Car Detection

1. Abstract
2. Introduction stating the problem and some context about the proposed solution.
2. Proposed Solution:
    1. Architecture Description with Visualization Figures.
    .2. Training Description:(Optimizer, Hyperparameters Tuning, etc)
2. Results and Evaluation:
    1. Evaluation Metrics Summary
    2. Summary of Results.
3. Conclusion

4. Reference

Abstract

Due to object detection's close relationship with video analysis and image understanding, it has attracted much research attention in recent years. Traditional object detection methods are built on handcrafted features and shallow trainable architectures. Their performance easily stagnates by constructing complex ensembles which combine multiple low-level image features with high-level context from object detectors and scene classifiers. With the rapid development in deep learning, more powerful tools, which are able to learn semantic, high-level, deeper features, are introduced to address the problems existing in traditional architectures. These models behave differently in network architecture, training strategy and optimization function, etc. In this paper, we provide a review on deep learning based object detection frameworks. Our review begins with a brief introduction on the history of deep learning and its representative tool, namely Convolutional Neural Network (CNN). Then we focus on typical generic object detection architectures along with some modifications and useful tricks to improve detection performance further. As distinct specific detection tasks exhibit different characteristics, we also briefly survey several specific tasks, including salient object detection, face detection and pedestrian detection. Experimental analyses are also provided to compare various methods and draw some meaningful conclusions. Finally, several promising directions and tasks are provided to serve as guidelines for future work in both object detection and relevant neural network based learning systems.

problem

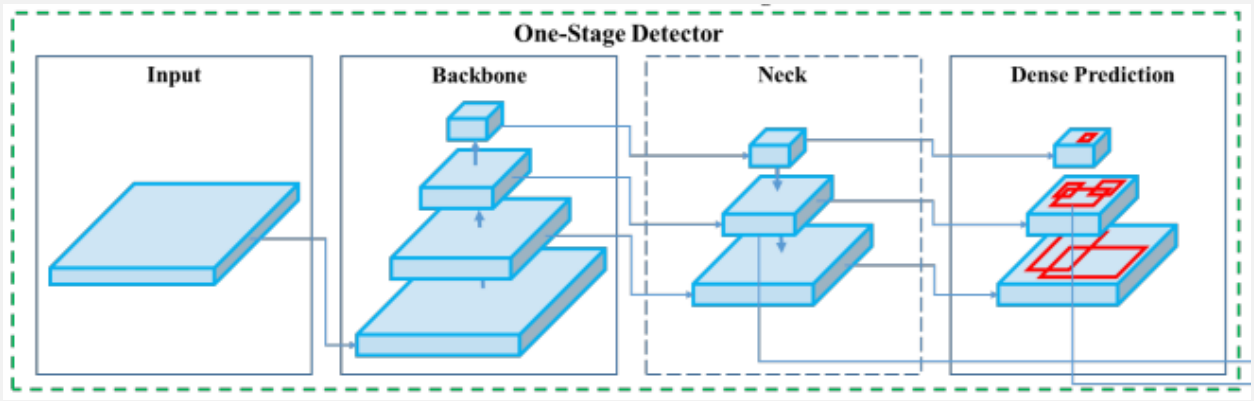
The main problem lies in the presence of cars within various environmental factors such as rain, dust, fog, sand, etc. These weather factors distort the features of the image, that is, the distinctive features of the image, which are many such as edges or colors, etc., which makes it difficult to discover the car within the image. we can use machine learning algorithms to help in detection

and one of the most famous of these algorithms or methods is yolo, which we will explain in detail

Architecture Description with Visualization Figures

General Architecture of an Object Detector

Although YOLO are one-stage detectors, there are also two-stage detectors like R-CNN, fast R-CNN and faster R-CNN which are accurate but slow. We will focus on the former ones. Let's take a look at the main components of a modern one-stage object detector.



Taken from YOLO v4 paper, source

Backbone

Models such as ResNet, DenseNet, VGG, etc, are used as feature extractors. They are pre-trained on image classification datasets, like ImageNet, and then fine-tuned on the detection dataset. Turns out that, these networks that produce different levels of features with higher semantics as the network gets deeper (more layers), are useful for latter parts of the object detection network.

Neck

These are extra layers that go in between the backbone and head. They are used to extract different feature maps of different stages of the backbone. The neck part can be for example a FPN[1], PANet[2], Bi-FPN[3], among others. For example, YOLOv3 uses FPN to extract features of different scales from the backbone.

What does a Feature Pyramid Network (FPN)?

Augments a standard convolutional network with a top-down pathway and lateral connections so the network efficiently constructs a rich, multi-scale feature pyramid from a single resolution input image [4]

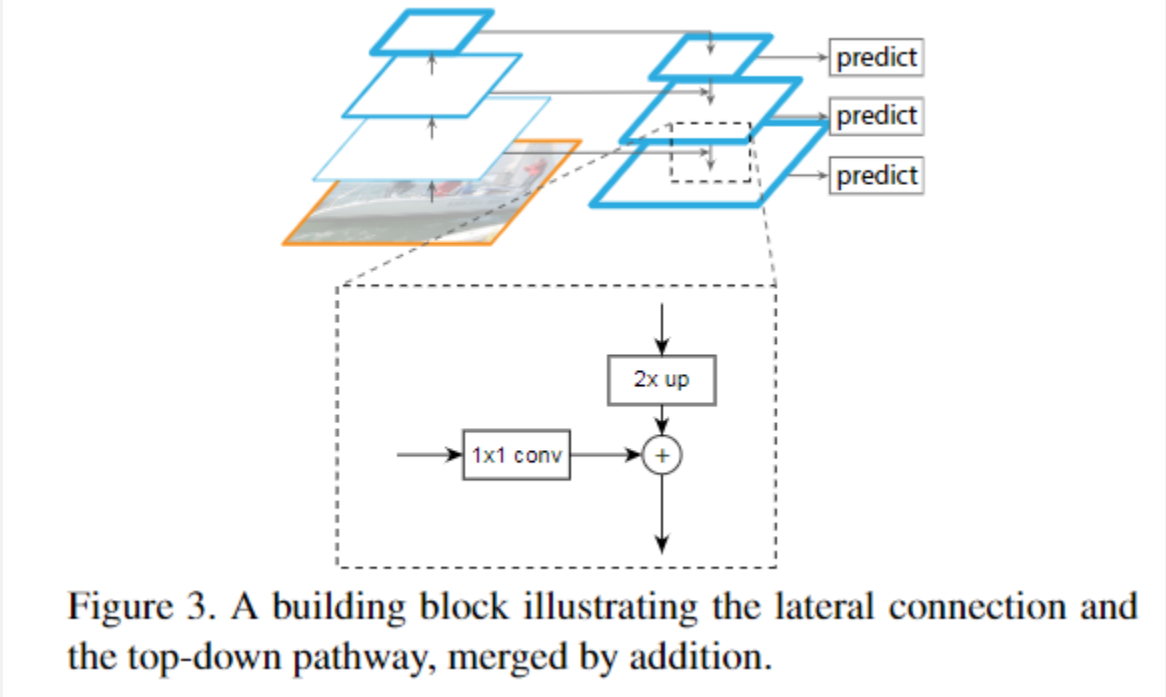


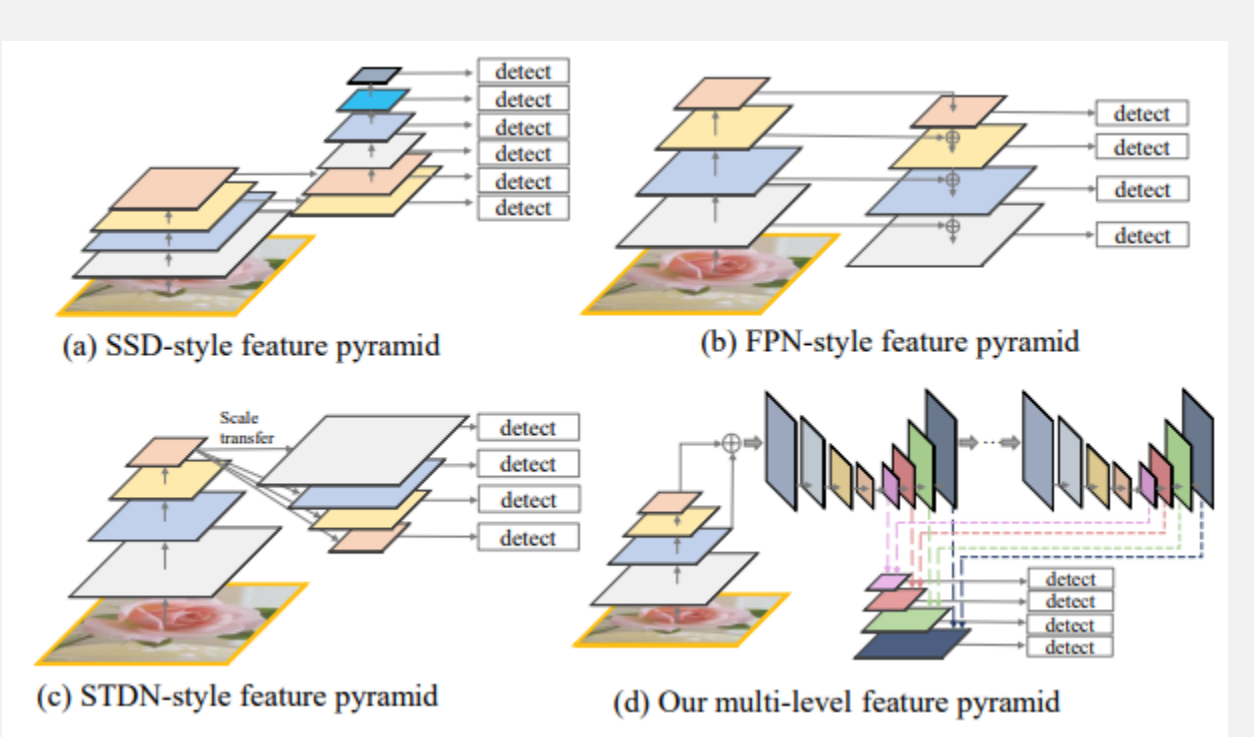
Figure 3. A building block illustrating the lateral connection and the top-down pathway, merged by addition.

Feature Pyramid Networks[1] for Object Detection

Each lateral connection merges the feature maps from the bottom-up pathway to the top-down pathway, producing different pyramid levels. Before merging the feature maps, the previous pyramid level is up-sampled by a factor of 2x in FPN[1] so they

have the same spatial size. The classification/regression network (the head) is then applied at each each level of the pyramid so that it helps to detect object of different sizes.

This idea of Feature Pyramid Networks can be applied to different backbones models, and as an example, the original FPN[1] paper used ResNets. There are also many modules that integrate FPN in different ways, such as SFAM [7], ASFF [9], and Bi-FPN[3].



Four types of feature pyramids. SFAM[7] module is (d)

Image *(a)* shows how features are extracted from the backbone in a Single Shot Detector architecture(SSD). The image above shows also three other different types of pyramid networks, but the idea behind them is the same as they help to:

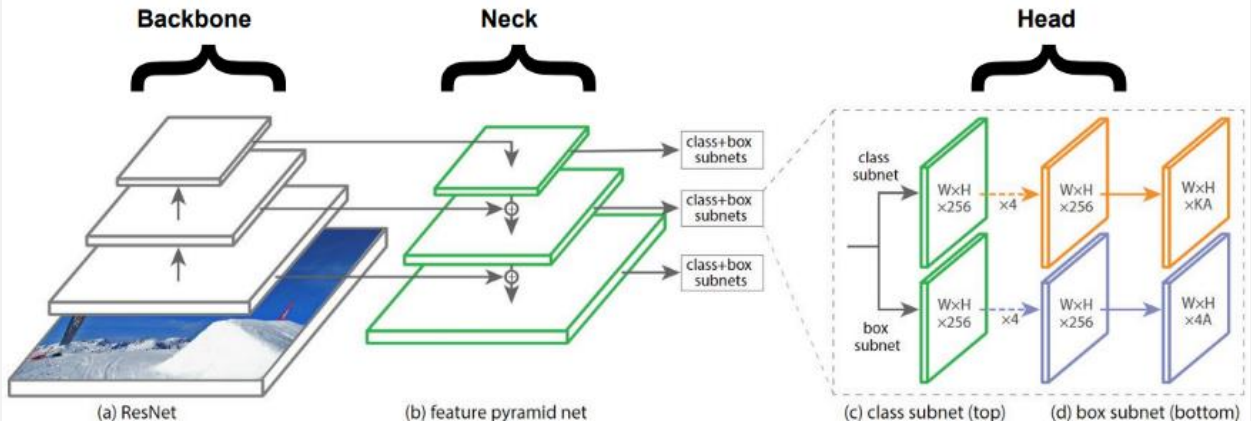
Alleviate the problem arising from scale variation across object instances [3].

ASFF[9] and Bi-FPN[3] are also interesting types of FPNs and show interesting results, but we will skip them here.

Head

This is a network in charge of actually doing the detection part (classification and regression) of bounding boxes. A single output may look like (depending on the implementation): 4 values describing the predicted bounding box (x, y, h, w) and the probability of k classes + 1 (one extra for background). Objected detectors *anchor-based*, like YOLO, apply the head network to each anchor box. Other popular one-stage detectors, which are *anchor-based*, are: Single Shot Detector[6] and RetinaNet[4].

Following illustration combines the three modules mentioned above.



Original image from RetinaNet[4] paper

Bag of freebies & Bag of specials

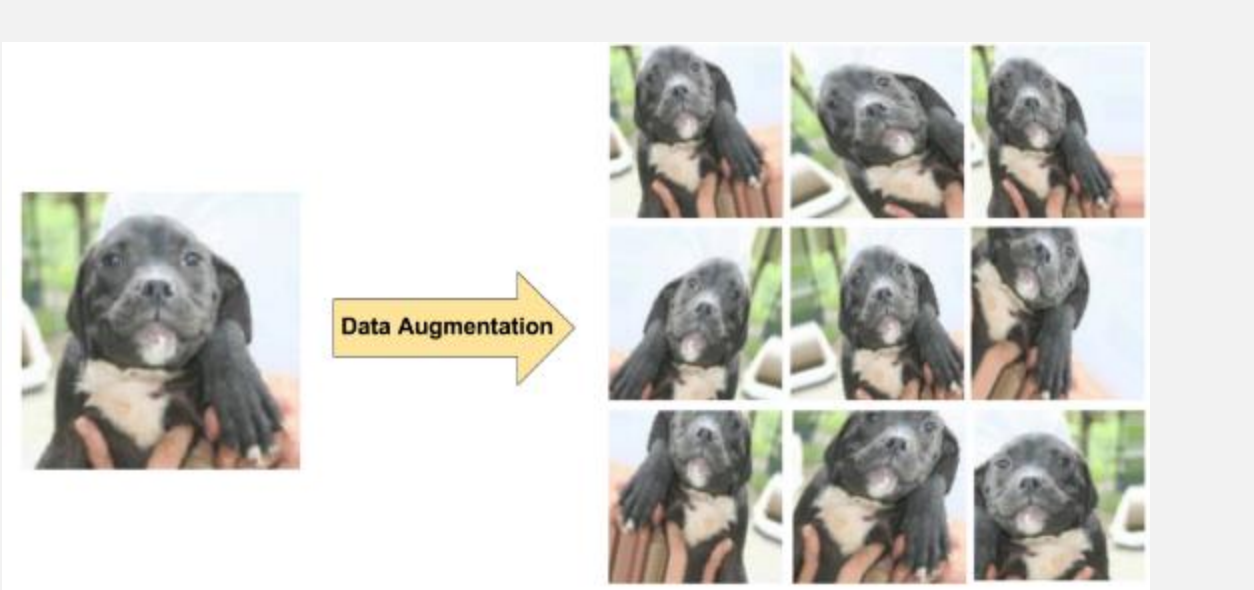
The authors of YOLO v4 paper[5] distinguish between two categories of methods that are used to improve the object detector's accuracy. They analyze different

methods in both categories, to achieve a fast operating-speed neural network with good 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. [5]

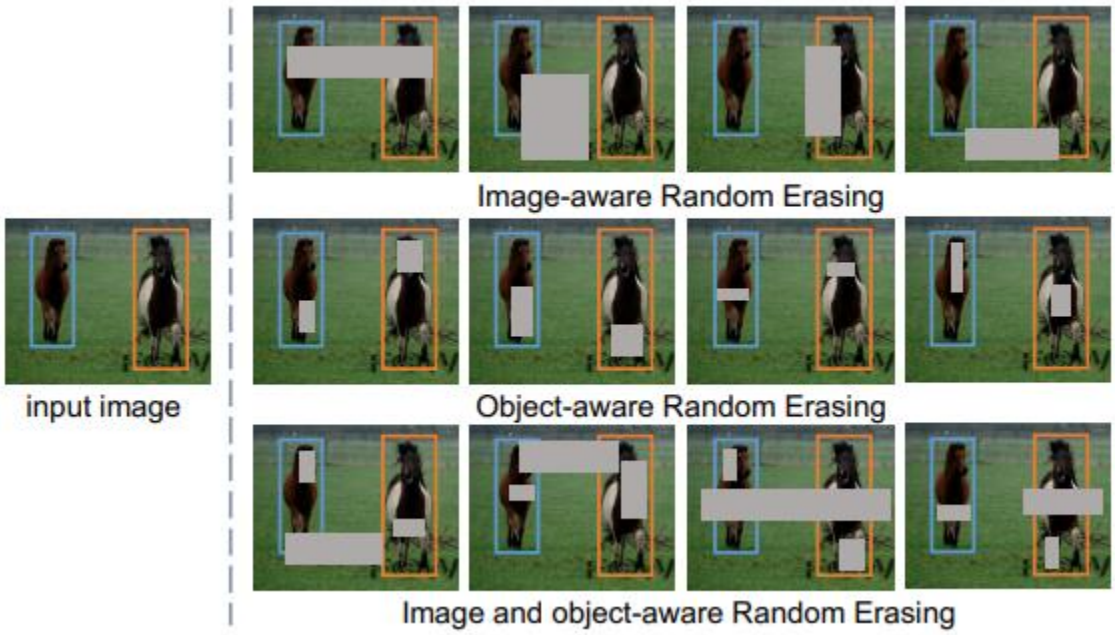
An example of BoF is data augmentation, which increases the generalization ability of the model. To do this we can do photo-metric distortions like: changing the brightness, saturation, contrast and noise or we can do geometric distortion of an image, like rotating it, cropping, etc. These techniques are a clear example of a BoF, and they help the detector accuracy!



Examples of geometric distortions, source

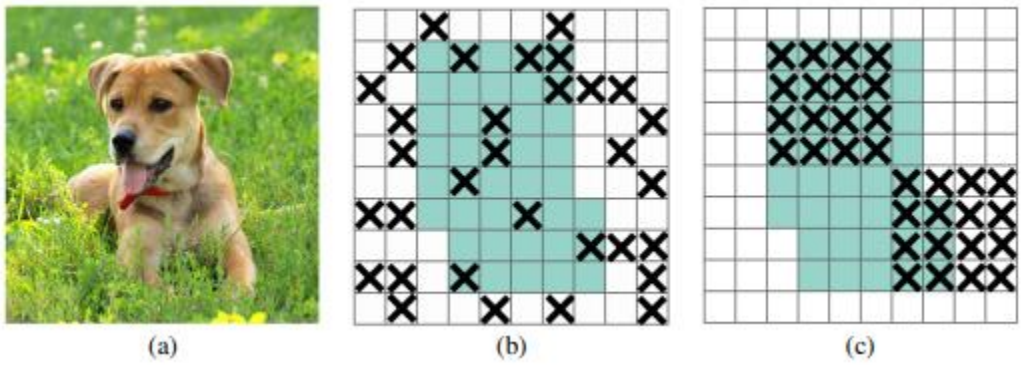
Note: for object detection tasks the bounding boxes should also have the same transformations applied

There are other interesting techniques of augmenting the images like CutOut[8] which randomly masks out square regions of input during training. This showed to improve robustness and performance of CNNs. Similarly, Random Erasing[10] selects rectangle regions in an image and erases its pixels with random values.



Example of Random Erasing for object detection, source

Other Back of Freebies are the regularization techniques used to avoid over-fitting, like: DropOut, DropConnect and DropBlock[13]. This last one actually shows very good results in CNNs and is used in YOLO v4 backbone.



From DropBlock paper[13]

Dropping activations at random (b) is not good to remove semantic information, because nearby activations contain closely related information. Instead, by dropping continuous regions it can remove certain semantic information (e.g., head or feet) and enforce remaining units to learn other features for classifying input image.

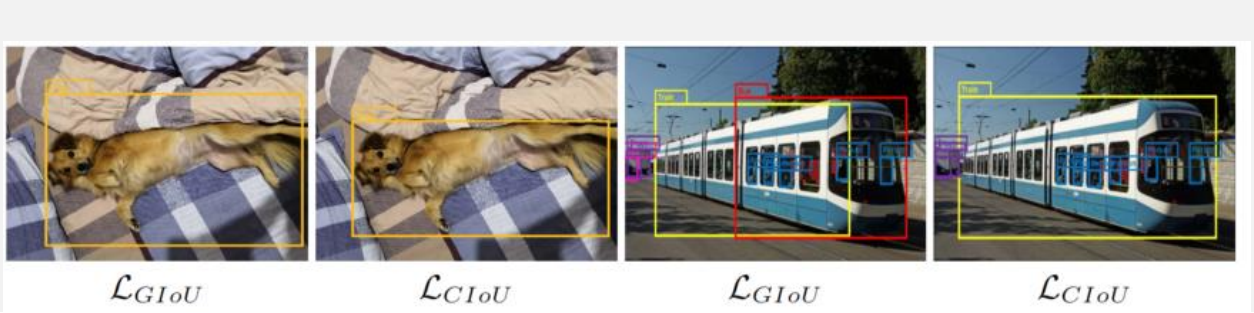
The cost function of the regression network also applies to the category. The traditional thing is to apply Mean Squared error to perform regression on the coordinates.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

As stated in the paper, this treats these points as independent variables but doesn't consider the integrity of the object itself. To improve this, IoU[12] loss has been proposed, which takes into consideration the area of the predicted Bounding Box(BBox) and the ground truth Bounding Box. This idea was improved furthermore by GIoU loss [11] by including the shape and orientation of an object in addition to the coverage area. On the other side, CIoU loss was also introduced and it takes into consideration the overlapping area, the distance between center points and aspect ratio. YOLO v4 uses CIoU loss as the loss for the Bounding Boxes, mainly because it leads to faster convergence and better performance compared to the others mentioned.

Note: one thing that might cause confusion is that although many models use MSE for BBox regression loss, they use IoU as a metric and not as a loss function like mentioned above.

Following illustration compares the same model with different IoU losses:



Comparison of losses, source

We can notice CIoU performs better than GIoU. These detections come from Faster R-CNN (Ren et al. 2015) which was trained on the same MS COCO dataset, with GIoU and CIoU losses.

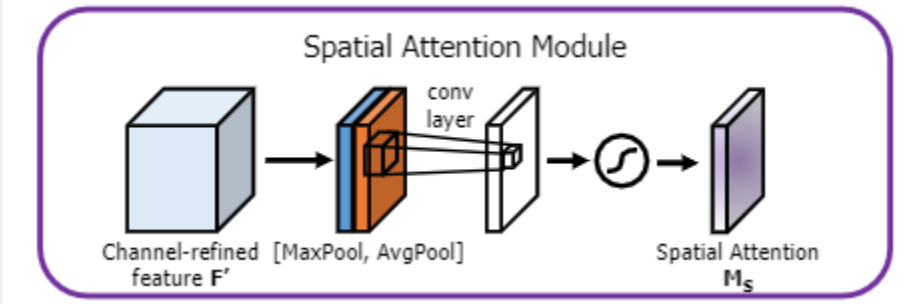
Bag of specials (BoS):

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 [5]

As stated in the paper, this kind of modules/methods usually involve: introducing attention mechanisms(*Squeeze-and-Excitation* and *Spatial Attention Module*), enlarging receptive field of model and strengthening feature integration capability, among others.

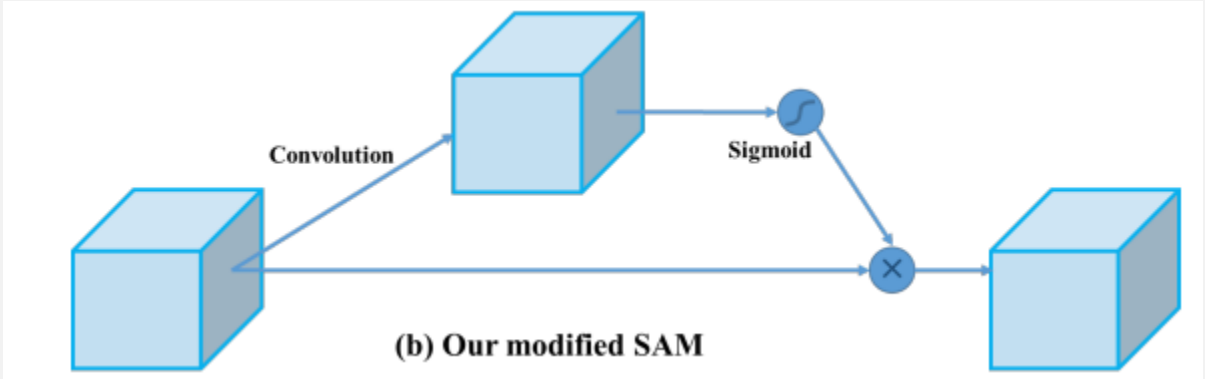
Common modules that are used to improve the receptive field are SPP, ASPP and RFB (YOLO v4 uses SPP).

Moreover, attention modules for CNNs are mainly divided in channel wise attention, like Squeeze-and-Excitation (SE)[15], and spatial-wise attention, like Spatial Attention Module (SAM)[16]. A reason why the latter is sometimes preferred, is because SE increases inference speed by a 10% on GPUs, which not desirable. Actually YOLO v4 considers SAM[16] module but not exactly as it was originally published in this paper. Note the following:



Original Spatial Attention Module [16]

Given a feature map F', the original implementation performed 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'.



YOLO v4 modified Spatial Attention Module, source[5]

YOLO v4 modified SAM, on the other hand, 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').

Feature Pyramids we discussed early like SFAM[7], ASFF[9] and Bi-FPN[3] also fall in this category of BoS, as do the activation functions. Since ReLU came out, there have been many variants of it, like LReLU, PReLU and ReLU6. Activations like ReLU6 and hard-Swish are specially designed for quantized networks used to make inference on embedded devices, like in the Google Coral Edge TPU.

On the other hand, YOLO v4 uses a lot Mish[14] activation function in the backbone. Take a look at the graph:

$$f(x) = x \tanh(\ln(1 + e^x))$$

Mish formula

Turns out that this activation function shows very promising results. For example, using a Squeeze Excite Network[15] with Mish (on CIFAR-100 dataset) resulted in an increase in Top-1 test accuracy by 0.494% and 1.671% as compared to the same network with Swish and ReLU respectively. [14]

You can check this desmos which contains some other activation functions graphed!

YOLO v4 design

Until now we have discussed methods used improve the model accuracy and different parts of an object detector(backbone, neck, head). Let us now talk about what is used in the new YOLO.

- Backbone: It uses the CSPDarknet53 as the feature-extractor model for the GPU version. For the VPU(Vision Processing Unit) they consider using EfficientNet-lite — MixNet — GhostNet or MobileNetV3. We will focus on the GPU version for now.

The following table shows different considered backbones for GPU version

Table 1: Parameters of neural networks for image classification.

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

source [5]

Certain backbones are more suitable for classification than for detection. For example, CSPDarknet53 showed to be better than CSPResNext50 in terms of detecting objects, and CSPResNext50 better than CSPDarknet53 for image classification. As stated in the paper, a backbone model for object detection

requires Higher input network size, for better detection in small objects, and more layers, for a higher receptive field.

- Neck: They use Spatial pyramid pooling (SPP) and Path Aggregation Network (PAN). The latter is not identical to the original PAN, but a modified version which replaces the addition with a concat. Illustration shows this:

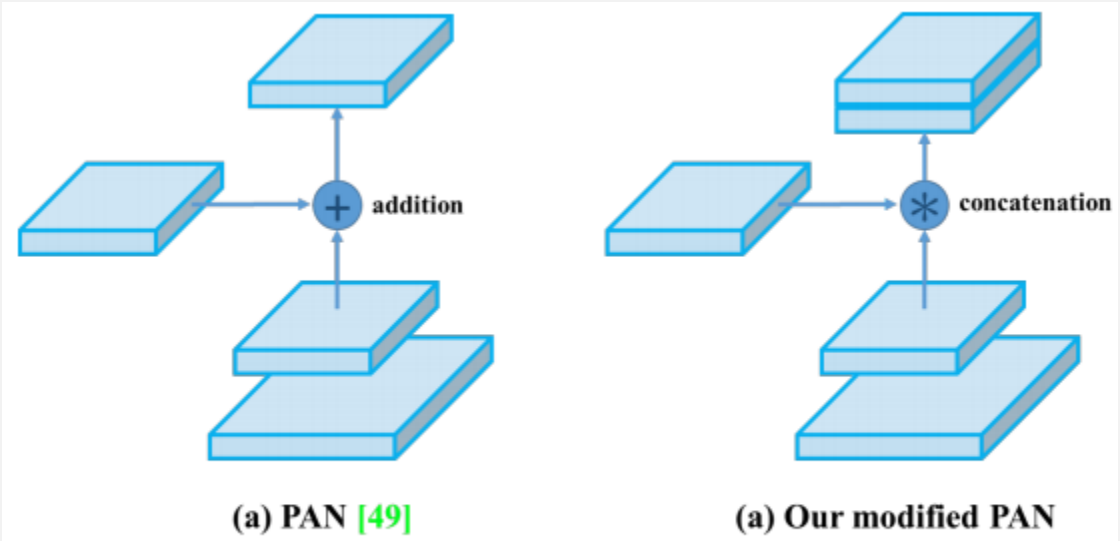
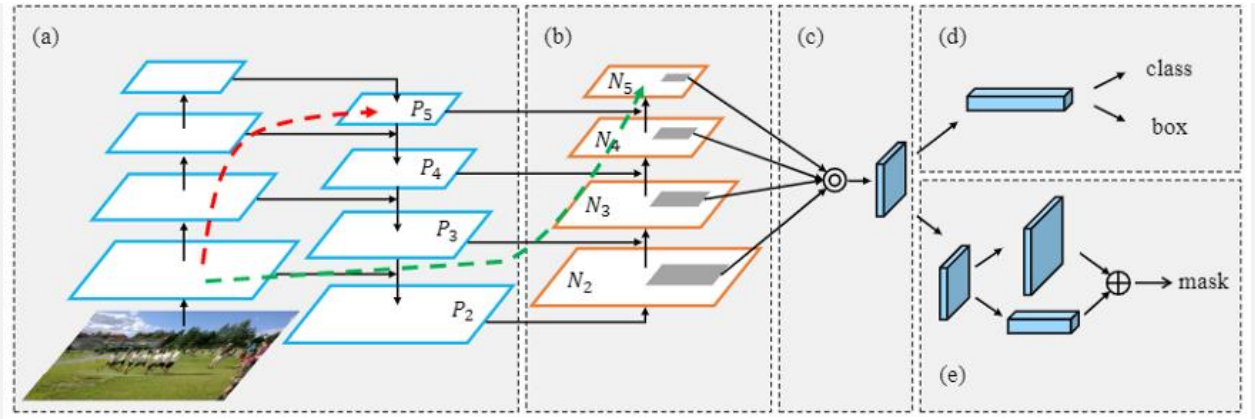


Figure 6: Modified PAN.

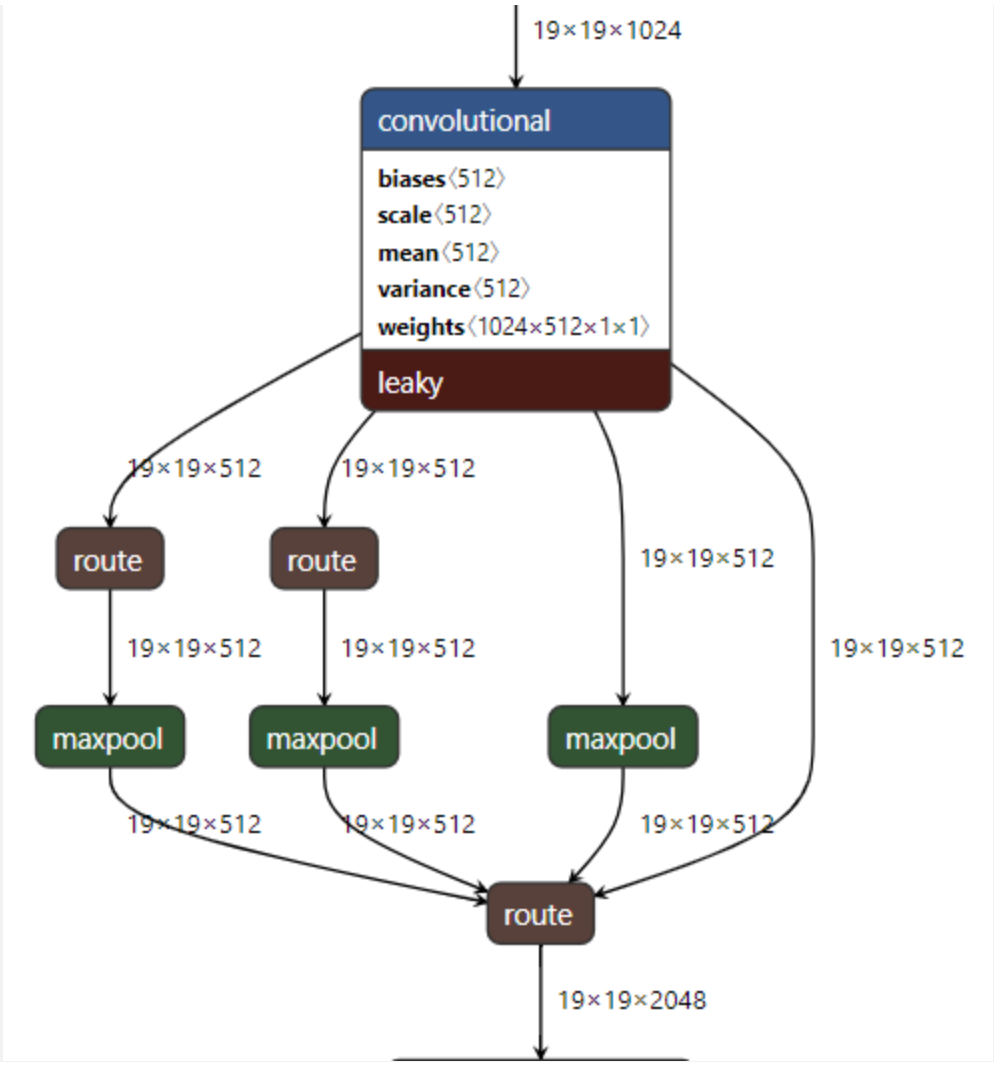
source [5]

Originally in PAN paper, after reducing the size of N4 to have the same spatial size as P5, they add this new down-sized N4 with P5. This is repeated at all levels of Pi+1 and Ni to produce Ni+1. In YOLO v4 instead of adding Ni with each Pi+1, they concatenate them (as shown in the image above).



Path Aggregation Network (PAN) source[2]

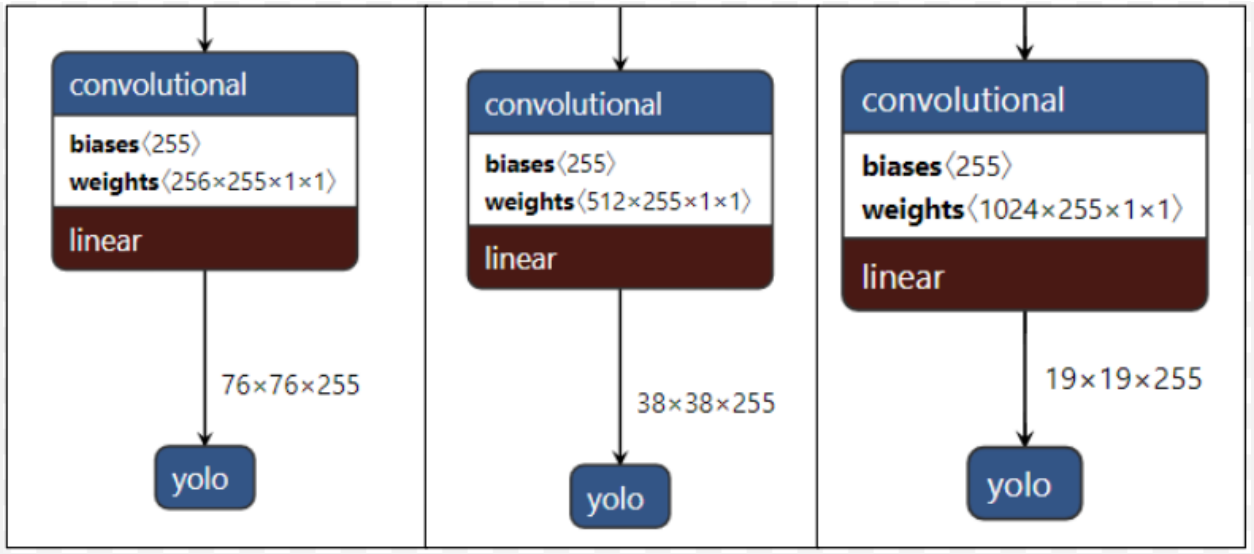
Looking at the SPP module, it basically performs max-pooling over the 19*19*512 feature map with different kernel sizes $k = \{ 5, 9, 13\}$ and 'same' padding (to keep the same spatial size). The four corresponding feature maps get then concatenated to form a 19*19*2048 volume. This increases the neck receptive field, thus improving the model accuracy with negligible increase of inference time.



SPP observed in yolov4.cfg

If you want to visualize different layers used in yolo, like in the image above, I recommend using this tool (either web/desktop version works) and then opening yolov4.cfg with it.

- Head: They use the same as YOLO v3.



The YOLO heads applied at different scales

These are the heads applied at different scales of the network, for detecting different-size objects. The number of channels is 255 because of (80 *classes* + 1 *for objectness* + 4 *coordinates*) * 3 *anchors.*

Summary BoF and BoS used

The different modules/methods of BoF and BoS used in the backbone and in the detector of YOLO v4 can be summarized as follows:

	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

Training Description

Optimizer:

This is the main form of Main square errore

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

The loss function remains the same as the YOLOv4 model, which consists of three parts: classification loss, regression loss, and confidence loss. Classification loss and confidence loss remain the same as the YOLOv3 model, but Complete Intersection over Union (CIoU) [33] is used to replace mean squared error (MSE) to optimize the regression loss.

The CIoU loss function is as follows:

$$LOSS_{CIoU}=1-IoU+\rho^2(b,b_{gt})c^2+\alpha\upsilon,$$

(2)

where $\rho^2(b,b_{gt})$ represents the Euclidean distance between the center points of the prediction box and the ground truth, c represents the diagonal distance of the smallest closed area that can simultaneously contain the prediction box and the ground truth. Figure 9 shows the structure of CIoU.

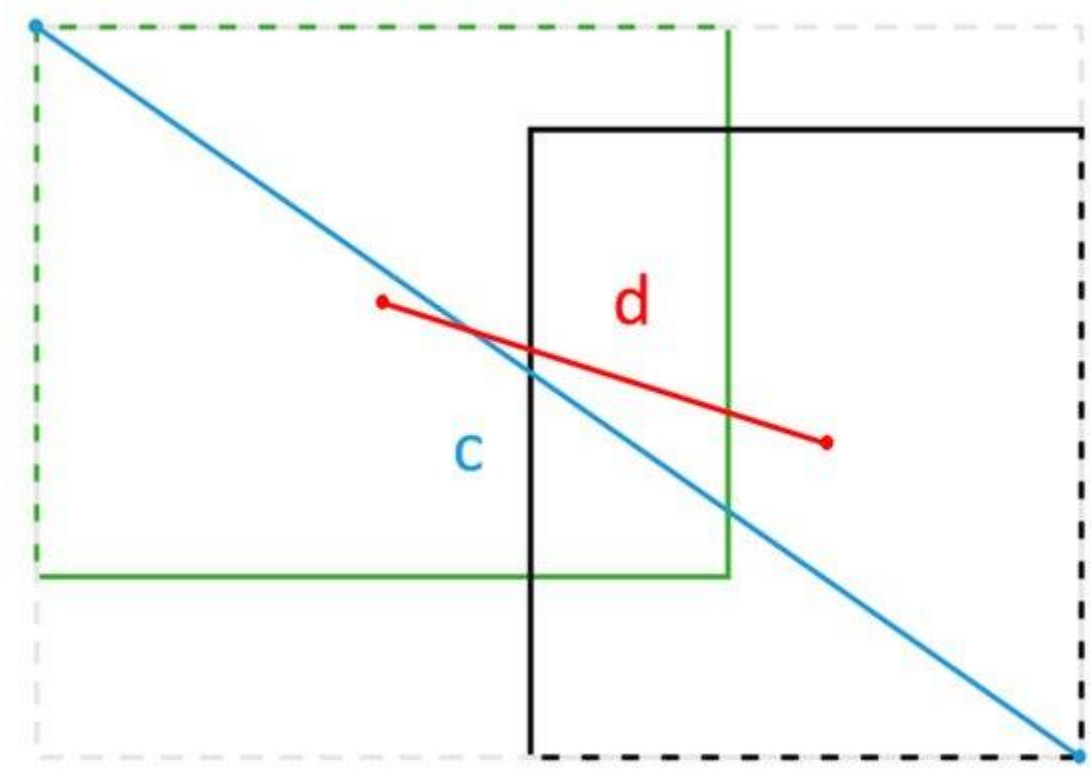


Figure 9. CIoU.

The formulas of α and υ are as follows:

$$\alpha=\upsilon 1-IoU+\upsilon, \tag{3}$$

$$\upsilon=4\pi 2(\arctan wgthgt-\arctan wh)2. \tag{4}$$

The total loss function of the YOLOv4 model is:

$$\alpha = \frac{v}{1 - IoU + v}, \tag{3}$$

$$v = \frac{4}{\pi^2}\left(\arctan\frac{w^{gt}}{h^{gt}} - \arctan\frac{w}{h}\right)^2. \tag{4}$$

The total loss function of the YOLOv4 model is:

$$\begin{aligned} LOSS = 1 - IoU + \frac{\rho^2\left(b, b^{gt}\right)}{c^2} + \alpha v - \\ \sum_{i=0}^{S^2}\sum_{j=0}^{B} I_{ij}^{obj}\left[\hat{C}_i \log\left(C_i\right) + \left(1 - \hat{C}_i\right)\log\left(1 - C_i\right)\right] - \\ \lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B} I_{ij}^{noobj}\left[\hat{C}_i \log\left(C_i\right) + \left(1 - \hat{C}_i\right)\log\left(1 - C_i\right)\right] - \\ \sum_{i=0}^{S^2} I_{ij}^{obj}\sum_{c \in classes}\left[\hat{p}_i\left(c\right)\log\left(p_i\left(c\right)\right) + \left(1 - \hat{p}_i\left(c\right)\right)\log\left(1 - p_i\left(c\right)\right)\right] \end{aligned} \tag{5}$$

(5)

where S^2 represents S × S grids, each grid generates B candidate boxes, and each candidate box gets corresponding bounding boxes through the network, finally, S × S × B bounding boxes are formed. If there is no object (noobj) in the box, only the confidence loss of the box is calculated. The confidence loss function uses cross entropy error and is divided into two parts: there is the object (obj) and noobj. The loss of noobj increases the weight coefficient λ, which is to reduce the contribution weight of the noobj calculation part. The classification loss function also uses cross entropy error. When the j-th anchor box of the *i*-th grid is responsible for certain ground truth, then the bounding box generated by this anchor box will calculate the classification loss function.

Resizing image :

During the training process, we change the size of the image frequently in order to avoid falling into the cases of overfit, which means what it means is memorizing the features. The image size is changed by multiples of 32. This method is a good way not only to avoid the problem of overfit, but also to help generalize more

Splitting data:

Training data 90%

Validation data 10%

This percentage was adopted for each of the classes, where we took 90 % for training and 10 % for verification

Hyperparameter value:

indx	Max_batch	Width image	Height image	chanel	subdivison	batch_size
1	6000	416	416	1	24	64
2	6000	480	480	1	32	64
3	6000	608	608	3	24	64

Result of hyperparameter :

TP: true positive

FP: false positive

FN : False Negitev

$$\text{Precision} = \frac{tp}{tp + fp}$$

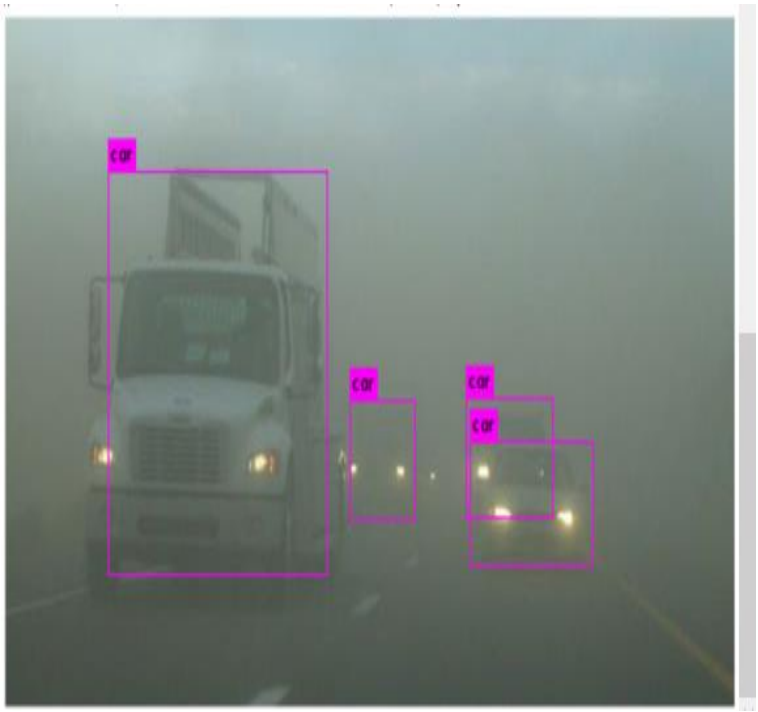
$$\text{Recall} = \frac{tp}{tp + fn}$$

$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{tp}}{\mathrm{tp} + \frac{1}{2}(\mathrm{fp} + \mathrm{fn})}.$$

Iou: Intersection over Union

Results and Evaluation:

Indx	Map	TP	FP	FN	Iou	precision	recall	F1-scor
1	51%	367	349	359	37	0.51	0.51	0.51
2	70%	454	112	272	62.80	0.80	0.63	0.70
3	87%	573	85	153	66.92	0.87	0.79	0.83



Summary of Results

Best parameter

indx	Max_batch	Width image	Height image	chanel	subdivison	batch_size
3	6000	608	608	3	24	64

Best result :

Indx	Map	TP	FP	FN	Iou	precision	recall	F1-scor
3	87%	573	85	153	66.92	0.87	0.79	0.83

Conclusion:

We can see through this work the great ability of machine learning to identify cars in different environmental conditions, even from the presence of a small number of images and high accuracy. In his currency if not better than him in this field

Reference

[1] Feature Pyramid Networks for Object Detection

[2] Path Aggregation Network for Instance Segmentation

[3] EfficientDet: Scalable and Efficient Object Detection

[4] Focal Loss for Dense Object Detection

[5] YOLOv4: Optimal Speed and Accuracy of Object Detection

[6] Single Shot MultiBox Detector (SSD)

[7] A Single-Shot Object Detector based on Multi-Level Feature Pyramid Network

[8] Improved Regularization of Convolutional Neural Networks with Cutout

[9] Learning Spatial Fusion for Single-Shot Object Detection

[10] Random Erasing Data Augmentation

[11] Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression

[12] UnitBox: An Advanced Object Detection Network

[13] DropBlock: A regularization method for convolutional networks

[14] Mish: A Self Regularized Non-Monotonic Neural Activation Function

[15] Squeeze-and-Excitation Networks

[16] CBAM: Convolutional Block Attention Module