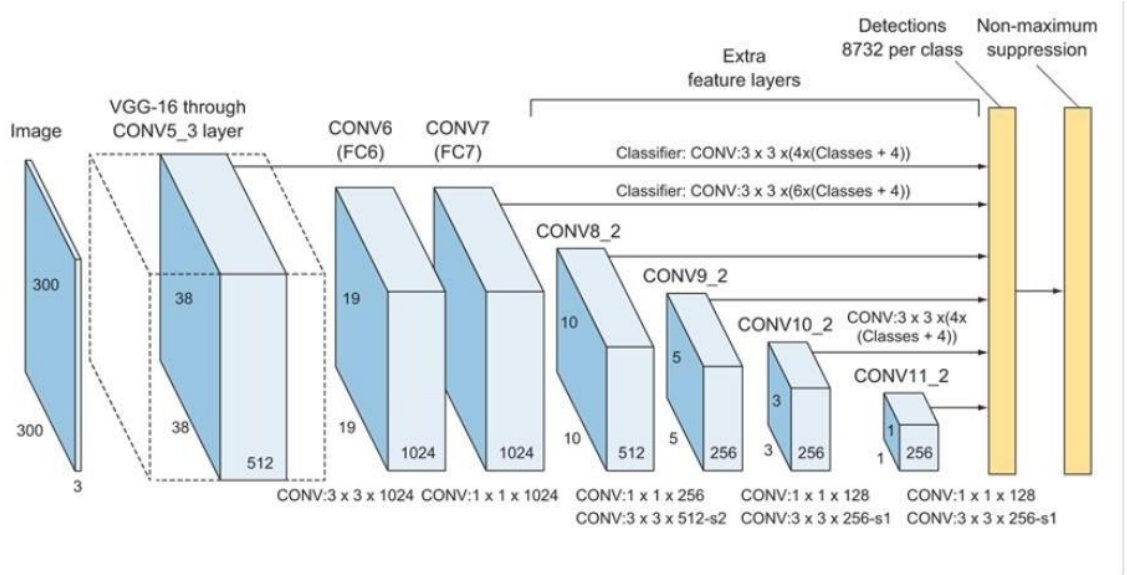


Figure 16. SSD Architecture

SSD does not use a delegated region proposal network but it computes both the location and class scores using small convolution filters. After extracting the feature maps, SSD applies 3 × 3 convolution filters for each cell to make predictions (these filters compute the results just like the regular CNN filters.)

In fact, given an input image and the goal of finding all the cones with its color and height the detection process goes as follows:

1. Similar to the Anchors concept in R-CNN, SSD overlays a grid of Anchors around the image and for each anchor, the network will create bounding boxes at its center.
2. The base network looks at each bounding box as a separate image. Within each bounding box, the network will ask the question: is there a cone in this box? In other words, it will ask: did I extract any features of a cone in this box?
3. When the network finds a bounding box that contains cone features, it sends its coordinates prediction and object classification to the non-maximum suppression layer.
4. Non-maximum suppression will then eliminate duplicate predictions pointing to the same object. First we sort the predictions by the confidence score and, starting from the top confidence prediction, evaluates whether any previously predicted boundary boxes have an IoU higher than 0.5 with the current prediction for the same class. If found, the current prediction will be ignored. In this way, we will eliminate different detection for the same object (the ones with IoU higher than 0.5) without removing the candidates for a different object (the ones with IoU less than 0.5).
5.

The training process involves the matching strategy, the choice of default boxes, the hard negative mining, the loss function and data augmentation strategies. Each of this process are discussed below:

Matching strategy: During training we need to determine which default boxes correspond to a ground truth detection and train the network accordingly. First, each ground truth box is matched to the default box having the maximum jaccard overlap. Then, each default box is matched to any ground truth box having a jaccard overlap > 0.5

***Default boxes choice*:** SSD relies heavily on default boxes, so it is very sensitive to their choice (specifically their scale and aspect ratios). In the paper, the authors design the default boxes such that each feature map corresponds to a specific scale of default boxes along with a list of aspect ratios for each scale. The minimum and maximum scales, s_min and s_max, are set to 0.2 and 0.9. For the kth feature map (among m feature maps), the scale is chosen as:

$$s_k = s_{min} + \frac{s_{max} - s_{min}}{m - 1} (k - 1)$$

For each feature map, the heights and widths of the boxes are computed as:

$$h = \frac{s_k}{\sqrt{a_r}}$$

$$w = s_k * \sqrt{a_r}$$

with $\alpha_r \epsilon \{1,2,3,\frac{1}{2},\frac{1}{3}\}$

Plus, for aspect ratio of 1, an additional scale s^* is added:

$$s^* = \sqrt{s_k * s_{k+1}}$$

resulting in 6 default boxes per feature map location. For layer conv4_3, conv10_2 and conv11_2 we only calculate 4 default boxes, omitting aspect ratio of $\frac{1}{3}$ and 3

Loss function: The default boxes that didn't get matched to any ground truth box are considered negative and contribute only to the confidence loss, whereas all the positively matched boxes add to the confidence loss as well as the localization loss. The confidence loss is essentially the cross-entropy loss while the localization loss is the Smooth L1 loss between the predicted box (*l*) and the ground truth box *(g*) parameters.

The localization loss is computed as:

$$L_{loc}(x, l, g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx,cy,w,h\}} x_{ij}^k \text{smooth}_{\text{L1}}(l_i^m - \hat{g}_j^m)$$

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \qquad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$

$$\hat{g}_j^w = \log\left(\frac{g_j^w}{d_i^w}\right) \qquad \hat{g}_j^h = \log\left(\frac{g_j^h}{d_i^h}\right)$$

with:

- *cx* and *cy* = offset to the default bounding box *d* of width *w* and height *h.*
- x_{ij}^p = *1* if the *IoU* between default box *i* and ground true box *j* on class *p* is greater than 0.5, 0 otherwise

The confidence loss is computed as:

$$L_{conf}(x,c) = -\sum_{i\in Pos}^{N} x_{ij}^{p} log(\hat{c}_i^p) - \sum_{i\in Neg} log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

The final loss is given by a combination of both the losses where α is the relative weighing of the localization loss and N is the number of positively matched boxes.

$$L(x,c,l,g) = \frac{1}{N}(L_{conf}(x,c) + \alpha L_{loc}(x,l,g))$$

Hard negative mining: Since after the matching step , most of the default boxes are negative, there is a huge imbalance between the positive and negative training examples (with the result that we are training the model to learn background instead of detecting object). So instead of using all the negatives, we sort those negative by their calculated confidence loss and picks the negative instances with top loss. Also, we make sure that the ratio between negative examples and positive examples is 3:1.

Data augmentation: In order to handle variance of object sizes and shapes, each training image is randomly sampled by one of the 3 following options:

- Use the original,
- Sample a patch so that the minimum Jaccard overlap with the objects is 0.1, 0.3, 0.5, 0.7 or 0.9 . The size of each sampled patch is [0.1,1] of the original image size and aspect ratio is between ½ and 2.
- Randomly sample a patch.

Then each sampled patch is resized to a fixed size and horizontally flipped with probability 0.5.

The impact of data augmentation and aspect ratio choices is evaluated in the paper, obtaining the following results on VOC 2007 dataset:

	SSD300				
more data augmentation?		✔	✔	✔	✔
include $\{\frac{1}{2}, 2\}$ box?	✔		✔	✔	✔
include $\{\frac{1}{3}, 3\}$ box?	✔			✔	✔
use atrous?	✔	✔	✔		✔
VOC2007 `test` mAP	65.5	71.6	73.7	74.2	**74.3**

Figure 17 Impact of design choices

As we can see, data augmentation leads to a big improvement because it allows better accuracy on small object.

The performance of SSD against Fast-RCNN on VOC 2007 dataset are showed in the next table:

Method	**mAP**	**FPS**
Faster R-CNN	73.2	7
SSD	74.3	46

This approach is definitely one of the candidates to solve the cone detection task, thanks to its good accuracy and high speed.

Maybe using another image classification network as Inception v3, Inception v4 could lead to better accuracy as well as using more default boxes.

Another strategy to improve performance could be the study of a better distribution of default boxes that fit a specific dataset.

7. YOLO v3

In this approach, we only predicts over a limited number of bounding boxes by splitting the input into a grid of cells and each cell directly predicts a bounding box and object classification. The result is a large number of candidate bounding boxes that are consolidated into a final prediction using non-maximum suppression. This leads to a much faster model, with some losses on accuracy.

The YOLO network splits the input image into a grid of S×S cells and each grid cell predicts B number of bounding boxes and their objectness score along with their class predictions as follows:

- Similar to previous detectors, YOLO predicts 4 coordinates for each B bounding box (bx,by,bw,bh). Where x and y are set to be offset of a cell location.
- It detects one objectness score through logistic regression in order be treated as a probability with a value range between 0 and 1
- If the bounding box contains an object, the network predicts the class probability of each of the K classes as $P(class_i|object)$

So for each bounding box, the prediction have:

- 4 values representing the bounding box coordinates
- 1 value representing the objectness score
- K class probabilities

Resulting in 5B + K values predicted for all bounding boxes of a cell. Then, if we have a SxS grid, the total number predicted is S*S*(5B +K).

In order to find the anchor boxes, the paper suggest to run k-means clustering (with k = 5) on the training set bounding boxes using the IoU as part of a distance metrics, defined as:

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

For the boundary box prediction, rather than directly predicting the bounding box dimensions (like in YOLO v1) , we simply predict the offset from our anchor box dimensions such that we can fine-tune our predicted bounding box dimensions. This reformulation makes the prediction task easier to learn. Figure 18 illustrates this process:

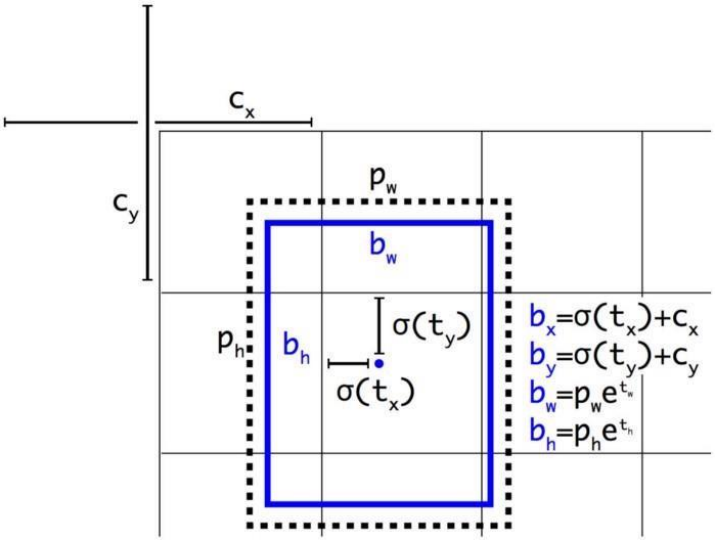


Figure 18. Bounding box predictions

with:

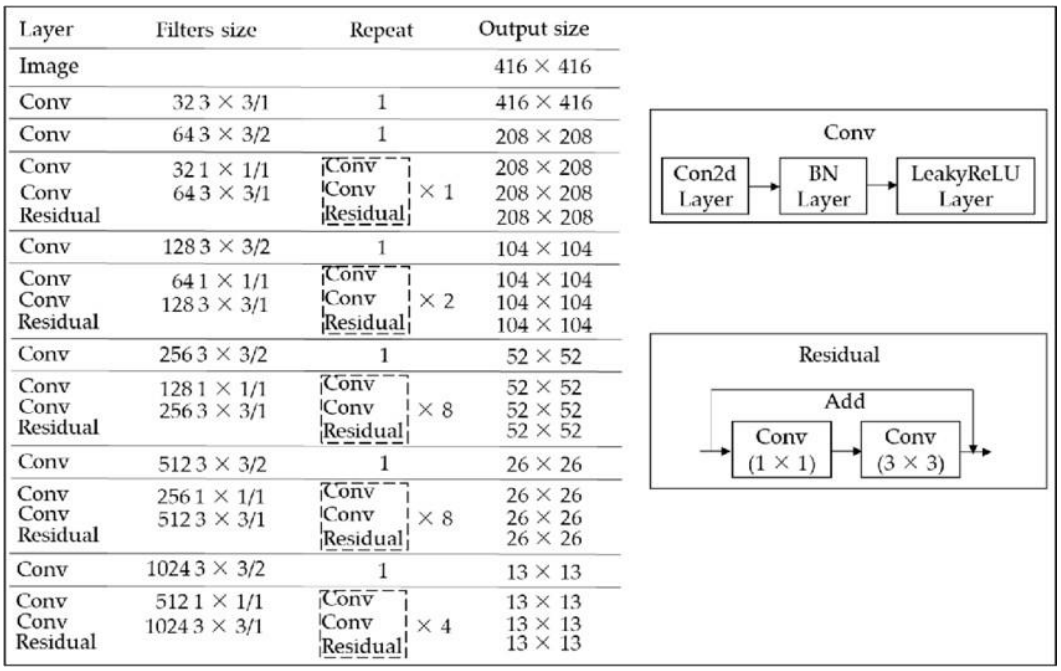
t_x, t_y, t_w , t_h are predictions made by YOLO

c_x, c_y is the top left corner of the grid cell of the anchor

p_w, p_h are the width and height of the anchor

b_x, b_y, b_w, b_h are the predicted boundary box

YOLO uses a variant of “Darknet-53” as feature extractor, which is mainly compose of 3 × 3 and 1× 1 filters with skip connections like the residual network in ResNet. The detailed structure is showed in Figure 19:



Layer	Filters size	Repeat	Output size
Image			416 × 416
Conv	32 3 × 3/1	1	416 × 416
Conv	64 3 × 3/2	1	208 × 208
Conv Conv Residual	32 1 × 1/1 64 3 × 3/1	Conv Conv Residual × 1	208 × 208 208 × 208 208 × 208
Conv	128 3 × 3/2	1	104 × 104
Conv Conv Residual	64 1 × 1/1 128 3 × 3/1	Conv Conv Residual × 2	104 × 104 104 × 104 104 × 104
Conv	256 3 × 3/2	1	52 × 52
Conv Conv Residual	128 1 × 1/1 256 3 × 3/1	Conv Conv Residual × 8	52 × 52 52 × 52 52 × 52
Conv	512 3 × 3/2	1	26 × 26
Conv Conv Residual	256 1 × 1/1 512 3 × 3/1	Conv Conv Residual × 8	26 × 26 26 × 26 26 × 26
Conv	1024 3 × 3/2	1	13 × 13
Conv Conv Residual	512 1 × 1/1 1024 3 × 3/1	Conv Conv Residual × 4	13 × 13 13 × 13 13 × 13

Figure 19. DarkNet structure

For the task of detection, 53 more layers are stacked onto it, giving us a 106 layer fully convolutional underlying architecture for YOLO v3

In order to handle different scales, YOLOv3 makes predictions in 3 different times:

1. In the last feature map layer.
2. Then it goes back 2 layers back and upsamples it by 2. YOLOv3 then takes a feature map with higher resolution and merge it with the upsampled feature map using element-wise addition. YOLOv3 apply convolutional filters on the merged map to make the second set of predictions.
3. Repeat 2 again so the resulted feature map layer has good high-level structure (semantic) information and good resolution spatial information on object locations.

The whole process can be viewed in Figure 20:

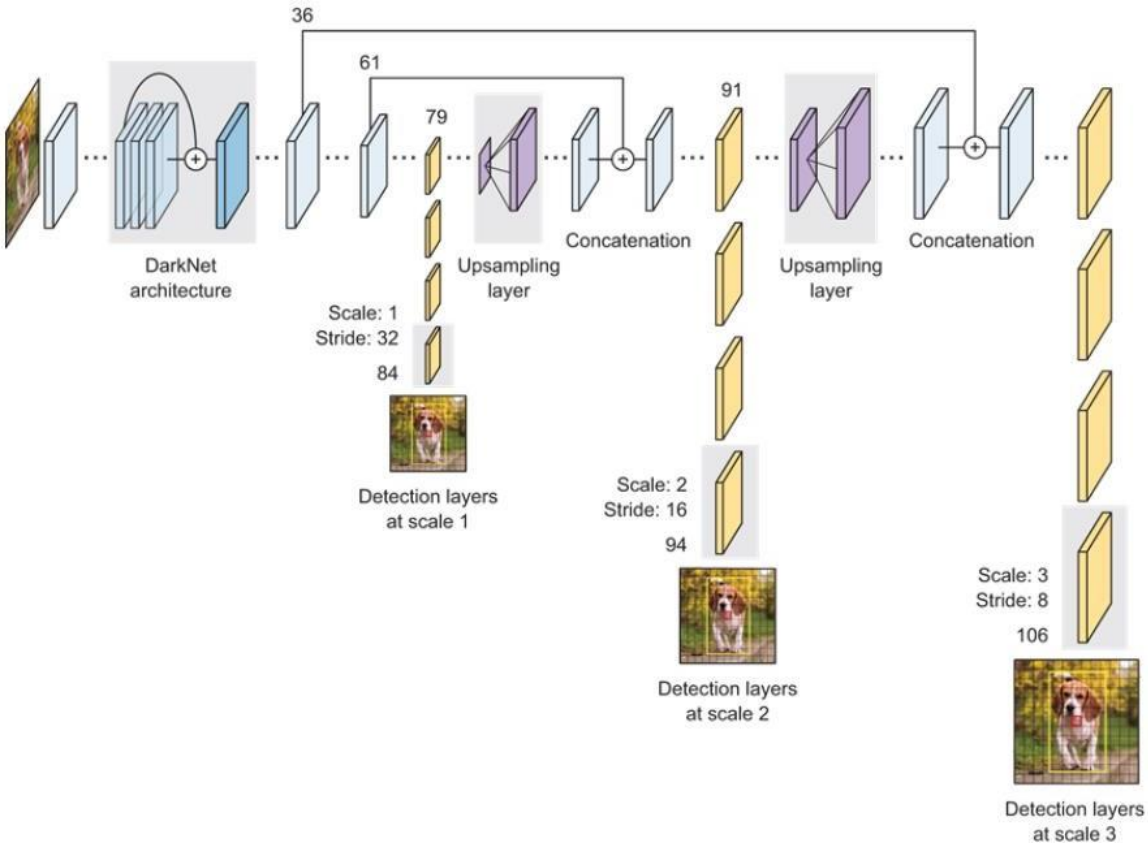


Figure 20. Multi Scale Detection

The input image (size 416x416) goes through Darknet-53 feature extractor , then it is down-sampled by the network until layer 79. Then the network branches out and continues to down-sample the image until it makes its first prediction at layer 84. This detection is made on a grid scale of 13x13 which is responsible for detecting large objects.

Then the feature map from layer 79 is then up-sampled by 2x to the dimensions of 26x26 and then concatenated with the feature map from layer 61. Then, the second detection is made by layer 94 on a grid scale of 26x26 which is responsible for detecting medium objects.

Finally, a similar procedure is followed again, where the feature map from layer 91 is subjected to few up-sampling convolutional layers before being depth concatenated with a feature map from layer 36. Then, a third prediction is made by layer 106 on a grid scale of 52x52 which is responsible for detecting small objects.

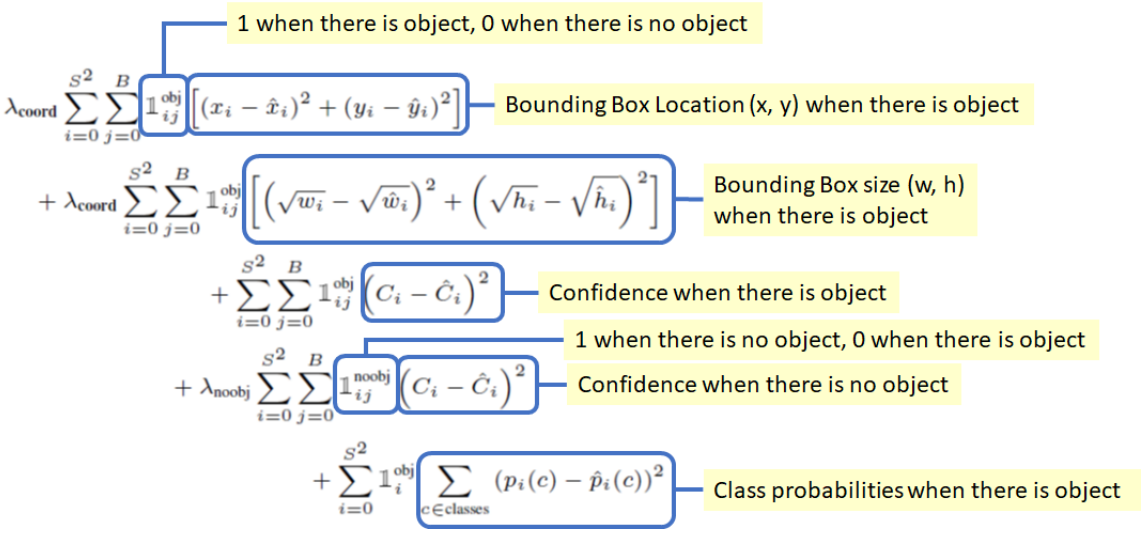
One of the most common problems with object detection algorithms is that rather than detecting an object just once, they might detect it multiple times.

The Non-Max Suppression technique cleans up this up so that we get only a single detection per object:

1. Discard all the boxes having probabilities (objectness score) less than or equal to a pre-defined threshold (say, 0.5)
2. For the remaining boxes:
    a) Pick the box with the highest probability and take that as the output prediction
    b) Discard any other box which has IoU greater than the threshold with the output box from the above step
3. Repeat step 2 until all the boxes are either taken as the output prediction or discarded

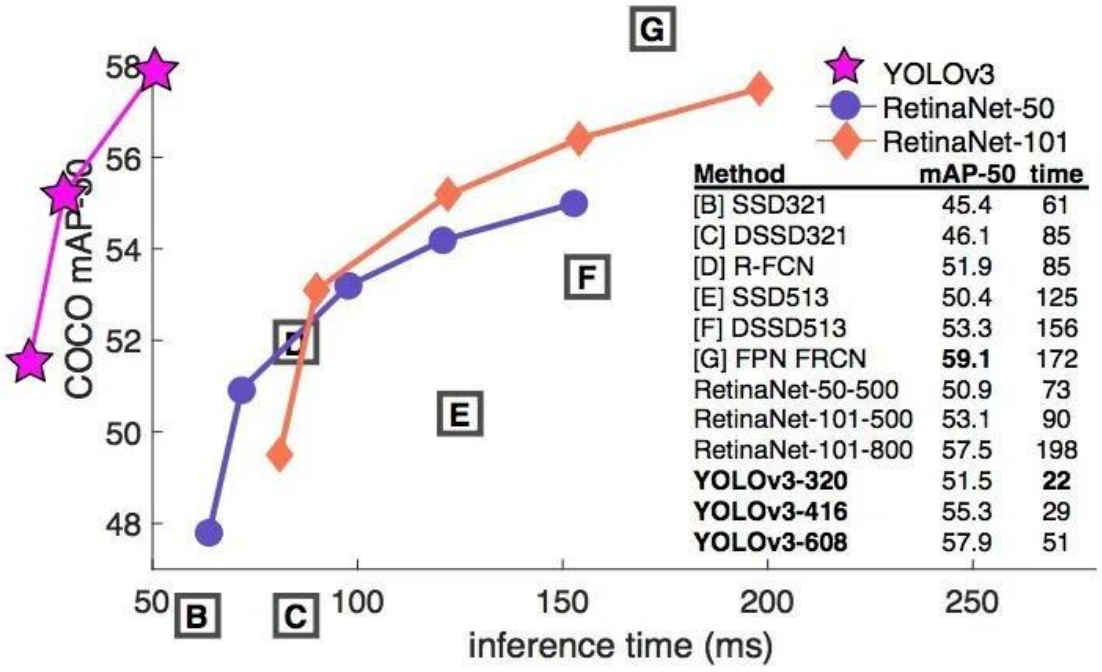
We repeat these steps until all the boxes have either been selected or compressed and we get the final bounding boxes

YOLO Loss function is defined as:



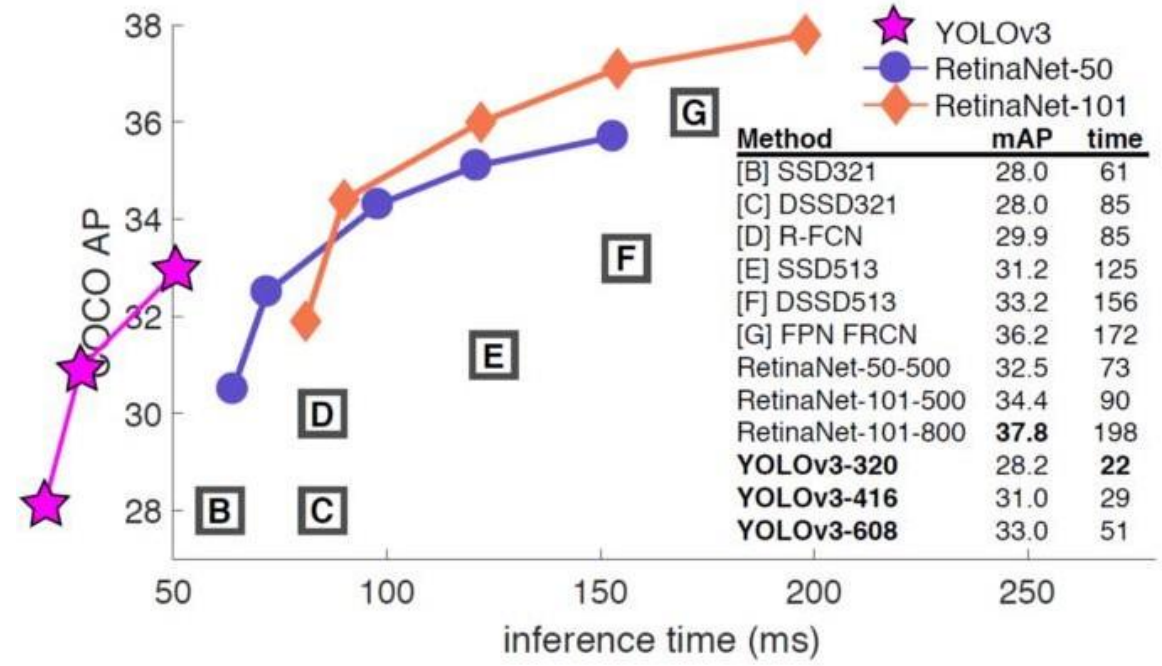
In YOLOv3, the first two term are replaced with the sum of squared error loss of bounding box location and bounding box size

In Figure 21 performance is analyzed. As you can see, YOLO v3 is better than SDD for mAP metric and is much faster.



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

Figure 21. Yolo results



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

Figure 22 Yolo results

8. YOLO v4

This architecture introduces a series of optimization in order to archive better accuracy and speed.

For the backbone, CSPDarknet53 was chosen over CSPResNext50 and EfficientNet

B3. The comparison between the backbones is showed in Figure 23:

Backbone model	Input network resolution	Receptive field size	Parameters	Average size of layer output (WxHxC)	BFLOPs (512x512 network resolution)	FPS (GPU RTX 2070)
CSPResNext50	512x512	425x425	20.6 M	**1058 K**	31 (15.5 FMA)	62
CSPDarknet53	512x512	725x725	**27.6 M**	950 K	52 (26.0 FMA)	**66**
EfficientNet-B3 (ours)	512x512	**1311x1311**	12.0 M	668 K	11 (5.5 FMA)	26

Figure 23 Backbones comparison

CSPDarknet53 was chosen for its balance between Receptive field size, the number of parameters ($width\ of\ kernel^2 * number\ of\ channel\ of\ input\ image * number\ of\ kernels$) and for the average size of layer output. This choice is also confirmed by experimental results on a RTX 2070 GPU.

A Spatial Pyramid Pooling (SPP) and a Path Aggregation Network (PAN) was added on top of the backbone:

- SPP is a pooling technique designed to handle multi-scale images. The technique performs pooling (ex: Max pooling) on the last convolution layer (either convolution or sub sampling) and produces a N*B dimensional vector (where N=Number of filters in the convolution layer, B= Number of Bins). The vector is then fed to the FC layer. Since the number of bins is a constant value, the vector dimension remains constant irrespective of the input image size. The whole process is showed on Figure 24:

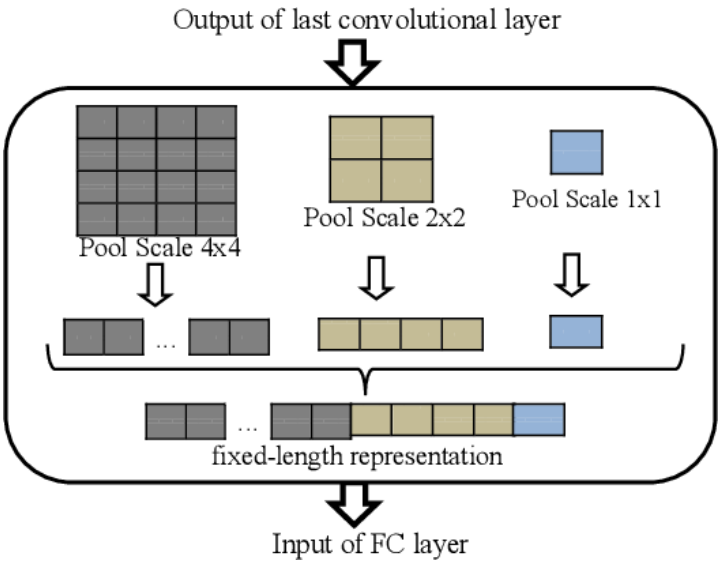


Figure 24. SPP

- PAN is a method of parameter aggregation similar to Feature Pyramid Network (which is the top-down pathway to combine multi-scale features used in YOLO v3). PAN adds an extra bottom up path aggregation network on top of FPN, as showed in Figure 25.1 and Figure 25.2:

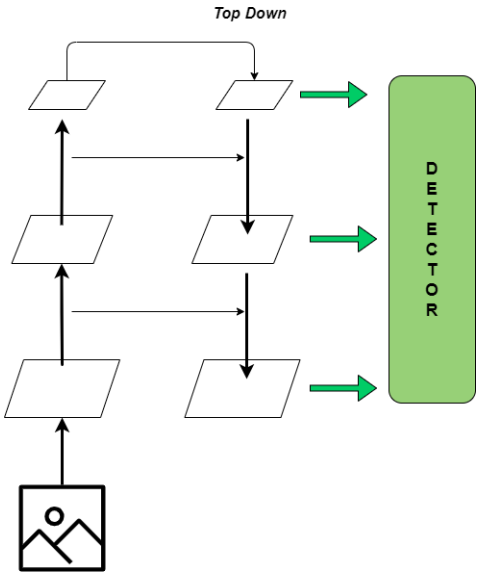


Figure 25.1. Feature Pyramid Network (FPN)

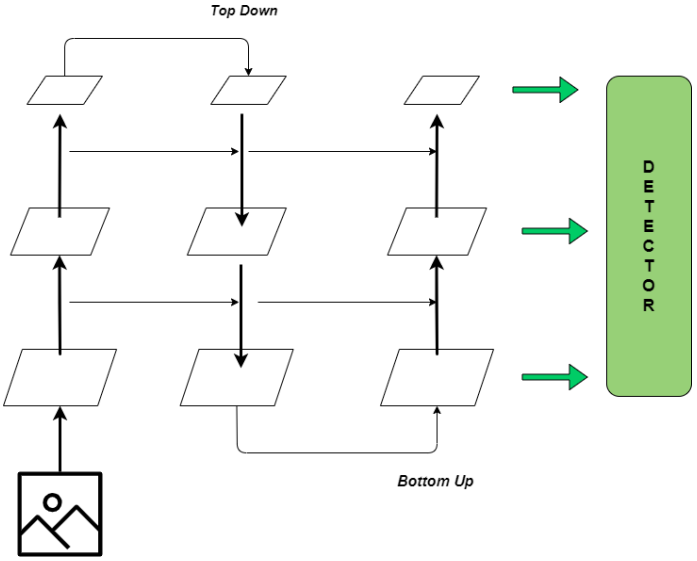


Figure 25.2. Path Aggregation Network (PAN)

The authors modified PAN replacing the addition with concatenation, as showed in Figure 26:

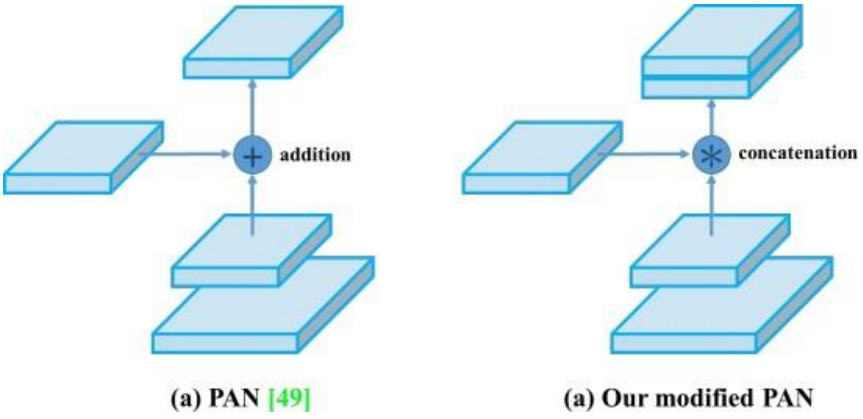


Figure 26 PAN vs Modified PAN

So the final architecture is the following:

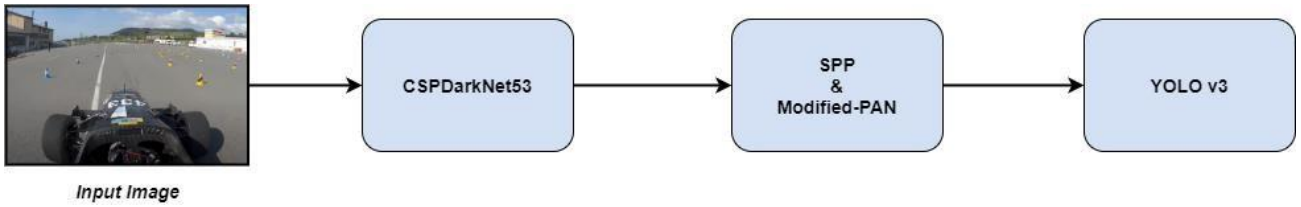


Figure 27. YOLO v4 architecture

The authors of YOLO v4 paper distinguish between two categories of methods that are used to improve the object detector's accuracy. These both categories are:

- Bag of freebies (BoF): Methods that can make the object detector receive better accuracy without increasing the inference cost. These methods only change the training strategy or only increase the training cost.
- Bag of specials (BoS): Those plugin modules and post-processing methods that only increase the inference cost by a small amount but can significantly improve the accuracy of object detection

The overview of the BoF and BoS used is showed in Figure 28:

	Backbone	Detector
Bag of Freebies (BoF)	• CutMix • Mosaic data augmentation • DropBlock • Class label smoothing	• CIoU-loss • Cross mini-Batch Normalization • DropBlock • Mosaic data augmentation • Self-Adversarial Training • Multiple anchors for a single ground truth • Cosine annealing scheduler • Optimal hyperparameters • Random training shapes
Bag of Specials (BoS)	• Mish activation • Cross-stage partial connections (CSP) • Multi-input weighted residual connections (MiWRC)	• Mish activation • SPP-block • SAM-block • PAN path-aggregation block • DIoU-NMS

Figure 28. BoF and BoS

Here are the details of some of these techniques:

- *CutMix*: augmentation technique in which we cut and paste random patches between the training images. The ground truth labels are mixed in proportion to the area of patches in the images. CutMix increases localization ability by making the model to focus on less discriminative parts of the object being classified and hence is also well suited for tasks like object detection.
- *Mosaic data augmentation*: new data augmentation method introduced by authors that mixes 4 training images. Thus 4 different contexts are mixed, while CutMix mixes only 2 input images. By doing this now, batch normalization calculates activation statistics from 4 different images on each layer. And so, it greatly reduces the need of selecting a large mini-batch size during training.
- *Self-Adversarial Training*: new data augmentation method introduced by authors in order to defend the model from adversarial attack. This technique operates in 2 forward backward stages. In the 1st stage the neural network alters the original image instead of the network weights. In this way the neural network executes an adversarial attack on itself, altering the original image to create the deception that there is no desired object on the image. In the 2nd stage, the neural network is trained to detect an object on this modified image in the normal way. This makes the model partially more robust, but it can increase the training complexity, and in some cases it fails completely.
- *Class Label Smoothing*: regularization technique that introduces noise for the labels. This accounts for the fact that datasets may contains mistakes in them. Assume for a small constant ε , the training set label y is correct with probability $1 - \varepsilon$ and incorrect otherwise.

Label Smoothing regularizes a model based on a softmax with *k* output values by replacing the hard 0 and 1 classification targets with targets of $\frac{\varepsilon}{k-1}$ and $1 - \varepsilon$ respectively.

- *Mish activation function*: defined as $f(x) = x * tanh * (\ln(1 + e^x))$ which reported an increase in Top-1 test accuracy by 0.9% and an increase in Top-5 test accuracy by 0.4% as compared to the same network with Leaky-ReLU..
- *Spatial Attenction Module (SAM)*: attention mechanism which take in input a feature map F', performs average-pooling and max-pooling operations along the channel axis and then concatenated them. Then a convolution layer is applied (with sigmoid as activation function) to generate an attention map (Ms), which is applied to the original F'. The author modified SAM so that doesn't apply max-pooling and average-pooling, but instead F' goes through a conv. layer (with sigmoid activation) which then multiplies the original feature map (F') (as illustrated in Figure 29).

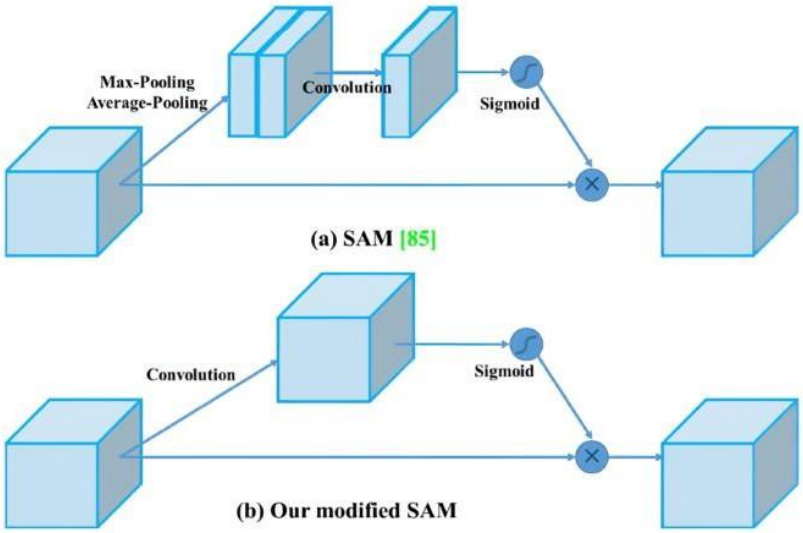


Figure 5: Modified SAM.

Figure 29. SAM vs Modified SAM

Figure 30-33 shows the results of YOLO v4 compared with other detectors using different GPU architecture (Maxwell, Pascal, Volta and Tesla V100):

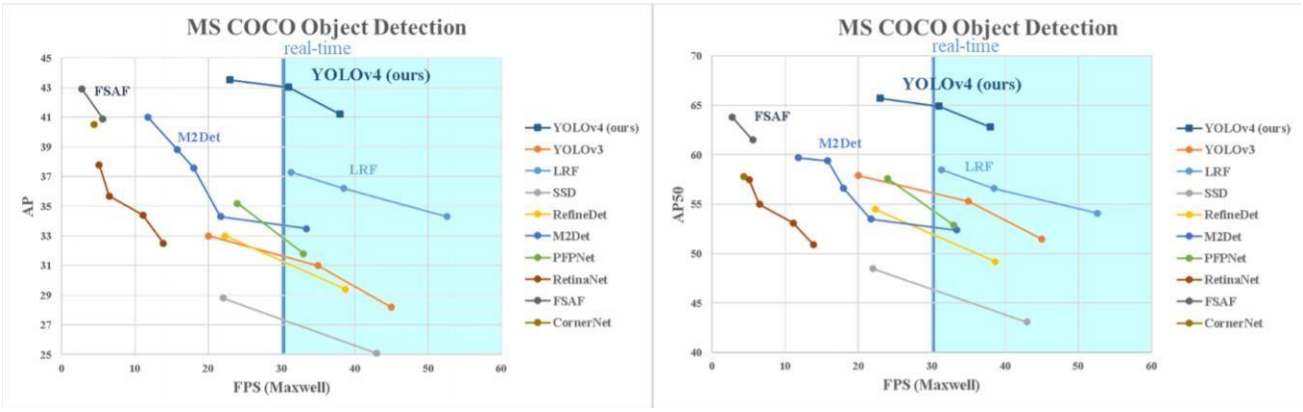


Figure 30. Performance of YOLO v4 vs other Detectors on Maxwell architecture

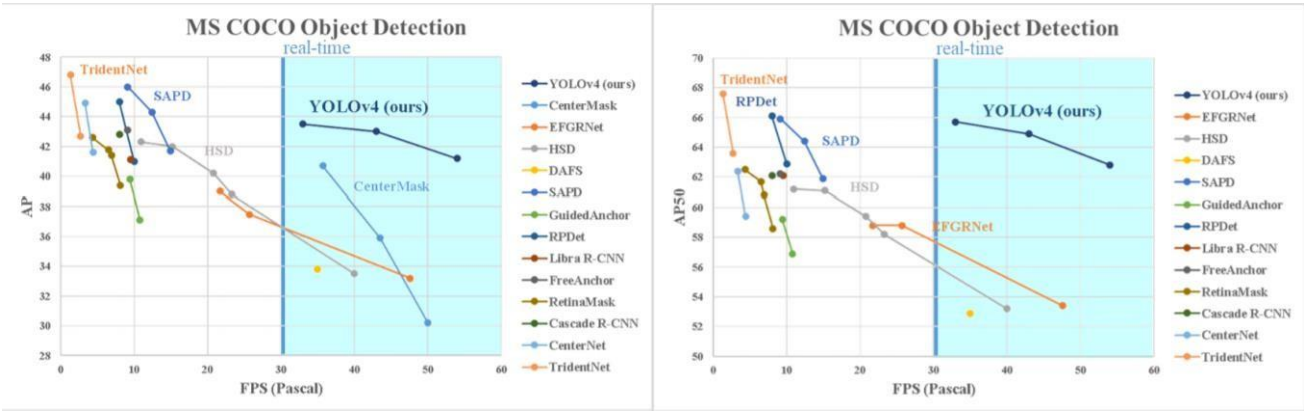


Figure 31. Performance of YOLO v4 vs other Detectors on Pascal architecture

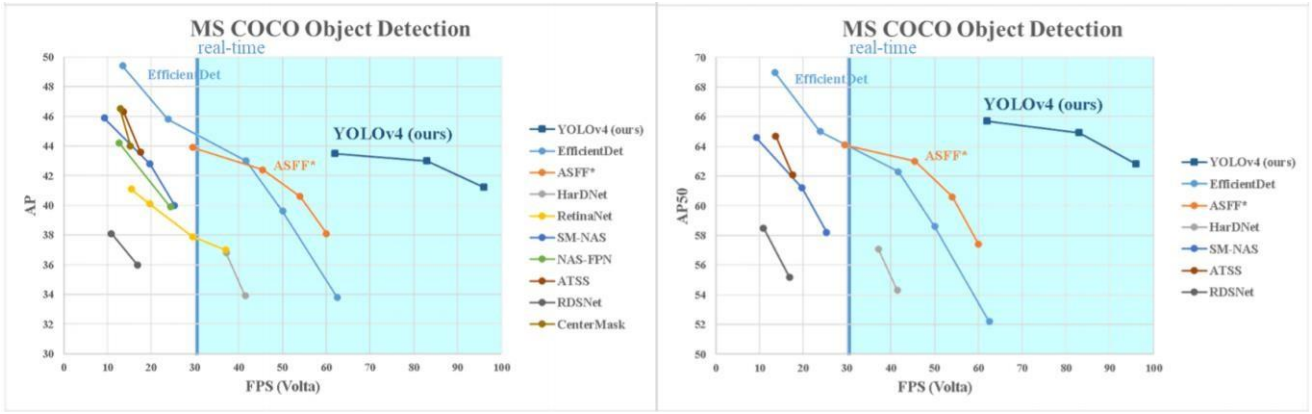


Figure 32. Performance of YOLO v4 vs other Detectors on Volta architecture

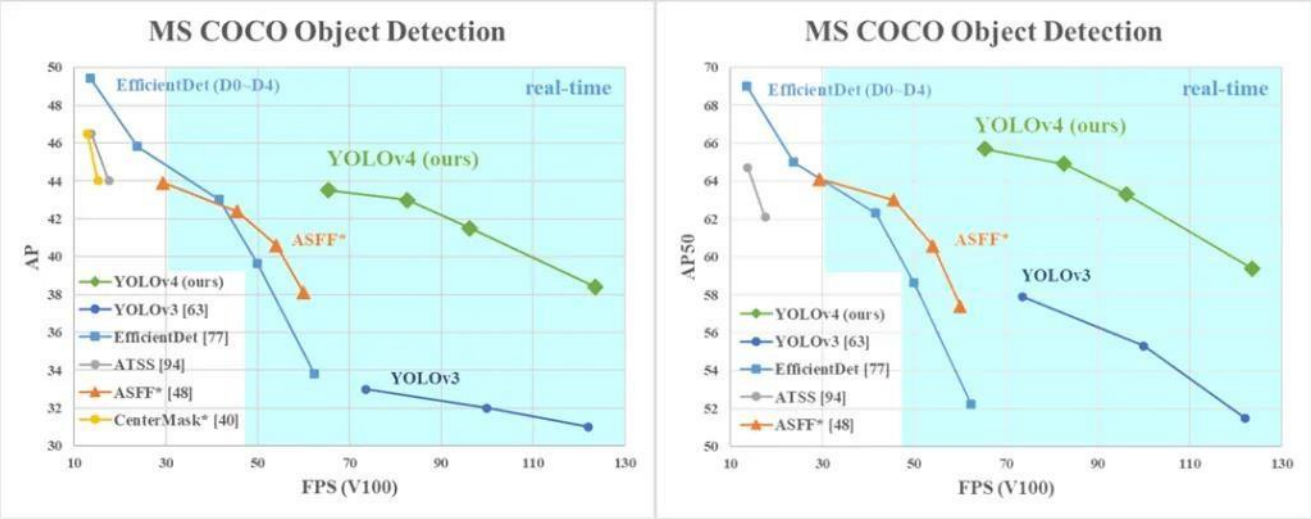


Figure 33. Performance of YOLO v4 vs other Detectors on Tesla V100

From these results, we can see that YOLO v4 outperforms in terms of AP all the other classifier when the frame-rate is higher than 30 FPS (so for real-time applications).

9. Final Considerations

In this report different architectures were compared, with the aim of choosing the best one for cone detection task presented in chapter 1.

Based on the results analysis of all the models considered, the best choice would seem to be **YOLO v4**.

Besides, we will use also Lidar Sensors to be able to detect the presence of cones and their distance from the machine. A possible strategy could be to collect data from Lidars and then focus the search for cones in the images only on the zones where the Lidars have detected object, in order to increase the detections speed (in this case, we might also consider using SSD since it has a better accuracy).

Another strategy could be to perform cone detection on the camera images and then focus the distance calculation from Lidars only on the zones where the detector have found a cone.

References

You Only Look Once: Unified, Real-Time Object Detection: https://arxiv.org/pdf/1506.02640v5.pdf

YOLO v3, an incremental improvement: https://pjreddie.com/media/files/papers/YOLOv3.pdf

YOLOv4: Optimal Speed and Accuracy of Object Detection: https://www.researchgate.net/publication/340883401_YOLOv4_Optimal_Speed_and_Accuracy_of_Object_Detection

Rich feature hierarchies for accurate object detection and semantic segmentation: https://arxiv.org/pdf/1311.2524.pdf

Selective Search for Object Recognition

https://www.researchgate.net/publication/262270555_Selective_Search_for_Object_Recognition

Fast R-CNN

https://arxiv.org/pdf/1504.08083.pdf

Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks: https://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf

SSD: Single Shot MultiBox Detector: https://arxiv.org/pdf/1512.02325.pdf

*Formula Student Germany Driverless Starters Workshop:*https://www.youtube.com/playlist?list=PLtuNXpGOPQ_aeLQNxB4rLzfb8uktPABU9

FSG Academy - powered by Waymo (only available for student who participated to Formula Student Germany) : https://www.formulastudent.de/academy/20200829-waymo/