YOLO

MALEKE CHAKER
BİLEL EL HADDAD

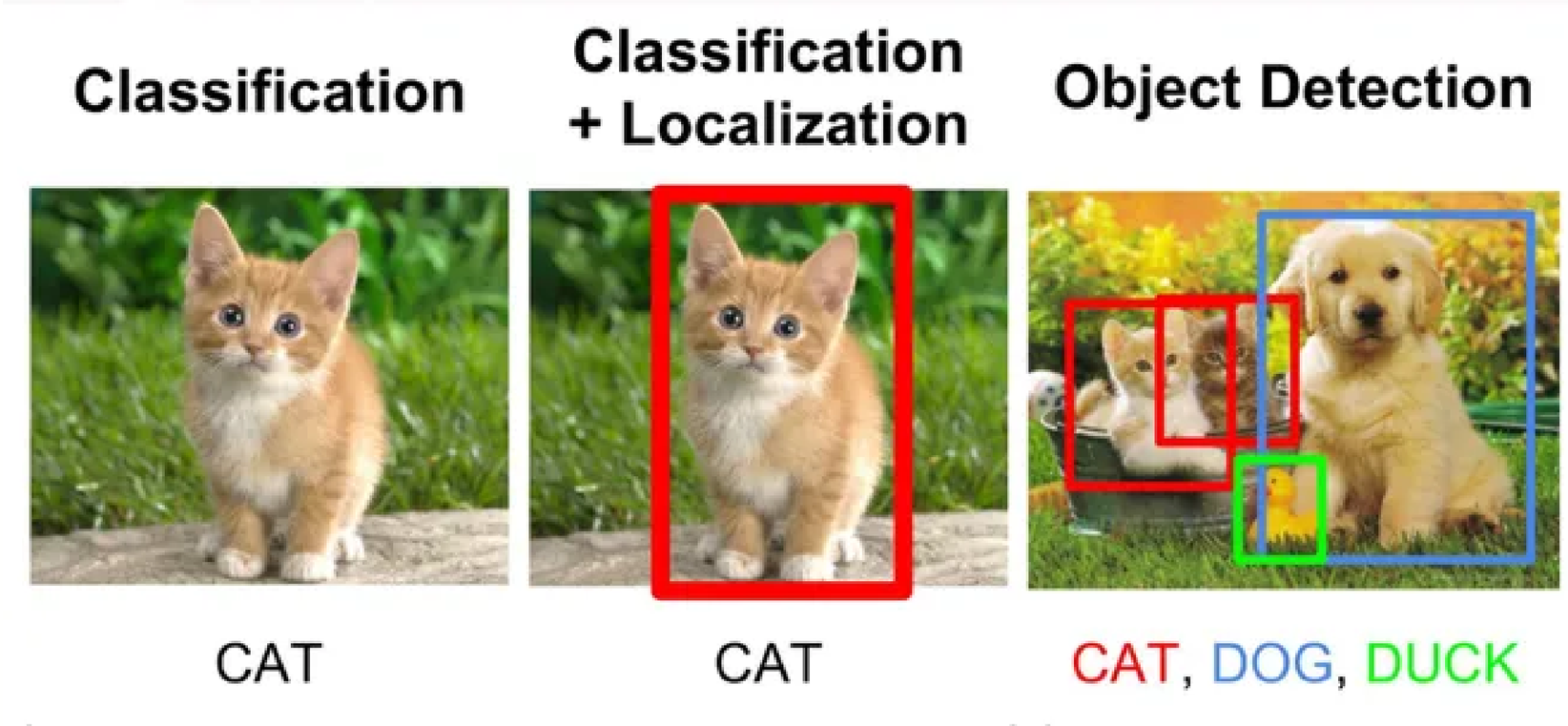
Introduction

In the vast realm of computer vision, Object Detection takes center stage, allowing computers to understand images in a way that goes beyond simple classification. It’s like giving machines the ability to not only recognize objects but also pinpoint their exact locations within an image.

Introduction

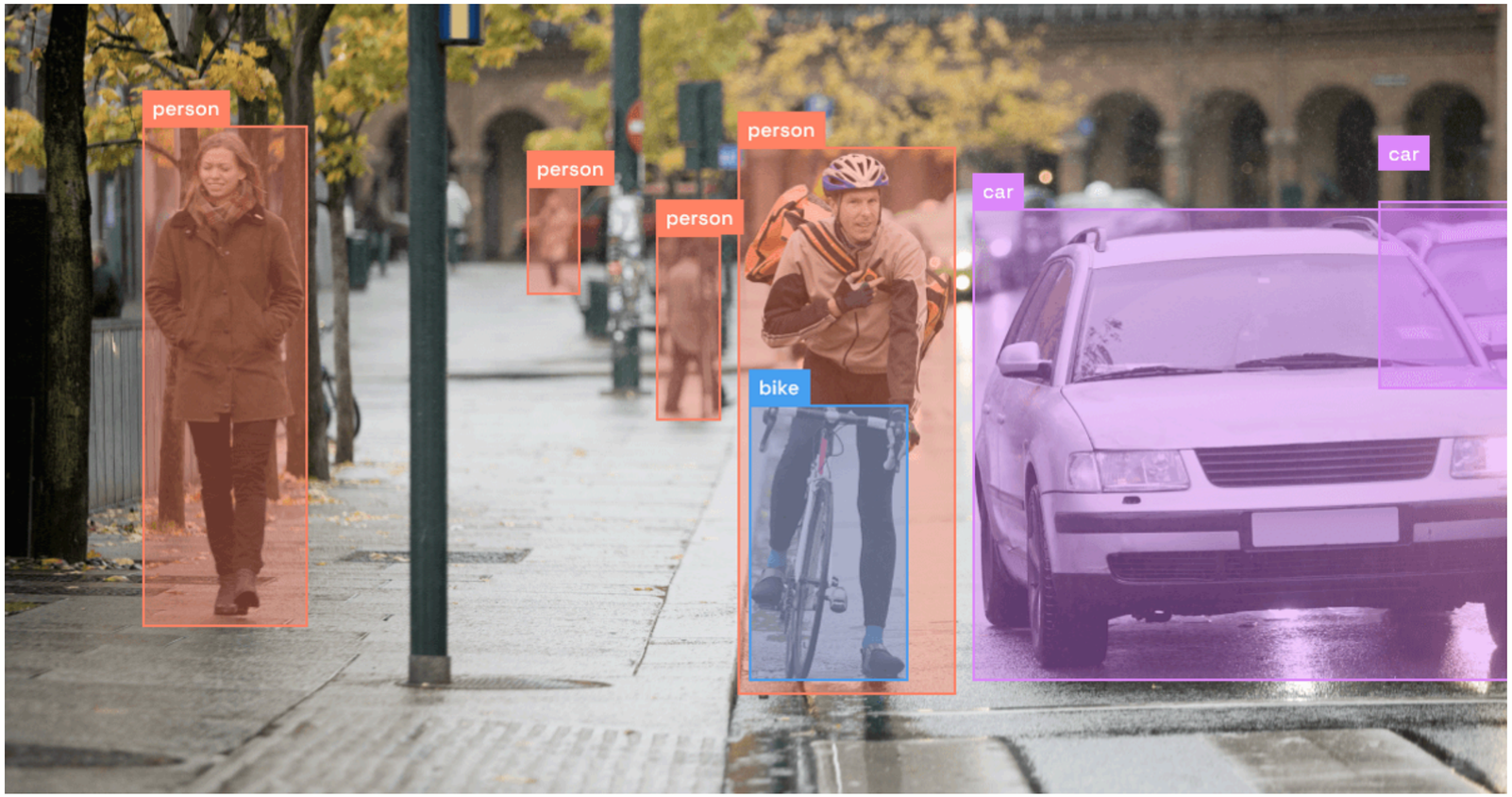
From self-driving cars and security cameras to healthcare tools and your self-checkout lane at the grocery store, Object Detection is everywhere. It helps cars see pedestrians, keeps an eye on who’s coming and going, assists in medical diagnoses, and even speeds up your trip through the store. Let’s take a journey into the domain of Object Detection and discover the state-of-art object detection model YOLO and its variants.

Object Detection :

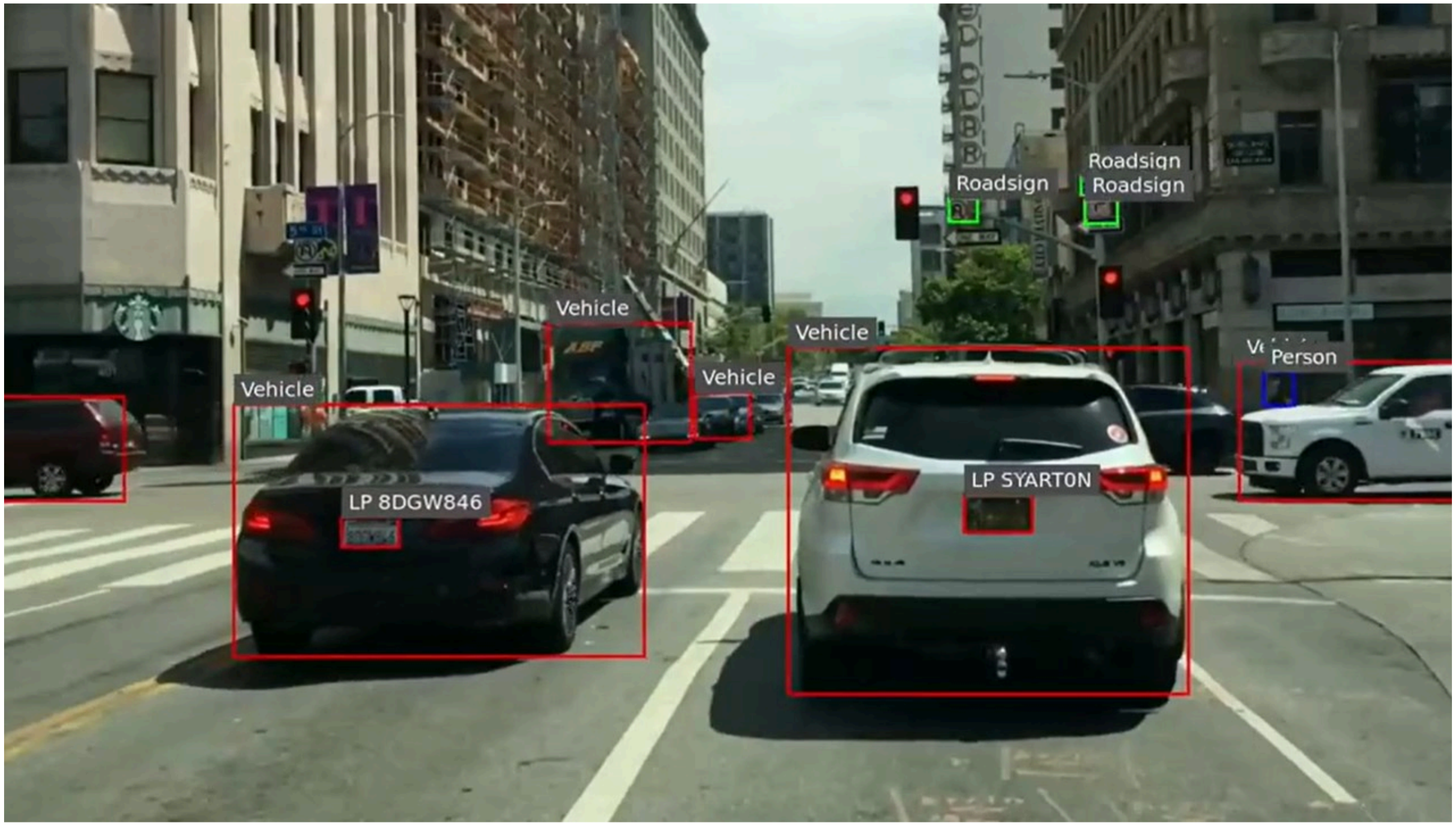


Object detection is an important task in computer vision. In layman's terms, Object detection is defined as Object Localization + Object Classification.

Object Detection :



Object Localization is the method of locating an object in the image using a bounding box and Object Classification is the method that tells what is in that bounding box.



There are numerous real-life applications for object detection. For example,in surveillance and monitoring, it is used in detecting trespassers, vehicle license plates...
In medical imaging, it's used for detecting certain cells, cancer detection, tumor detection, etc. and the list goes on.

Object Detection using Classical Computer Vision

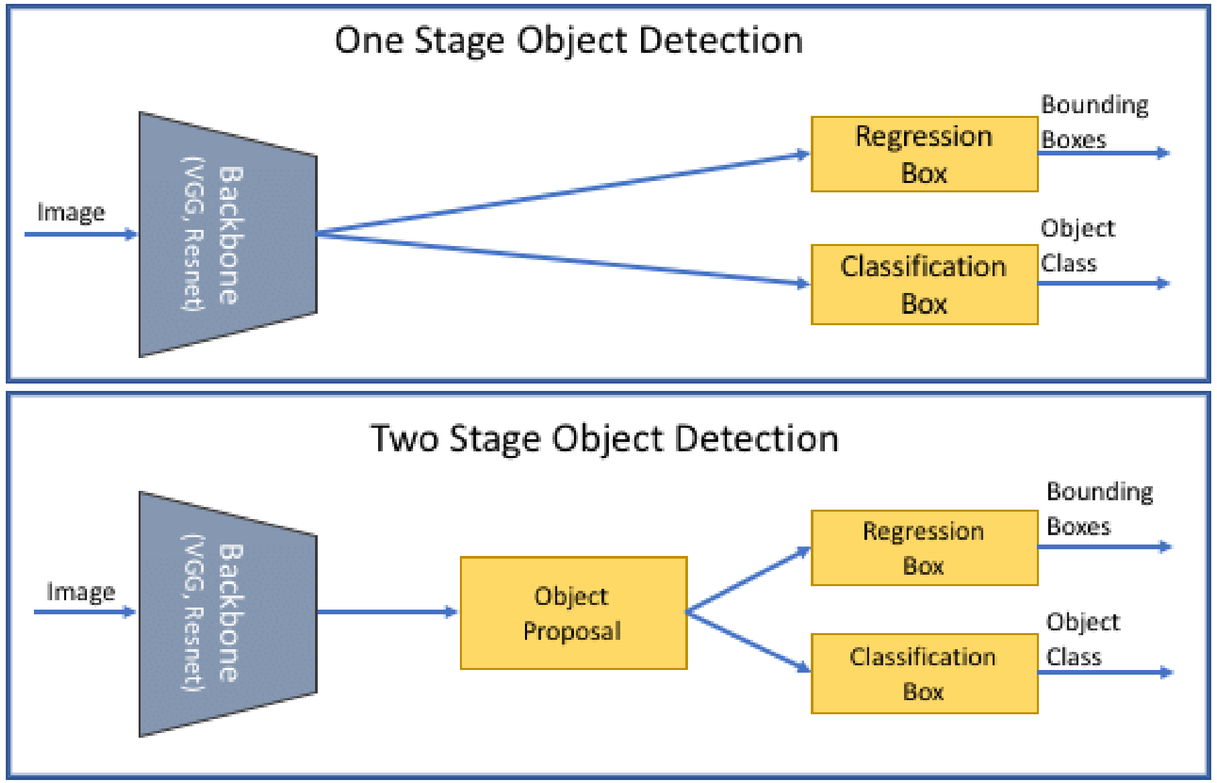
Earlier, detection algorithms used to be based on classical computer vision techniques such as template matching, Haar cascade, Feature detection and matching using SIFT or SURF, HOG Detector, Deformable Part-based Model (DPM), etc. However, most of these methods are very specific to the use case and don’t have generalization capabilities, which redirected the research toward deep learning-based methods.

Object Detection using Deep Learning

The Overfeat paper was a pioneer in deep learning-based object detection as it was a single network model employed to perform object classification and localization.

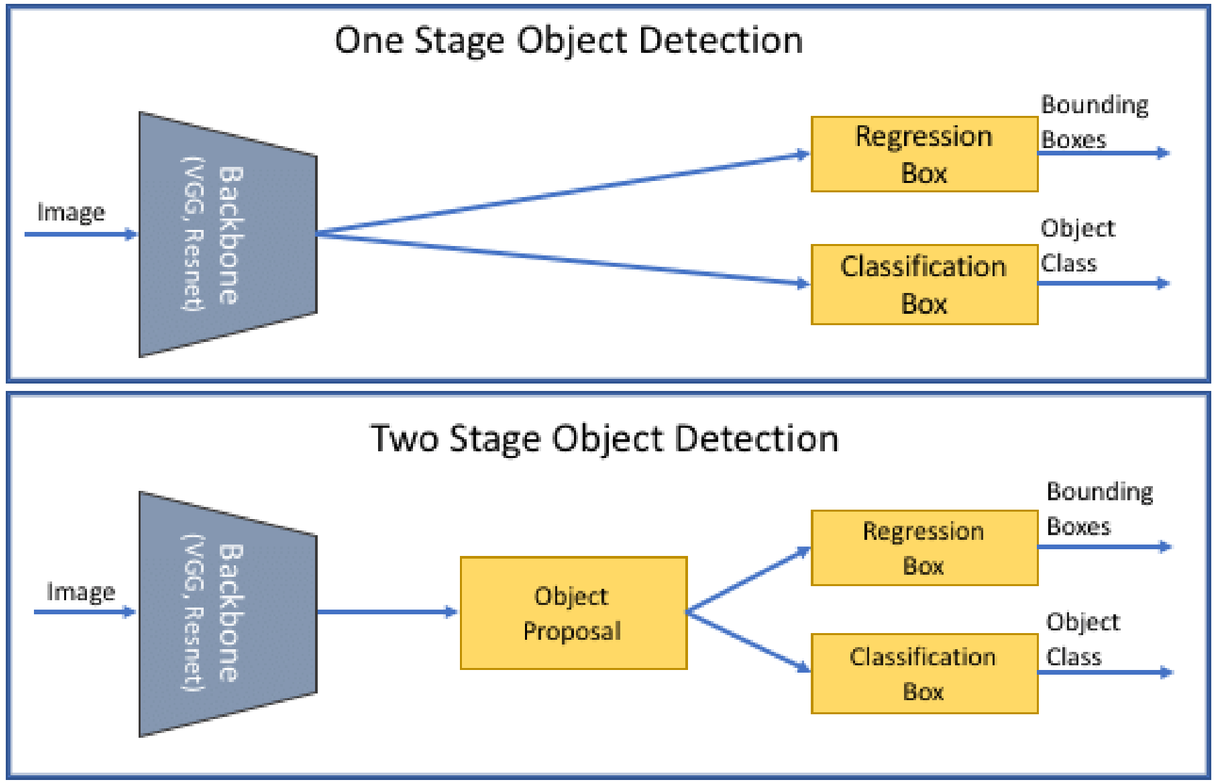
The Overfeat architecture closely resembles the **AlexNet** architecture. It does image classification on different scales in a sliding window fashion and carries out bounding box regression on the same CNN layer. Later other models like RCNN, FastRCNN, SPPNet, YOLO etc. emerged in this field.

One-Stage Vs. Two-Stage Detectors

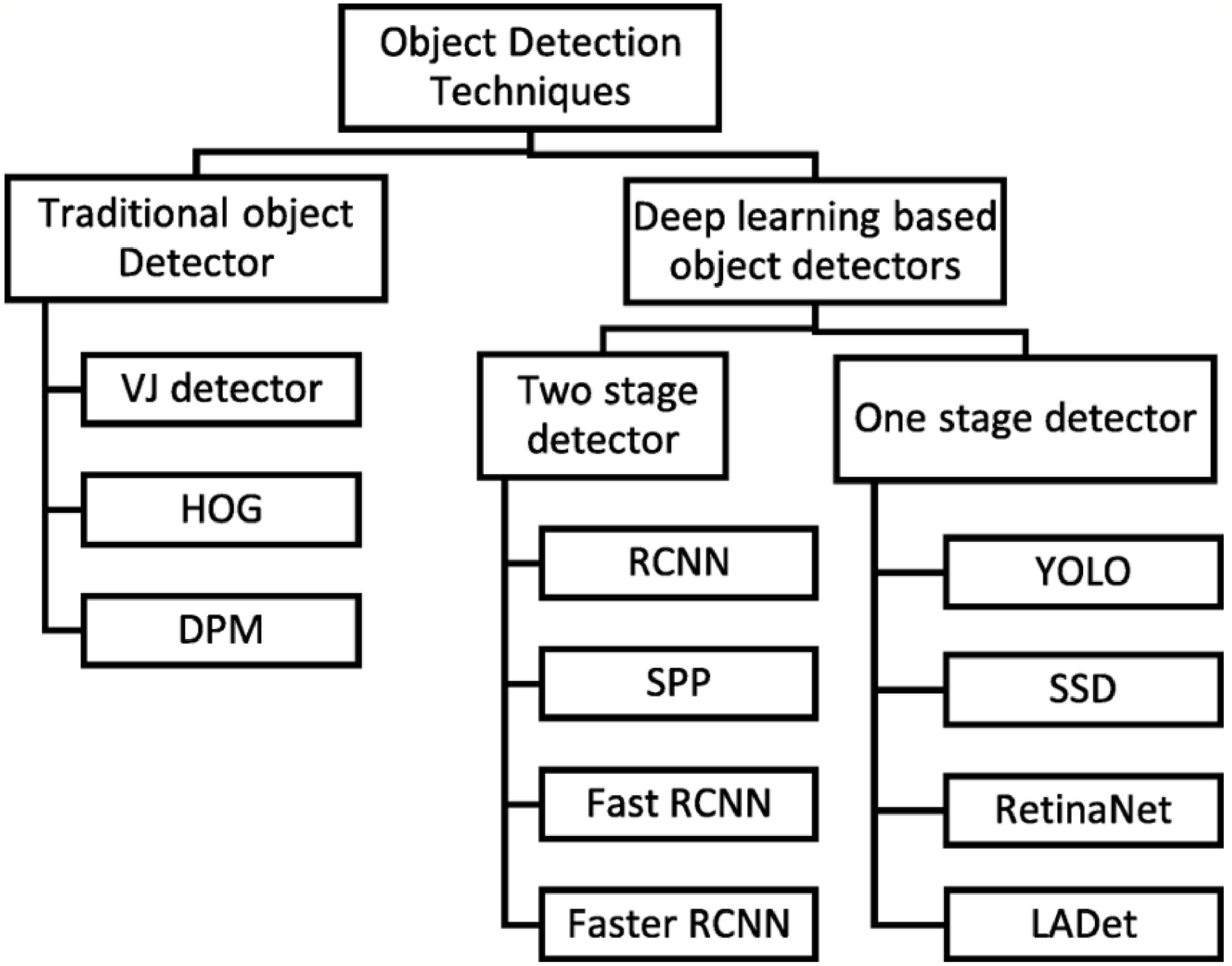


Initial models were two-stage detectors, which means there was a step for region proposals and finding out the region of interest. It was also a step for predicting the bounding box coordinates and class using the proposed region.

One-Stage Vs. Two-Stage Detectors



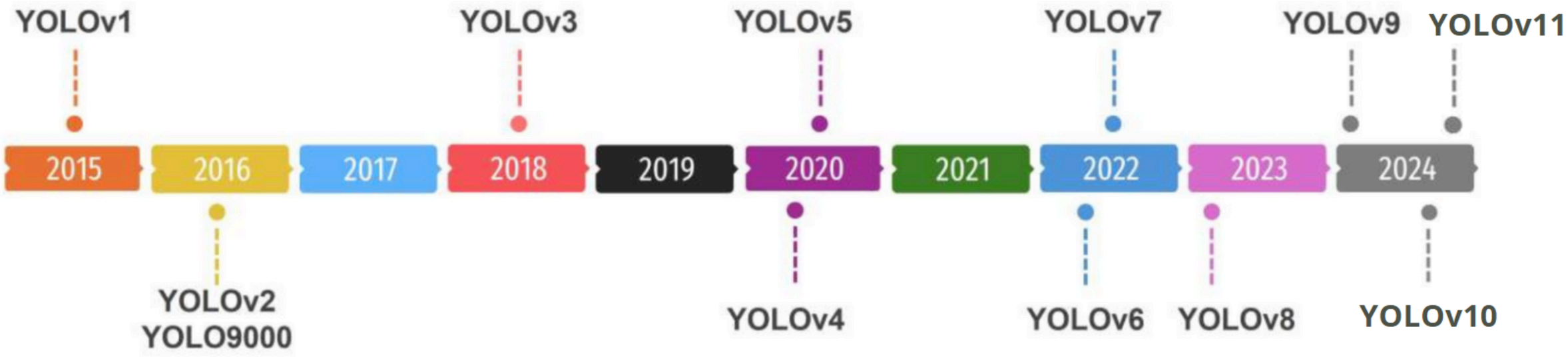
Later the YOLO authors proposed a single-stage strategy that overlooked the Region Proposal step and ran detection directly over the entire image. This helped speed up the training and inference, preserve global context and keep the accuracy on par.


Object Detection Techniques
Traditional object Detector
Deep learning based object detectors
VJ detector
HOG
DPM
Two stage detector
One stage detector
RCNN
SPP
Fast RCNN
Faster RCNN
YOLO
SSD
RetinaNet
LADet

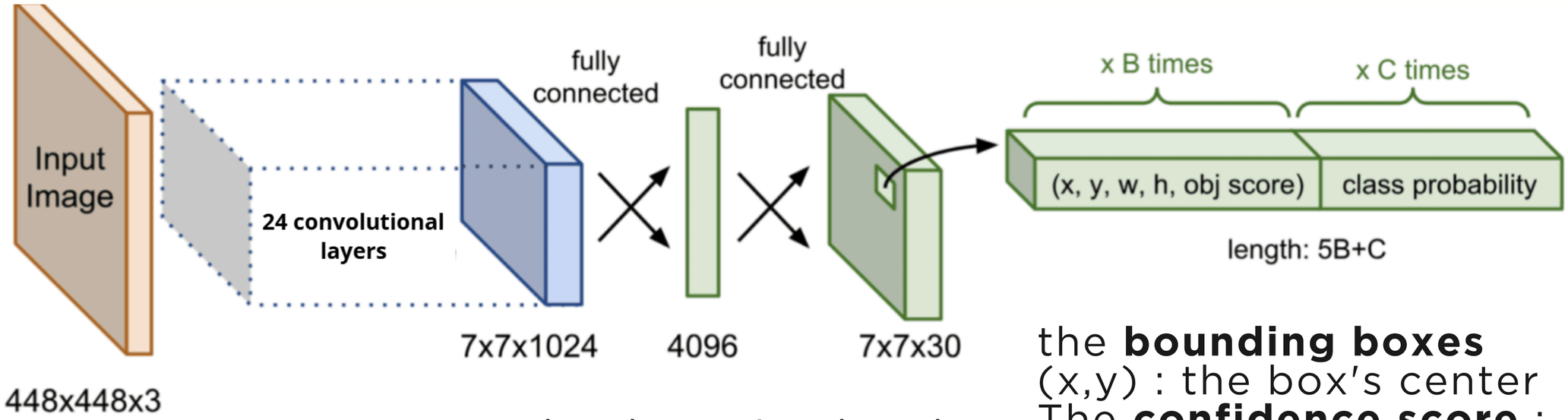
One-Stage Vs Two Stage Detectors

Aspect	One-Stage Detector	Two Stage Detector
Process	Directly predicts bounding boxes & classes	Region proposals + classification
Examples	YOLO, SSD, RetinaNet	Faster R-CNN, Mask R-CNN
Speed	Faster, real-time capable	Slower, not ideal for real-time
Accuracy	Slightly lower, especially on small objects	Higher accuracy, especially for small or dense objects
Use Case	Real-time detection, applications needing speed	Applications requiring accuracy over speed

Timeline of YOLO Versions



YOLOv1

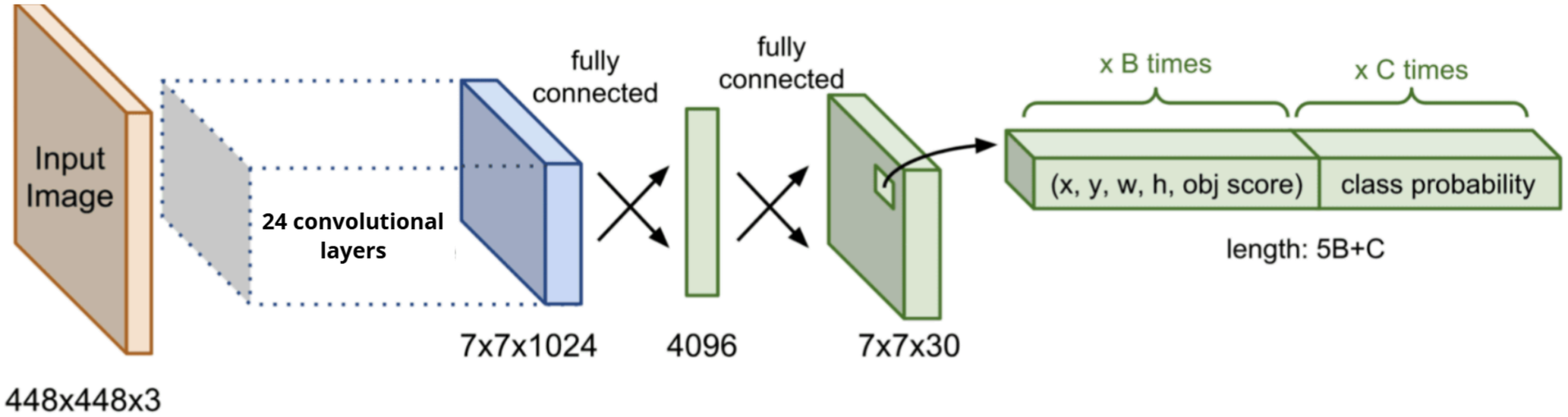


the detection head

the **backbone** : the core feature extraction network, inspired by GoogleNet's architecture.

the **bounding boxes**
(x,y) : the box's center
The **confidence score** : indicates how likely it contains an object, as well as its accuracy
the **Class Probability .**

YOLOv1

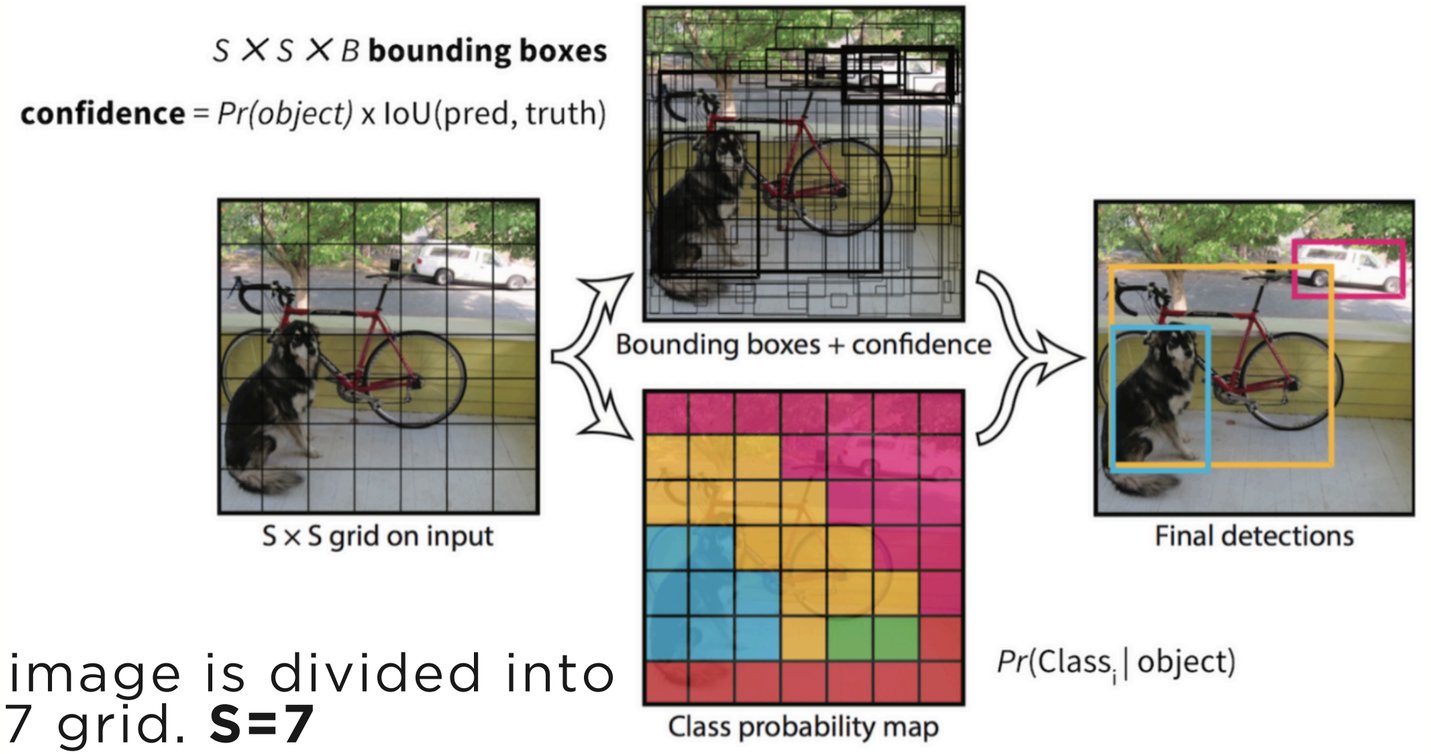


To avoid **overfitting**:
- dropout rate of 0.5
- extensive augmentation.

three parts to the **loss function**:
- localization loss,
- confidence loss,
- classification loss.

YOLOv1

Each grid cell predicts 2 bounding boxes and confidence scores. **B=2**

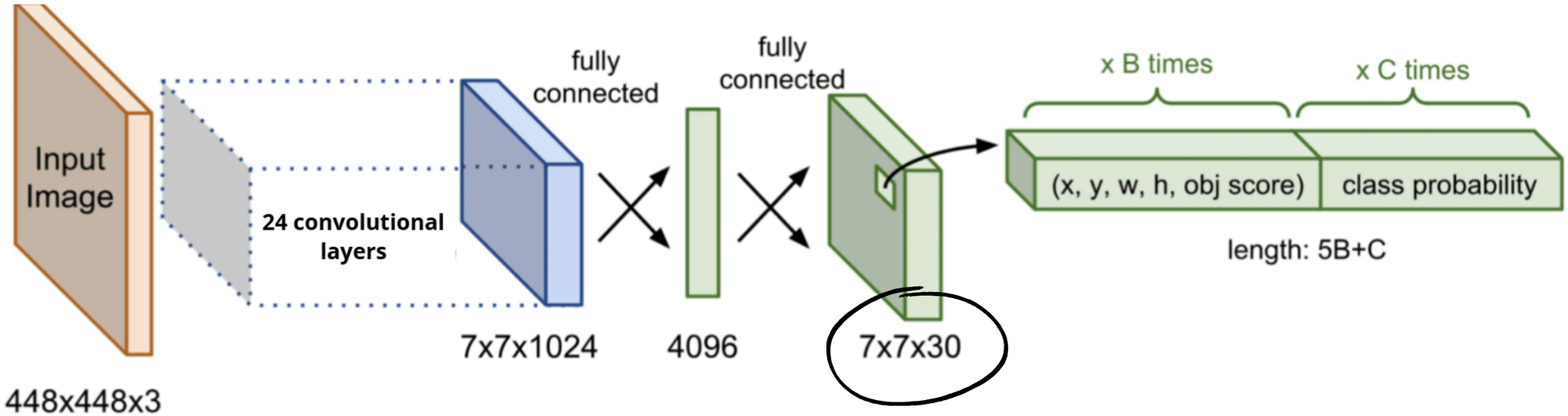


Total Predictions: 2 bounding boxes ×5 values (x,y,w,h,c) + 20 class scores = 30 values

The image is divided into a 7x7 grid. **S=7**

Each grid cell predicts the probability of each class. **C=20**

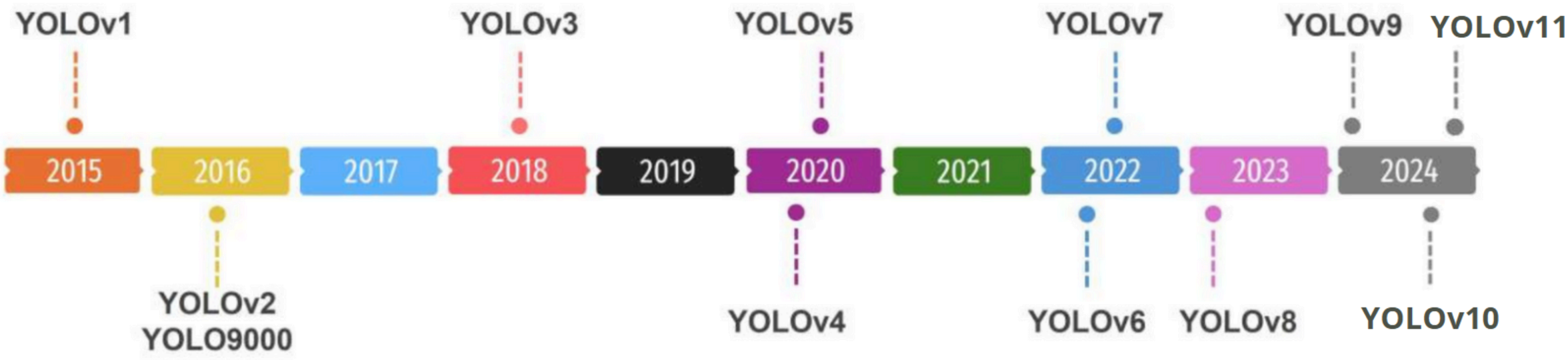
YOLOv1



- Speed
- Simplicity

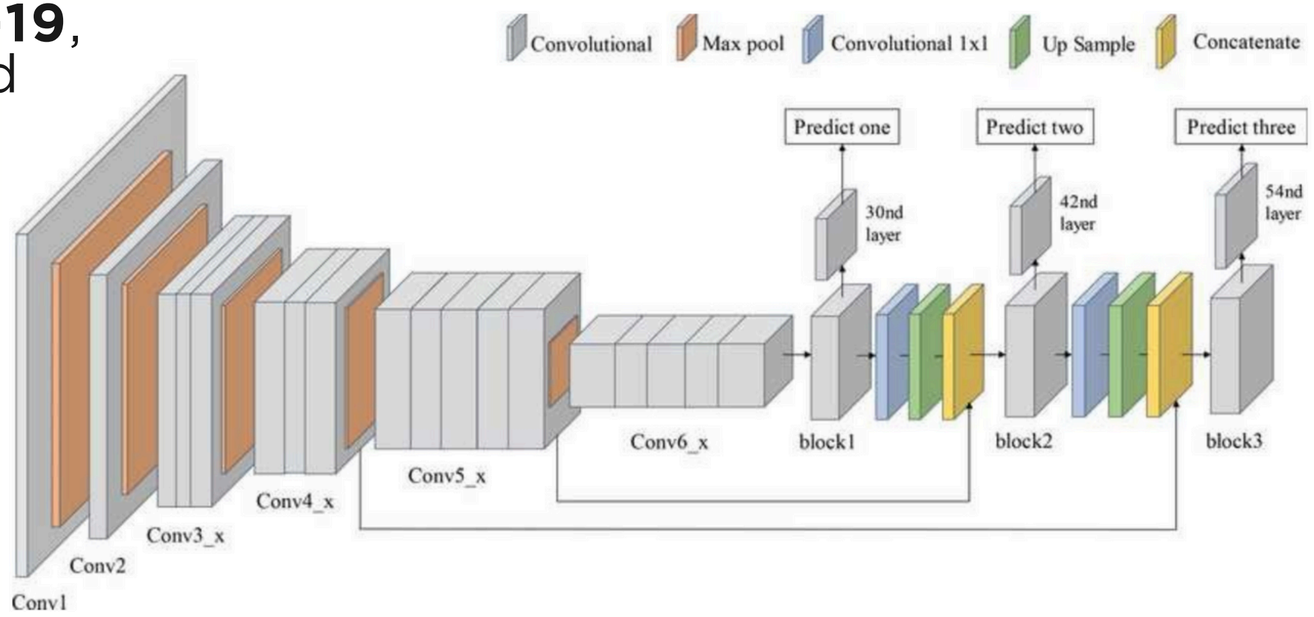
- Low Localization Accuracy: only 2 bounding boxes.
- Rough Predictions: The 7x7 grid size.
- Low Recall: particularly for smaller objects.

Timeline of YOLO Versions



YOLOv2

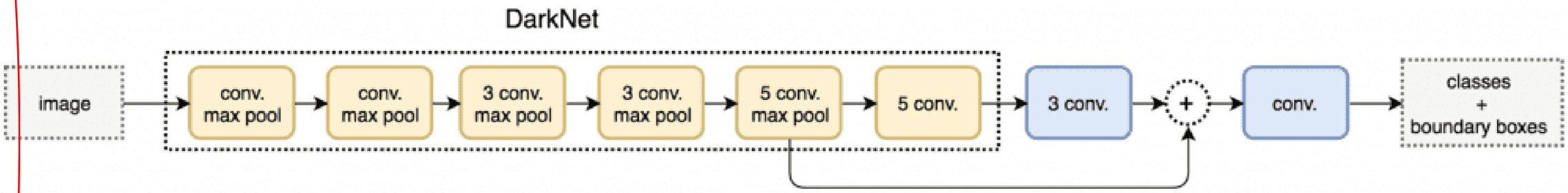
- Introduces **Darknet-19**, a more powerful and deeper architecture



passthrough layer that combines low-level, high-resolution features from earlier layers with high-level, low-resolution features.

YOLOv2

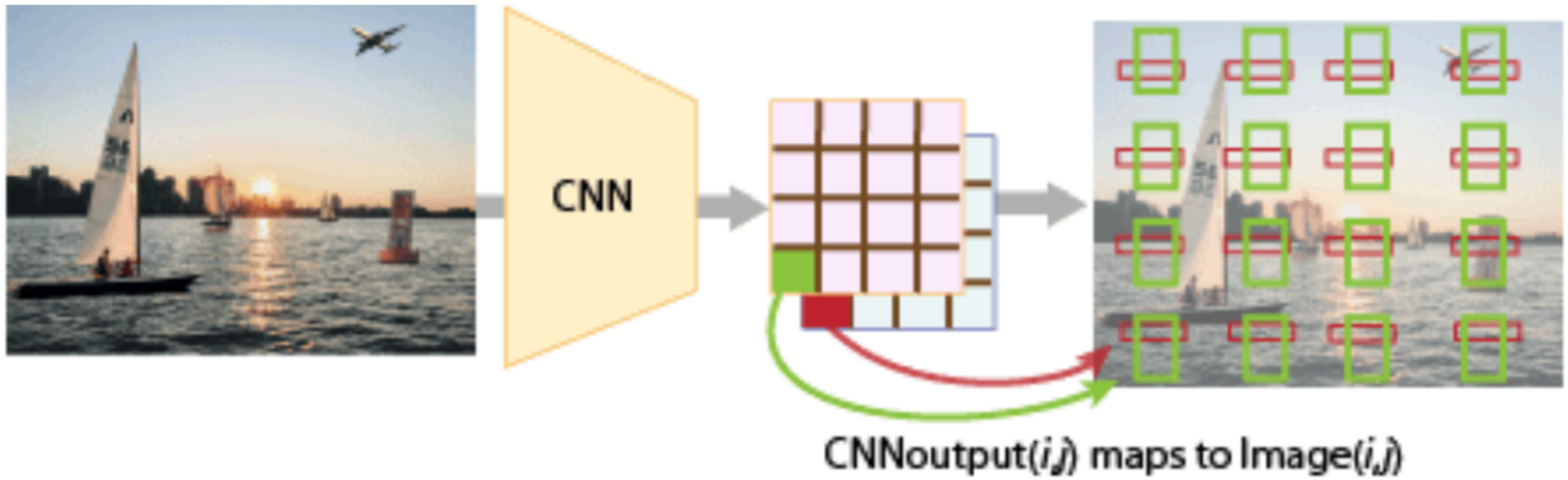
- Introduces **Darknet-19**, with max-pooling layers.



- allowing different image resolutions :
  ranging from 320x320 to 608x608 pixels
- Implements **batch normalization** across all convolutional layers

YOLOv2

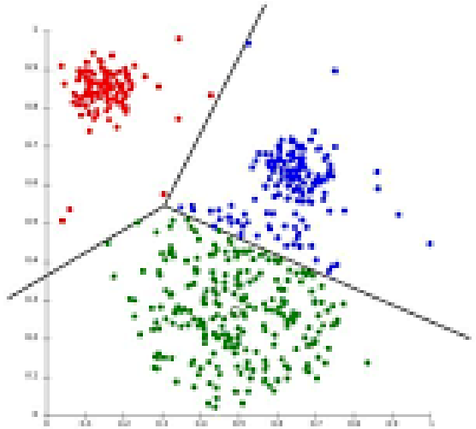
- Introduction to **Anchor Boxes**: predefined bounding boxes.
- “starting points” for predicting object locations :



Instead of predicting the bounding boxes directly from scratch, **the model predicts adjustments** to these anchor boxes to better fit objects in the image.

YOLOv2

- the authors proposed using **k-means clustering** on all the bounding boxes of the dataset to Set Anchor Box Dimensions.

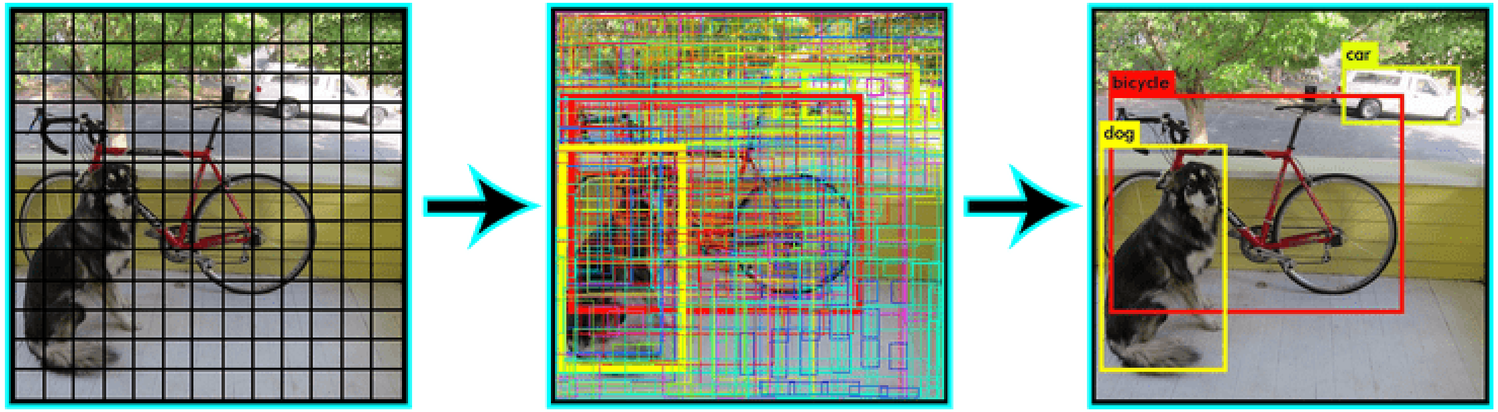


Anchor boxes reduce the complexity of the model by giving it reference bounding boxes to work from rather than predicting each box completely from scratch.

YOLOv2

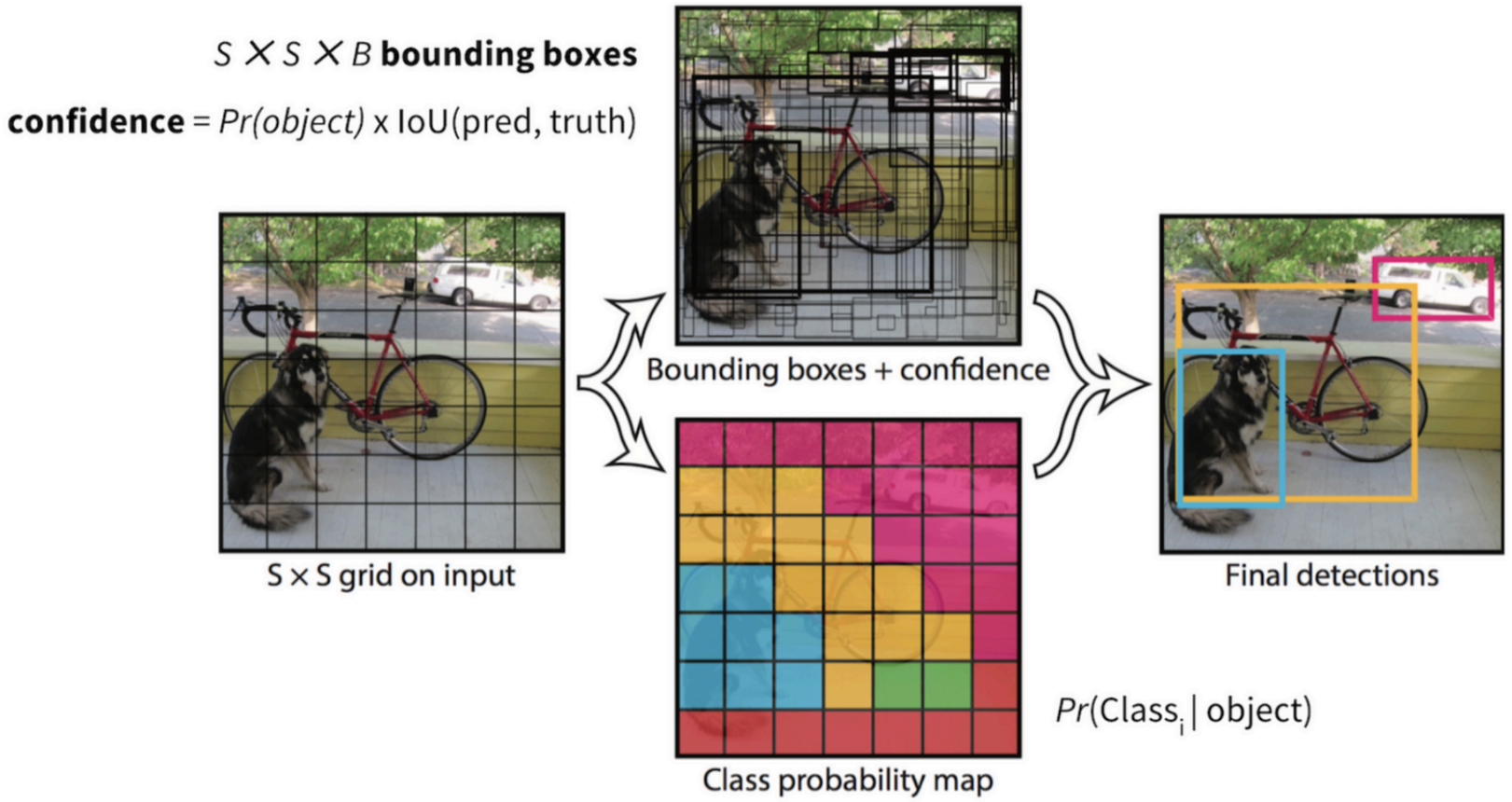
Each anchor box predicts (5 values):

- Adjustments to its position and size to match the detected object.
- A confidence score (probability that an object is within the box).
- Class probabilities for each object type.



Ability to Detect Multiple Objects in One Cell

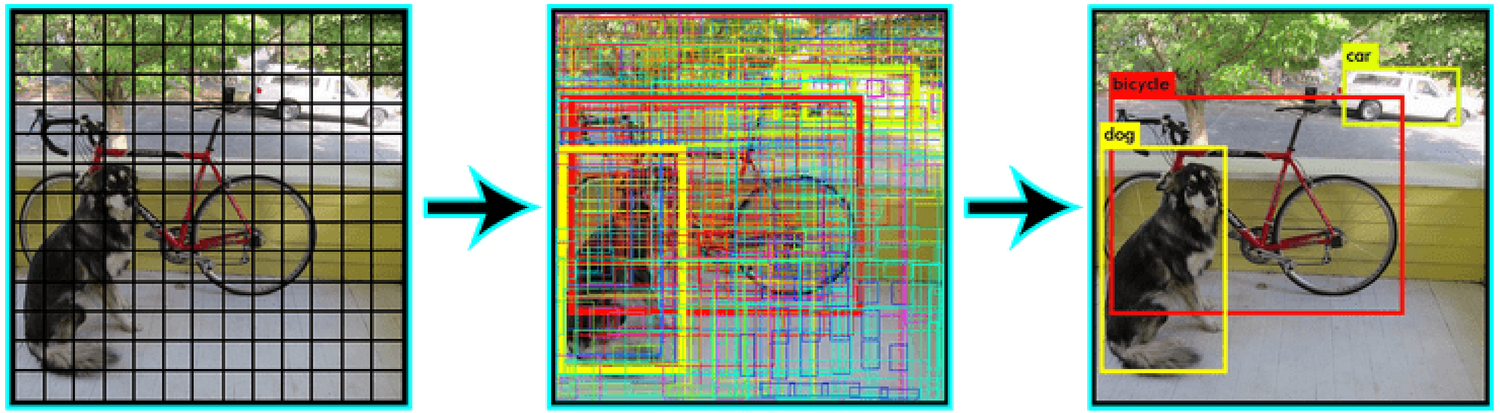
YOLOv1



detect 1 Objects for each Cell

YOLOv2

Each anchor boxe predicts the probability of each class. C=**80**



Total Predictions: 5 anchor boxes × (5 values (x,y,w,h,c) + 80 class scores) = 425 values

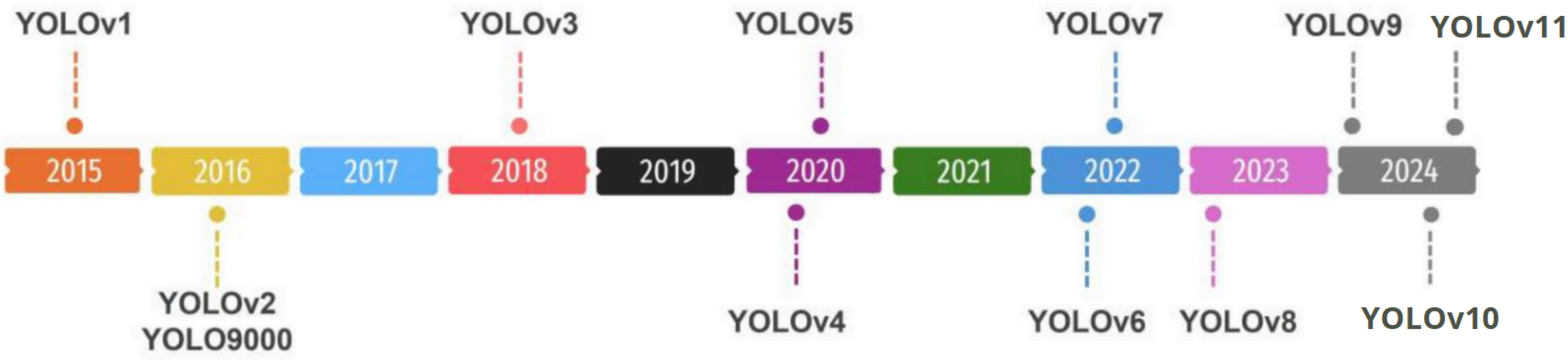
The image is divided into a **13x13** grid.

Each grid cell predicts **5 anchor boxes** and confidence scores.

- This results in a **13×13×425 output** tensor for the entire image

Aspect	YOLOv1	YOLOv2
Backbone Network	Custom CNN (24 conv + 2 FC layers)	Darknet-19 (19 conv + 5 max-pooling layers)
Bounding Box Prediction	Direct prediction (2 boxes per grid cell)	Anchor boxes (5 per grid cell, k-means clustered)
Input Resolution	Fixed at 448x448	Multi-scale (320x320 to 608x608)
Feature Fusion	Single scale features	Passthrough layer for fine-grained features
Pre-training	Trained from scratch	Uses ImageNet pre-trained weights
Performance	~45 FPS, lower mAP	~67 FPS, higher mAP
Additional Features	Basic object detection	YOLO9000 (9000+ classes), improved detection

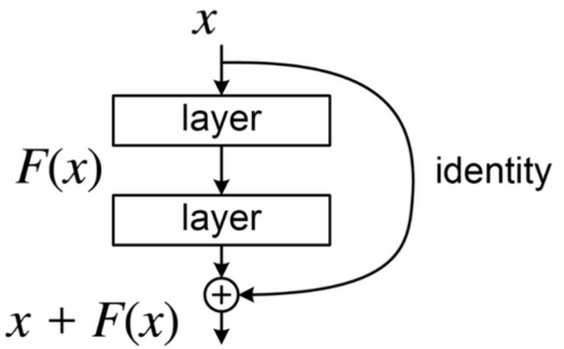
Timeline of YOLO Versions



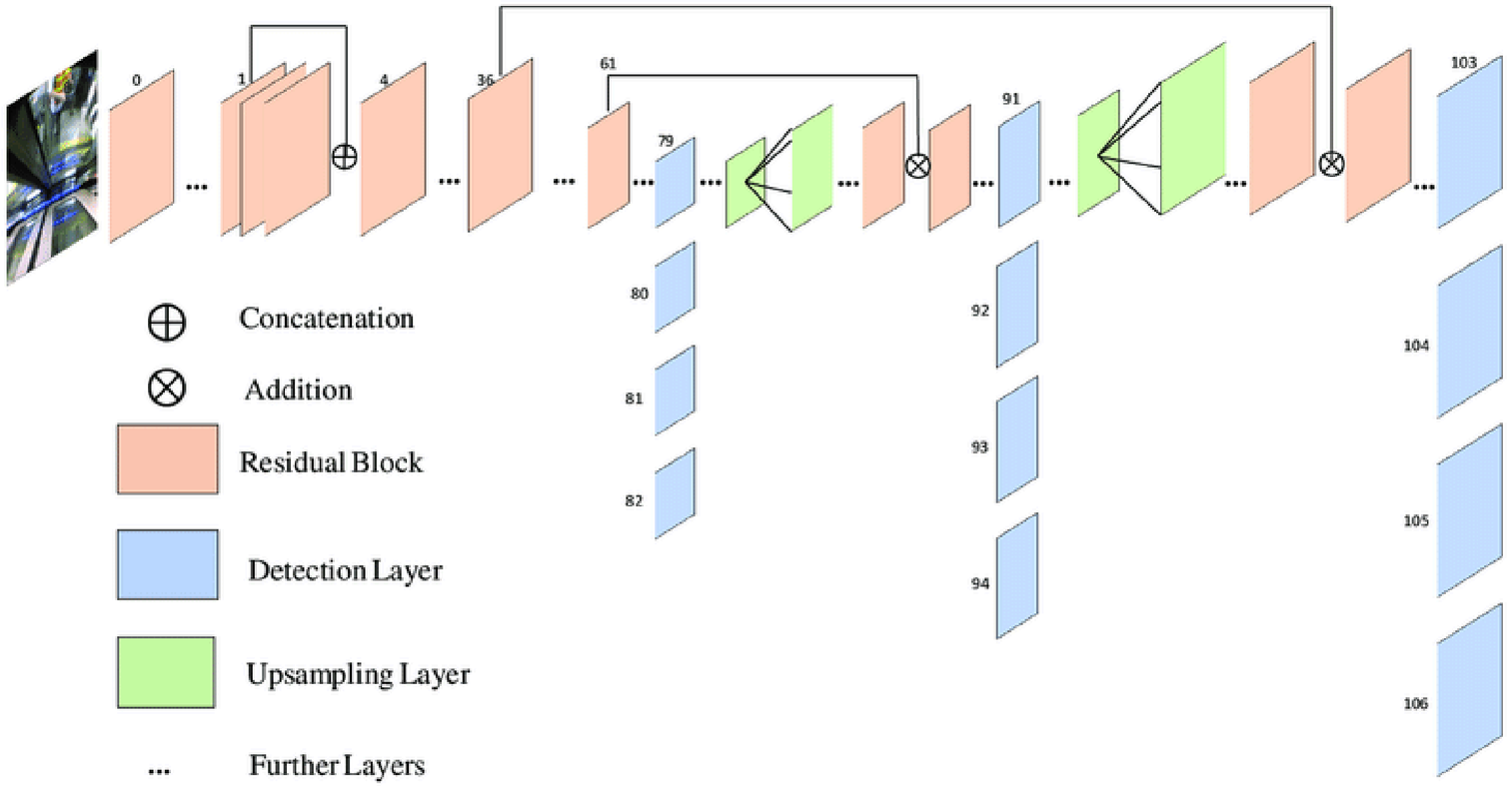
YOLOv3

- Upgraded to **Darknet-53** with 53 convolutional layers, Incorporating **residual blocks** throughout the network,

Residual blocks solve the problem of vanishing or exploding gradients in very deep networks.



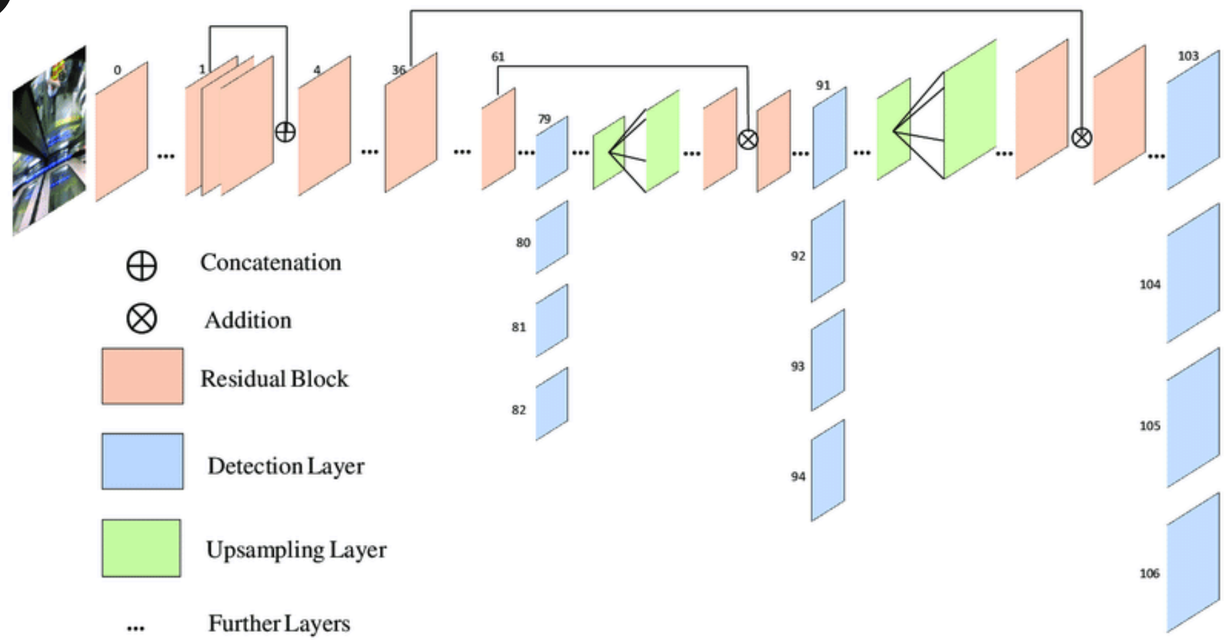
YOLOv3



- Introduces **multi-scale prediction** by making predictions at three different scales. (13x13, 26x26, 52x52)

YOLOv3

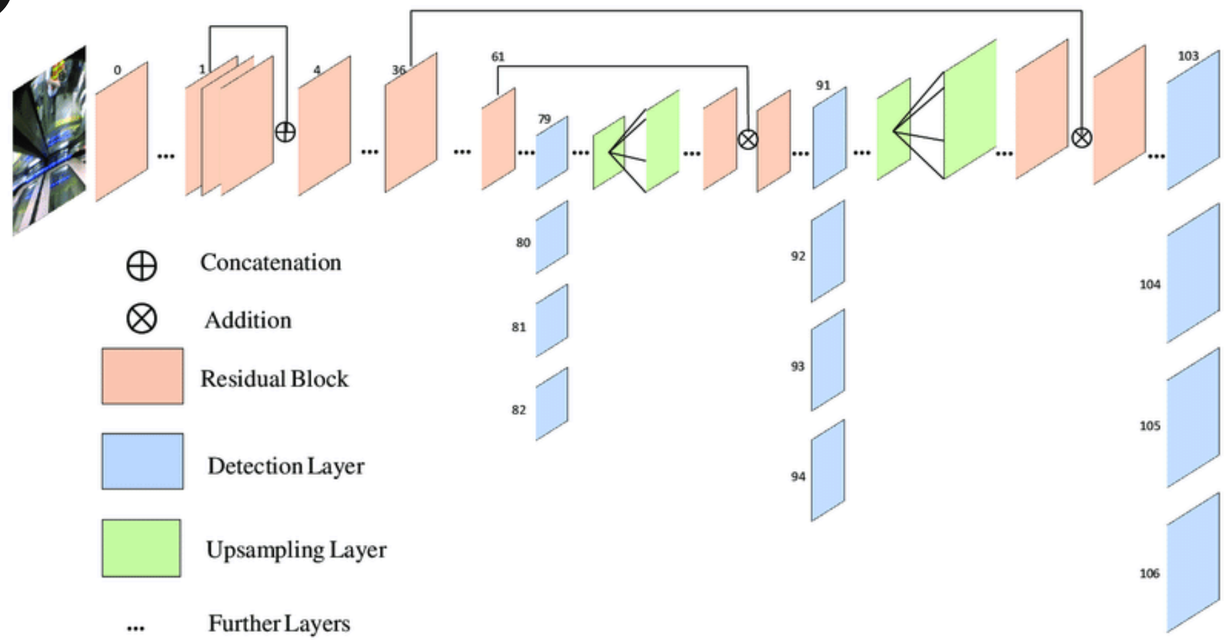
- Each detection layer is assigned **3 anchor boxes**, (9 anchor boxes divided across the three different scales.)



- **Total Predictions:** 3 anchor boxes ×(5 values + 80 class scores) = 255 values for each scale
- This results in a **13×13×255 | 26×26×255 | 52×52×255 output** tensor for the entire image.

YOLOv3

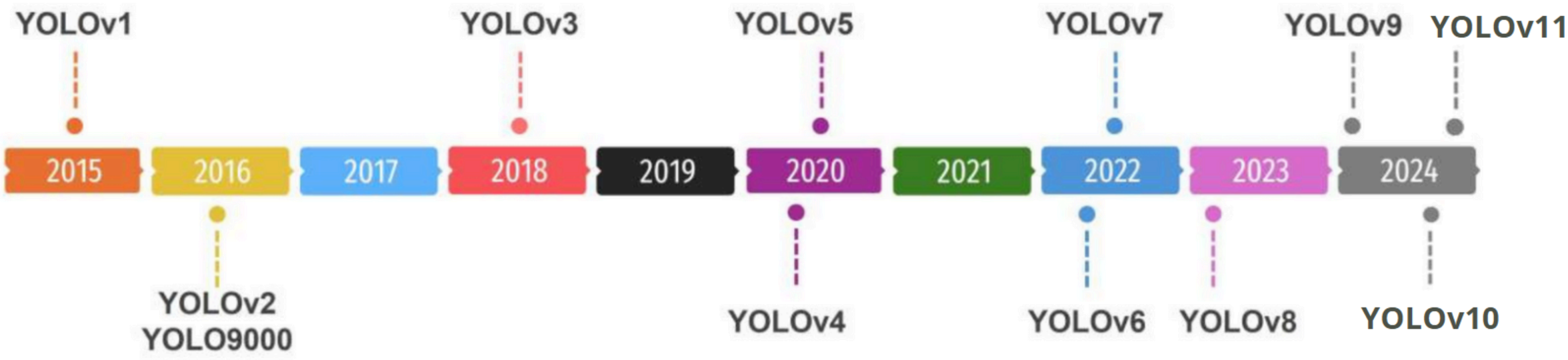
- Each detection layer is assigned **3 anchor boxes**, (9 anchor boxes divided across the three different scales.)



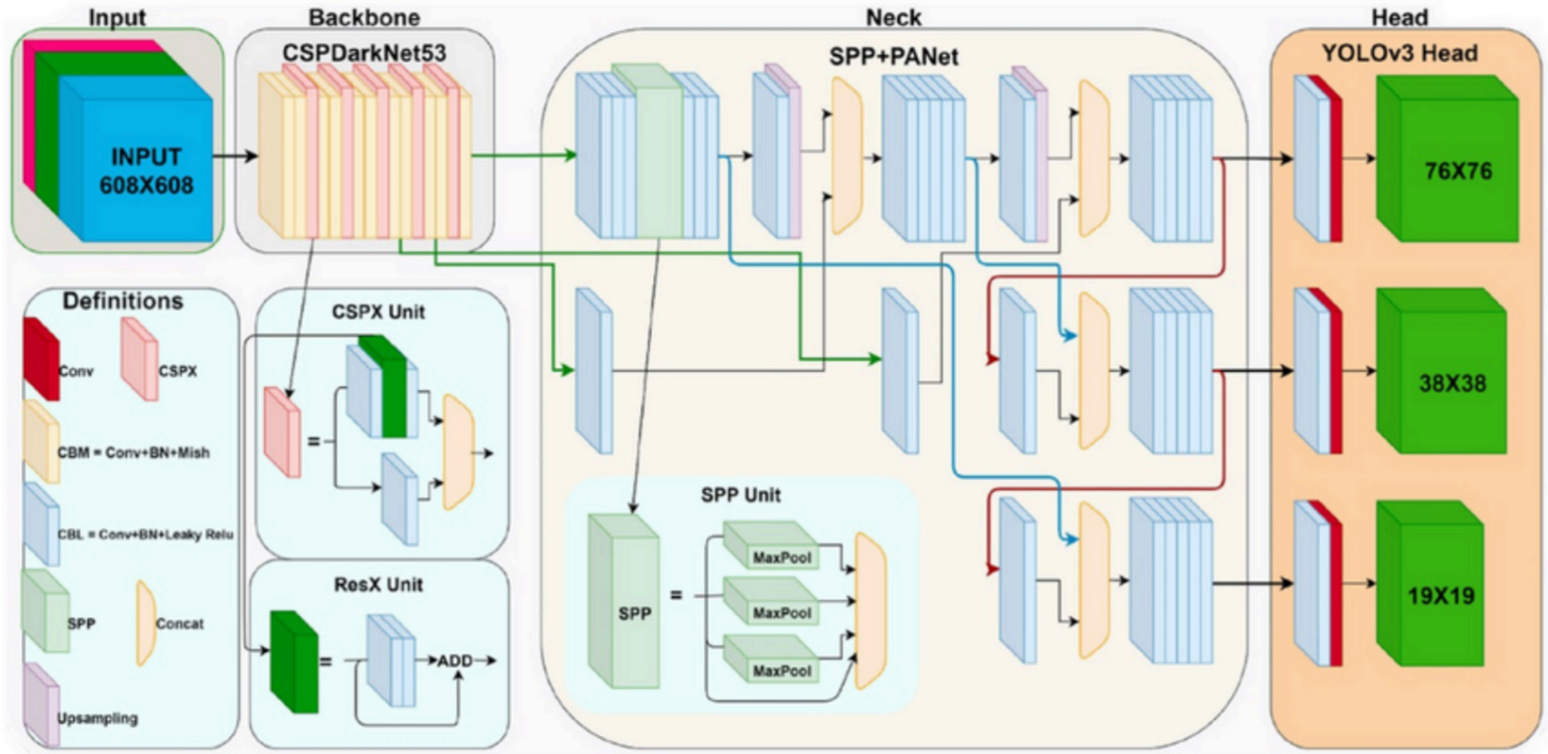
Replaces the softmax with independent logistic classifiers for each class, allowing **multi-label classification**
an object can belong to more than one class

Aspect	YOLOv2	YOLOv3
Backbone Network	Darknet-19	Darknet-53 with residual connections
Detection Scale	Single-scale	Multi-scale (3 scales for better accuracy)
Anchor Boxes	5 anchor boxes, single detection layer	9 anchor boxes, 3 scales (3 per scale)
Classification	Softmax (single-label classification)	Independent logistic classifiers (multi-label)
Objectness Score	Single sigmoid for object confidence	Refined logistic regression for better localization
Residual Blocks	Not used	Used, improving gradient flow and accuracy
Small Object Detection	Limited	Improved through multi-scale prediction
FPS (Frames per Second)	~45-67 FPS	~30-45 FPS
Overall Accuracy	Good for large objects	Improved accuracy, especially on small/multi-scale objects

Timeline of YOLO Versions

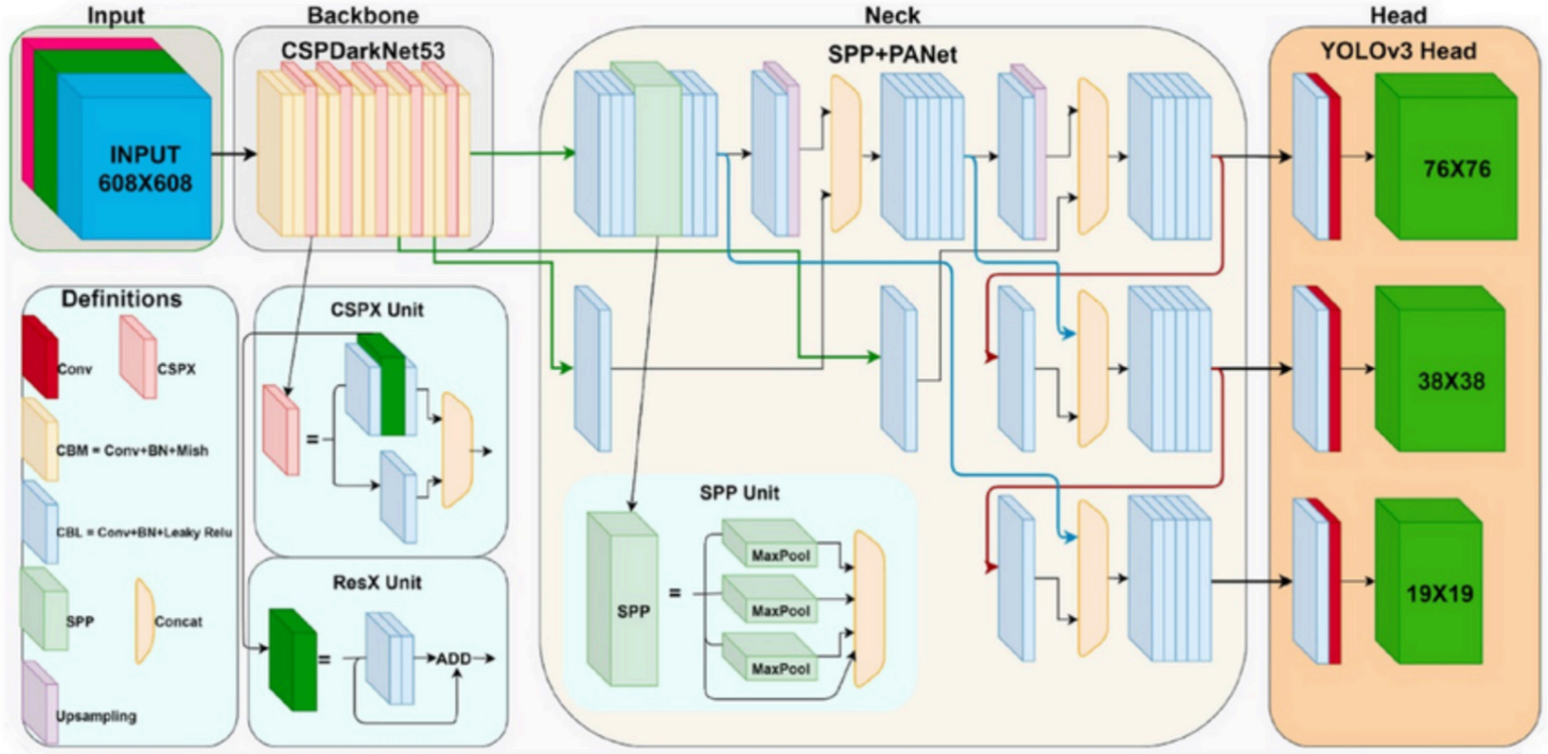


YOLOv4



- Uses **CSPDarknet-53** as the backbone, an improved version of Darknet-53.

YOLOv4



from YOLOv3 and YOLOv4, the authors introduced **the Neck**, placed between the backbone and the head.

- The use of this block is to collect features of different stages from the backbone and pass them to the head.

YOLOv4

In the paper, they introduced two concepts:
bag of freebies and bag of specials.

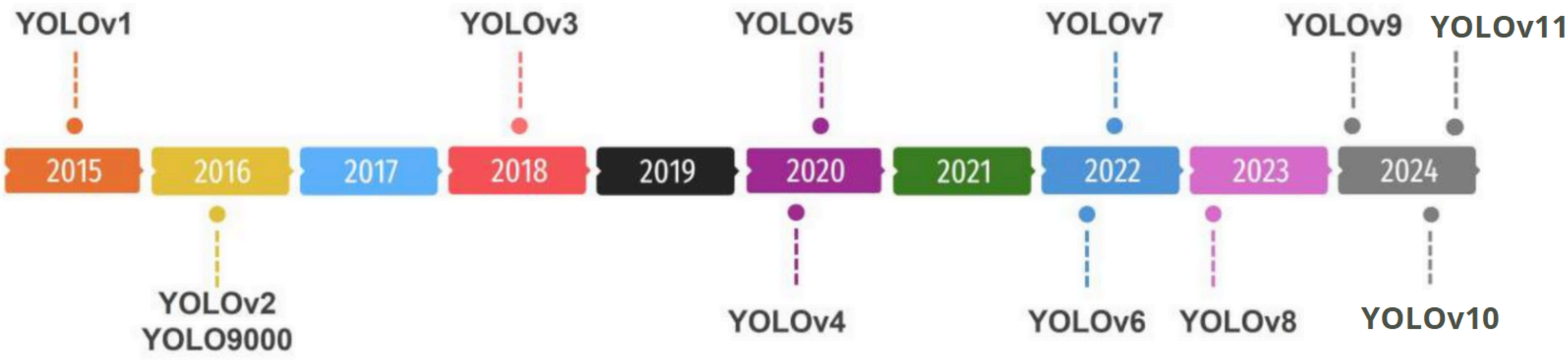
The concept of "**Bag of Freebies**" refers to techniques that modify the training approach or raise the training cost without affecting the cost during inference.
Data augmentation is a typical example of such a method.

In contrast, "**Bag of Specials**" includes methods that marginally increase the cost at inference time but substantially enhance accuracy.
Specific examples like SPP (Spatial Pyramid Pooling) and PANet (Path Aggregation Network)

Feature	YOLOv3	YOLOv4
Backbone Network	Darknet-53	CSPDarknet-53
Neck	None	PANet + SPP
Bag of Freebies	None	Mosaic Augmentation, SAT, CIoU
Bag of Specials	None	Mish, DIoU NMS, CmBN
Activation Function	Leaky ReLU	Mish
Speed (FPS)	~45 FPS	~45 FPS (but higher mAP)
Accuracy (COCO mAP)	~33%	~43%

mean Average Precision (mAP) on COCO Dataset

Timeline of YOLO Versions



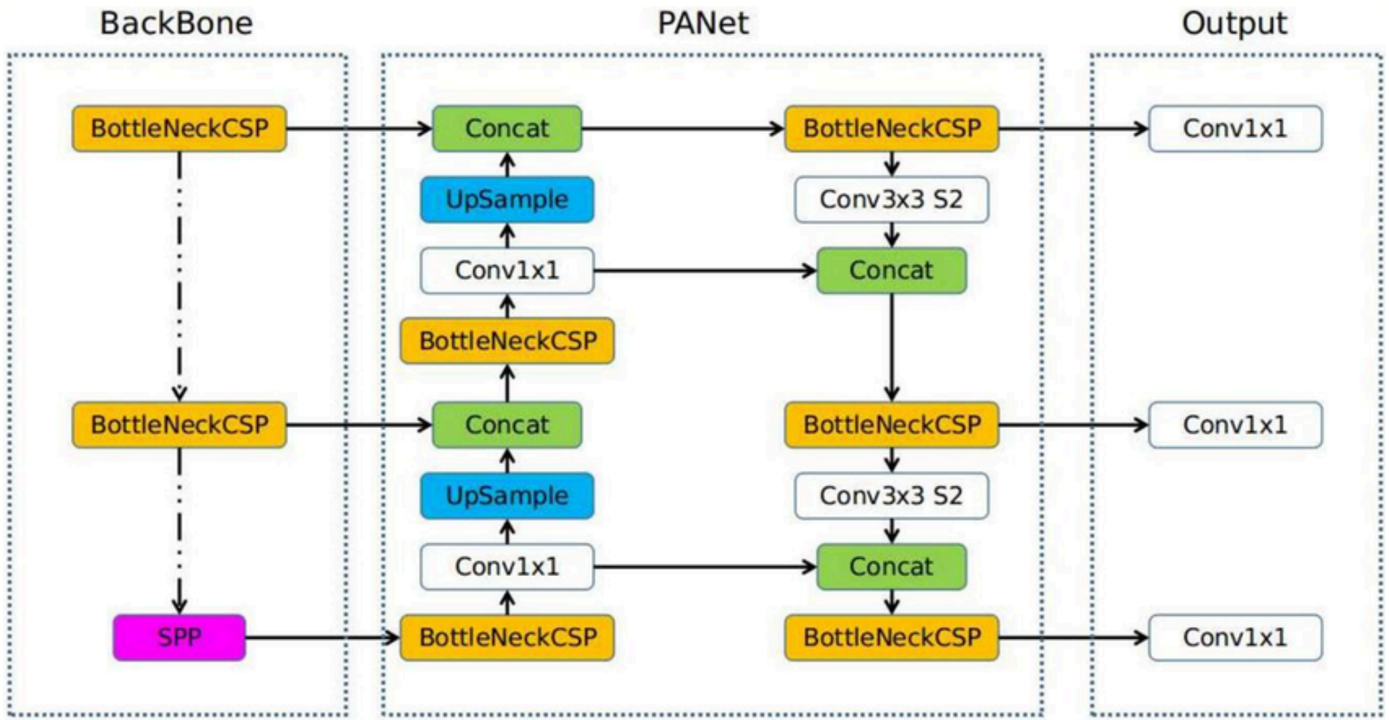
YOLOv5

While YOLOv1 to 4 was developed by the original YOLO community researchers with C-based framework,
YOLOv5 was created by Ultralytics, implemented in **PyTorch**,

This makes YOLOv5 more accessible and easy to deploy and adapt in various machine learning pipelines.

Uses a simpler, more efficient backbone and neck optimized directly for PyTorch

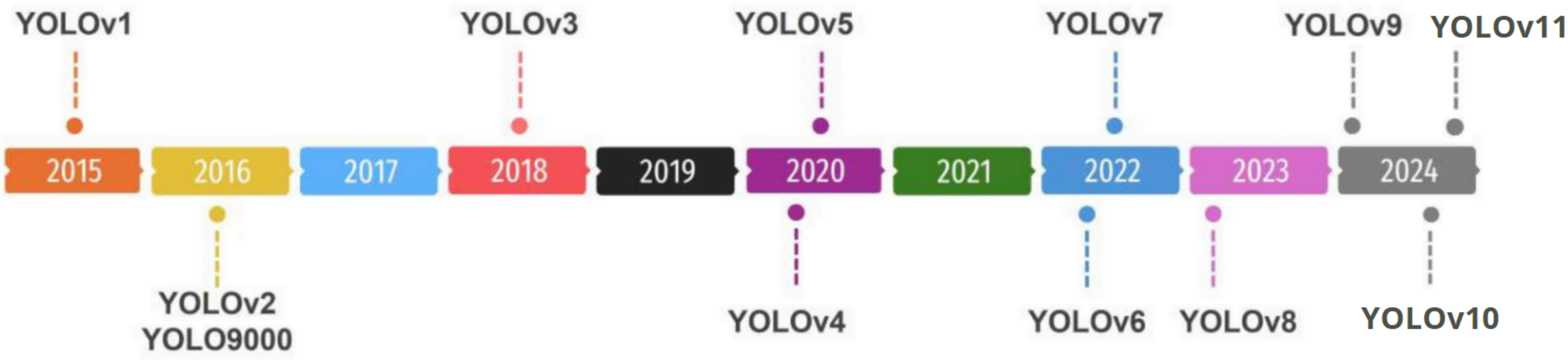
YOLOv5



it provides four model sizes (YOLOv5s, YOLOv5m, YOLOv5l, YOLOv5x) to balance speed and accuracy.

Aspect	YOLOv4	YOLOv5
Framework	Darknet	PyTorch
Backbone	CSPDarknet53	Custom backbone optimized for PyTorch
Neck	PANet + SPP	PANet with optimizations
Head	Three output layers for three scales	Three output layers for three scales
Model Sizes	Single size (manually configurable)	YOLOv5s, YOLOv5m, YOLOv5l, YOLOv5x
Training Complexity	Complex (requires Darknet with advanced tuning)	Simple (end-to-end in PyTorch)
Inference Speed	Slower than YOLOv3 but improved mAP	Faster, highly optimized for edge devices
Performance (COCO mAP)	~43%	~45% (varies by model size)
Deployment Flexibility	Limited by Darknet	High, with native PyTorch and ONNX support

Timeline of YOLO Versions



YOLOv7

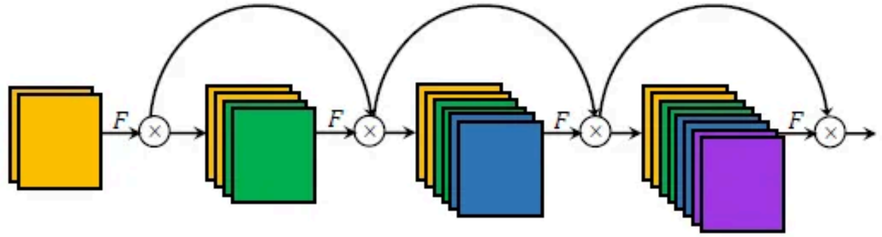
Introduced in July 2022, YOLOv7 marked a significant leap forward from its predecessors, showcasing improved accuracy and speed enhancements ranging from 5 FPS to 160 FPS.

The YOLOv7 model introduces novel modifications such as E-ELAN, Model Scaling, Planned re-parameterized convolution, Coarse for auxiliary, and penalty for lead loss.

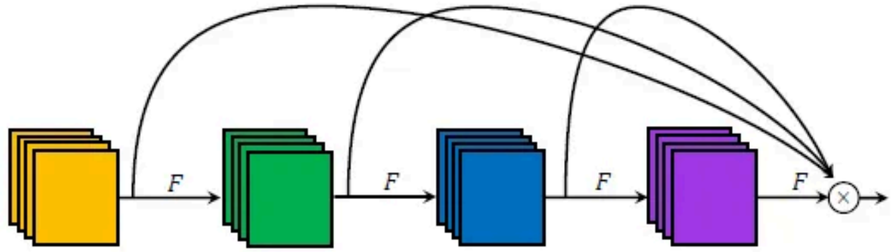
YOLOv7

1
2
3
4
5
6
7
8
9

Extended-ELAN (E-ELAN):



(a) Dense Aggregation (DenseNet)



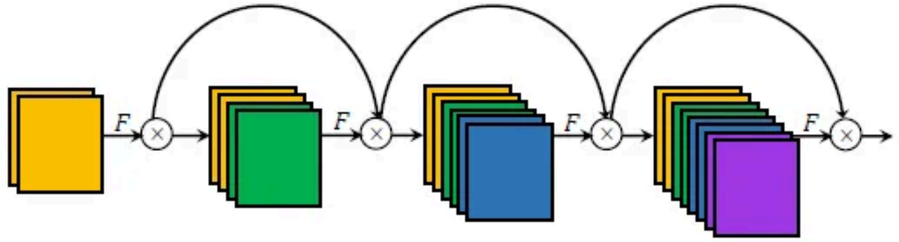
(b) One-Shot Aggregation (VoVNet)

E-ELAN is an Extended efficient layer aggregation network, a variant of ELAN. ELAN is inspired by VoVNet and CSPNet. We already know about CSPNet, and VoVNet is nothing but an Object Detection Network composed of cascaded OSA modules.

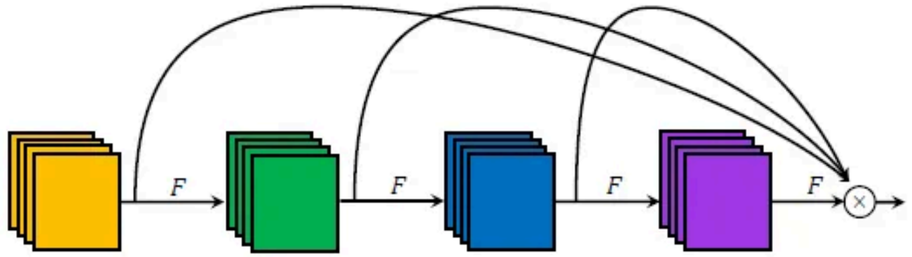
YOLOv7

OSA means One-shot Aggregation. It is more efficient than Denses Block in DenseNet and optimized for GPU computation. The below image clears the concept of OSA,

It is similar to DensesNet, but we concatenate the features after a few conv blocks here.



(a) Dense Aggregation (DenseNet)



(b) One-Shot Aggregation (VoVNet)

YOLOv7

1
2
3
4
5
6
7
8
9

Compound Model Scaling:

The authors used model scaling to adjust parameters to generate models of different scales. By different scales, it means a change in model parameters. It helps to meet the need for different inference speeds.
In this model, they used NAS (Network Architecture Search), which is a commonly used model scaling method, authors also showed that this method can be further improved using the compound model scaling approach.

YOLOv7

Model Re-parameterization:

Model re-parameterization means model weight averaging. There are two ways of doing model averaging,

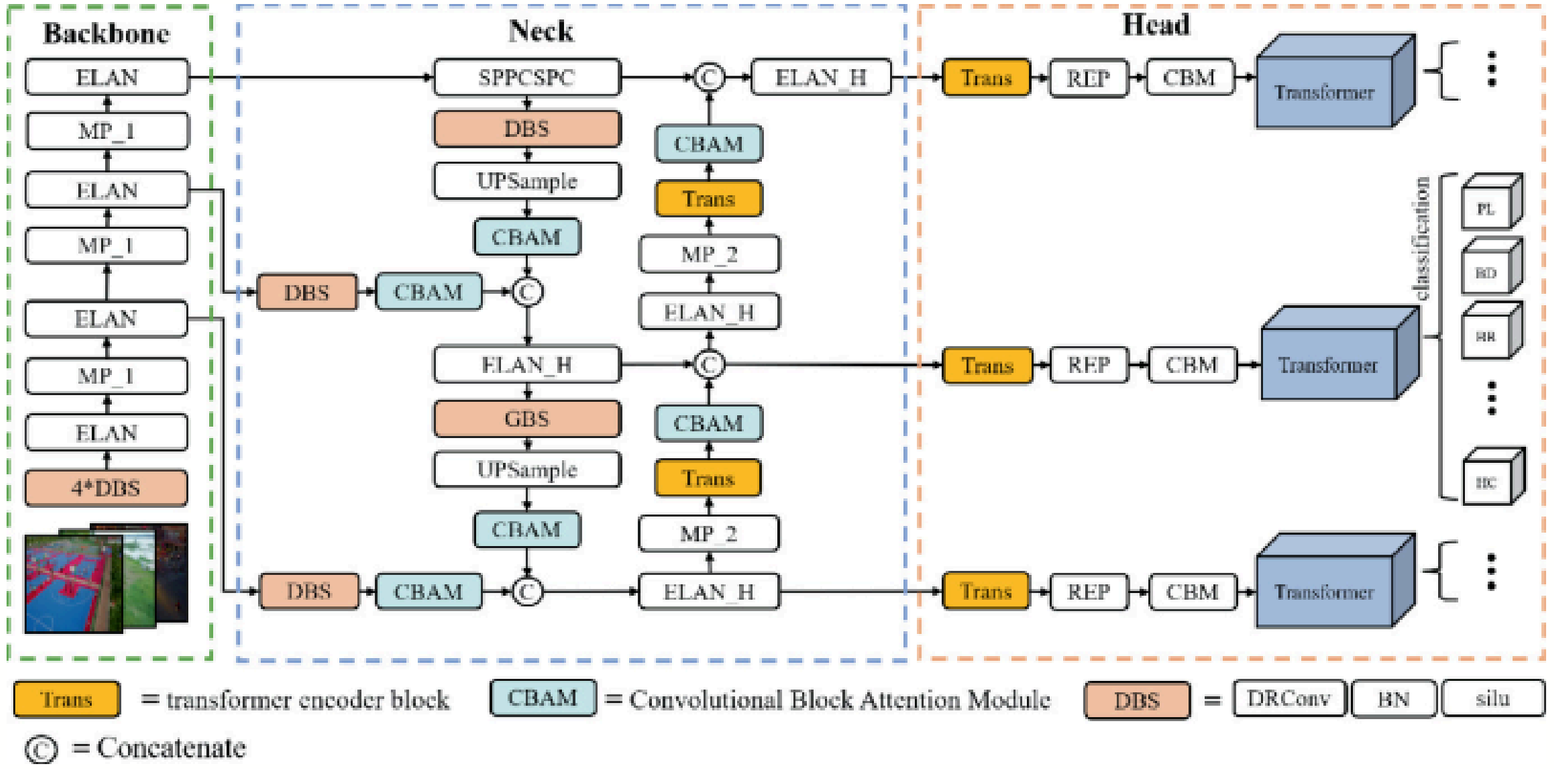
- Averaging the weights of the same model trained on different folds of data.
- Averaging the model weights of different epochs.

This is a very popular ensemble technique. Pytorch has an implementation of the same named SWA(Stochastic Weight Averaging).

YOLOv7

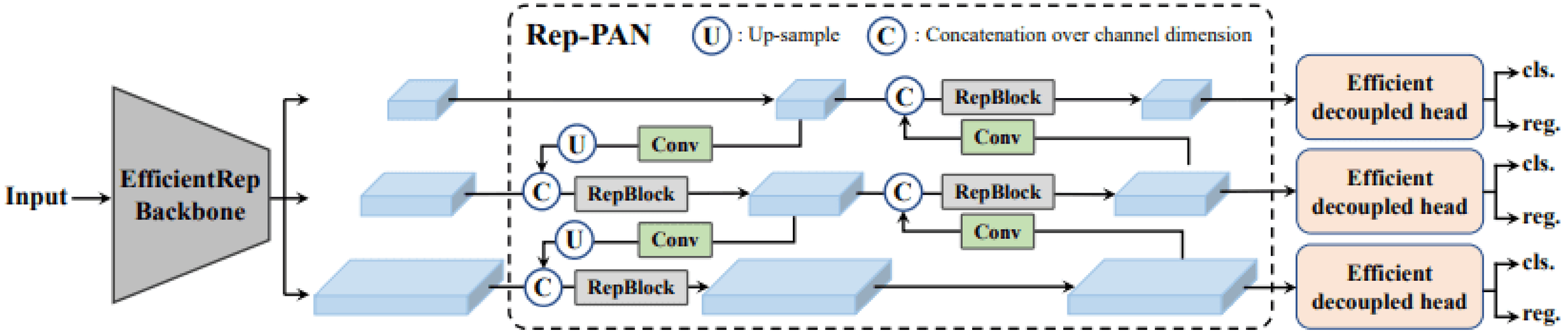
Coarse for Auxiliary and Fine for Lead Loss:

As you can see, the model architecture has multiple heads predicting the same thing. The head responsible for the final output is the lead head, and the other heads assist in the model training. With the help of an assistant loss, the weights of the auxiliary heads are updated.



YOLOv6

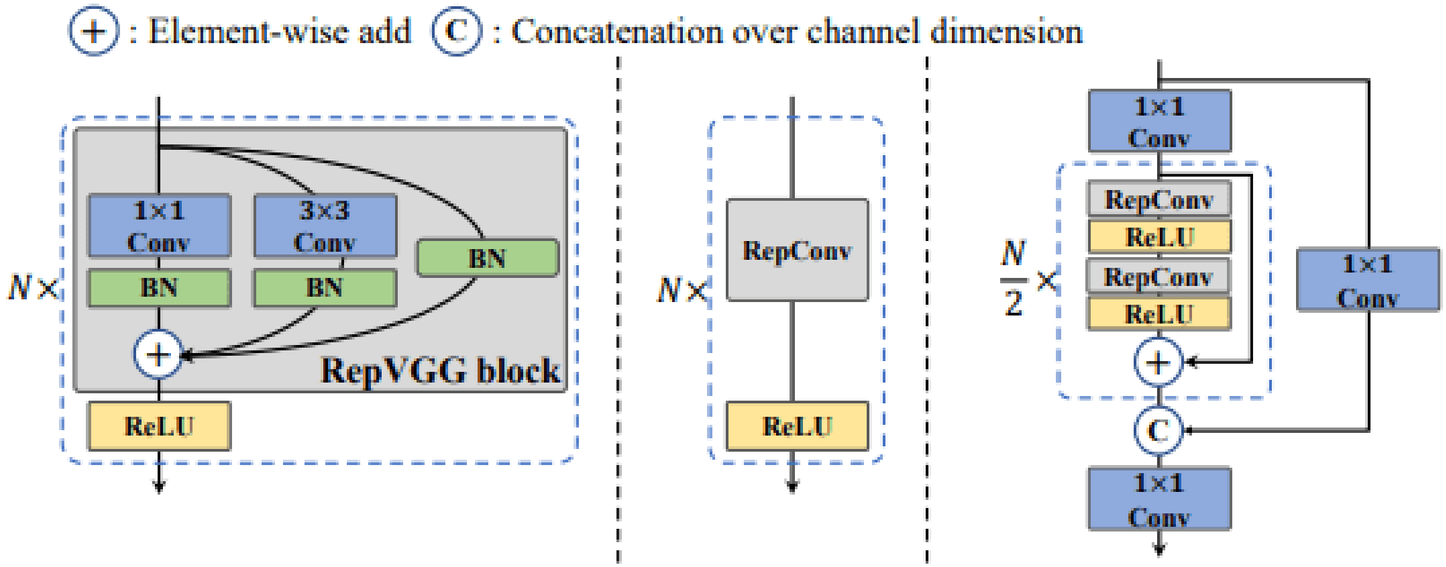
YOLOv6 is an anchor-free, decoupled head architecture with a unique backbone named EfficientRep.



Instead of predicting the bounding boxes directly from scratch, **the model predicts adjustments** to these anchor boxes to better fit objects in the image.

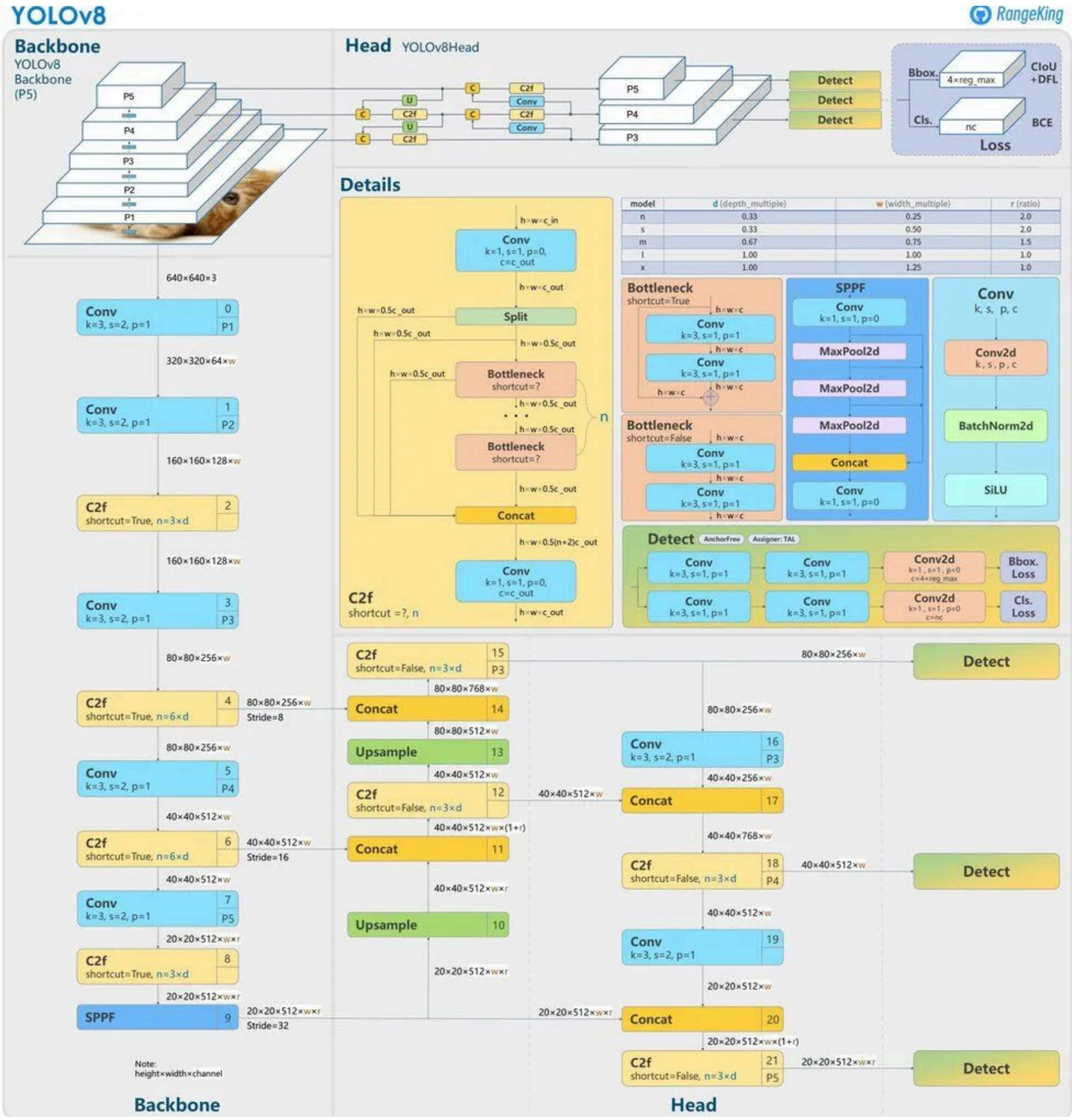
YOLOv6

EfficientRep



Compared with the CSP-Backbone used by YOLOv5, this backbone can efficiently utilize the computing power of hardware (such as GPU) and also has strong representation capabilities.

YOLOv8


YOLOv8
RangeKing
Backbone
YOLOv8 Backbone (P5)
Head
YOLOv8Head
Detect
Detect
Detect
Bbox.
4×reg_max
CIoU +DFL
Cls.
nc
BCE
Loss
Details
model
d (depth_multiple)
w (width_multiple)
r (ratio)
n 0.33 0.25 2.0
s 0.33 0.50 2.0
m 0.67 0.75 1.5
l 1.00 1.00 1.0
x 1.00 1.25 1.0
Bottleneck
shortcut=True
Bottleneck
shortcut=False
SPPF
Conv
k, s, p, c
Conv2d
BatchNorm2d
SiLU
C2f
shortcut =?, n
Detect
Anchorfree
Assigner: TAL
Bbox. Loss
Cls. Loss
640×640×3
Conv k=3, s=2, p=1 0 P1
320×320×64×w
Conv k=3, s=2, p=1 1 P2
160×160×128×w
C2f shortcut=True, n=3×d 2
160×160×128×w
Conv k=3, s=2, p=1 3 P3
80×80×256×w
C2f shortcut=True, n=6×d 4
80×80×256×w Stride=8
80×80×256×w
Conv k=3, s=2, p=1 5 P4
40×40×512×w
C2f shortcut=True, n=6×d 6
40×40×512×w Stride=16
40×40×512×w
Conv k=3, s=2, p=1 7 P5
20×20×512×w×r
C2f shortcut=True, n=3×d 8
20×20×512×w×r
SPPF 9
20×20×512×w×r Stride=32
Note: height×width×channel
Backbone
C2f shortcut=False, n=3×d 15 P3
80×80×768×w
Concat 14
80×80×512×w
Upsample 13
40×40×512×w
C2f shortcut=False, n=3×d 12
40×40×512×w×(1+r)
Concat 11
40×40×512×w×r
Upsample 10
20×20×512×w×r
80×80×256×w
Conv k=3, s=2, p=1 16 P3
40×40×256×w
Concat 17
40×40×768×w
C2f shortcut=False, n=3×d 18 P4
40×40×512×w
Conv k=3, s=2, p=1 19
20×20×512×w
Concat 20
20×20×512×w×(1+r)
C2f shortcut=False, n=3×d 21 P5
20×20×512×w×r
Detect
Detect
Detect
Head

YOLOv8

The YOLOv8 architecture follows the same architecture as YOLOv5, with a few slight adjustments, such as the use of the c2f module instead of CSPNet module, which is just a variant of CSPNet, (CSPNet followed by two convolutional networks).

They designed YOLOv8 anchor-free using YOLOv5 as a based model.

They also introduced Decoupled heads to independently process objectness, classification and bounding box prediction tasks. They used the sigmoid layer as the last layer for objectness score prediction and softmax for classification.

YOLOv8

YOLOv8 uses CIoU and DFL loss functions for bounding box loss and binary cross-entropy for classification loss. These losses have improved object detection performance, particularly when dealing with smaller objects. They also introduced the YOLOv8-Seg model for semantic segmentation by applying minimal changes to the original model.

YOLOv9

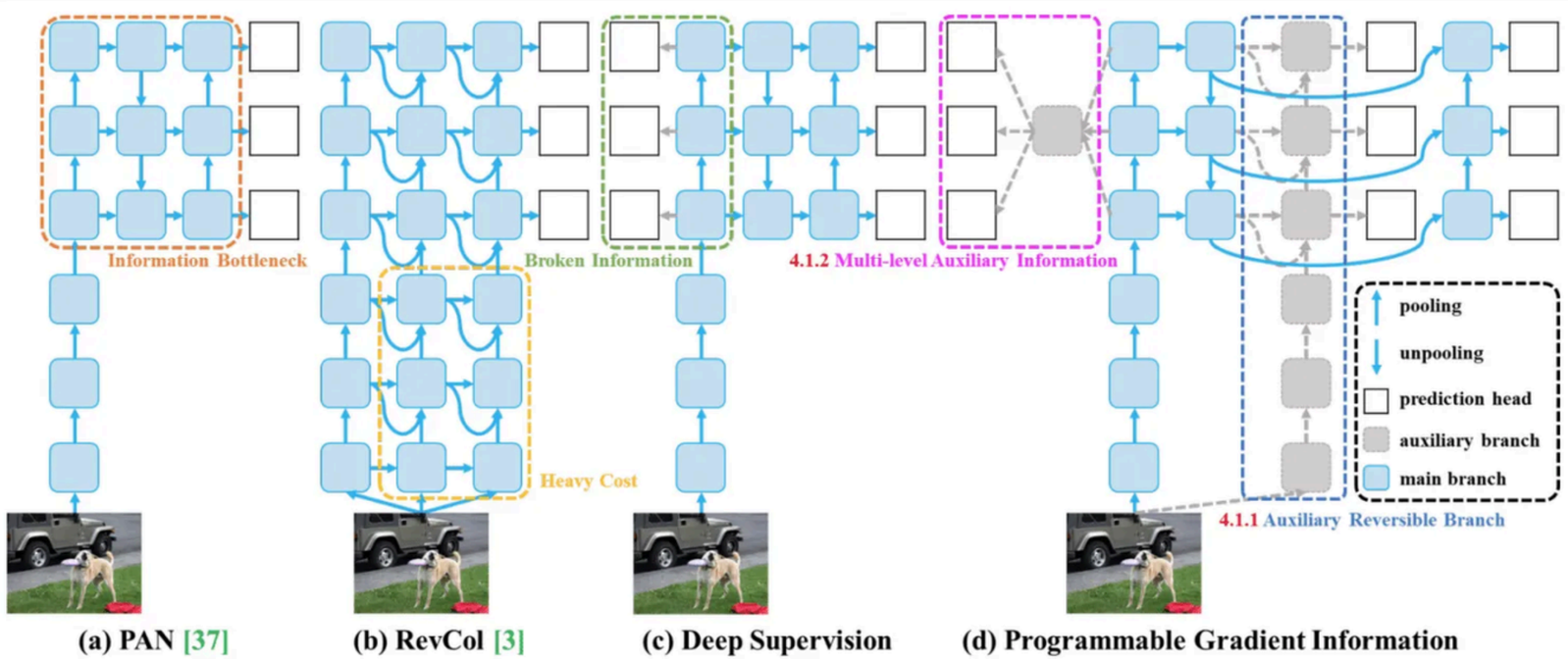


Figure 3. PGI and related network architectures and methods. (a) Path Aggregation Network (PAN)) [37], (b) Reversible Columns (RevCol) [3], (c) conventional deep supervision, and (d) our proposed Programmable Gradient Information (PGI). PGI is mainly composed of three components: (1) main branch: architecture used for inference, (2) auxiliary reversible branch: generate reliable gradients to supply main branch for backward transmission, and (3) multi-level auxiliary information: control main branch learning plannable multi-level of semantic information.

YOLOv9 boasts two key innovations: the Programmable Gradient Information (PGI) framework and the Generalized Efficient Layer Aggregation Network (GELAN).

YOLOv9

The PGI framework:

PGI is a novel concept introduced in YOLOv9 to combat the information bottleneck problem, ensuring the preservation of essential data across deep network layers. This allows for the generation of reliable gradients, facilitating accurate model updates and improving the overall detection performance.

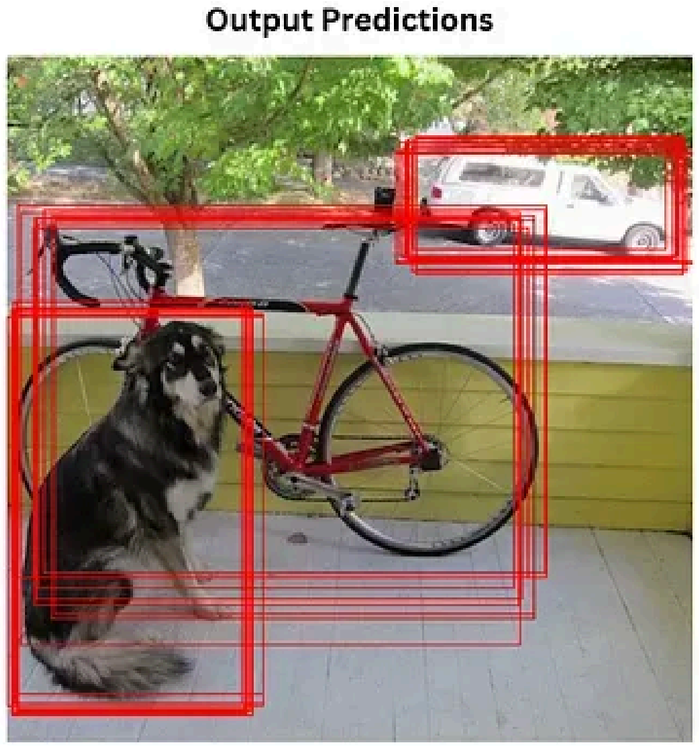
YOLOv9

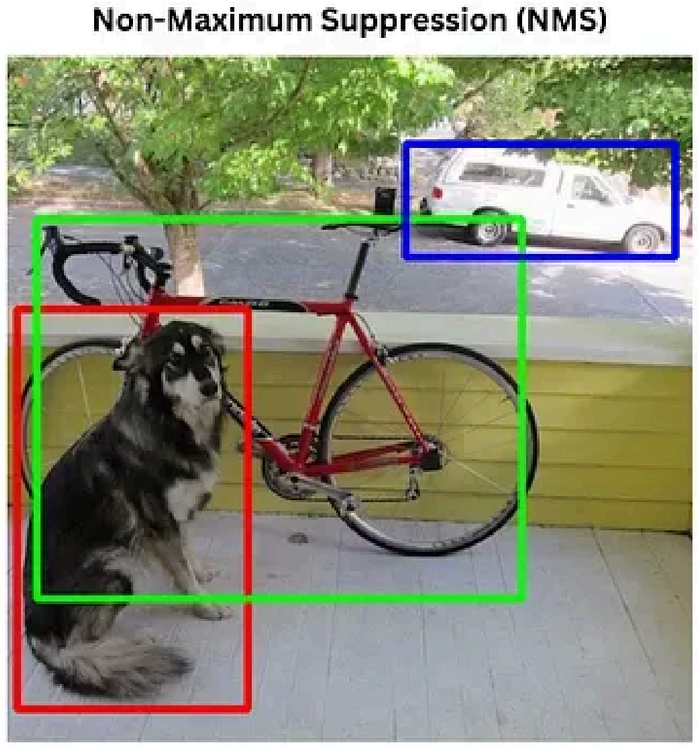
The GELAN architecture:

GELAN represents a strategic architectural advancement, enabling YOLOv9 to achieve superior parameter utilization and computational efficiency. Its design allows for flexible integration of various computational blocks, making YOLOv9 adaptable to a wide range of applications without sacrificing speed or accuracy.

By combining the above two frameworks (PGI and GELAN), YOLOv9 presents a significant advancement in lightweight object detection.

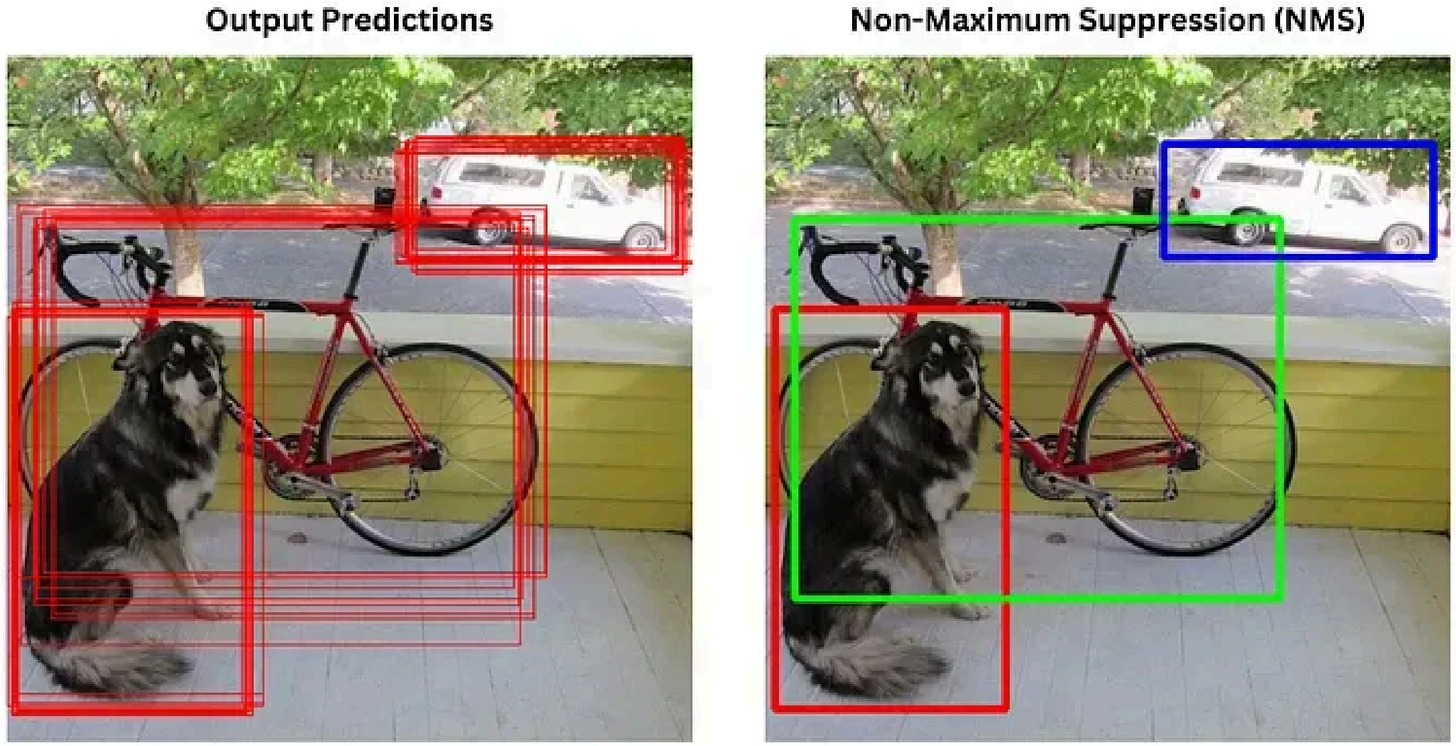
YOLOv10





YOLOv10 distinguishes itself by completely eliminating the reliance on non-maximum suppression (NMS) during post-processing, which is a significant step forward in enhancing inference speed.

YOLOv10



This model adopts a novel NMS-free training approach using dual label assignments, allowing for a harmonious integration of accuracy and speed by ensuring that the model remains computationally efficient while still capturing essential detection features.

Version	Date	Contributions	Framework
v1	2015	One-shot object detector	Darknet
v2	2016	Multi-scale training, dimensional clustering	Darknet
v3	2018	SPP block, Darknet-53	Darknet
v4	2020	Mish-based activation, CSPDarknet-53 backbone	Darknet

Version	Date	Contributions	Framework
v5	2020	Anchor-free detection, SWISH-based activation, PANet	PyTorch
v6	2022	Self-attention, anchor-free object detection	PyTorch
v7	2022	Transformers, E-ELAN reparameterization	PyTorch
v8	2023	GANs, anchor-free detections	PyTorch

Version	Date	Contributions	Framework
v9	2024	Programmable Gradient Information (PGI), Generalized Efficient Layer Aggregation Network (GELAN)	PyTorch
v10	2024	NMS-free training approach, dual label assignments, holistic model design for enhanced accuracy and efficiency	PyTorch

EMOTION DETECTION USING YOLO

1

2

3

4

5

6

7

8

9

Context

Emotion detection is key in **security, customer service,** and **mental health.**

Why YOLO:

- YOLO offers fast, efficient detection, ideal for real-time needs.
- Adapting YOLO enables instant emotional recognition in live video.

Objective

Detect and classify people emotions in real time, achieve a certain level of accuracy

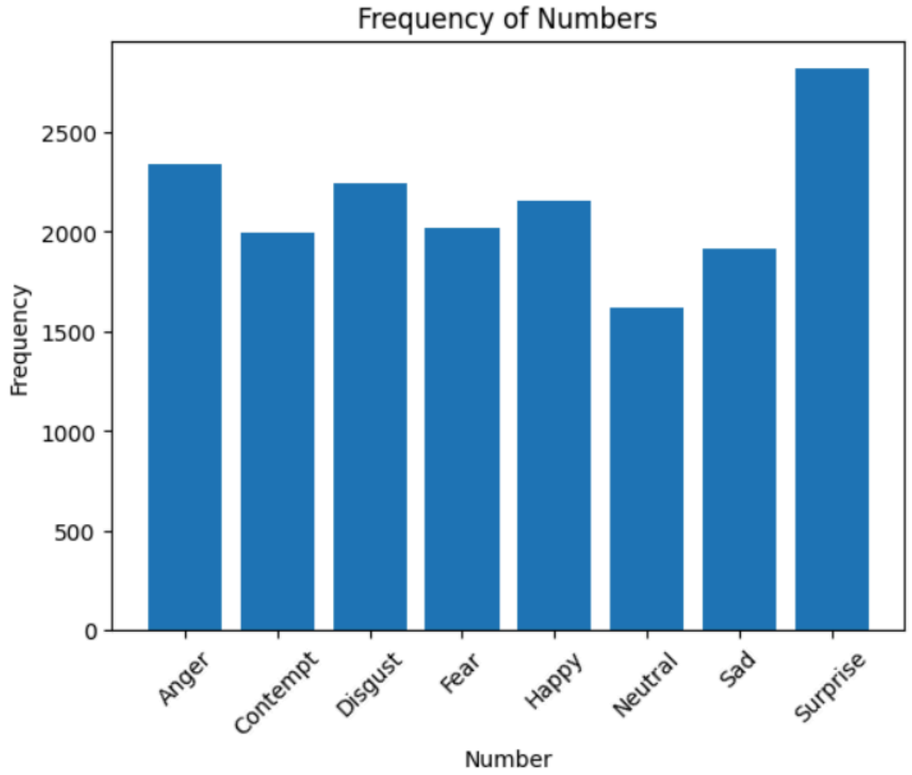
1
2
3
4
5
6
7
8
9

Data Understanding

- affectnet-yolo-format
  - YOLO_format
    - test
      - images
      - labels
    - train
      - images
      - labels
    - valid
      - images
      - labels
    - data.yaml

Data Understanding

- affectnet-yolo-format
  - YOLO_format
    - test
      - images
      - labels
    - train
      - images
      - labels
    - valid
      - images
      - labels
    - data.yaml

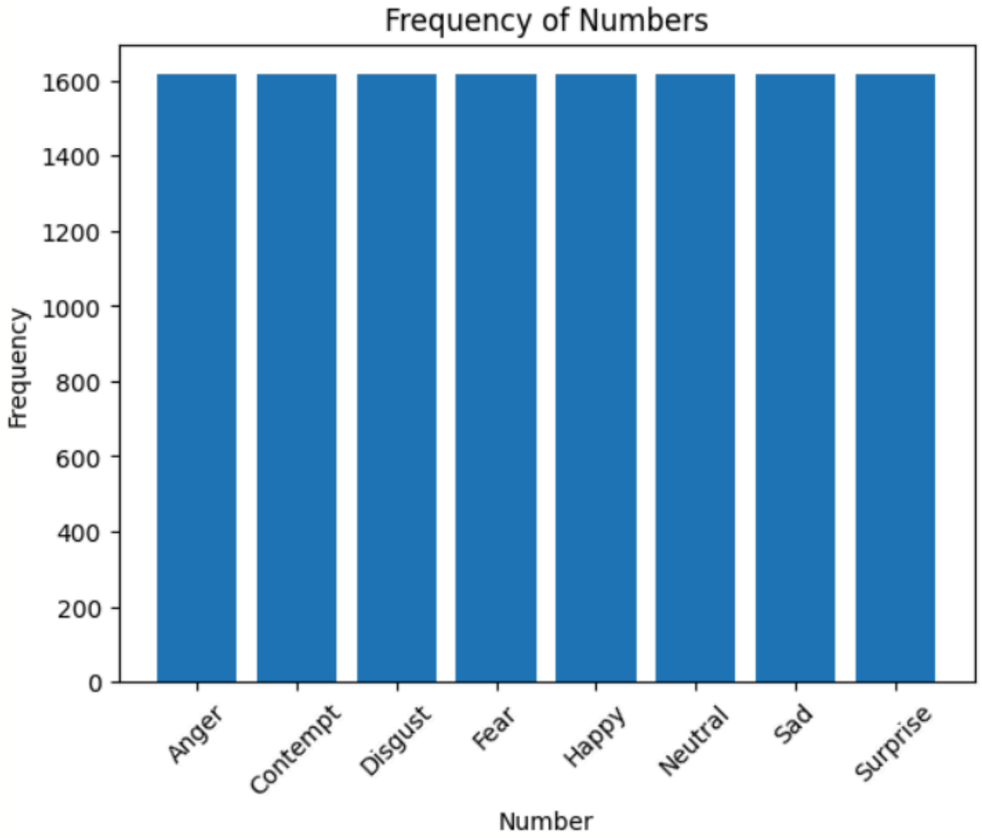


Valid Files: 5406

Test Files: 2755

Train Files: 17101

Data Preparation



Balanced dataset creation:

- Created a new yolo-data folder:
- Copied the valid and test folders.
- Balanced the train folder.
- Updated the data path in data.yaml.

HyperParameter Fine-tuning

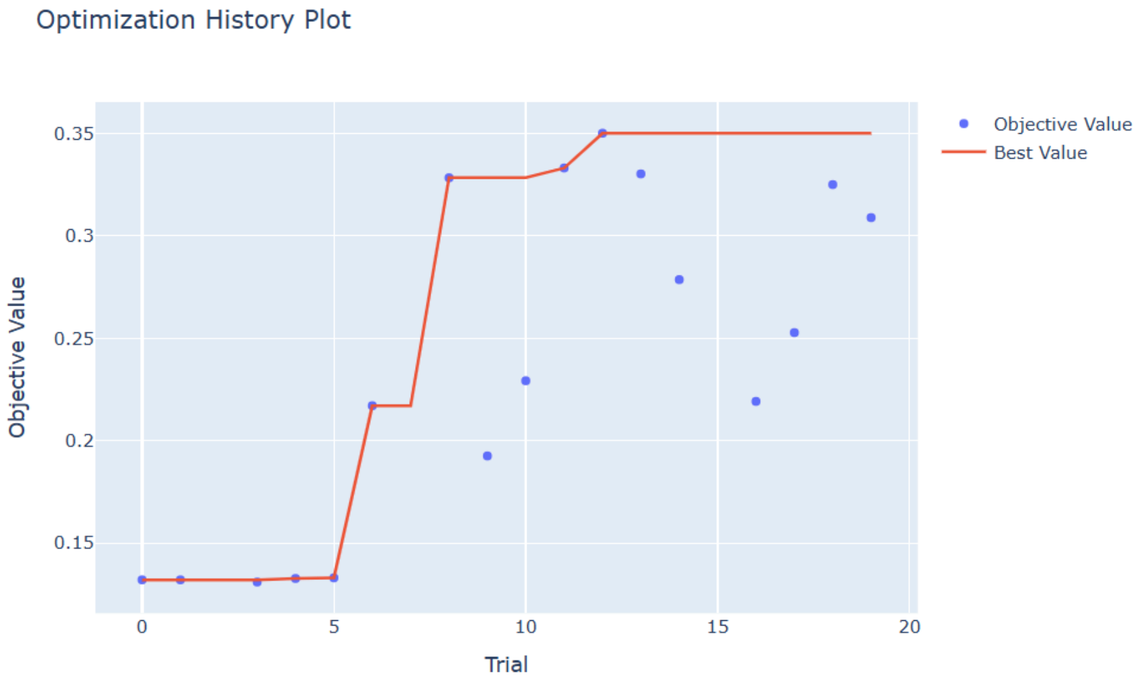
Using **Optuna** to test **20 combinations** of hyperparameters and maximize the model's performance the validation metric (**mAP**).

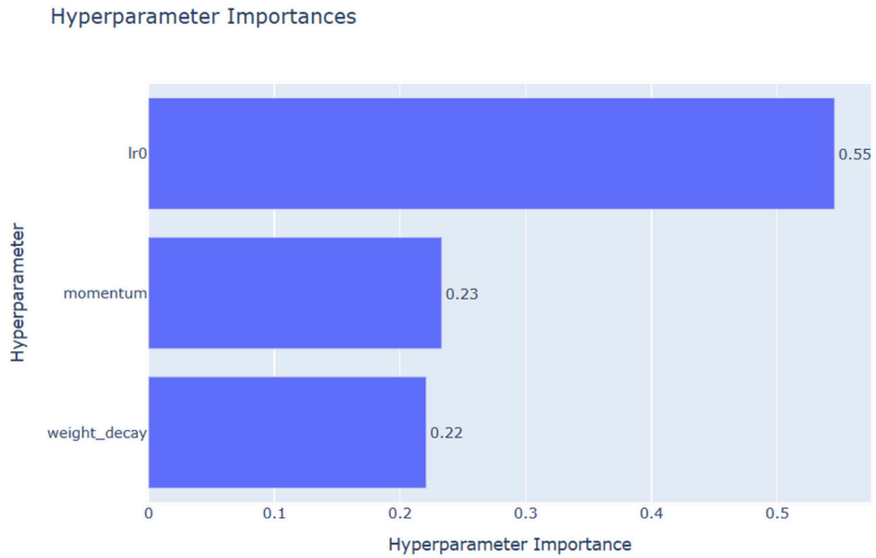
- Learning rate: between 0.0001 et 0.1 (log-uniform).
- Momentum: between 0.8 and 0.98.
- Weight decay: between 0.000001 et 0.001 (log-uniform).

Training the model for **2 epochs** to evaluate its performance.

HyperParameter Fine-tuning

- Lr0: 0.00010446991982747811
- momentum: 0.881082406956181
- weight_decay: 7.273003893562258e-06





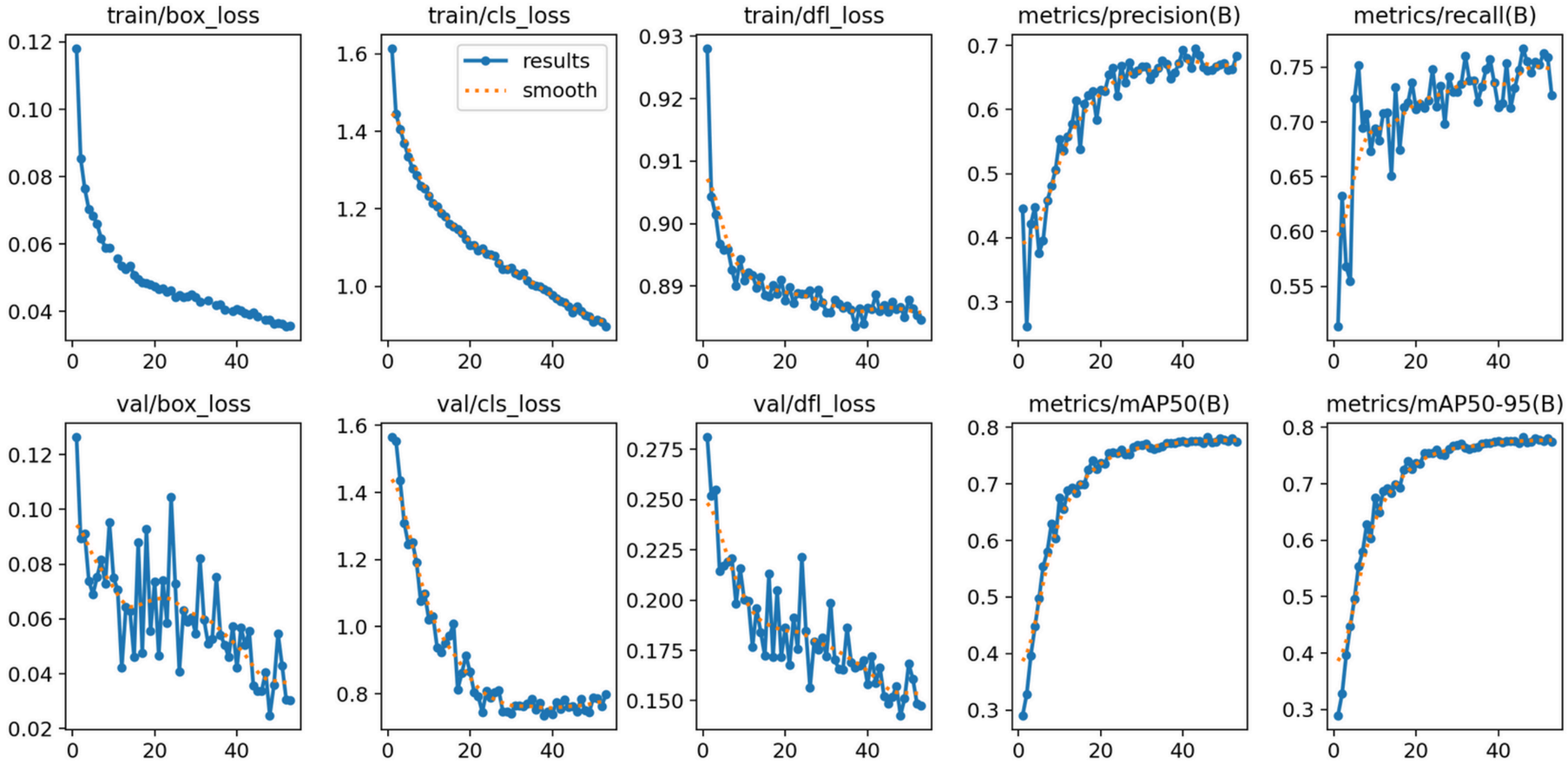
1
2
3
4
5
6
7
8
9

Model Training

Optuna automatically identifies the best hyperparameter combination, which is then used to train the model for **100 epochs** with an early stopping **patience of 7 epochs.**

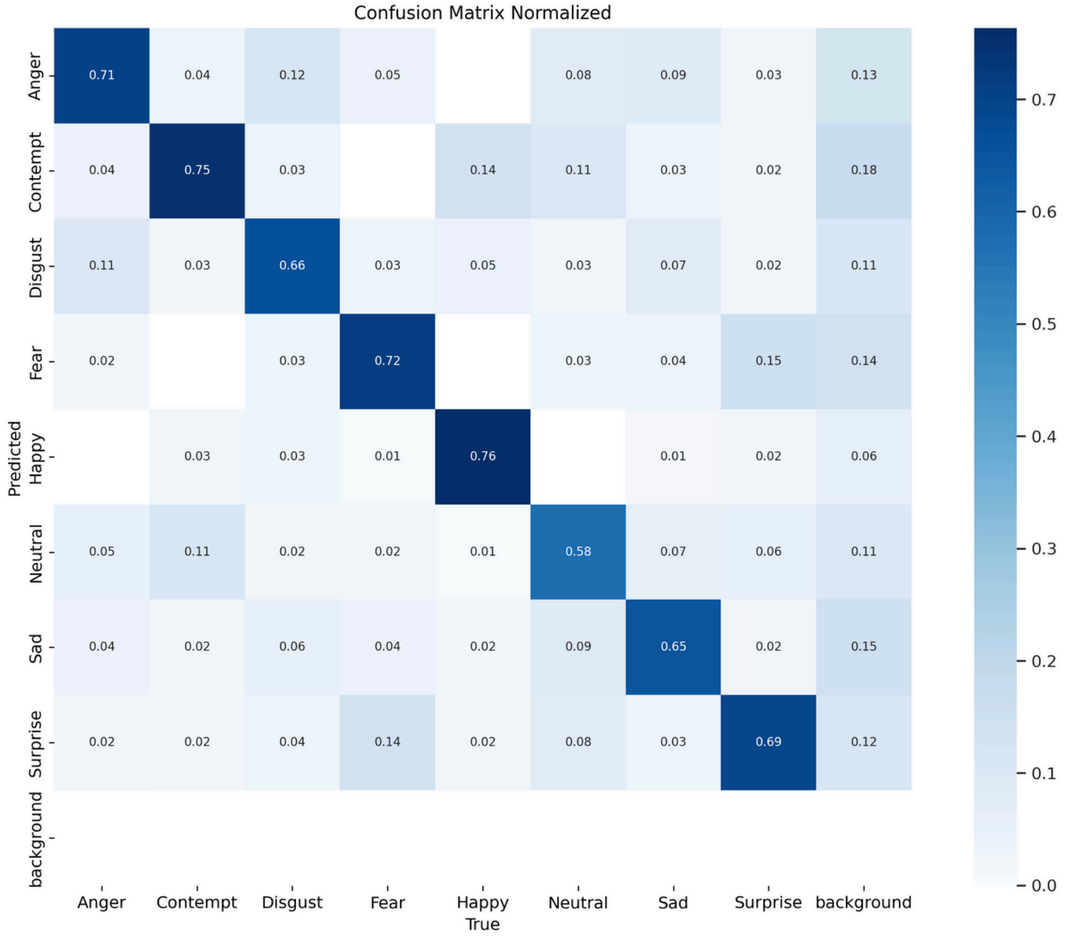
Model Evaluation

YOLO automatically logs metrics like loss and mAP during training.



Model Evaluation

We get the mAP : 0.78



Testing

ffhq_4685.png

image0021959.jpg

ffhq_4031.png

image0030789.jpg

A local demonstration application.

THANK YOU