



*Faculty of Computers and Artificial Intelligence,
Beni-Suef University, Egypt*



BLIND GUIDE

A senior project submitted in partial fulfillment of the requirements for the degree of Bachelor of
Computers and Artificial Intelligence.

Computer Science Department && Medical Department

Supervisor:

Dr. Hossam Moftah

Presented by:

1. Omar Adly Abd El Halim Nagdy
2. .
3. .
4. .
5. .

Acknowledgement

We would like to convey Our heartfelt gratitude **Dr. Hossam Moftah** for her tremendous support and assistance in the completion of our project. I would also like to thank **Eng. Muhamad Gamal El-Dien**, for his time and efforts he provided throughout the year. And also, for providing us with this wonderful opportunity to work on a project. The completion of the project would not have been possible without their help and insights.

Purpose of documentation

This document aims to describe the key elements of the project starting with the project idea, the software development life cycle of the project, discussing the design of the user interface, the algorithms that used of the project, and outline the technical aspects of the project in detail what parts that will be implemented and which parts to be considered as future work. With all of this information, this document becomes an absolute reference for the designers and developers during the implementation phase of this project.

Abstract

Sight is one of the five senses that help us know what is happening in the world around us. And we see things with our eyes, for the eyes are the organs that take (light and images) and turn them into (electrical impulses) so that the brain can understand them. But there are people who have lost their sight or were born without it. they find it difficult in their lives. Therefore, our project aims to identify objects and describe them, then convert that description into speech for the blind to hear, Through a mobile application. The idea of the application is to take pictures or video in real time and send it to the deep learning model, which in turn identifies the objects and send the output to a function that describes the location of the objects in the image, then translates the description into the desired language, then converts the description into speech using APIs.

Table Of Content

Chapter 1 (Introduction)	!!
Chapter 2 (Related work)	!!
2.1 Introduction	!!
2.2 A Brief Review of Object Detection Research	!!
2.3 Object Detection Approaches	!!
2.4 Current Research Problems	!!
2.5 Open Problems and Future Directions	!!
2.6 Conclusion	!!
Chapter 3 (Project software tools and technology)	!!
3.1. Machine learning	!!
3.2. Deep learning	!!
3.3. Computer vision	!!
3.4. Object Detection	!!
3.5. Existing method	!!
3.6. Yolo	!!
3.7. Python	!!
3.8. Tensorflow	!!
3.9. Pytorch	!!
3.10. Google Colab	!!
Chapter 4 ()	!!
Chapter 5 ()	!!
Chapter 6 ()	!!
Reference	!!

Chapter 2

Related Work

Object detection is a key ability required by most computer and robot vision systems. The latest research on this area has been making great progress in many directions. In the current manuscript, we give an overview of past research on object detection, outline the current main research directions, and discuss open problems and possible future directions.

2.1 introduction

During the last years, there has been a rapid and successful expansion on computer vision research. Parts of this success have come from adopting and adapting machine learning methods, while others from the development of new representations and models for specific computer vision problems or from the development of efficient solutions. One area that has attained great progress is object detection. The present works gives a perspective on object detection research.

Given a set of object classes, object detection consists in determining the location and scale of all object instances, if any, that are present in an image. Thus, the objective of an object detector is to find all object instances of one or more given object classes regardless of scale, location, pose, view with respect to the camera, partial occlusions, and illumination conditions.

In many computer vision systems, object detection is the first task being performed as it allows to obtain further information regarding the detected object and about the scene. Once an object instance has been detected (e.g., a face), it is possible to obtain further information, including: (i) to recognize the specific instance (e.g., to identify the subject's face), (ii) to track the object over an image sequence (e.g., to track the face in a video), and (iii) to extract further information about the object (e.g., to determine the subject's gender), while it is also possible to (a) infer the presence or location of other objects in the scene (e.g., a hand may be near a face and at a similar scale) and (b) to better estimate further information about the scene (e.g., the type of scene, indoor versus outdoor, etc.), among other contextual information.

Object detection has been used in many applications, with the most popular ones being: (i) human-computer interaction (HCI), (ii) robotics (e.g., service robots), (iii) consumer electronics (e.g., smartphones), (iv) security (e.g., recognition, tracking), (v) retrieval (e.g., search engines, photo management), and (vi) transportation (e.g., autonomous and assisted driving). Each of these applications has different requirements, including: processing time (off-line, on-line, or real-time), robustness to occlusions, invariance to rotations (e.g., in-plane rotations), and detection under pose changes. While many applications consider the detection of a single object class (e.g., faces) and from a single view (e.g., frontal faces), others require the detection of multiple object classes (humans, vehicles, etc.), or of a single class from multiple views (e.g., side and frontal view of vehicles). In general, most systems can detect only a single object class from a restricted set of views and poses.

Several surveys on detection and recognition have been published during the last years [see Hjelmås and Low (2001), Yang et al. (2002), Sun et al. (2006), Li and Allinson (2008), Enzweiler and Gavrila (2009), Dollar et al. (2012), Andreopoulos and Tsotsos (2013), Li et al. (2015), and Zafeiriou et al. (2015)], and

there are four main problems related to object detection. The first one is object localization, which consists of determining the location and scale of a single object instance known to be present in the image; the second one is object presence classification, which corresponds to determining whether at least one object of a given class is present in an image (without giving any information about the location, scale, or the number of objects), while the third problem is object recognition, which consist in determining if a specific object instance is present in the image. The fourth related problem is view and pose estimation, which consist of determining the view of the object and the pose of the object.

The problem of object presence classification can be solved using object detection techniques, but in general, other methods are used, as determining the location and scale of the objects is not required, and determining only the presence can be done more efficiently. In some cases, object recognition can be solved using methods that do not require detecting the object in advance [e.g., using methods based on Local Interest Points such as Tuytelaars and Mikolajczyk (2008) and Ramanan and Niranjan (2012)]. Nevertheless, solving the object detection problem would solve (or help simplifying) these related problems. An additional, recently addressed problem corresponds to determining the “objectness” of an image patch, i.e., measuring the likeliness for an image window to contain an object of any class [e.g., Alexe et al. (2010), Endres and Hoiem (2010), and Huval et al. (2013)]. In the following, we give a summary of past research on object detection, present an overview of current research directions, and discuss open problems and possible future directions, all this with a focus on the classifiers and architectures of the detector, rather than on the used features.

2.2 A Brief Review of Object Detection Research

Early works on object detection were based on template matching techniques and simple part-based models [e.g., Fischler and Elschlager (1973)]. Later, methods based on statistical classifiers (e.g., Neural Networks, SVM, Adaboost, Bayes, etc.) were introduced [e.g., Osuna et al. (1997), Rowley et al. (1998), Sung and Poggio (1998), Schneiderman and Kanade (2000), Yang et al. (2000a,b), Fleuret and Geman (2001), Romdhani et al. (2001), and Viola and Jones (2001)]. This initial successful family of object detectors, all of them based on statistical classifiers, set the ground for most of the following research in terms of training and evaluation procedures and classification techniques. Because face detection is a critical ability for any system that interacts with humans, it is the most common application of object detection. However, many additional detection problems have been studied [e.g., Papageorgiou and Poggio (2000), Agarwal et al. (2004), Alexe et al. (2010), Everingham et al. (2010), and Andreopoulos and Tsotsos (2013)]. Most cases correspond to objects that people often interact with, such as other humans [e.g., pedestrians (Papageorgiou and Poggio, 2000; Viola and Jones, 2002; Dalal and Triggs, 2005; Bourdev et al., 2010; Paisitkriangkrai et al., 2015)] and body parts [(Kölsch and Turk, 2004; Ong and Bowden, 2004; Wu and Nevatia, 2005; Verschae et al., 2008; Bourdev and Malik, 2009) e.g., faces, hands, and eyes], as well as vehicles [(Papageorgiou and Poggio, 2000; Felzenszwalb et al., 2010b), e.g., cars and airplanes], and animals [e.g., Fleuret and Geman (2008)]. Most object detection systems consider the same basic scheme, commonly known as sliding window: in order to detect the objects appearing in the image at different scales and locations, an exhaustive search is applied. This search makes use of a classifier, the core part of the detector, which indicates if a given image patch, corresponds to the object or not. Given that the classifier basically works at a given scale and patch size, several versions of the input image are generated at different scales, and the classifier is used to classify

all possible patches of the given size, for each of the downscaled versions of the image. Basically, three alternatives exist to the sliding window scheme. The first one is based on the use of bag-of-words (Weinland et al., 2011; Tsai, 2012), method sometimes used for verifying the presence of the object, and that in some cases can be efficiently applied by iteratively refining the image region that contains the object [e.g., Lampert et al. (2009)]. The second one samples patches and iteratively searches for regions of the image where it is likely that the object is present [e.g., Prati et al. (2012)]. These two schemes reduce the number of image patches where to perform the classification, seeking to avoid an exhaustive search over all image patches. The third scheme finds key-points and then matches them to perform the detection [e.g., Azzopardi and Petkov (2013)]. These schemes cannot always guarantee that all object's instances will be detected.

2.3 Object Detection Approaches

Object detection methods can be grouped in five categories, each with merits and demerits: while some are more robust, others can be used in real-time systems, and others can be handle more classes, etc. Table 1 gives a qualitative comparison.

Method	Coarse-to-fine and boosted classifiers	Dictionary based	Deformable part-based models	Deep learning	Trainable image processing architectures
Accuracy	++	+=	++	++	+=
Generality	==	++	+=	++	+=
Speed	++	+=	==	+=	+=
Advantages	Real-time, it can work at small resolutions	Representation can be shared across classes	It can handle deformations and occlusions	Representation can be transferred to other classes	General-purpose architecture that can be used is several modules of a system
Drawbacks/requirements	Features are predefined	It may not detect all object instances	It can not detect small objects	Large training sets specialized hardware (GPU) for efficiency	The obtained system may be Too specialized for a particular setting
Typical applications	Robotics, security	Retrieval, search	Transportation pedestrian detection	Retrieval, search	HCI, health, robotics

Accuracy: ++, High; +=, Good; ==, Low.

Speed: ++, real-time (15 fps or more); +=, online (10–5 fps); ==, offline (5 fps or more).

Generality: ++ (+=), applicable to many (some) object classes; ==, depend on features designed for specific classes.

2.3.1 Coarse-to-Fine and Boosted Classifiers

The most popular work in this category is the boosted cascade classifier of Viola and Jones (2004). It works by efficiently rejecting, in a cascade of test/filters, image patches that do not correspond to the object. Cascade methods are commonly used with boosted classifiers due to two main reasons: (i) boosting generates an additive classifier, thus it is easy to control the complexity of each stage of the cascade and (ii) during training, boosting can be also used for feature selection, allowing the use of large (parametrized) families of features. A coarse-to-fine cascade classifier is usually the first kind of classifier to consider when efficiency is a key requirement. Recent methods based on boosted classifiers include Li and Zhang (2004), Gangaputra and Geman (2006), Huang et al. (2007), Wu and Nevatia (2007), Verschae et al. (2008), and Verschae and Ruiz-del-Solar (2012).

2.3.2 Dictionary Based

The best example in this category is the Bag of Word method [e.g., Serre et al. (2005) and Mutch and Lowe (2008)]. This approach is basically designed to detect a single object per image, but after removing a detected object, the remaining objects can be detected [e.g., Lampert et al. (2009)]. Two problems with this approach are that it cannot robustly handle well the case of two instances of the object appearing near each other, and that the localization of the object may not be accurate.

2.3.3 Deformable Part-Based Model

This approach considers object and part models and their relative positions. In general, it is more robust than other approaches, but it is rather time consuming and cannot detect objects appearing at small scales. It can be traced back to the deformable models (Fischler and Elschlager, 1973), but successful methods are recent (Felzenszwalb et al., 2010b). Relevant works include Felzenszwalb et al. (2010a) and Yan et al. (2014), where efficient evaluation of deformable part-based model is implemented using a coarse-to-fine cascade model for faster evaluation, Divvala et al. (2012), where the relevance of the part-models is analyzed, among others [e.g., Azizpour and Laptev (2012), Zhu and Ramanan (2012), and Girshick et al. (2014)].

2.3.4 Deep Learning

One of the first successful methods in this family is based on convolutional neural networks (Delakis and Garcia, 2004). The key difference between this and the above approaches is that in this approach the feature representation is learned instead of being designed by the user, but with the drawback that a large number of training samples is required for training the classifier. Recent methods include Dean et al. (2013), Huval et al. (2013), Ouyang and Wang (2013), Sermanet et al. (2013), Szegedy et al. (2013), Zeng et al. (2013), Erhan et al. (2014), Zhou et al. (2014), and Ouyang et al. (2015).

2.3.5 Trainable Image Processing Architectures

In such architectures, the parameters of predefined operators and the combination of the operators are learned, sometimes considering an abstract notion of fitness. These are general-purpose architectures, and thus they can be used to build several modules of a larger system (e.g., object recognition, key point detectors and object detection modules of a robot vision system). Examples include trainable COSFIRE filters (Azzopardi and Petkov, 2013, 2014), and Cartesian Genetic Programming (CGP) (Harding et al., 2013; Leitner et al., 2013).

2.4 Current Research Problems

Table 2 presents a summary of solved, current, and open problems. In the present section we discuss current research directions.

Solved problems	Single-class	Single-view	Small deformations	Multi-scale
Current directions	Multi-class (scalability and efficiency)	Multi-view/pose Multi-resolution	Occlusions, deformable Interlaced object and background	Contextual information Temporal features
Open	Incremental learning	Object-part relation	Pixel-level detection Background objects	Multi-modal

2.4.1 Multi-Class

Many applications require detecting more than one object class. If a large number of classes is being detected, the processing speed becomes an important issue, as well as the kind of classes that the system can handle without accuracy loss. Works that have addressed the multi-class detection problem include Torralba et al. (2007), Razavi et al. (2011), Benbouzid et al. (2012), Song et al. (2012), Verschae and Ruiz-del-Solar (2012), and Erhan et al. (2014). Efficiency has been addressed, e.g., by using the same representation for several object classes, as well as by developing multi-class classifiers designed specifically to detect multiple classes. Dean et al. (2013) presents one of the few existing works for very large-scale multi-class object detection, where 100,000 object classes were considered.

2.4.2 Multi-View, Multi-Pose, Multi-Resolution

Most methods used in practice have been designed to detect a single object class under a single view, thus these methods cannot handle multiple views, or large pose variations; with the exception of deformable part-based models which can deal with some pose variations. Some works have tried to detect objects by learning subclasses (Wu and Nevatia, 2007) or by considering views/poses as different classes (Verschae and Ruiz-del-Solar, 2012); in both cases improving the efficiency and robustness. Also, multi-pose models [e.g., Erol et al. (2007)] and multi-resolution models [e.g., Park et al. (2010)] have been developed.

2.4.3 Efficiency and Computational Power

Efficiency is an issue to be taken into account in any object detection system. As mentioned, a coarse-to-fine classifier is usually the first kind of classifier to consider when efficiency is a key requirement [e.g., Viola et al. (2005)], while reducing the number of image patches where to perform the classification [e.g., Lampert et al. (2009)] and efficiently detecting multiple classes [e.g., Verschae and Ruiz-del-Solar (2012)] have also been used. Efficiency does not imply real-time performance, and works such as Felzenszwalb et al. (2010b) are robust and efficient, but not fast enough for real-time problems. However, using specialized hardware (e.g., GPU) some methods can run in real-time (e.g., deep learning).

2.4.4 Occlusions, Deformable Objects, and Interlaced Object and Background

Dealing with partial occlusions is also an important problem, and no compelling solution exists, although relevant research has been done [e.g., Wu and Nevatia (2005)]. Similarly, detecting objects that are not “closed,” i.e., where objects and background pixels are interlaced with background is still a difficult

problem. Two examples are hand detection [e.g., Kölsch and Turk (2004)] and pedestrian detection [see Dollar et al. (2012)]. Deformable part-based model [e.g., Felzenszwalb et al. (2010b)] have been to some extend successful under this kind of problem, but further improvement is still required.

2.4.5 Contextual Information and Temporal Features

Integrating contextual information (e.g., about the type of scene, or the presence of other objects) can increase speed and robustness, but “when and how” to do this (before, during or after the detection), it is still an open problem. Some proposed solutions include the use of (i) spatio-temporal context [e.g., Palma-Amestoy et al. (2010)], (ii) spatial structure among visual words [e.g., Wu et al. (2009)], and (iii) semantic information aiming to map semantically related features to visual words [e.g., Wu et al. (2010)], among many others [e.g., Torralba and Sinha (2001), Divvala et al. (2009), Sun et al. (2012), Mottaghi et al. (2014), and Cadena et al. (2015)]. While most methods consider the detection of objects in a single frame, temporal features can be beneficial [e.g., Viola et al. (2005) and Dalal et al. (2006)].

2.5 Open Problems and Future Directions

In the following, we outline problems that we believe have not been addressed, or addressed only partially, and may be interesting relevant research directions.

2.5.1 Open-World Learning and Active Vision

An important problem is to incrementally learn, to detect new classes, or to incrementally learn to distinguish among subclasses after the “main” class has been learned. If this can be done in an unsupervised way, we will be able to build new classifiers based on existing ones, without much additional effort, greatly reducing the effort required to learn new object classes. Note that humans are continuously inventing new objects, fashion changes, etc., and therefore detection systems will need to be continuously updated, adding new classes, or updating existing ones. Some recent works have addressed these issues, mostly based on deep learning and transfer learning methods [e.g., Bengio (2012), Mesnil et al. (2012), and Kotzias et al. (2014)]. This open-world learning is of particular importance in robot applications, case where active vision mechanisms can aid in the detection and learning [e.g., Paletta and Pinz (2000) and Correa et al. (2012)].

2.5.2 Object-Part Relation

During the detection process, should we detect the object first or the parts first? This is a basic dilemma, and no clear solution exists. Probably, the search for the object and for the parts must be done concurrently where both processes give feedback to each other. How to do this is still an open problem and is likely related to how to use of context information. Moreover, in cases the object part can be also decomposed in subparts, an interaction among several hierarchies emerges, and in general it is not clear what should be done first.

2.5.3 Multi-Modal Detection

The use of new sensing modalities, in particular depth and thermal cameras, has seen some development in the last years [e.g., Fehr and Burkhardt (2008) and Correa et al. (2012)]. However, the methods used for processing visual images are also used for thermal images, and to a lesser degree for

depth images. While using thermal images makes easier to discriminate the foreground from the background, it can only be applied to objects that irradiate infrared light (e.g., mammals, heating, etc.). Using depth images is easy to segment the objects, but general methods for detecting specific classes has not been proposed, and probably higher resolution depth images are required. It seems that depth and thermal cameras alone are not enough for object detection, at least with their current resolution, but further advances can be expected as the sensing technology improves.

2.5.4 Pixel-Level Detection (Segmentation) and Background Objects

In many applications, we may be interested in detecting objects that are usually considered as background. The detection of such “background objects,” such as rivers, walls, mountains, has not been addressed by most of the here mentioned approaches. In general, this kind of problem has been addressed by first segmenting the image and later labeling each segment of the image [e.g., Peng et al. (2013)]. Of course, for successfully detecting all objects in a scene, and to completely understand the scene, we will need to have a pixel level detection of the objects, and further more, a 3D model of such scene. Therefore, at some point object detection and image segmentation methods may need to be integrated. We are still far from attaining such automatic understanding of the world, and to achieve this, active vision mechanisms might be required [e.g., Aloimonos et al. (1988) and Cadena et al. (2015)].

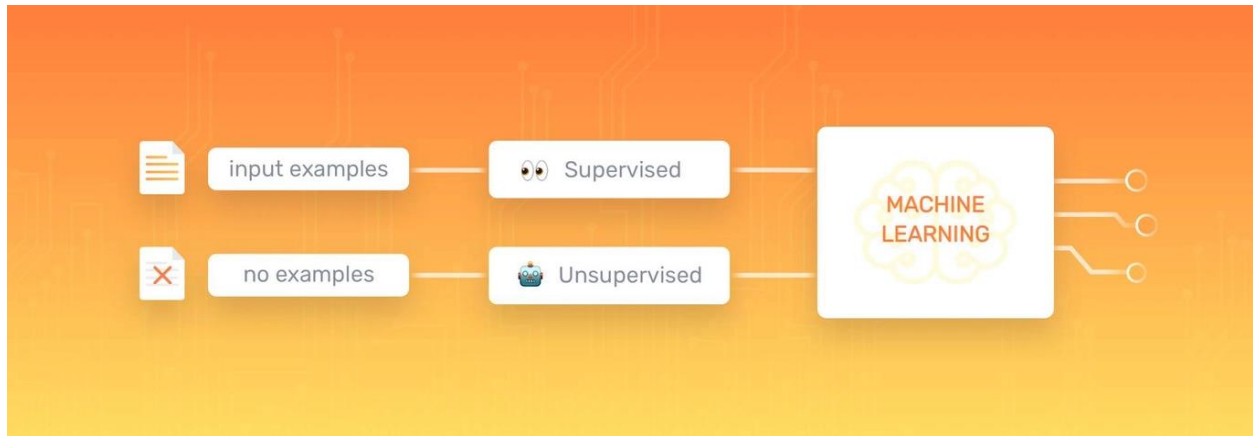
2.6 Conclusion

Object detection is a key ability for most computer and robot vision system. Although great progress has been observed in the last years, and some existing techniques are now part of many consumer electronics (e.g., face detection for auto-focus in smartphones) or have been integrated in assistant driving technologies, we are still far from achieving human-level performance, in particular in terms of open-world learning. It should be noted that object detection has not been used much in many areas where it could be of great help. As mobile robots, and in general autonomous machines, are starting to be more widely deployed (e.g., quad-copters, drones and soon service robots), the need of object detection systems is gaining more importance. Finally, we need to consider that we will need object detection systems for nano-robots or for robots that will explore areas that have not been seen by humans, such as depth parts of the sea or other planets, and the detection systems will have to learn to new object classes as they are encountered. In such cases, a real-time open-world learning ability will be critical.

Chapter 3

3.1 Machine learning

Machine learning (ML) is a branch of artificial intelligence (AI) that enables computers to “self-learn” from training data and improve over time, without being explicitly programmed. Machine learning algorithms are able to detect patterns in data and learn from them, in order to make their own predictions. In short, machine learning algorithms and models learn through experience.



Machine learning can be put to work on massive amounts of data and can perform much more accurately than humans. It can help you save time and money on tasks and analyses, like solving customer pain points to improve customer satisfaction, support ticket automation, and data mining from internal sources and all over the internet. To understand how machine learning works, you’ll need to explore different machine learning methods and algorithms, which are basically sets of rules that machines use to make decisions. Below, you’ll find the four most common and most used types of machine learning:

1-supervised learning models make predictions based on labeled training data. Each training sample includes an input and a desired output.

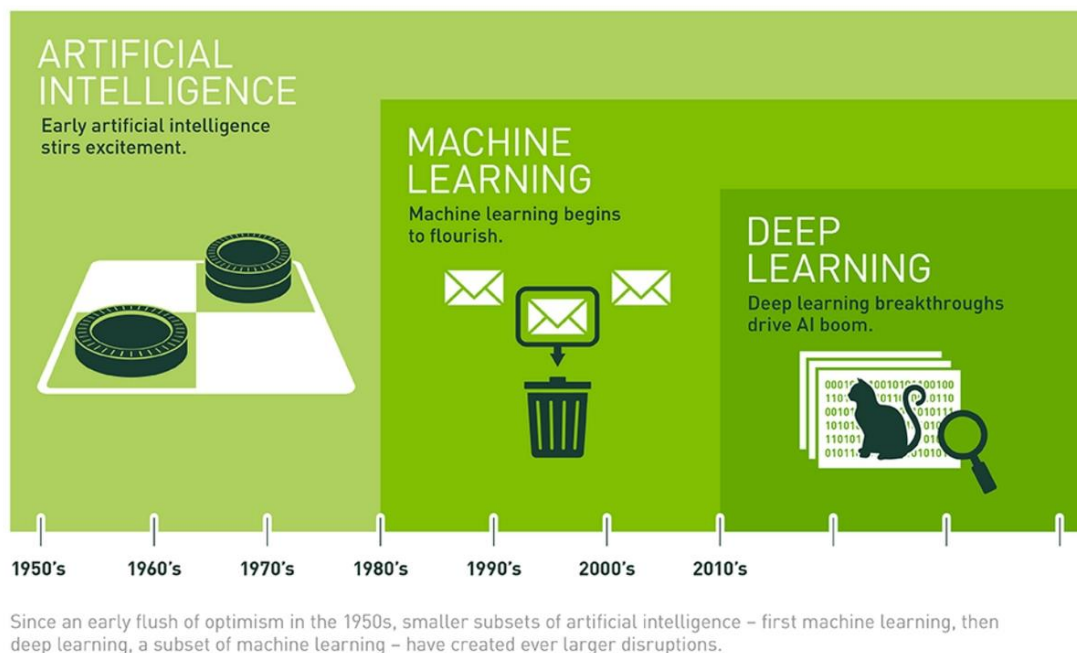
2-Unsupervised learning algorithms uncover insights and relationships in unlabeled data. In this case, models are fed input data but the desired outcomes are unknown, so they have to make inferences based on circumstantial evidence, without any guidance or training.

3-In semi-supervised learning, training data is split into two. A small amount of labeled data and a larger set of unlabeled data.

4-Reinforcement learning (RL) is concerned with how a software agent (or computer program) ought to act in a situation to maximize the reward. In short, reinforced machine learning models attempt to determine the best possible path they should take in a given situation.

3.2 Deep learning

Deep learning models can be supervised, semi-supervised, or unsupervised (or a combination of any or all of the three). They're advanced machine learning algorithms used by tech giants, like Google, Microsoft, and Amazon to run entire systems and power things, like self-driving cars and smart assistants.



Evolution of Deep Learning

Deep learning is based on Artificial Neural Networks (ANN), a type of computer system that emulates the way the human brain works. Deep learning algorithms or neural networks are built with multiple layers of interconnected neurons, allowing multiple systems to work together simultaneously, and step-by-step. When a model receives input data – which could be image, text, video, or audio – and is asked to perform a task (for example, text classification with machine learning), the data passes through every layer, enabling the model to learn progressively. It's kind of like a human brain that evolves with age and experience! Deep learning is common in image recognition, speech recognition, and Natural Language Processing (NLP). Deep learning models usually perform better than other machine learning algorithms for complex problems and massive sets of data. However, they generally require millions upon millions of pieces of training data, so it takes quite a lot of time to train them.

3.3 Computer vision

Computer vision is a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs — and take actions or make recommendations based on that information. If AI enables computers to think, computer vision enables them to see, observe and understand.

Computer vision works much the same as human vision, except humans have a head start. Human sight has the advantage of lifetimes of context to train how to tell objects apart, how far away they are, whether they are moving and whether there is something wrong in an image.

Computer vision trains machines to perform these functions, but it has to do it in much less time with cameras, data and algorithms rather than retinas, optic nerves and a visual cortex. Because a system trained to inspect products or watch a production asset can analyze thousands of products or processes a minute, noticing imperceptible defects or issues, it can quickly surpass human capabilities.

Computer vision tasks can focus and validate projects and applications and make it easier to get started.

Here are a few examples of established computer vision tasks:

Image classification sees an image and can classify it (a dog, an apple, a person's face). More precisely, it is able to accurately predict that a given image belongs to a certain class. For example, a social media company might want to use it to automatically identify and segregate objectionable images uploaded by users.

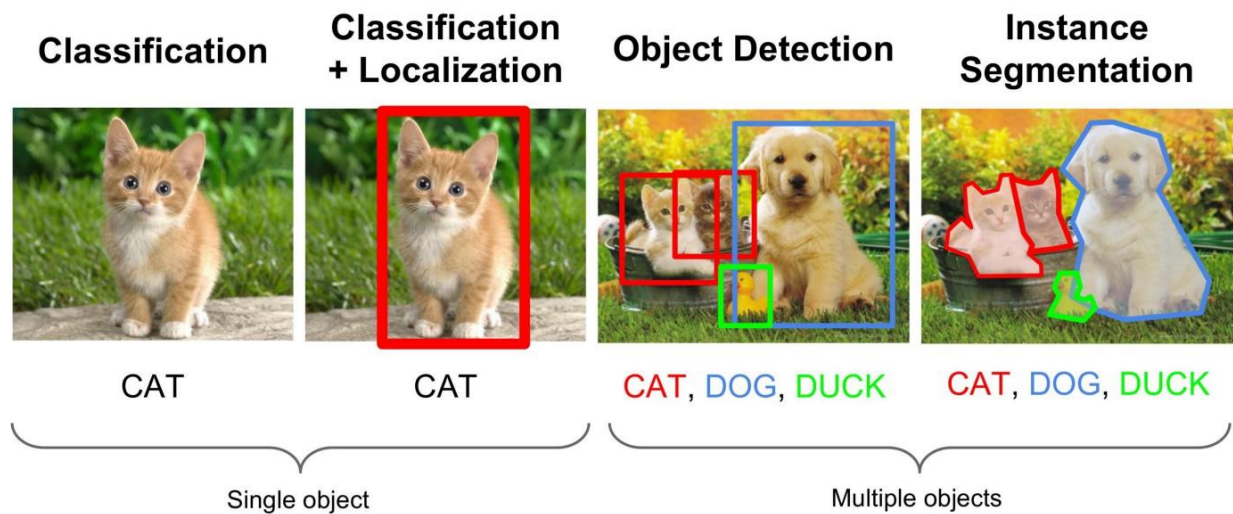
Object detection can use image classification to identify a certain class of image and then detect and tabulate their appearance in an image or video. Examples include detecting damages on an assembly line or identifying machinery that requires maintenance.

Object tracking follows or tracks an object once it is detected. This task is often executed with images captured in sequence or real-time video feeds. Autonomous vehicles, for example, need to not only classify and detect objects such as pedestrians, other cars and road infrastructure, they need to track them in motion to avoid collisions and obey traffic laws.

Content-based image retrieval uses computer vision to browse, search and retrieve images from large data stores, based on the content of the images rather than metadata tags associated with them. This task can incorporate automatic image annotation that replaces manual image tagging. These tasks can be used for digital asset management systems and can increase the accuracy of search and retrieval.

3.4 Object Detection

Object detection is an important task, yet challenging vision task. It is a critical part of many applications such as image search, image auto-annotation and scene understanding, object tracking. Moving object tracking of video image sequences was one of the most important subjects in computer vision. It had already been applied in many computer vision fields, such as smart video surveillance (Arun Hampapur 2005), artificial intelligence, military guidance, safety detection and robot navigation, medical and biological application. In recent years, a number of successful single-object tracking system appeared, but in the presence of several objects, object detection becomes difficult and when objects are fully or partially occluded, they are obtruded from the human vision which further increases the problem of detection. Decreasing illumination and acquisition angle. The proposed MLP based object tracking system is made robust by an optimum selection of unique features and also by implementing the Adaboost strong classification method.



3.5 Existing method

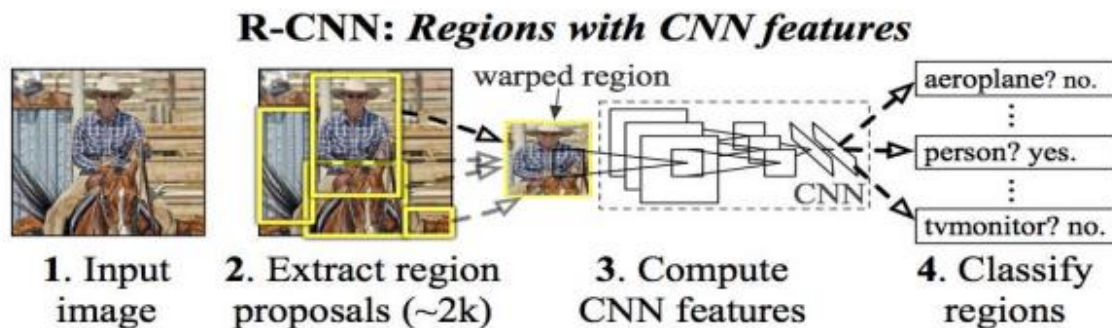
3.5.1 ResNet

To train the network model in a more effective manner, we herein adopt the same strategy as that used for DSSD (the performance of the residual network is better than that of the VGG network). The goal is to improve accuracy. However, the first implemented for the modification was the replacement of the VGG network which is used in the original SSD with ResNet. We will also add a series of convolution feature layers at the end of the underlying network. These feature layers will gradually be reduced in size that allowed prediction of the detection results on multiple scales. When the input size is given as 300 and 320, although the ResNet-101 layer is deeper than the VGG-16 layer, it is experimentally known that it replaces the SSD's underlying convolution network with a residual network, and it does not improve its accuracy but rather decreases it.

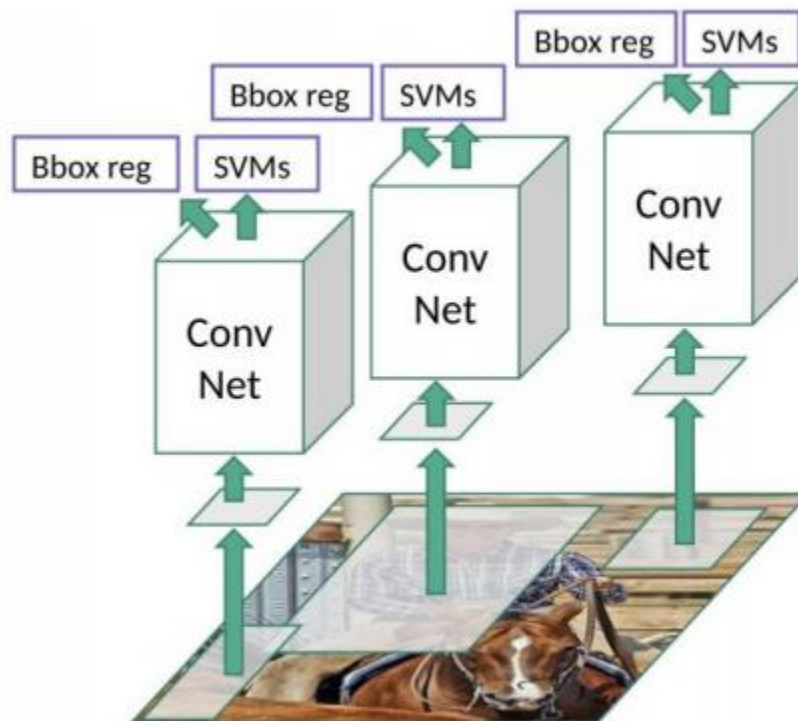
3.5.2 R-CNN

To circumvent the problem of selecting a huge number of regions, Ross Girshick et al. proposed a method where we use the selective search to extract just 2000 regions from the image and he called them region proposals. Therefore, instead of trying to classify the huge number of regions, you can just work with 2000 regions. These 2000 region proposals are generated by using the selective search algorithm which is written below. Selective Search:

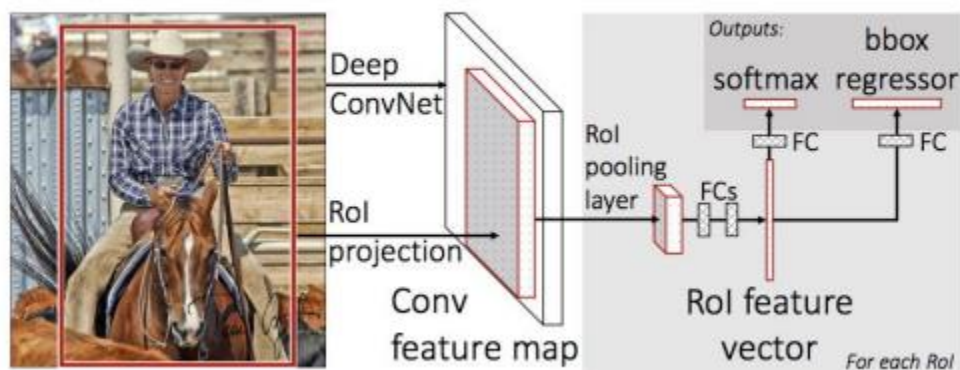
1. Generate the initial sub-segmentation, we generate many candidate regions
2. Use the greedy algorithm to recursively combine similar regions into larger ones
3. Use generated regions to produce the final candidate region proposals



These 2000 candidate regions which are proposals are warped into a square and fed into a convolutional neural network that produces a 4096-dimensional feature vector as output. The CNN plays a role of feature extractor and the output dense layer consists of the features extracted from the image and the extracted features are fed into an SVM for the classify the presence of the object within that candidate region proposal. In addition to predicting the presence of an object within the region proposals, the algorithm also predicts four values which are offset values for increasing the precision of the bounding box. For example, given the region proposal, the algorithm might have predicted the presence of a person but the face of that person within that region proposal could have been cut in half. Therefore, the offset values which is given help in adjusting the bounding box of the region proposal.

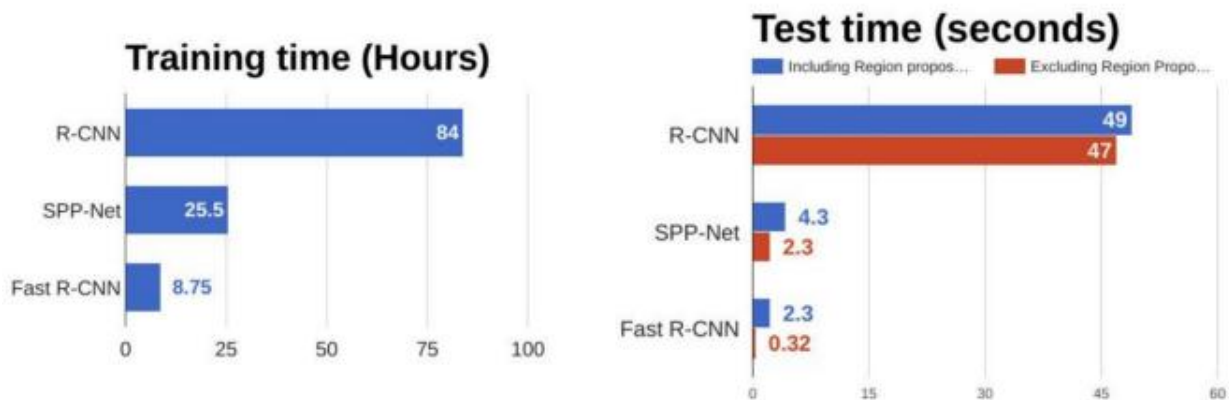


3.5.3 Fast R-CNN

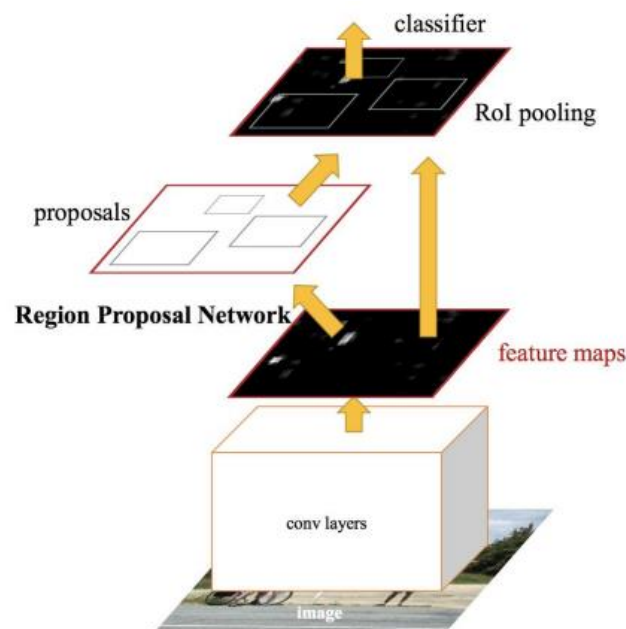


The same author of the previous paper(R-CNN) solved some of the drawbacks of R-CNN to build a faster object detection algorithm and it was called Fast R-CNN. The approach is similar to the R-CNN algorithm. But, instead of feeding the region proposals to the CNN, we feed the input image to the CNN to generate a convolutional feature map. From the convolutional feature map, we can identify the region of the proposals and warp them into the squares and by using an RoI pooling layer were shape them

into the fixed size so that it can be fed into a fully connected layer. From the RoI feature vector, we can use a softmax layer to predict the class of the proposed region and also the offset values for the bounding box. The reason “Fast R-CNN” is faster than R-CNN is because you don’t have to feed 2000 region proposals to the convolutional neural network every time. Instead, the convolution operation is always done only once per image and a feature map is generated from it.

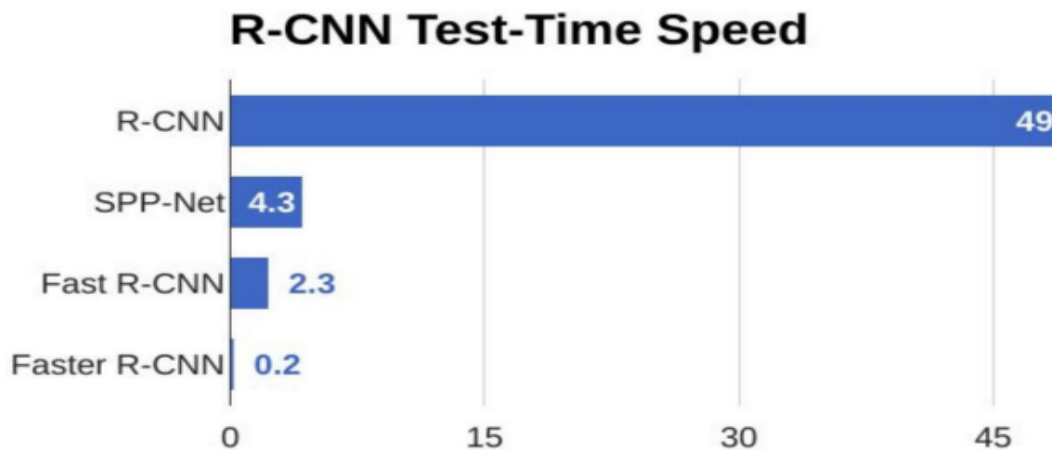


3.5.4 Faster R-CNN



Both of the above algorithms (R-CNN & Fast R-CNN) use selective search to find out the region proposals. Selective search is the slow and time-consuming process which affects the performance of the network. Similar to Fast R-CNN, the image is provided as an input to a convolutional network which provides a convolutional feature map. Instead of using the selective search algorithm for the feature

map to identify the region proposals, a separate network is used to predict the region proposals. The predicted region proposals are then reshaped using an RoI pooling layer which is used to classify the image within the proposed region and predict the offset values for the bounding boxes.

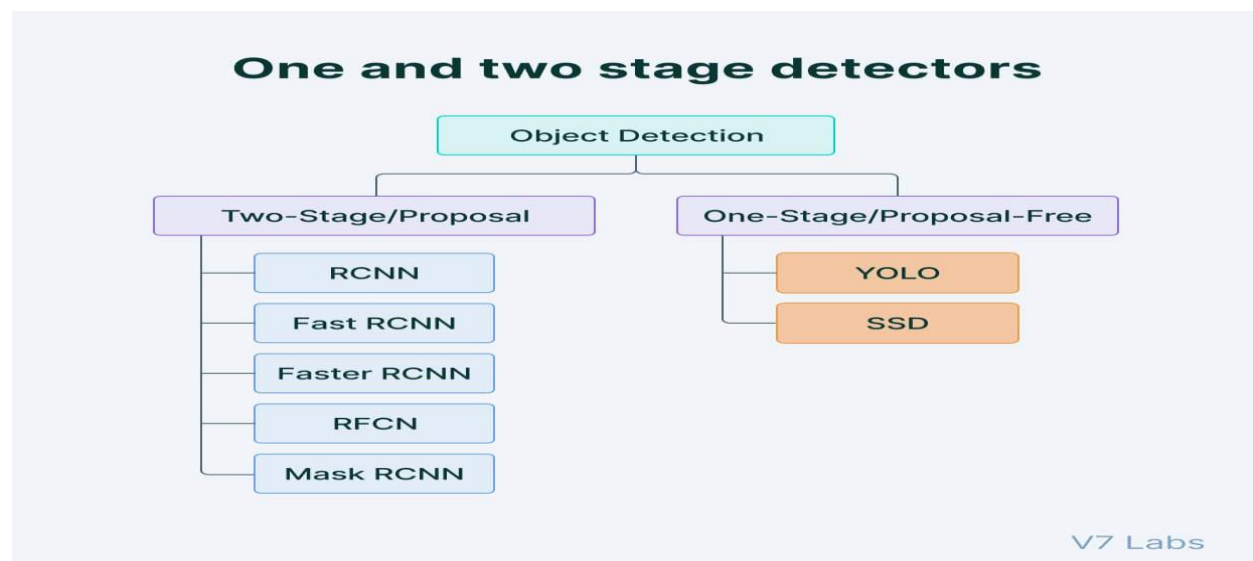


From the above graph, you can see that Faster R-CNN is much faster than its predecessors. Therefore, it can even be used for real-time object detection.

3.6 YOLO

Before we talk about Yolo, we must understand some concepts: -

Object detection algorithms are broadly classified into two categories based on how many times the same input image is passed through a network.



3.6.1 Single-shot object detection

Single-shot object detection uses a single pass of the input image to make predictions about the presence and location of objects in the image. It processes an entire image in a single pass, making them computationally efficient. However, single-shot object detection is generally less accurate than other methods, and it's less effective in detecting small objects. Such algorithms can be used to detect objects in real time in resource-constrained environments. YOLO is a single-shot detector that uses a fully convolutional neural network (CNN) to process an image. We will dive deeper into the YOLO model in the next section.

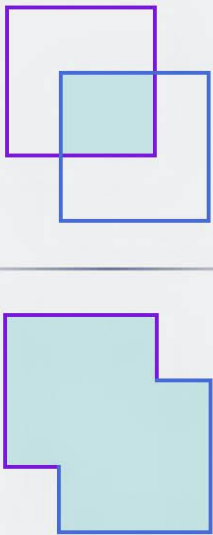
3.6.2 Two-shot object detection

Two-shot object detection uses two passes of the input image to make predictions about the presence and location of objects. The first pass is used to generate a set of proposals or potential object locations, and the second pass is used to refine these proposals and make final predictions. This approach is more accurate than single-shot object detection but is also more computationally expensive. Overall, the choice between single-shot and two-shot object detection depends on the specific requirements and constraints of the application. Generally, single-shot object detection is better suited for real-time applications, while two-shot object detection is better for applications where accuracy is more important.

3.6.3 Object detection models performance evaluation metrics To determine and compare the predictive performance of different object detection models, we need standard quantitative metrics. The two most common evaluation metrics are Intersection over Union (IoU) and the Average Precision (AP) metrics.

3.6.3.1 Intersection over Union (IoU)

Intersection over Union is a popular metric to measure localization accuracy and calculate localization errors in object detection models. To calculate the IoU between the predicted and the ground truth bounding boxes, we first take the intersecting area between the two corresponding bounding boxes for the same object. Following this, we calculate the total area covered by the two bounding boxes— also known as the “Union” and the area of overlap between them called the “Intersection”. The intersection divided by the Union gives us the ratio of the overlap to the total area, providing a good estimate of how close the prediction bounding box is to the original bounding box.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}} =$$


V7

3.6.3.2 Average Precision (AP)

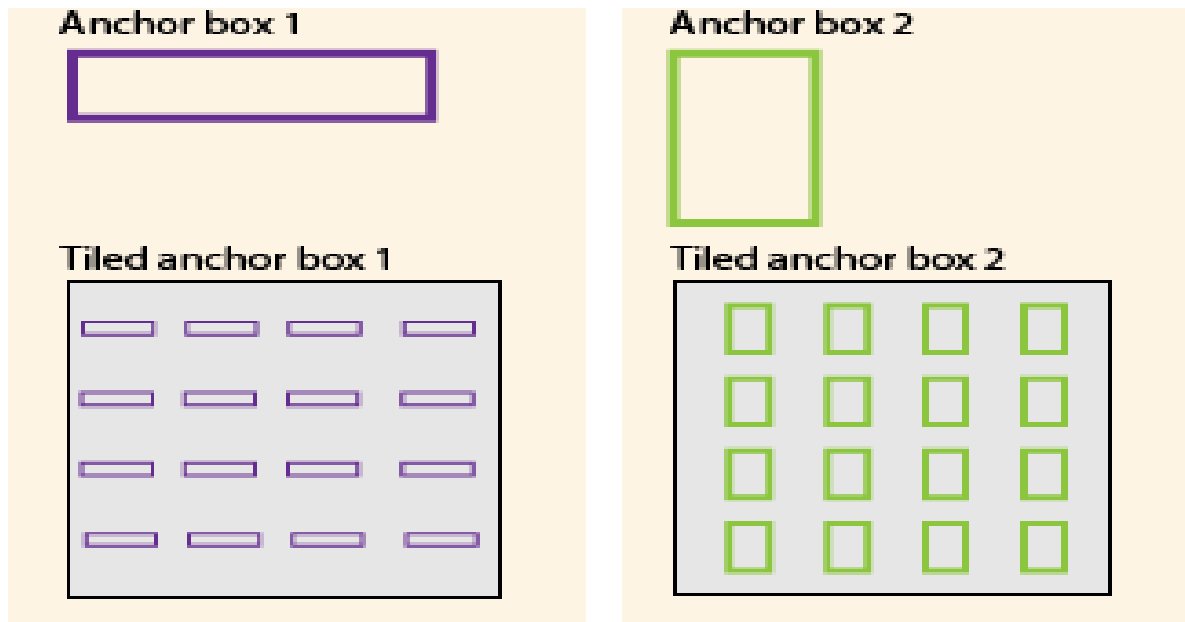
Average Precision (AP) is calculated as the area under a precision vs. recall curve for a set of predictions.

Recall is calculated as the ratio of the total predictions made by the model under a class with a total of existing labels for the class. Precision refers to the ratio of true positives with respect to the total predictions made by the model. Recall and precision offer a trade-off that is graphically represented into a curve by varying the classification threshold. The area under this precision vs. recall curve gives us the Average Precision per class for the model. The average of this value, taken over all classes, is called mean Average Precision (mAP).

3.6.4 Anchor boxes

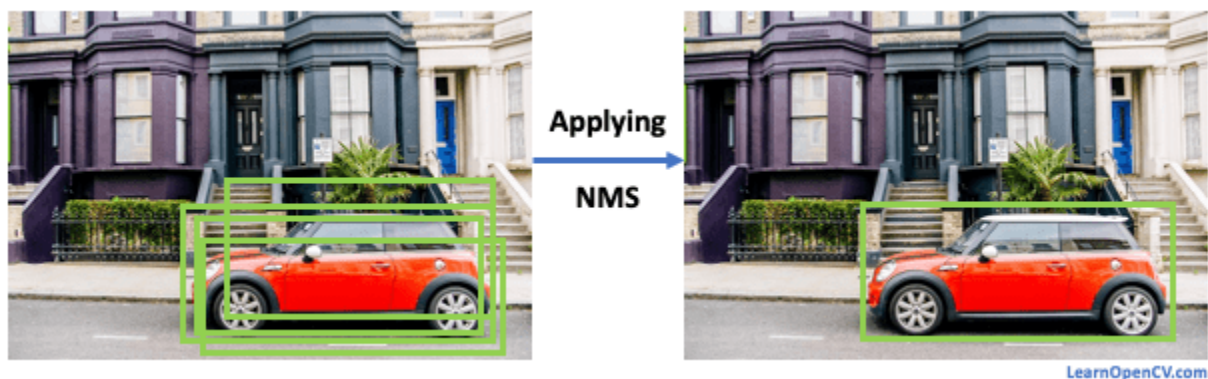
Anchor boxes are a set of predefined bounding boxes of a certain height and width. These boxes are defined to capture the scale and aspect ratio of specific object classes you want to detect and are typically chosen based on object sizes in your training datasets. During detection, the predefined anchor boxes are tiled across the image. The network predicts the probability and other attributes, such as background, intersection over union (IoU) and offsets for every tiled anchor box. The predictions are

used to refine each individual anchor box. You can define several anchor boxes, each for a different object size. Anchor boxes are fixed initial boundary box guesses.



3.6.5 Non Maximum Suppression

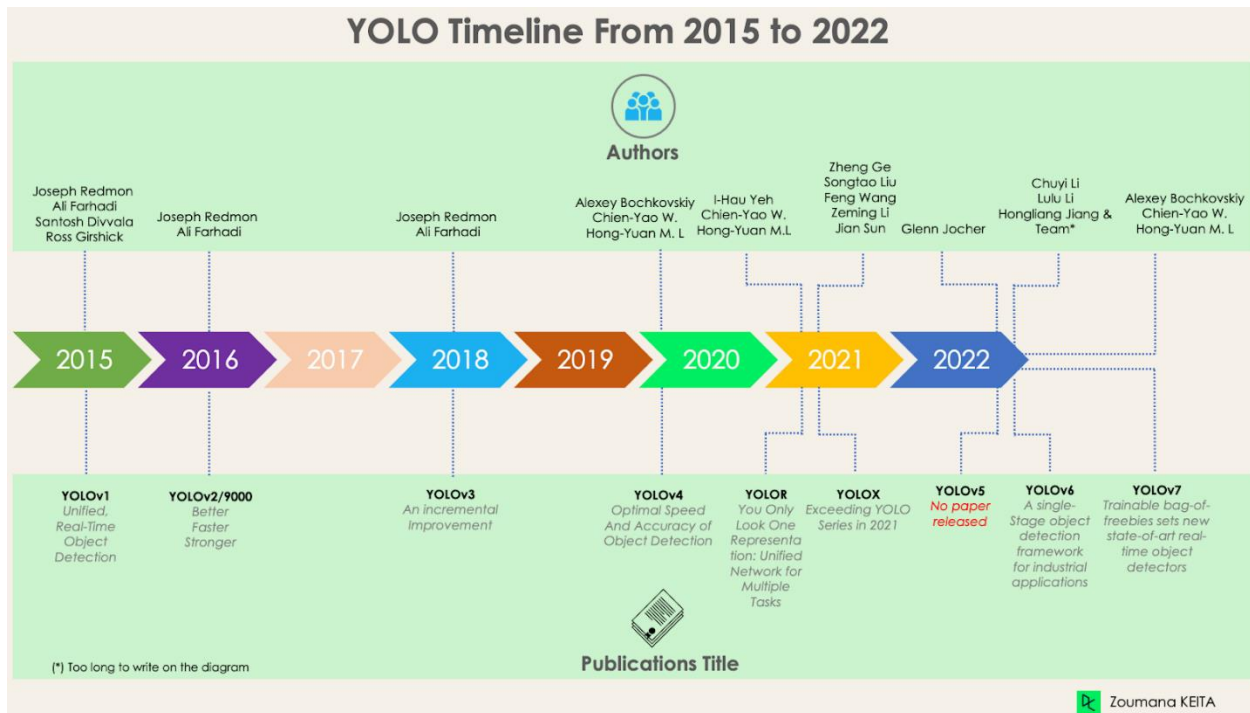
Non Maximum Suppression (NMS) is a technique used in numerous computer vision tasks. It is a class of algorithms to select one entity (e.g., bounding boxes) out of many overlapping entities. We can choose the selection criteria to arrive at the desired results. The criteria are most commonly some form of probability number and some form of overlap measure (e.g. Intersection over Union).



3.6.6 What is YOLO?

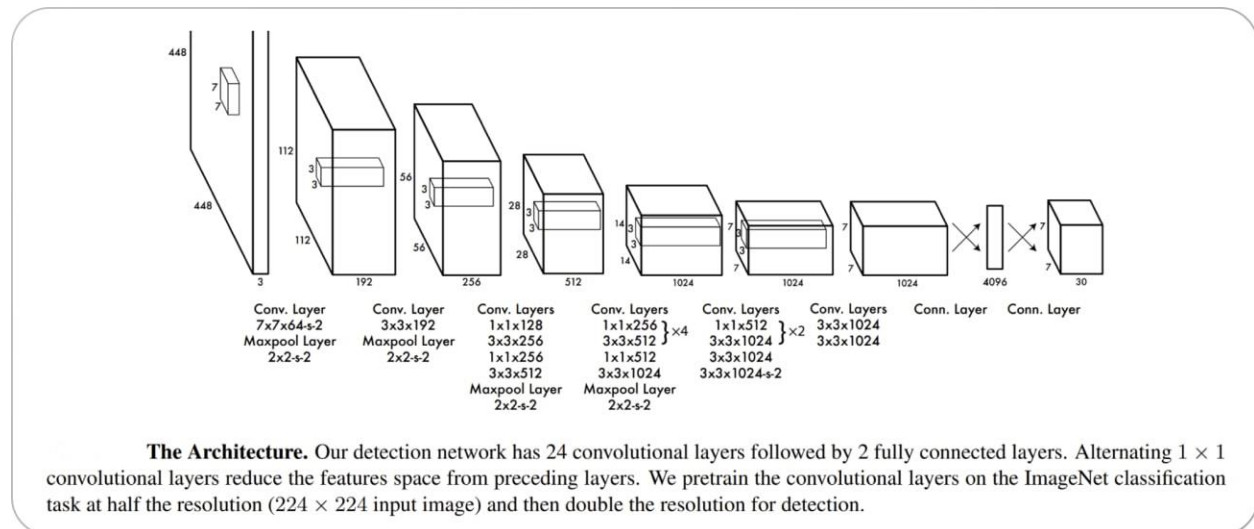
You Only Look Once (YOLO) proposes using an end-to-end neural network that makes predictions of bounding boxes and class probabilities all at once. It differs from the approach taken by previous object detection algorithms, which repurposed classifiers to perform detection.

Several new versions of the same model have been proposed since the initial release of YOLO in 2015, each building on and improving its predecessor. Here's a timeline showcasing YOLO's development in recent years.



3.6.7 How does YOLO work? YOLO Architecture

The YOLO algorithm takes an image as input and then uses a simple deep convolutional neural network to detect objects in the image. The architecture of the CNN model that forms the backbone of YOLO is shown below.



The first 20 convolution layers of the model are pre-trained using ImageNet by plugging in a temporary average pooling and fully connected layer. Then, this pre-trained model is converted to perform detection since previous research showcased that adding convolution and connected layers to a pre-trained network improves performance. YOLO's final fully connected layer predicts both class probabilities and bounding box coordinates.

YOLO divides an input image into an $S \times S$ grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and how accurate it thinks the predicted box is.

YOLO predicts multiple bounding boxes per grid cell. At training time, we only want one bounding box predictor to be responsible for each object. YOLO assigns one predictor to be "responsible" for predicting an object based on which prediction has the highest current IOU with the ground truth. This leads to specialization between the bounding box predictors. Each predictor gets better at forecasting certain sizes, aspect ratios, or classes of objects, improving the overall recall score.

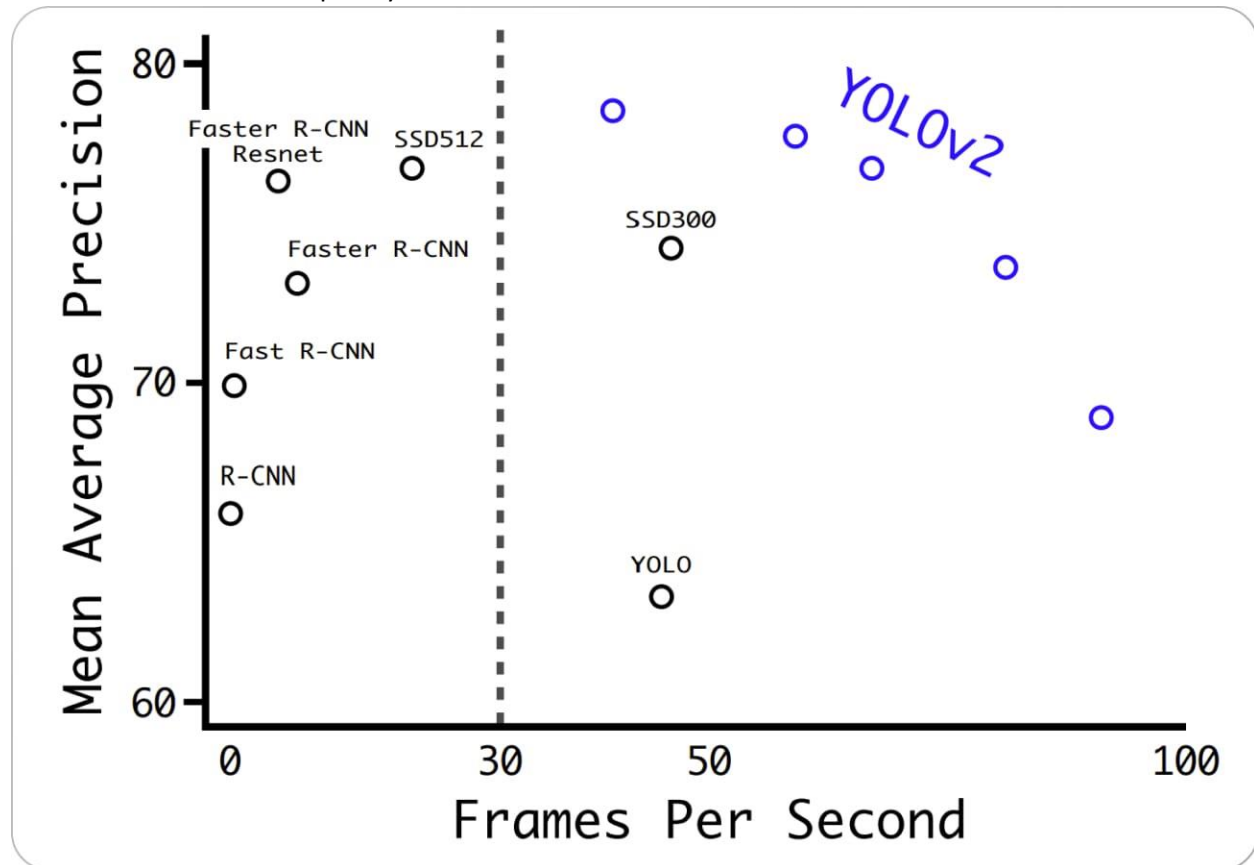
One key technique used in the YOLO models is non-maximum suppression (NMS). NMS is a post-processing step that is used to improve the accuracy and efficiency of object detection. In object detection, it is common for multiple bounding boxes to be generated for a single object in an image. These bounding boxes may overlap or be located at different positions, but they all represent the same object. NMS is used to identify and remove redundant or incorrect bounding boxes and to output a single bounding box for each object in the image.

Now, let us look into the improvements that the later versions of YOLO have brought to the parent model.

3.6.8 YOLOv2

YOLO v2, also known as YOLO9000, was introduced in 2016 as an improvement over the original YOLO algorithm. It was designed to be faster and more accurate than YOLO and to be able to detect a wider

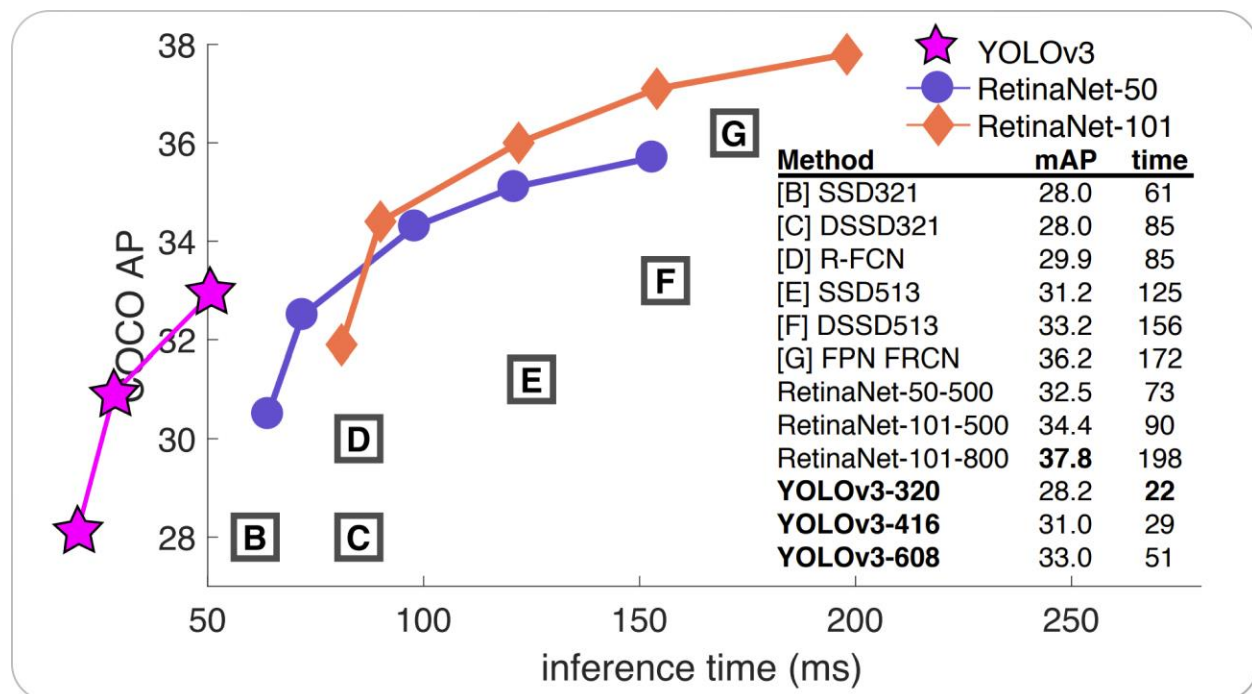
range of object classes. This updated version also uses a different CNN backbone called Darknet-19, a variant of the VGG Net architecture with simple progressive convolution and pooling layers. One of the main improvements in YOLO v2 is the use of anchor boxes. Anchor boxes are a set of predefined bounding boxes of different aspect ratios and scales. When predicting bounding boxes, YOLO v2 uses a combination of the anchor boxes and the predicted offsets to determine the final bounding box. This allows the algorithm to handle a wider range of object sizes and aspect ratios. Another improvement in YOLO v2 is the use of batch normalization, which helps to improve the accuracy and stability of the model. YOLO v2 also uses a multi-scale training strategy, which involves training the model on images at multiple scales and then averaging the predictions. This helps to improve the detection performance of small objects. YOLO v2 also introduces a new loss function better suited to object detection tasks. The loss function is based on the sum of the squared errors between the predicted and ground truth bounding boxes and class probabilities. The results obtained by YOLO v2 compared to the original version and other contemporary models are shown below.



3.6.9 YOLOv3

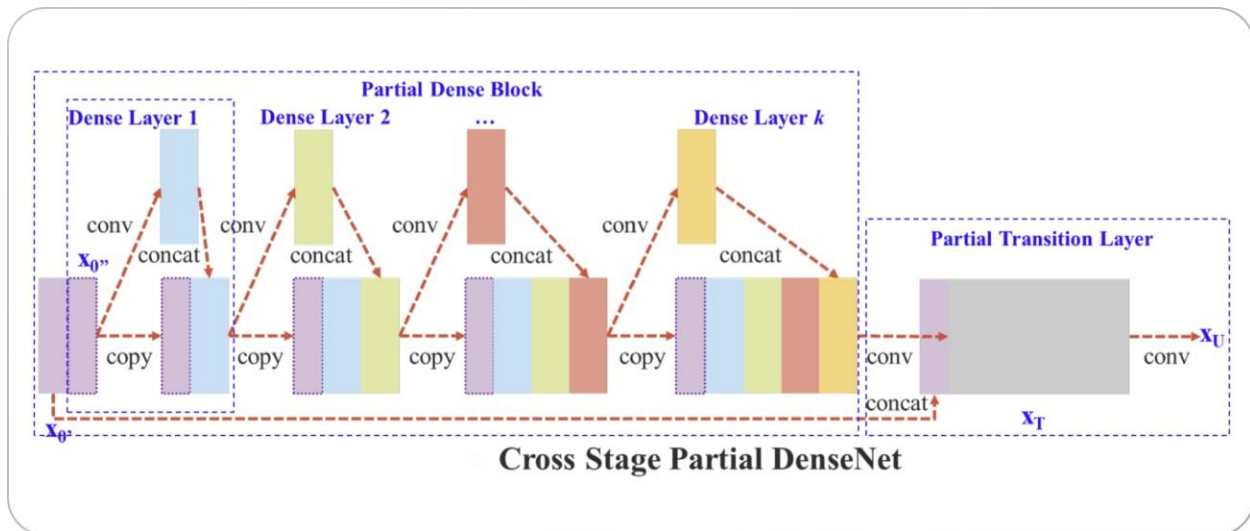
YOLO v3 is the third version of the YOLO object detection algorithm. It was introduced in 2018 as an improvement over YOLO v2, aiming to increase the accuracy and speed of the algorithm. One of the main improvements in YOLO v3 is the use of a new CNN architecture called Darknet-53. Darknet-53 is a variant of the ResNet architecture and is designed specifically for object detection tasks. It has 53

convolutional layers and is able to achieve state-of-the-art results on various object detection benchmarks. Another improvement in YOLO v3 are anchor boxes with different scales and aspect ratios. In YOLO v2, the anchor boxes were all the same size, which limited the ability of the algorithm to detect objects of different sizes and shapes. In YOLO v3 the anchor boxes are scaled, and aspect ratios are varied to better match the size and shape of the objects being detected. YOLO v3 also introduces the concept of "feature pyramid networks" (FPN). FPNs are a CNN architecture used to detect objects at multiple scales. They construct a pyramid of feature maps, with each level of the pyramid being used to detect objects at a different scale. This helps to improve the detection performance on small objects, as the model is able to see the objects at multiple scales. In addition to these improvements, YOLO v3 can handle a wider range of object sizes and aspect ratios. It is also more accurate and stable than the previous versions of YOLO.

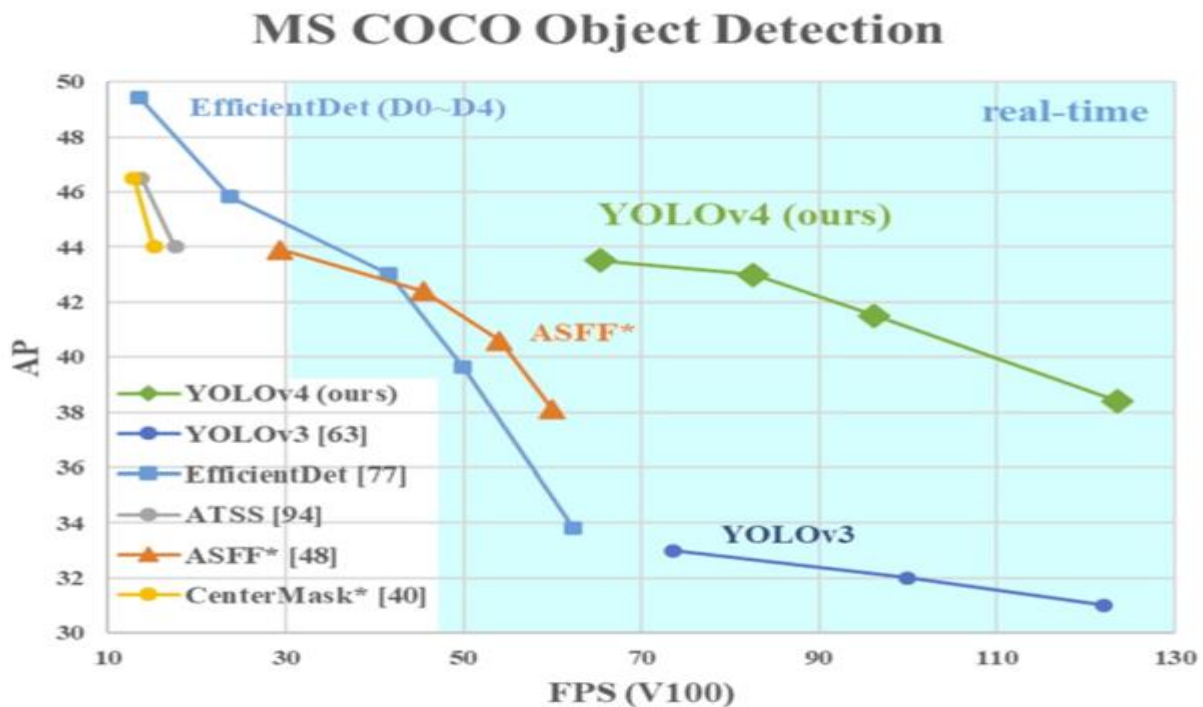


3.6.10 YOLOv4

YOLO v4 is the fourth version of the YOLO object detection algorithm introduced in 2020 by Bochkovskiy et al. as an improvement over YOLO v3. The primary improvement in YOLO v4 over YOLO v3 is the use of a new CNN architecture called CSPNet (shown below). CSPNet stands for "Cross Stage Partial Network" and is a variant of the ResNet architecture designed specifically for object detection tasks. It has a relatively shallow structure, with only 54 convolutional layers. However, it can achieve state-of-the-art results on various object detection benchmarks.

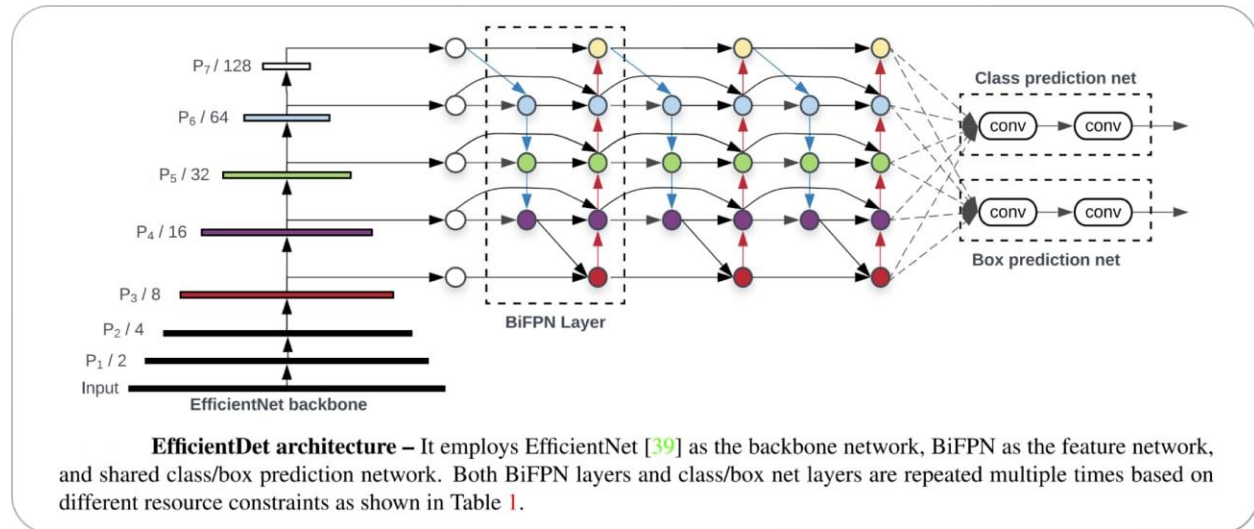


Both YOLO v3 and YOLO v4 use anchor boxes with different scales and aspect ratios to better match the size and shape of the detected objects. YOLO v4 introduces a new method for generating the anchor boxes, called "k-means clustering." It involves using a clustering algorithm to group the ground truth bounding boxes into clusters and then using the centroids of the clusters as the anchor boxes. This allows the anchor boxes to be more closely aligned with the detected objects' size and shape. While both YOLO v3 and YOLO v4 use a similar loss function for training the model, YOLO v4 introduces a new term called "GHM loss." It's a variant of the focal loss function and is designed to improve the model's performance on imbalanced datasets. YOLO v4 also improves the architecture of the FPNs used in YOLO v3.



3.6.11 YOLOv5

YOLO v5 was introduced in 2020 by the same team that developed the original YOLO algorithm as an open-source project and is maintained by Ultralytics. YOLO v5 builds upon the success of previous versions and adds several new features and improvements. Unlike YOLO, YOLO v5 uses a more complex architecture called EfficientDet (architecture shown below), based on the EfficientNet network architecture. Using a more complex architecture in YOLO v5 allows it to achieve higher accuracy and better generalization to a wider range of object categories.



Another difference between YOLO and YOLO v5 is the training data used to learn the object detection model. YOLO was trained on the PASCAL VOC dataset, which consists of 20 object categories. YOLO v5, on the other hand, was trained on a larger and more diverse dataset called D5, which includes a total of 600 object categories. YOLO v5 uses a new method for generating the anchor boxes, called "dynamic anchor boxes." It involves using a clustering algorithm to group the ground truth bounding boxes into clusters and then using the centroids of the clusters as the anchor boxes. This allows the anchor boxes to be more closely aligned with the detected objects' size and shape. YOLO v5 also introduces the concept of "spatial pyramid pooling" (SPP), a type of pooling layer used to reduce the spatial resolution of the feature maps. SPP is used to improve the detection performance on small objects, as it allows the model to see the objects at multiple scales. YOLO v4 also uses SPP, but YOLO v5 includes several improvements to the SPP architecture that allow it to achieve better results. YOLO v4 and YOLO v5 use a similar loss function to train the model. However, YOLO v5 introduces a new term called "CloU loss," which is a variant of the IoU loss function designed to improve the model's performance on imbalanced datasets.

3.6.12 YOLOv6

YOLO v6 was proposed in 2022 by Li et al. as an improvement over previous versions. One of the main differences between YOLO v5 and YOLO v6 is the CNN architecture used. YOLO v6 uses a variant of the EfficientNet architecture called EfficientNet-L2. It's a more efficient architecture than EfficientDet used

in YOLO v5, with fewer parameters and a higher computational efficiency. It can achieve state-of-the-art results on various object detection benchmarks. The framework of the YOLO v6 model is shown below.

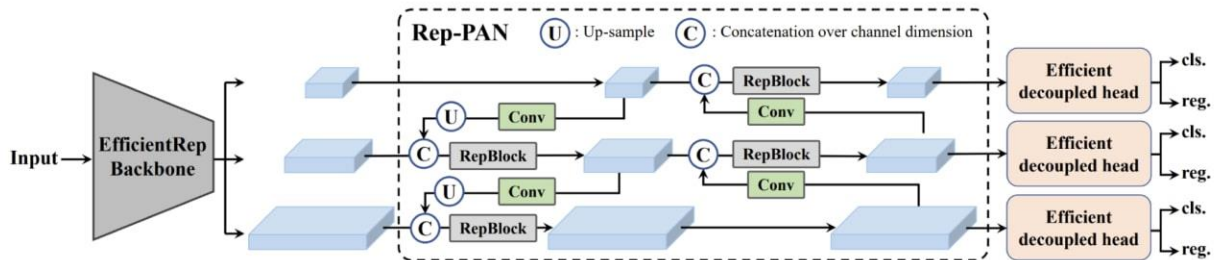
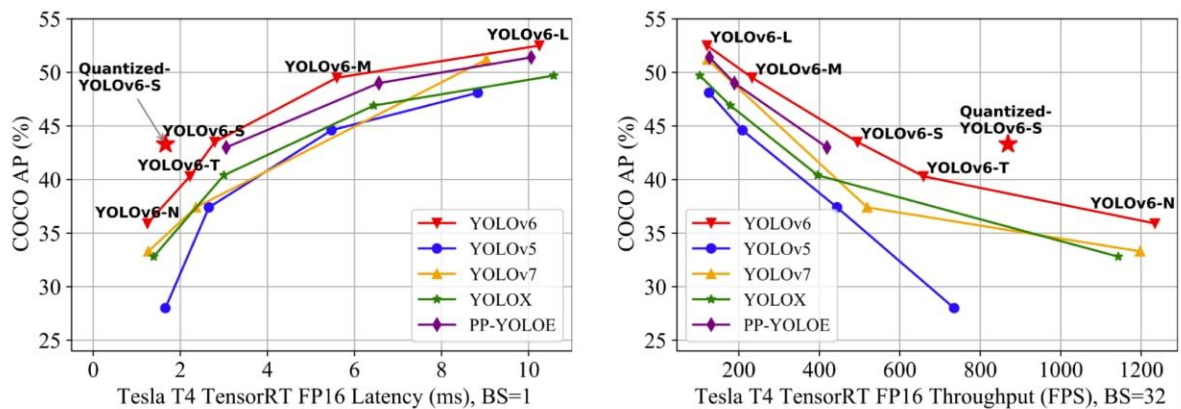


Figure 2: The YOLOv6 framework (N and S are shown). Note for M/L, RepBlocks is replaced with CSPStackRep.

YOLO v6 also introduces a new method for generating the anchor boxes, called "dense anchor boxes."

The results obtained by YOLO v6 compared to other state-of-the-art methods are shown below.

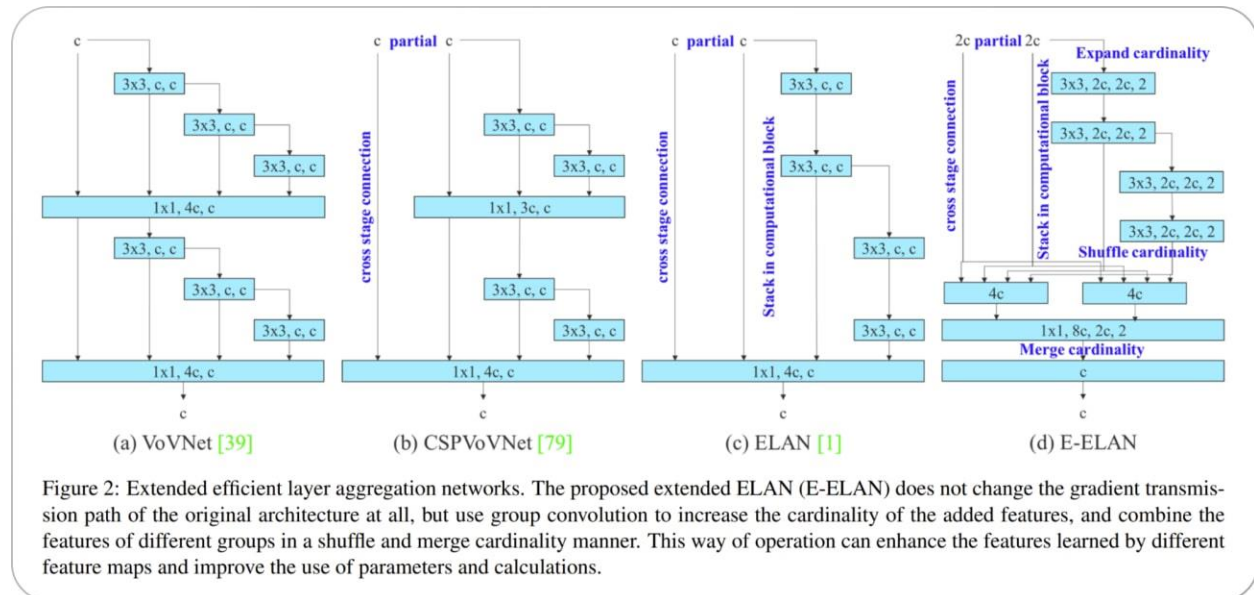


Comparison of state-of-the-art efficient object detectors. Both latency and throughput (at a batch size of 32) are given for a handy reference. All models are test with TensorRT 7 except that the quantized model is with TensorRT 8.

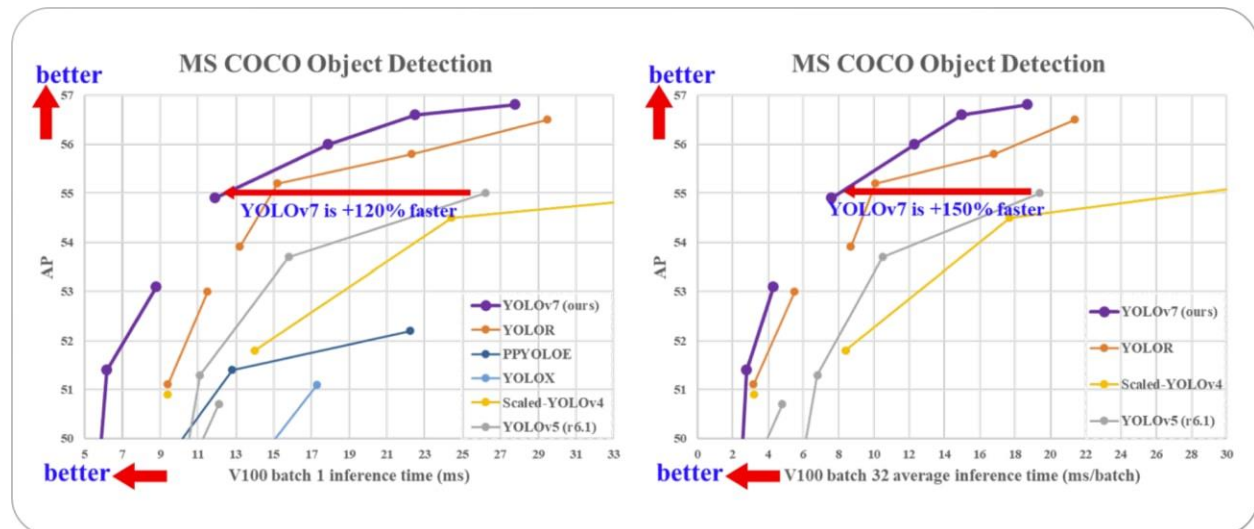
3.6.13 YOLOv7

YOLO v7, the latest version of YOLO, has several improvements over the previous versions. One of the main improvements is the use of anchor boxes. Anchor boxes are a set of predefined boxes with different aspect ratios that are used to detect objects of different shapes. YOLO v7 uses nine anchor boxes, which allows it to detect a wider range of object shapes and sizes compared to previous versions, thus helping to reduce the number of false positives.

A key improvement in YOLO v7 is the use of a new loss function called “focal loss.” Previous versions of YOLO used a standard cross-entropy loss function, which is known to be less effective at detecting small objects. Focal loss battles this issue by down-weighting the loss for well-classified examples and focusing on the hard examples—the objects that are hard to detect. YOLO v7 also has a higher resolution than the previous versions. It processes images at a resolution of 608 by 608 pixels, which is higher than the 416 by 416 resolution used in YOLO v3. This higher resolution allows YOLO v7 to detect smaller objects and to have a higher accuracy overall.



One of the main advantages of YOLO v7 is its speed. It can process images at a rate of 155 frames per second, much faster than other state-of-the-art object detection algorithms. Even the original baseline YOLO model was capable of processing at a maximum rate of 45 frames per second. This makes it suitable for sensitive real-time applications such as surveillance and self-driving cars, where higher processing speeds are crucial.



Regarding accuracy, YOLO v7 performs well compared to other object detection algorithms. It achieves an average precision of 37.2% at an IoU (intersection over union) threshold of 0.5 on the popular COCO dataset, which is comparable to other state-of-the-art object detection algorithms. The quantitative comparison of the performance is shown below.

Model	#Param.	FLOPs	Size	AP ^{val}	AP ^{val} ₅₀	AP ^{val} ₇₅	AP ^{val} _S	AP ^{val} _M	AP ^{val} _L
YOLOv4 [3]	64.4M	142.8G	640	49.7%	68.2%	54.3%	32.9%	54.8%	63.7%
YOLOv4-u5 (r6.1) [81]	46.5M	109.1G	640	50.2%	68.7%	54.6%	33.2%	55.5%	63.7%
YOLOv4-CSP [79]	52.9M	120.4G	640	50.3%	68.6%	54.9%	34.2%	55.6%	65.1%
YOLOv4-CSP [81]	52.9M	120.4G	640	50.8%	69.5%	55.3%	33.7%	56.0%	65.4%
YOLOv7	36.9M	104.7G	640	51.2%	69.7%	55.5%	35.2%	56.0%	66.7%
improvement	-43%	-15%	-	+0.4	+0.2	+0.2	+1.5	=	+1.3
YOLOv4-CSP-X [81]	96.9M	226.8G	640	52.7%	71.3%	57.4%	36.3%	57.5%	68.3%
YOLOv7-X	71.3M	189.9G	640	52.9%	71.1%	57.5%	36.9%	57.7%	68.6%
improvement	-36%	-19%	-	+0.2	-0.2	+0.1	+0.6	+0.2	+0.3
YOLOv4-tiny [79]	6.1	6.9	416	24.9%	42.1%	25.7%	8.7%	28.4%	39.2%
YOLOv7-tiny	6.2	5.8	416	35.2%	52.8%	37.3%	15.7%	38.0%	53.4%
improvement	+2%	-19%	-	+10.3	+10.7	+11.6	+7.0	+9.6	+14.2
YOLOv4-tiny-3l [79]	8.7	5.2	320	30.8%	47.3%	32.2%	10.9%	31.9%	51.5%
YOLOv7-tiny	6.2	3.5	320	30.8%	47.3%	32.2%	10.0%	31.9%	52.2%
improvement	-39%	-49%	-	=	=	=	-0.9	=	+0.7
YOLOv4-E6 [81]	115.8M	683.2G	1280	55.7%	73.2%	60.7%	40.1%	60.4%	69.2%
YOLOv7-E6	97.2M	515.2G	1280	55.9%	73.5%	61.1%	40.6%	60.3%	70.0%
improvement	-19%	-33%	-	+0.2	+0.3	+0.4	+0.5	-0.1	+0.8
YOLOv4-D6 [81]	151.7M	935.6G	1280	56.1%	73.9%	61.2%	42.4%	60.5%	69.9%
YOLOv7-D6	154.7M	806.8G	1280	56.3%	73.8%	61.4%	41.3%	60.6%	70.1%
YOLOv7-E6E	151.7M	843.2G	1280	56.8%	74.4%	62.1%	40.8%	62.1%	70.6%
improvement	=	-11%	-	+0.7	+0.5	+0.9	-1.6	+1.6	+0.7

However, it should be noted that YOLO v7 is less accurate than two-stage detectors such as Faster R-CNN and Mask R-CNN, which tend to achieve higher average precision on the COCO dataset but also require longer inference times.

Limitations of YOLO v7

YOLO v7 is a powerful and effective object detection algorithm, but it does have a few limitations.

- YOLO v7, like many object detection algorithms, struggles to detect small objects. It might fail to accurately detecting objects in crowded scenes or when objects are far away from the camera.
- YOLO v7 is also not perfect at detecting objects at different scales. This can make it difficult to detect objects that are either very large or very small compared to the other objects in the scene.
- YOLO v7 can be sensitive to changes in lighting or other environmental conditions, so it may be inconvenient to use in real-world applications where lighting conditions may vary.
- YOLO v7 can be computationally intensive, which can make it difficult to run in real-time on resource-constrained devices like smartphones or other edge devices.

3.6.14 YOLOv8

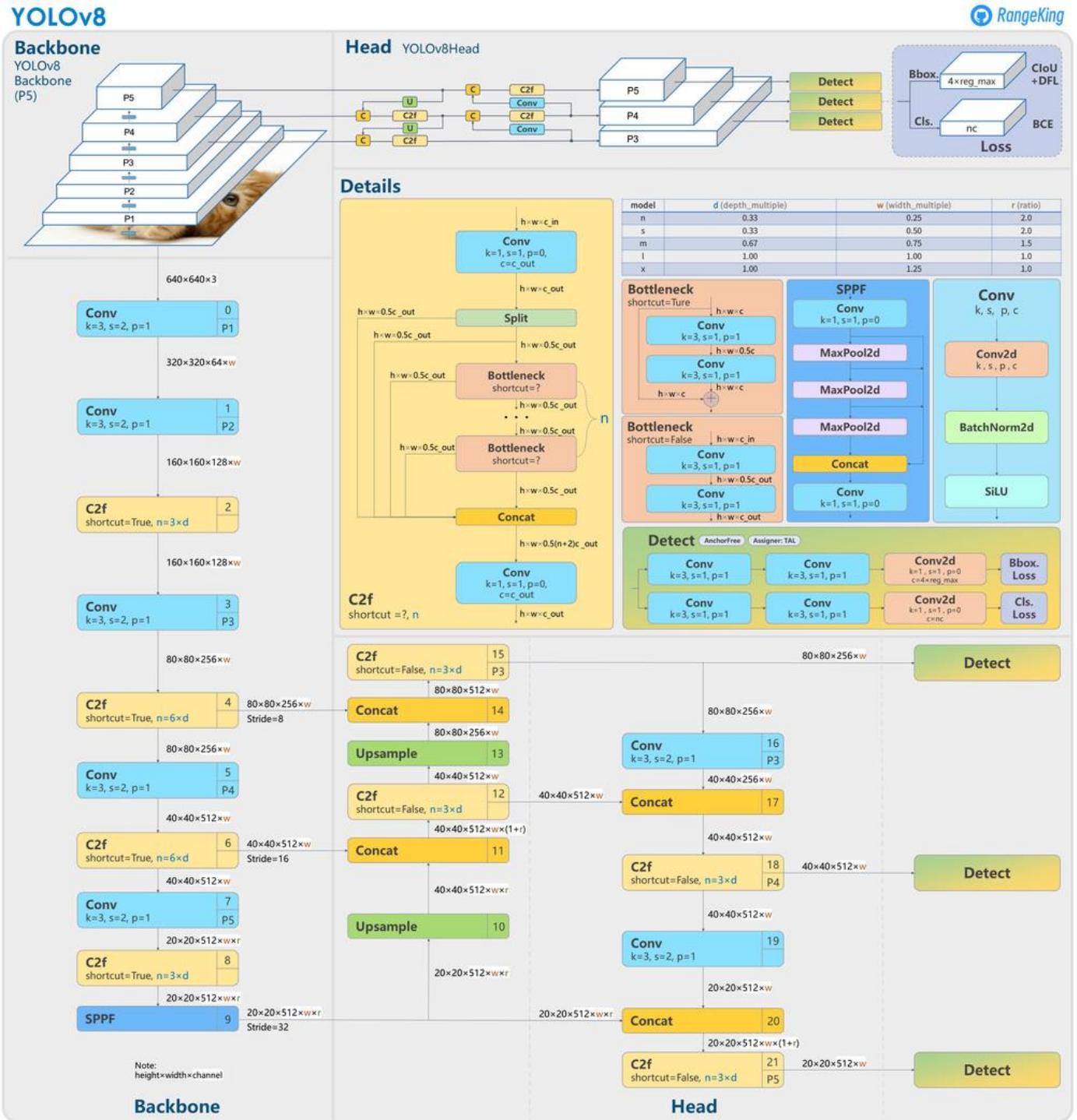
YOLOv8 is the newest state-of-the-art YOLO model that can be used for object detection, image classification, and instance segmentation tasks. YOLOv8 was developed by Ultralytics, who also created the influential and industry-defining YOLOv5 model. YOLOv8 includes numerous architectural and developer experience changes and improvements over YOLOv5.

Here are a few main reasons why you should consider using YOLOv8 for your next computer vision project:

- YOLOv8 has a high rate of accuracy measured by COCO and Roboflow 100.
- YOLOv8 comes with a lot of developer-convenience features, from an easy-to-use CLI to a well-structured Python package.
- There is a large community around YOLO and a growing community around the YOLOv8 model, meaning there are many people in computer vision circles who may be able to assist you when you need guidance.

YOLOv8 achieves strong accuracy on COCO. For example, the YOLOv8m model -- the medium model -- achieves a 50.2% mAP when measured on COCO. When evaluated against Roboflow 100, a dataset that specifically evaluates model performance on various task-specific domains, YOLOv8 scored substantially better than YOLOv5. More information on this is provided in our performance analysis later in the article. Furthermore, the developer-convenience features in YOLOv8 are significant. As opposed to other models where tasks are split across many different Python files that you can execute, YOLOv8 comes with a CLI that makes training a model more intuitive. This is in addition to a Python package that provides a more seamless coding experience than prior models.

The following image made by GitHub user RangeKing shows a detailed visualisation of the network's architecture.



Anchor Free Detection

YOLOv8 is an anchor-free model. This means it predicts directly the center of an object instead of the offset from a known anchor box.

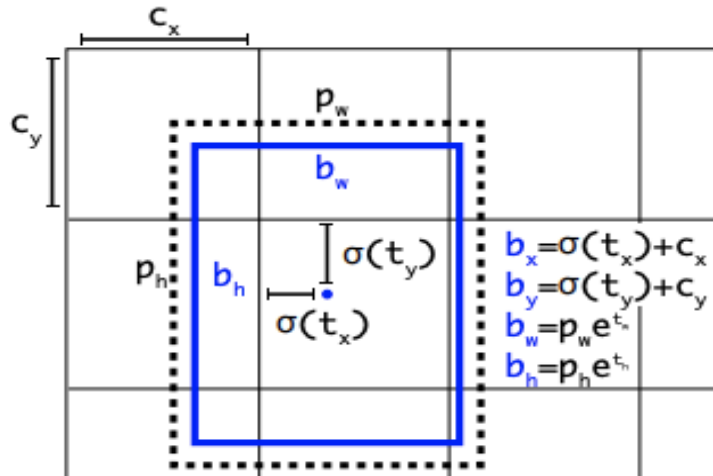


Figure 1 Visualization of an anchor box in YOLO

Anchor boxes were a notoriously tricky part of earlier YOLO models, since they may represent the distribution of the target benchmark's boxes but not the distribution of the custom dataset.

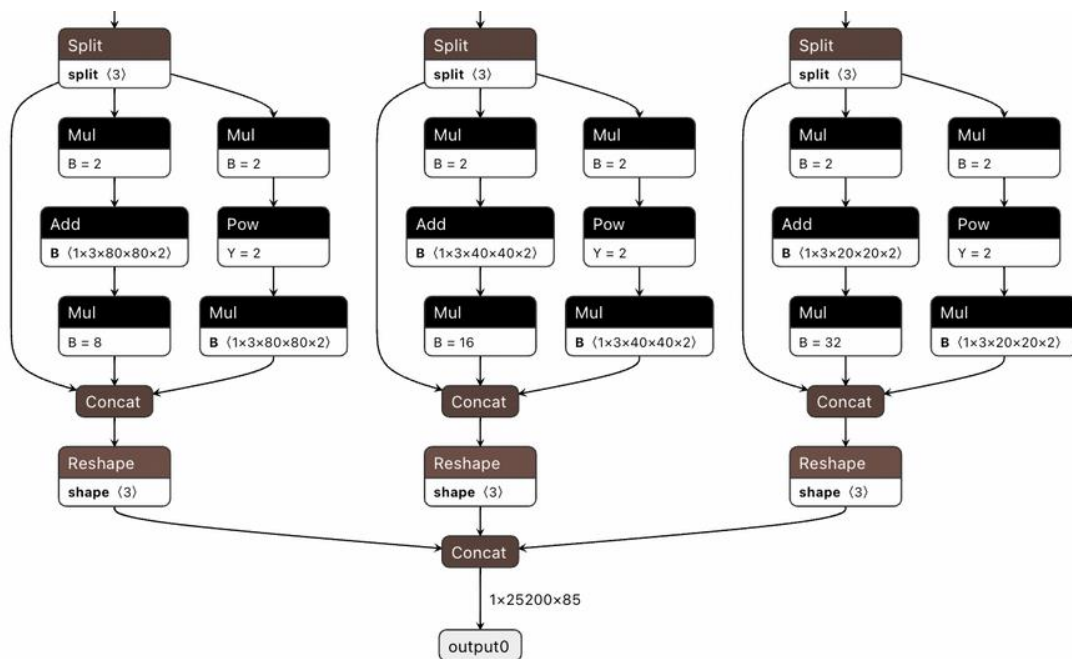


Figure 2 The detection head of YOLOv5

Anchor free detection reduces the number of box predictions, which speeds up Non-Maximum Suppression (NMS), a complicated post processing step that sifts through candidate detections after inference.

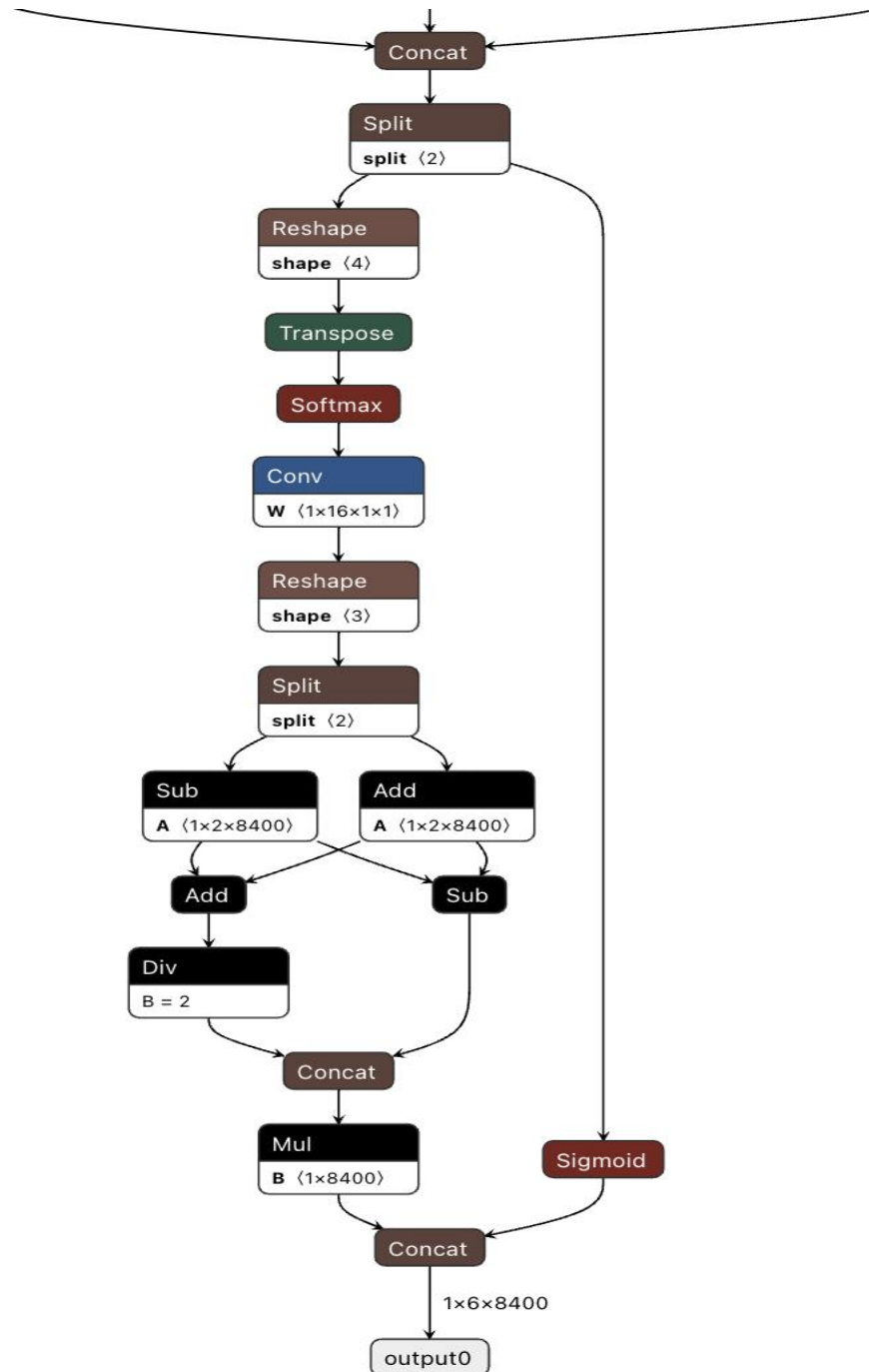


Figure 3 The detection head for YOLOv8

New Convolutions

The stem's first 6x6 conv is replaced by a 3x3, the main building block was changed, and C2f replaced C3. The module is summarized in the picture below, where "f" is the number of features, "e" is the expansion rate and CBS is a block composed of a Conv, a BatchNorm and a SiLU later. In C2f, all the outputs from the Bottleneck (fancy name for two 3x3 convs with residual connections) are concatenated. While in C3 only the output of the last Bottleneck was used.

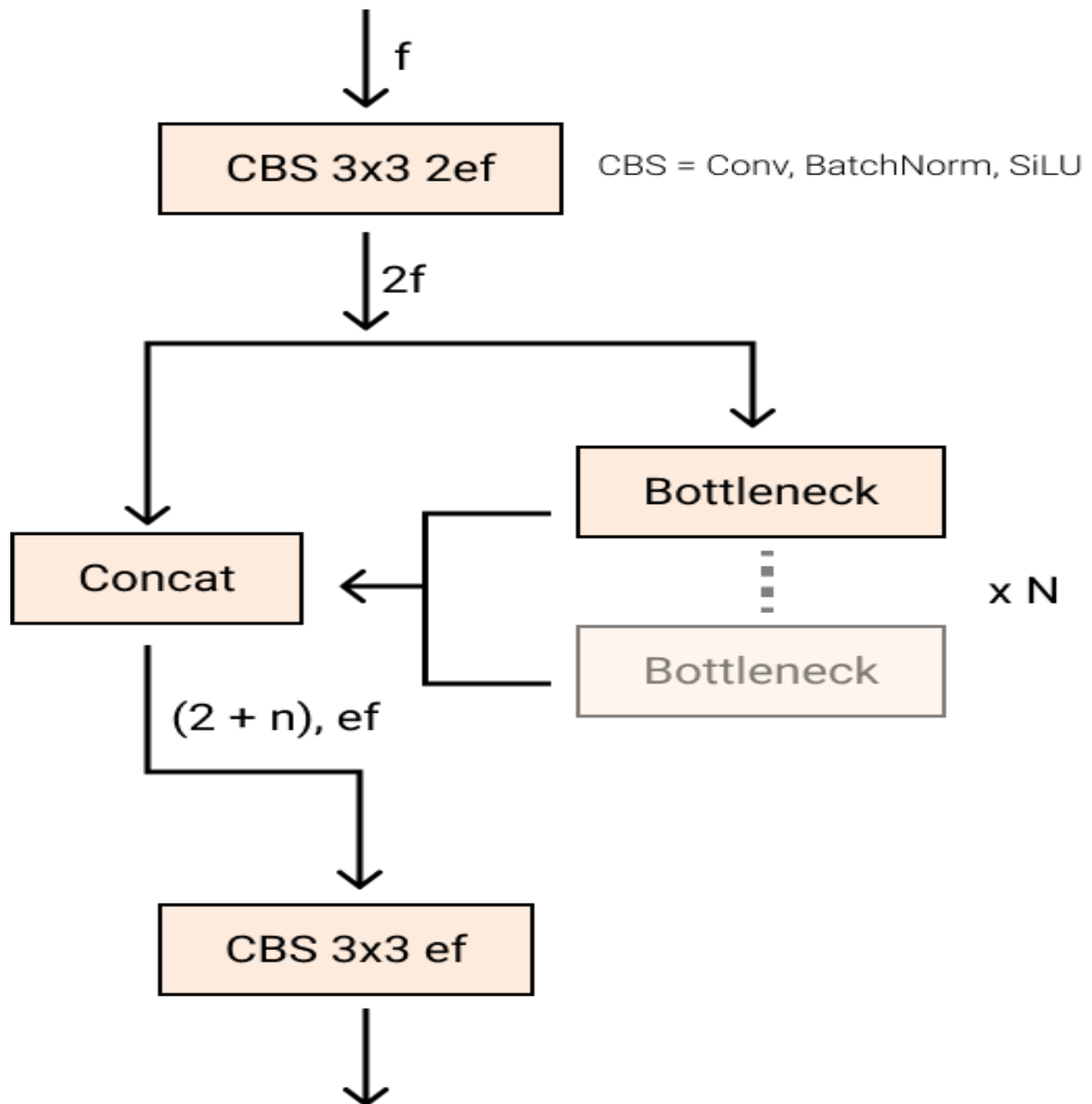


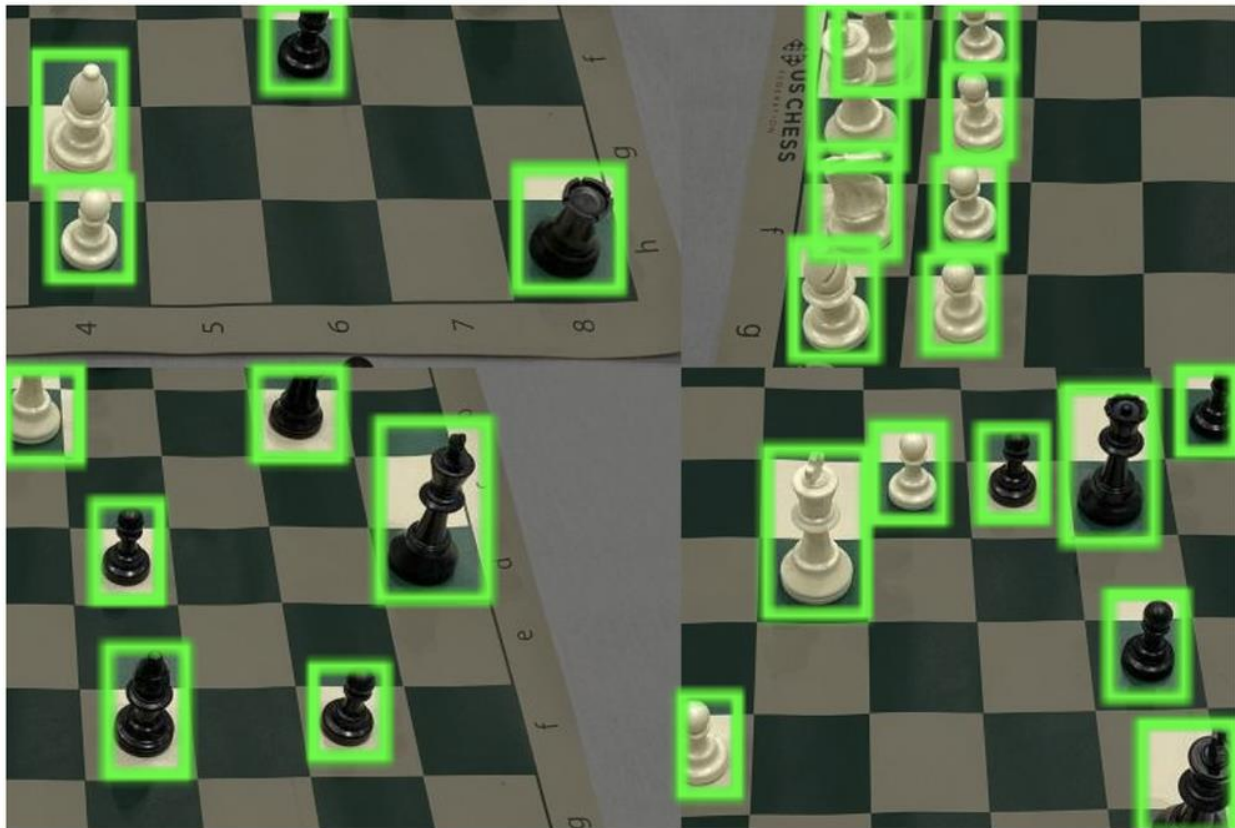
Figure 4New YOLOv8 C2f module

The Bottleneck is the same as in YOLOv5 but the first conv's kernel size was changed from 1x1 to 3x3. From this information, we can see that YOLOv8 is starting to revert to the ResNet block defined in 2015.

In the neck, features are concatenated directly without forcing the same channel dimensions. This reduces the parameters count and the overall size of the tensors.

Closing the Mosaic Augmentation

Deep learning research tends to focus on model architecture, but the training routine in YOLOv5 and YOLOv8 is an essential part of their success. YOLOv8 augments images during training online. At each epoch, the model sees a slightly different variation of the images it has been provided. One of those augmentations is called mosaic augmentation. This involves stitching four images together, forcing the model to learn objects in new locations, in partial occlusion, and against different surrounding pixels.



However, this augmentation is empirically shown to degrade performance if performed through the whole training routine. It is advantageous to turn it off for the last ten training epochs. This sort of change is exemplary of the careful attention YOLO modeling has been given in overtime in the YOLOv5 repo and in the YOLOv8 research.

YOLOv8 COCO Accuracy

COCO (Common Objects in Context) is the industry standard benchmark for evaluating object detection models. When comparing models on COCO, we look at the mAP value and FPS measurement for inference speed. Models should be compared at similar inference speeds. The image below shows the accuracy of YOLOv8 on COCO, using data collected by the Ultralytics team and published in their YOLOv8 README:

▼ Detection

Model	size (pixels)	mAP ^{val} 50-95	Speed CPU (ms)	Speed T4 GPU (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	-	-	3.2	8.7
YOLOv8s	640	44.9	-	-	11.2	28.6
YOLOv8m	640	50.2	-	-	25.9	78.9
YOLOv8l	640	52.9	-	-	43.7	165.2
YOLOv8x	640	53.9	-	-	68.2	257.8

- mAP^{val} values are for single-model single-scale on [COCO val2017](#) dataset.
Reproduce by `yolo mode=val task=detect data=coco.yaml device=0`
- Speed averaged over COCO val images using an [Amazon EC2 P4d](#) instance.
Reproduce by `yolo mode=val task=detect data=coco128.yaml batch=1 device=0/cpu`

3.7 PYTHON

Python is a high-level, interpreted, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library. Guido van Rossum began working on Python in the late 1980s as a successor to the ABC programming language and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features such as list comprehensions, cycle-detecting garbage collection, reference counting, and Unicode support. Python 3.0, released in 2008, was a major revision that is not completely backward-compatible with earlier versions. Python 2 was discontinued with version 2.7.18 in 2020. Python consistently ranks as one of the most popular programming languages.

Pros of Python:

- Python is easy to learn and read.
- Python enhances productivity.
- Has a vast collection of libraries.
- Python is free, open-source, and has a vibrant community.
- Portable programming language.(A portable programming language is one that can work on any platform without requiring the developer to make changes to the code.)
- Used widely for AI, ML and DL with a huge support community and library.

Cons of Python:

- Python has speed limitations.
- Python can have runtime errors.
- Python consumes a lot of memory space.
- Python is not easy to test.

3.8 TENSORFLOW

TensorFlow is an end-to-end open-source platform for machine learning

TensorFlow is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML, and gives developers the ability to easily build and deploy ML-powered applications. TensorFlow provides a collection of workflows with intuitive, high-level APIs for both beginners and experts to create machine learning models in numerous languages. Developers have the option to deploy models on a number of platforms such as on servers, in the cloud, on mobile and edge devices, in browsers, and on many other JavaScript platforms. This enables developers to go from model building and training to deployment much more easily.

Easy model building

Build and train ML models easily using intuitive high-level APIs like Keras with eager execution, which makes for immediate model iteration and easy debugging.

Robust ML production anywhere

Easily train and deploy models in the cloud, on-prem, in the browser, or on-device no matter what language you use.

Powerful experimentation for research

A simple and flexible architecture to take new ideas from concept to code, to state-of-the-art models, and to publication faster.

How Google uses TensorFlow

TensorFlow is used by many internal Google products and teams including: Search, Gmail, Translate, Maps, Android, Photos, Speech, YouTube, and Play.

3.9 PyTorch

PyTorch is pythonic in nature, which means it follows the coding style that uses Python's unique features to write readable code. Python is also popular for its use of dynamic computation graphs. It enables developers, scientists and neural network debuggers to run and test a portion of code in real time instead of waiting for the entire program to be written.

PyTorch provides the following key features:

- **Tensor computation.** Similar to NumPy array -- an open source library of Python that adds support for large, multidimensional arrays -- tensors are generic n-dimensional arrays used for arbitrary numeric computation and are accelerated by graphics processing units. These multidimensional structures can be operated on and manipulated with application program interfaces (APIs).
- **TorchScript.** This is the production environment of PyTorch that enables users to seamlessly transition between modes. TorchScript optimizes functionality, speed, ease of use and flexibility.
- **Dynamic graph computation.** This feature lets users change network behavior on the fly, rather than waiting for all the code to be executed.
- **Automatic differentiation.** This technique is used for creating and training neural networks. It numerically computes the derivative of a function by making backward passes in neural networks.
- **Python support.** Because PyTorch is based on Python, it can be used with popular libraries and packages such as NumPy, SciPy, Numba and Cynthon.
- **Variable.** The variable is enclosed outside the tensor to hold the gradient. It represents a node in a computational graph.
- **Parameter.** Parameters are wrapped around a variable. They're used when a parameter needs to be used as a tensor, which isn't possible when using a variable.
- **Module.** Modules represent neural networks and are the building blocks of stateful computation. A module can contain other modules and parameters.
- **Functions.** These are the relationships between two variables. Functions don't have memory to store any state or buffer and have no memory of their own.

PyTorch benefits

Using PyTorch can provide the following benefits:

- Offers developers an easy-to-learn, simple-to-code structure that's based on Python.
- Enables easy debugging with popular Python tools.
- Offers scalability and is well-supported on major cloud platforms.

- Provides a small community focused on open source.
- Exports learning models to the Open Neural Network Exchange (ONNX) standard format.
- Offers a user-friendly interface.
- Provides a C++ front-end interface option.
- Includes a rich set of powerful APIs that extend the PyTorch library.

3.10 Google Colab

Colaboratory, or “Colab” for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. Colab resources are not guaranteed and not unlimited, and the usage limits sometimes fluctuate. This is necessary for Colab to be able to provide resources free of charge. For more details, see Resource Limits.

Users who are interested in more reliable access to better resources may be interested in Colab Pro. Resources in Colab are prioritized for interactive use cases. We prohibit actions associated with bulk computation, actions that negatively impact others, as well as actions associated with bypassing colab policies.

Used in projects to run the ML-Module and API and for saving some data in Google Drive.

Colab Pros:

- Pre-built with lots of python library
- Quick Start of python learning
- No infra setup required
- No charges for GPU usage
- Can run your code for 24 hours without interruptions but not more than that
- Your notebooks is saved in google drive only

Colab Cons:

- Need to install all specific libraries which does not come with standard python (Need to repeat this with every session)
- Google Drive is your source and target for Storage, there are other like local (which eats your bandwidth if dataset is big)

2023

- Google provided the code to connect and use with google drive but that will not work with lots of other data format
- Google Storage is used with the current session, so if you have downloaded some file and want to use it later, better save it before closing the session.
- Difficult to work with BIGGER datasets as you have to download and store them in Google drive (15 GB Free space with Gmail id, additional required a payment towards google)

Reference

1. Agarwal, S., Awan, A., and Roth, D. (2004). Learning to detect objects in images via a sparse, part-based representation. *IEEE Trans. Pattern Anal. Mach. Intell.* 26, 1475–1490. doi:10.1109/TPAMI.2004.108
2. Alexe, B., Deselaers, T., and Ferrari, V. (2010). “What is an object?,” in *Computer Vision and Pattern Recognition (CVPR)*, 2010 IEEE Conference on (San Francisco, CA: IEEE), 73–80. doi:10.1109/CVPR.2010.5540226
3. Aloimonos, J., Weiss, I., and Bandyopadhyay, A. (1988). Active vision. *Int. J. Comput. Vis.* 1, 333–356. doi:10.1007/BF00133571
4. Andreopoulos, A., and Tsotsos, J. K. (2013). 50 years of object recognition: directions forward. *Comput. Vis. Image Underst.* 117, 827–891. doi:10.1016/j.cviu.2013.04.005
5. Azizpour, H., and Laptev, I. (2012). “Object detection using strongly-supervised deformable part models,” in *Computer Vision-ECCV 2012* (Florence: Springer), 836–849.
6. Azzopardi, G., and Petkov, N. (2013). Trainable cosfire filters for keypoint detection and pattern recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* 35, 490–503. doi:10.1109/TPAMI.2012.106
7. Azzopardi, G., and Petkov, N. (2014). Ventral-stream-like shape representation: from pixel intensity values to trainable object-selective cosfire models. *Front. Comput. Neurosci.* 8:80. doi:10.3389/fncom.2014.00080
8. Benbouzid, D., Busa-Fekete, R., and Keggl, B. (2012). “Fast classification using sparse decision dags,” in *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, ICML ‘12, eds J. Langford and J. Pineau (New York, NY: Omnipress), 951–958.
9. Bengio, Y. (2012). “Deep learning of representations for unsupervised and transfer learning,” in *ICML Unsupervised and Transfer Learning*, Volume 27 of *JMLR Proceedings*, eds I. Guyon, G. Dror, V. Lemaire, G. W. Taylor, and D. L. Silver (Bellevue: JMLR.Org), 17–36.
10. Bourdev, L. D., Maji, S., Brox, T., and Malik, J. (2010). “Detecting people using mutually consistent poselet activations,” in *Computer Vision – ECCV2010 – 11th European Conference on Computer Vision*, Heraklion, Crete, Greece, September 5–11, 2010, *Proceedings, Part VI*, Volume 6316 of *Lecture Notes in Computer Science*, eds K. Daniilidis, P. Maragos, and N. Paragios (Heraklion: Springer), 168–181.
11. Bourdev, L. D., and Malik, J. (2009). “Poselets: body part detectors trained using 3d human pose annotations,” in *IEEE 12th International Conference on Computer Vision*, ICCV 2009, Kyoto, Japan, September 27 – October 4, 2009 (Kyoto: IEEE), 1365–1372.
12. Cadena, C., Dick, A., and Reid, I. (2015). “A fast, modular scene understanding system using context-aware object detection,” in *Robotics and Automation (ICRA)*, 2015 IEEE International Conference on (Seattle, WA).

13. Correa, M., Hermosilla, G., Verschae, R., and Ruiz-del-Solar, J. (2012). Humandetection and identification by robots using thermal and visual information indomestic environments. *J. Intell. Robot Syst.* 66, 223–243. doi:10.1007/s10846-011-9612-2
14. Dalal, N., and Triggs, B. (2005). “Histograms of oriented gradients for humandetection,” in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, Vol. 1 (San Diego, CA: IEEE), 886–893. doi:10.1109/CVPR.2005.177
15. Erhan, D., Szegedy, C., Toshev, A., and Anguelov, D. (2014). “Scalable object detection using deep neural networks,” in *Computer Vision and Pattern Recognition Frontiers in Robotics and AI* www.frontiersin.org November 2015
16. <https://monkeylearn.com/machine-learning>
17. <https://www.forbes.com/sites/bernardmarr/2019/04/08/7-amazing-examples-of-computer-and-machine-vision-in-practice/#3dbb3f751018> (Link resides outside ibm.com)
18. <https://hackernoon.com/a-brief-history-of-computer-vision-and-convolutional-neural-networks-8fe8aacc79f3> (Link resides outside ibm.com)
19. Optical character recognition, Wikipedia (Link resides outside ibm.com)
20. Intelligent character recognition, Wikipedia (Link resides outside ibm.com)
21. A Brief History of Computer Vision (and Convolutional Neural Networks), Rostyslav Demush, Hacker Noon, February 27, 2019 (Link resides outside ibm.com)
22. 7 Amazing Examples of Computer And Machine Vision In Practice, Bernard Marr, Forbes, April 8, 2019 (Link resides outside ibm.com)
23. 7. The 5 Computer Vision Techniques That Will Change How You See The World, James Le, Heartbeat, April 12, 2018 (Link resides outside ibm.com)
24. <https://medium.com/@whatdhack/reflections-on-non-maximum-suppression-nms-d2fce148ef0a>
25. <https://www.jeremyjordan.me/evaluating-image-segmentation-models/>
26. <https://github.com/rbgirshick/fast-rcnn/blob/master/lib/utils/nms.py>
27. <https://medium.com/koderunners/intersection-over-union-516a3950269c>
28. <https://blog.roboflow.com/whats-new-in-yolov8/>
29. <https://opensource.google/projects/tensorflow>
30. <https://www.techtarget.com/searchenterpriseai/definition/PyTorch>
31. Agarwal, S., Awan, A., and Roth, D. (2004). Learning to detect objects in images via a sparse, part-based representation. *IEEE Trans. Pattern Anal. Mach. Intell.* 26, 1475–1490. doi: 10.1109/TPAMI.2004.108
32. Alexe, B., Deselaers, T., and Ferrari, V. (2010). “What is an object?,” in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on* (San Francisco, CA: IEEE), 73–80. doi:10.1109/CVPR.2010.5540226
33. Aloimonos, J., Weiss, I., and Bandyopadhyay, A. (1988). Active vision. *Int. J. Comput. Vis.* 1, 333–356. doi:10.1007/BF00133571
34. Andreopoulos, A., and Tsotsos, J. K. (2013). 50 years of object recognition: directions forward. *Comput. Vis. Image Underst.* 117, 827–891. doi:10.1016/j.cviu.2013.04.005

35. Azizpour, H., and Laptev, I. (2012). "Object detection using strongly-supervised deformable part models," in *Computer Vision-ECCV 2012* (Florence: Springer), 836–849.
36. Azzopardi, G., and Petkov, N. (2014). Ventral-stream-like shape representation: from pixel intensity values to trainable object-selective cosfire models. *Front. Comput. Neurosci.* 8:80. doi:10.3389/fncom.2014.00080
37. Wu, L., Hoi, S., and Yu, N. (2010). Semantics-preserving bag-of-words models and applications. *IEEE Trans. Image Process.* 19, 1908–1920. doi:10.1109/TIP.2010.2045169
38. Wu, L., Hu, Y., Li, M., Yu, N., and Hua, X.-S. (2009). Scale-invariant visual language modeling for object categorization. *IEEE Trans. Multimedia* 11, 286–294. doi:10.1109/TMM.2008.2009692
39. Yan, J., Lei, Z., Wen, L., and Li, S. Z. (2014). "The fastest deformable part model for object detection," in *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on* (Columbus, OH: IEEE), 2497–2504.
40. Yang, M.-H., Ahuja, N., and Kriegman, D. (2000a). "Mixtures of linear subspaces for face detection," in *Proc. Fourth IEEE Int. Conf. on Automatic Face and Gesture Recognition* (Grenoble: IEEE), 70–76.
41. Yang, M.-H., Roth, D., and Ahuja, N. (2000b). "A SNoW-based face detector," in *Advances in Neural Information Processing Systems 12* (Denver: MIT press), 855–861.
42. Yang, M.-H., Kriegman, D., and Ahuja, N. (2002). Detecting faces in images: a survey. *IEEE Trans. Pattern Anal. Mach. Intell.* 24, 34–58. doi:10.1109/34.982883
43. Zafeiriou, S., Zhang, C., and Zhang, Z. (2015). A survey on face detection in the wild: past, present and future. *Comput. Vis. Image Underst.* 138, 1–24. doi:10.1016/j.cviu.2015.03.015
44. Zeng, X., Ouyang, W., and Wang, X. (2013). "Multi-stage contextual deep learning for pedestrian detection," in *Computer Vision (ICCV), 2013 IEEE International Conference on* (Washington, DC: IEEE), 121–128.
45. Zhou, B., Khosla, A., Lapedriza, À., Oliva, A., and Torralba, A. (2014). Object Detectors Emerge in Deep Scene Cnns. *CoRR*, abs/1412.6856.
46. Zhu, X., and Ramanan, D. (2012). "Face detection, pose estimation, and landmark localization in the wild," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on* (Providence: IEEE), 2879–2886.