

**Detection of Teeth details and Root canal deviation along
with its dimensions using YOLOv5;
A Custom Object Detection**

Submitted in partial fulfillment of the requirements for the award of
Master of Science degree in Bioinformatics and Data science

By

Mr. Anantha Reddi Yaswanth (40739001)



**DEPARTMENT OF BIOINFORMATICS AND DATASCIENCE
SCHOOL OF BIO AND CHEMICAL ENGINEERING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Accredited with Grade "A" by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119 MAY - 202

MAY - 2022



DEPARTMENT OF Bionformatics and Datascience

BONAFIED CERTIFICATE

This is to certify that this project report is the bonafied work of **Anantha Reddi Yaswanth, (40739001)** who carried out the project entitled **“Detection of Teeth details and Root canal deviation alongwith its dimension using YOLOv5; A Custom Object Detection”** under my supervision from October 2021 to April 2022.

Internal Guide

(Dr. H. Jhemmy Christy)

Head of the Department

(Dr. H. Jhemmy Christy)

Submitted for Viva voce Examination held on_____

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of Management of SATHYABAMA for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I would like to express my sincere and deep sense of gratitude to my Project Guide Dr./Mr./Ms **Dr. H. Jhemmy Christy** for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of Bioinformatics and Data science who were helpful in many ways for the completion of the project.

ABSTRACT

The object detection is a helpful process for the identification of names or objects by augmenting the data. In Dentistry, the object detection plays a role identifying the tooth details like whether the tooth is inline or in in good condition or not or the details of the tooth. In this project, We developed an project to detect the name of the tooth along with its root canal dimensions and details. In this process X-ray panoramic images will play an crucial role as they are the source for the identification. The detection of teeth from dental panoramic x-ray photographs is a critical step in the development of computerised tooth applications. we described a method that use Convolutional Neural Networks using YOLOV5 architecture to detect the names of teeth and the position of root canals in dental panoramic images. Around 1000 images are collected as dataset and augmented or annotated the data using labeling and then used the YOLOV5 architecture, one stage algorithm employed to conduct multi class object detection test for panoramic X-ray dental images. The prediction method's accuracy exceeds 90%, and it can be used as a first step in determining the name, number, and root canal position of a tooth.

KEYWORDS: Object detection,X-ray panoromicimages,YOLOv5, CNN, Labelimg

CONTENTS

S.NO	TITLE	PAGE NUMBER
(i)	Abstract	4
(ii)	List of Tables	6
(iii)	List of figures	7
1	Introduction	8
2	Objectives	11
3	Review of literature	14
4	Materials and Methods	25
5	Results and discussion	77
6	Summary and Conclusion	84
7	References	85

List of Tables

S.No	TITLE OF THE TABLES
4.10	Colab environment Configuration
4.10.1	CPU Configuration
4.13	Hyperparameter Configuration
5.1	Experimental Results

List of Figures

S.No	List of Figures
1.2.1	Facial recognition example
1.2.2	Number plate recognition process
1.2.4	Health Care- Medical Image Analysis
1.3	Objective
4.2	Block diagram of object detection
4.4	Annotation
4.6.1	GPU speed of YOLOv5 versions
5.1	Confusion matrix for the root canals
5.2	m@p curve
5.3	Precision Recall curve
5.4	Loss functions
5.7	Detected image

CHAPTER-1

INTRODUCTION

In everyday clinical practice, radiograph analysis is an important part of the diagnostic process. Teeth detection and numbering are examples of deep learning. In this project, we propose a novel solution based on deep learning technique i.e., object detection, performs this task for panoramic radiographs.

1.1 Introduction

In recent years dental biometric information has been playing an promising role for human identification as like fingerprint and DNA. Teeth will not be damaged after disaster because of their hard and durable enamel. For every task like detection of diseases, dental surgeries and operations every tooth must be localised initially with dental images. Thus detection of each tooth's details in dental images is a crucial task.

Actually, there are two types of X-ray images in dentistry which are intra oral and extra oral. The intra oral X-ray images are retrieved by placing the image receptor inside the patient's oral cavity and they contain a few teeth. The other one, extra oral X-ray images are the panoramic images that are taken by rotating the X-ray source around the head and they shows the entire mouth, including all the teeth in the upper and lower arch in a single image. Panoramic X-rays are used to have an idea about what sort of treatment should be given to the patient and diagnose tumors and cysts.

In recent years, the deep learning came into limelight with adverse applications for computer vision and speech recognition and helped to improve many other domains.

Object detection is technologically challenging and practically useful case in the field of computer vision. The goal of object detection is to find the location of an object in a given X-ray image accurately and mark the according to its class. In the present aspect the goal is to detect the name of the tooth along with its number and root canal position.

However, solving this problem is not easy as the size of the object(tooth), its orientation, its altitude and its location in the image can vary greatly. Unlike the human eye, a computer processes the X-ray images into 2 dimensions.

Object detection deals with the identification of various objects in an individual image in which success has been achieved in controlled environment but it remains unsolved when the environment is uncontrolled. For example, it is easy to train a robot to detect a thing(viz.,pencil,machine,bat) with nothing else in the image. On the other hand assume the difficulty of that robot in detecting the machine on a factory that is clustered by various other machines, tools, etc. The object recognition process in such scenario is very difficult. A lot of research is going on in the field of object recognition from two decades without having proper solutions. The research on Object detection is an multi disciplinary and often involves other aspects like image processing, Machine learning, linear algebra, topology, Probability, Optimization, etc and getting primary summary of these approaches is quite difficult and time consuming.

This approach incorporates four computer vision and ML tasks such as sliding windows to extract sub images from the image, feature extraction, SVN for the classification of objects in sub- image and Principal component analysis (PCA) for better efficiency. In this motivating problem, we have been focussed on the detection of the details of the tooth in panoramic X-ray images in particular their names and number.

1.2 APPLICATIONS

1.2.1 Facial Recognition

A deep learning facial recognition system called "DeepFace" has been developed by a team of researchers from Facebook, identify human face in a digital image effectively. Mobile companies using facial recognition technology for unlocking the various applications in their respective user's devices. Google uses its own facial recognition system in Google Photos, Automatically separates all photos based on the person in the photo.

Many components have been involved in facial recognition such as eyes, nose, mouth, and eyebrows and masks especially in Apple Inc mobiles because of the Covid-19.

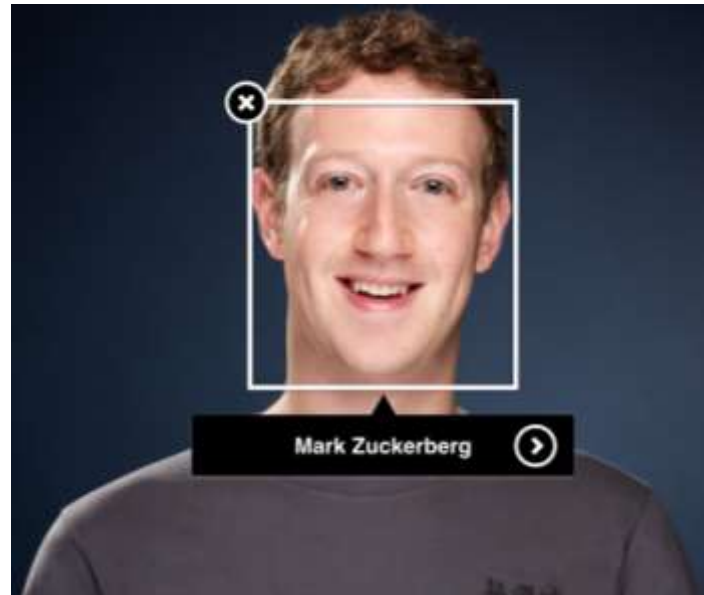


Fig 1.2.1: Facial Recognition example

1.2.2 Number Plate recognition

Automatic license plate recognition (ANPR) is a technique that uses optical character recognition in an image to read a vehicle's license plate and create vehicle position data. We can use an existing security camera, a road control camera specifically designed for this task. ANPR is used by police around the world for law enforcement purposes, such as verifying that a vehicle is registered or licensed. It is also used as a way to collect electronic tolls on toll roads and catalog traffic movements by highway authorities.

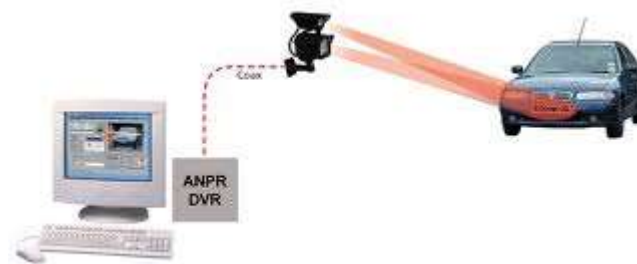


Fig 1.2.2 : Number plate recognition process

1.2.3 Industrial quality Analysis

Object detection or recognition plays an quality role in detecting the manufacturing defects incase of hardware's, damaged vegetables through visual inspection or even sorting. Automatic object counting and localisation allows to improve accuracy of the inventory incase of real time tracking.

1.2.4 Health Care- Medical Image Analysis

In medical image analysis, Object detection is an important discipline that helps detecting various tumours and diseases or infections like Pneumonia, Cardiomegaly, brain tumours, etc with help of CT scan images .

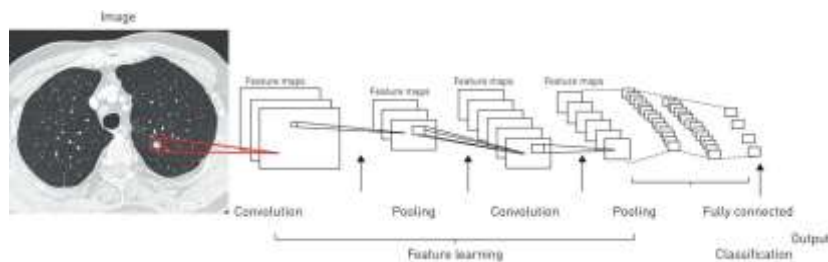


Fig 1.2.4 Health Care- Medical Image Analysis

1.3Objective

The main objective of this project is that the detection of all instances of objects from a known class such as 1_MLTM, 2_MLSM, 3_MLFM (Molars) in an X-ray image. In the case of rigid object one example is enough to detect the functioning of the process but usually several multiple training examples are needed to gain certain aspects of the class variability.

Object detection is one of the hot fields in computer vision and digital image analysis or processing. Because of the extensive use and applications of deep learning technologies, object detection algorithms have been employed rapidly. Considering the complexity and diversity of the scene(Tooth detection), this is still an challenging topic and there are many potential aspects to be improved.

The algorithms employing can be roughly classified into two categories. Object detection algorithms can be divided into two categories.

The first is an algorithm of the RCNN family based on an area proportion, whose representative network is RCNN (Girshick, Donahue, Darrell and Malik, 2014), Rapid RCNN (Girshick, 2015), SPPNet (He, Zhang, Ren & Sun, 2015), faster RCNN (Ren, He, Girshick & Sun, 2015), FPN (Lin et al., 2017) and MaskRCNN (He, Gkioxari, Dollar & Girshick, 2017). There are two-steps, that requires the algorithm to generate the object's candidate box, namely the location of the object, and then do the classification and regression candidate box.

One stage detection algorithm, which doesn't require the region proposal phase, can directly generate the category probability and the location coordinate value of the object. This representative network is the YOLO series, which includes YOLO (Redmon, Divvala, Girshick & Farhadi, 2016), YOLOv2 (Redmon & Farhadi, 2017), YOLOv3 (Redmon & Farhadi, 2018), YOLOv4 (Bochkovskiy, Wang & Liao, 2020), YOLOv5, SSD (Liu et al., 2016) and RetinaNet (Lin, Goyal, Girshick, He & Dollar, 2017).

In recent years, most object detection algorithms have high accuracy such as RCNN, Fast RCNN, Faster RCNN often does not meet the real-time requirements of the visual object detection industry due to low speed. Now, one step algorithm represented by YOLO has launched and received widespread recognition for precision speed and performance. Until September 2020, the YOLO algorithm is gone through five iterative releases and has made great strides in speed and accuracy.

There has been a lot of work on the tooth regarding the cavity condition and numbering and classification according to the general 4 types of classes namely incisors, canines, premolars and molars.

Compared with the general image classification techniques, visual object detection through Panoramic X-ray images is more in line with the dental needs, because it is often not an easy task for every practitioner to predict which teeth is this by watching X-ray images.

Based on extensive research on object detection algorithms depending on speed and accuracy, this project adopts an object detection algorithm based on deep learning, the YOLOv5 algorithm based on one stage idea and applies it to the Panoramic X-ray images dataset collected locally.

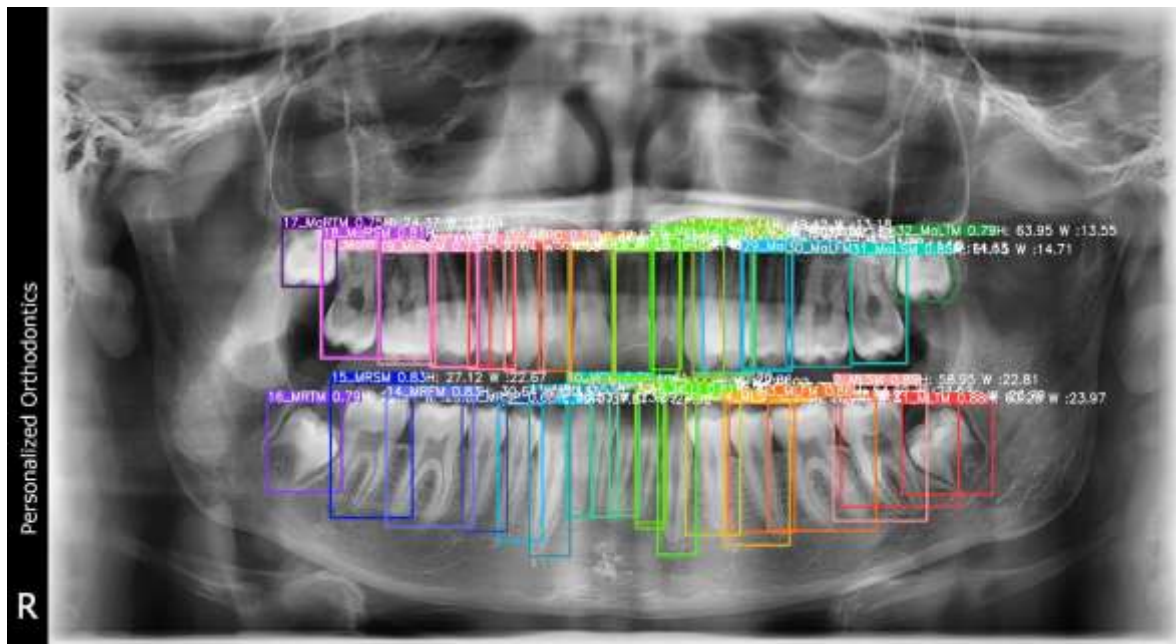


Fig 1.3 Objective

CHAPTER-2

LITERATURE SURVEY

2 Introduction

2.1.1 Tooth

The teeth are the hardest substance and one of the most important organs in the human body. Besides being essential for chewing, the teeth play an important role in speech. A normal adult has 32 teeth which have been developed by the age of 13-14 and it has four types of teeth namely Incisors (8), Canines (4), Premolars (8), and Molars (12). Tooth numbering is a notation used by dentists to denote and specify information linked with a particular tooth, allowing for easy identification, communication, and dental records maintenance in a precise manner.

In this project, the above process is done with the help of the Object Detection technique.

2.1.2 Literature survey

Many researchers contributed their precious insights in the field of dentistry using image processing tools or techniques likewise Ayse Betul Oktay detected Tooth with Convolutional Neural Networks by coloring and differentiating the teeth.

Abdolvahab Ehsani Radet. al. (2013) developed a method that has been implemented by using a level-set method for segmentation after image enhancement. Texture statistics techniques by grey-level co-occurrence matrix are used to take out multiple features of dental. Obtaining data is used in automatic dental systems such as human identification or dental diagnosis systems. Preliminary experiments demonstrate the significance of the proposed method to remove teeth from an x-ray image

Many methods are presented for isolating and segmentation and classification the teeth. Many of them are proposed as the initial step of human identification methods and they use low-level image processing techniques.

Automated dental identification has been done for human identification with intra oral images(G. Fahmy et al.,2004). The recognition of the tooth and numbering has been done by a process called pattern recognition using X-ray images(P.L. Lin, et al., 2010)

CNN has been used to classify the teeth into seven classes in cone-beam CT images and the performance has been evaluated based on deep convolutional neural networks(Y. Miki, et al., 2017)

There is a lot of research on tooth classification based on disease aspects and the research on object detection is going on but there have been relatively few previous studies on Tooth detection and classification.

2.2 Object Detection

Object recognition is a computer vision technique to recognize various classes of objects in an image/video. Object detection provides more detailed information by identifying an object in an image and specifying its location using a bounding box which is often shown in the form of a minimum bounding rectangle. Semantic segmentation is similar to object detection but provides a more detailed approximation of an object by classifying every pixel in an image as a class and then drawing a mask on that object in an image.

2.2.1 Literature review

Image classification and object detection are important research methods in computer vision. These technologies help machines understand and recognize real-time objects and their environments. In essence, object detection is also an image classification technique. In addition to classification, this technique can also identify the location of object instances from a large number of predefined categories in natural images. Object detection is one of the most basic and challenging tasks in computer vision (Zou, Shi, Guo & Ye, 2019). From the past, nearly two decades of development, the object detection algorithm of natural images can be roughly divided into the period based on traditional manual features before 2013 and the period based on deep learning after 2013.

In terms of technology development, object detection has gone through many milestones such as bounding box regression, multi-references boxes (anchors), hard example mining and focusing, and multi-scale and multi-port detection (Zou, Shi, Guo & Ye, 2019).

Early object detection algorithms are mostly based on manual features. Due to the lack of effective image feature expression methods before the birth of deep learning, people have to design more diversified detection algorithms to make up for the defects of manual feature expression ability. In 2001, Viola and Jones published a cross-era paper on CVPR, and later generations called the face detection algorithm in the paper as Viola-Jones (VJ) detector (Viola & Jones, 2001). VJ detector realizes the real-time face detection for the first time with very limited computing resources, and its speed is tens or even hundreds of times faster than the detection algorithm in the same period, which greatly promotes the commercialization process of face detection application. The idea of VJ detectors has profoundly influenced the development of the object detection field for at least 10 years. The VJ detector adopts the most traditional and conservative object detection method -- sliding window detection, which is to traverse every scale and every pixel position in the image and judges whether the current window is a face target one by one. The cost of this approach is huge computation (Viola, & Jones, 2004).

2.2.3 Panoramic X-ray Object Detection based on Deep Learning

In the year 2012, a variety of artificial neural network design methods emerged to replace the traditional neural network.

After CNN achieved great success in the 2012 Imagenet classification task (Hinton, Krizhevsky & Sutskever, 2012), Girshick and others seized the opportunity to break the deadlock and first proposed the Region-Based Convolutional Neural Networks (R-CNN) in 2014. Since then, the field of object detection has developed at an unprecedented speed (Girshick, Donahue, Darrell & Malik, 2014). With the deepening of CNN layers, the network's abstract ability, anti translation ability and anti-scale change ability become stronger and stronger.

For the image classification task, this is beneficial, but for the detection task, it brings another problem: the accurate position of the object bounding box is becoming more

and more difficult to obtain. Therefore, if we want the detection algorithm to obtain stronger translation invariance and scale invariance, we must sacrifice the sensitivity of features in the position and scale changes of the object bounding box to a certain extent. On the contrary, if we want to get more accurate bounding box localization results, we must make some compromises on translation invariance and scale invariance. (Dai & R-FCN, 2016). This forced people to give up the detection scheme based on feature maps and sliding windows, turning their attention to finding a more accurate object candidate detection algorithm.

At present, deep learning methods in the field of object detection are mainly divided into two development directions: two-stage based and one-stage based. The former refers to the algorithm generating a series of candidate frames as samples, and then classifying the samples through CNN; the latter does not need to generate candidate frames and directly transforms the problem of object border location into a regression problem. The performance of the two methods is also different. The former is superior in detection accuracy and positioning accuracy, while the latter is superior in algorithm speed. The development of these two ideas and their representative algorithms are described below. Two Stages Algorithm In the early development process of deep learning technology, research was mainly carried out around classification problems. This is because the unique structural output of neural networks combines probability statistics and classification problems to provide an intuitive and easy way of thinking. Although many researchers are also working on integrating other areas such as object detection and deep learning, they have not achieved much success. This situation was not solved until the emergence of the R-CNN algorithm. R-CNN R-CNN algorithm was proposed for the first time in 2014, and its algorithm structure also became the classic structure of subsequent two-stage (Girshick, Donahue, Darrell & Malik, 2014).The R-CNN algorithm evaluates the feature similarity of adjacent image subblocks by Selective Search algorithm. By scoring the similar image regions after the merging, the candidate box of the region of interest is selected as the sample and input into the CNN structure.

The corresponding feature vector is formed by the positive and negative sample features composed of network learning candidate box and calibration box. Then support vector machine design classifier to classify the feature vectors. Finally, the localization of object detection is achieved by border regression operation. Although

R-CNN algorithm has achieved 50% performance improvement compared with the traditional object detection algorithm, it also has defects: the positive and negative sample candidate areas of the training network are generated by the traditional algorithm, which limits the speed of the algorithm. In addition, CNN does feature extraction for each generated candidate region respectively, which will lead to a large number of repeated operations and further restrict the performance of the algorithm. SPP-Net In 2015, SPP-NET algorithm was proposed for the repetitive operation of convolutional neural networks (He, Zhang, Ren & Sun, 2015).

This algorithm adds a space pyramid pooling structure between the convolutive layer and the full connection layer, and 15 optimizes the method of R-CNN algorithm to cut and scale each candidate region before the input of CNN to make the image sub-block size consistent. This spatial pyramid pooling structure effectively avoids the problems of image object clipping and shape distortion caused by image region clipping and scale operation in R-CNN algorithm.

More importantly, it solves the problem of extracting repeated features from images by CNN, thus greatly improving the speed of generating candidate boxes and saving calculation costs. However, just like R-CNN algorithm, when the image size of training data is inconsistent, the ROI of the candidate box will be greatly enhanced, and BP cannot be used to efficiently update the weight. Fast R-CNN Girshick proposed a Fast R-CNN algorithm in 2015 to improve SPP-NET algorithm (Girshick, 2015).

This algorithm designs a pooling layer structure of ROI pooling, which effectively solves the operation that R-CNN algorithm must crop and scale the image areas to the same size. The idea of multi-task loss function is proposed. Gradients can be transmitted directly through the ROI Pooling layer.

But it still does not get rid of the problem of generating positive and negative sample candidate box in the selective search algorithm. Faster R-CNN To solve the defects of Fast R-CNN algorithm, Faster R-CNN algorithm was proposed in 2015 (Ren, He, Girshick & Sun, 2015).

Region Proposal Network (RPN) that assists in generating samples was designed. Its advantage is that the whole network process can share the feature information extracted by CNN, which saves the calculation cost, solves the problem that Fast R-CNN algorithm is slow in generating positive and negative sample candidate boxes, and avoids the decrease of algorithm accuracy caused by too much extraction of candidate boxes.

RPN network can generate multi-size candidate boxes in the convolution feature map of fixed size, resulting in the inconsistency between the size of the variable object and the fixed receptive field. This is a shortcoming of the Faster R-CNN algorithm.

MR-CNN The MR-CNN algorithm was proposed by Gidaris and Komodakis in 2015. The algorithm decomposed the detection problem into classification and location problem (Gidaris & Komodakis, 2015). The classification problem is composed of Multi-Region CNN Model and Semantic Segmentation-Aware CNN Model. The candidate box of the former is obtained by Selective Search. For each sample area, 10 areas are extracted separately and then spliced, which forces the network to capture different aspects of the object. After the entire image is input into the Activation Maps module, it is output through a series of convolution operation feature map. In this part of the network, various classic network structures can be used, such as AlexNet, VGG16, etc. As for positioning, the algorithm adopts three sample Bounding correction methods for precise positioning, namely, Bbox Regression, Iterative localization and Bounding box voting (Liu, Du, Tian & Wen, 2019). HyperNet HyperNet algorithm is a variant algorithm with excellent performance proposed by Tsinghua University in 2016. The main improvement of HyperNet lies in the collection of multi-layer Feature maps to obtain multi-scale Hyper Feature, which has the advantages of multi-level abstraction, appropriate resolution and calculation timeliness. Compared with Faster R-CNN, HyperNet is better at dealing with small objects.

It has more advantages in high IOU (Kong, Yao, Chen & Sun, 2016). CRAFT The first stage of R-CNN series algorithm is to generate object proposals, and the second stage is to classify the object proposals. In 2016, the craft algorithm proposed by the Institute of automation of Chinese Academy of Sciences improved

the two stages of fast R-CNN (Yang, Yan, Lei & Li, 2016). For the generated object proposals stage, a binary fast R-CNN classifier is added after the RPN to further filter the RPN generated proposals, leaving some high-quality proposals; for the object proposals classification in the second 17 stage, the original classifier is cascaded with N binary classifiers (excluding background class) for more precise object detection. R-FCN With the emergence of full convolutional network, R-FCN algorithm was proposed in 2016 (Dai, Li, He & Sun, 2016). R-FCN is improved on the framework of Faster R-CNN. This algorithm proposes the idea of Position-Sensitive Score Maps to solve the position sensitivity problem of object detection. It makes full use of full convolutional network to reduce the total computation, which enables feature sharing to be realized in the whole network. However, it does not take into account the global information and semantic information of the regional proposal. MS-CNN MS-CNN was proposed by Cai in 2016.

The basic idea is to propose a multi-scale convolutional neural network. In view of the different advantages of feature maps at different levels, for example, feature maps at lower levels are relatively good at detecting small objects due to their small sensing fields. However, the higher level can ignore a lot of noise because it takes into account the information of large perception field, so it is more accurate to detect large objects. Considering this feature, the author designed detectors of different scales for different layers on the feature map for the first time.

At the same time, the deconvolution layer of feature map is used to replace the upsampling of input image, which improves the speed and accuracy (Cai, Fan, Feris & Vasconcelos, 2016). PVANet At the end of 2016, the Intel Graphics technology team proposed a lightweight network PVANet, achieving remarkable results (Kim, Hong, Roh, Cheon & Park, 2016). Network is based on basic design principles: more layers with less channels. In addition, the author uses “c. ReLU” and Inception structure to reduce the redundancy of the network.

FPN In 2017, Lin proposed the FPN algorithm, which uses feature maps of different layers to predict objects of different sizes (Ghiasi, Lin & Le, 2019).

Most of the original object detection algorithms only use the deep features for prediction, and the low-level feature semantic information is less, but the object position information is accurate. The high-level feature semantic information is rich,

but the object location is rough. In addition, although some algorithms adopt multi-scale feature fusion, they generally adopt the fused features for prediction, while the difference of FPN algorithm lies in that the prediction is carried out independently at different feature layers, and the deep features are used for up-sampling and low-level features for fusion.

Mask R-CNN algorithm was proposed by Kaiming He, which achieved excellent recognition effect (He, Gkioxari, Dollar & Girshick, 2017). Mask R-CNN is an extended form of Faster R-CNN. The original idea of Mask R - CNN is to add another branch to Faster R-CNN to add an output, namely object Mask, which means that the original two tasks (classification + regression) will be changed to three tasks (classification + regression + segmentation).

Combining a binary Mask with the classification and bounding box from Faster R-CNN produces a surprisingly precise image separation effect. Fast R-CNN The Fast R-CNN algorithm was proposed by Carnegie Mellon University in 2017, which introduced Generative Adversarial Networks (GAN) into object detection problems, mainly focusing on solving occlusion and deformation problems (Wang, Shrivastava & Gupta, 2017).

The author designed two GAN: ASDN and ASTN, corresponding to occlusion and deformation respectively. ASDN and ASTN provide two kinds of different changes. By combining these two kinds of deformation (ASDN output is the input of ASTN), the detector can be trained more robust.

CoupleNet In view of the problem that R-FCN algorithm fails to consider the global information and semantic information of regional proposal.

The Institute of Automation of the Chinese Academy of Sciences proposed CoupleNet algorithm in 2017, which introduced the global and semantic information of proposal on the basis of the original R-FCN, and improved the detection accuracy by combining local, global and semantic information (Zhu et al., 2017).

MegDet Research on CNN-based object detection has been making continuous progress. From RCNN to Fast/Faster R-CNN and then Mask R-CNN, the main

improvement points are all in the new network architecture, new paradigm or new loss function design. However, the key factor of mini-batch size in the training has not been fully studied. This also makes it impossible for the existing deep learning framework to train the object detection model of large Mini-batch, while the object detection algorithm of small Mini-batch often introduces unstable gradients, inaccurate BN layer statistics, imbalance of positive and negative sample proportion and excessively long training time. Therefore, In December 2017, Peng proposed MegDet, a large Mini-batch object detection algorithm (Peng et al.,2018).

The MegDet algorithm can use a training network much larger than the previous Mini-batch size (for example, increasing from 16 to 256), which can also efficiently utilize the combined training of multiple GPUs (up to 128 were used in the experiment of the paper) and greatly reduce the training time, such as from 33 hours to just 4 hours. At the same time, the algorithm can achieve higher accuracy. Light-Head R-CNN In 2017, light-head RCNN was proposed, mainly discussing how R-CNN balances accuracy and speed in object detection (Li et al., 2017).

Light Head R-CNN is the combination of Faster R-CNN and R-FCN. The detector can achieve the optimal tradeoff between speed and accuracy. If it is based on the ResNet101 network, it performs better than Mask RCNN and RetinaNet.

One Stage Algorithm In view of the existence of RPN structure, the two-stage method represented by R-CNN 20 algorithm, although the detection accuracy is getting higher and higher, its speed encounters a bottleneck, which makes it difficult to meet the real-time requirements of some scenes. Therefore, a one-stage object detection algorithm based on regression method appears.

The one-stage algorithm can give the category and location information directly through the trunk network without using the RPN network. This algorithm is faster. However, the accuracy is slightly lower than the two-stage object detection algorithm.

OverFeat algorithm was proposed by Yann Lecun's team at New York University in 2013. The algorithm uses sliding Windows and rule blocks to generate candidate boxes, and then uses multi-scale sliding Windows to increase detection results to solve the problem of complex shapes and different sizes of image objects. Finally,

it uses convolutional neural network and regression model to categorize and locate objects (Sermanet et al., 2013). OverFeat algorithm makes full use of the feature extraction function of CNN. The features extracted in the classification process are also used for various tasks such as positioning and detection. Different tasks can be realized only by changing the last few layers of the network, instead of training the parameters of the whole network from the beginning.

This algorithm solved the three computer vision tasks of classification, positioning and detection together for the first time, and won the champion of ILSVRC2013 Task 3 (classification + positioning) in the same year, but it was soon replaced by R-CNN algorithm in the same period. G-CNN In 2016, Najibi et al. from the University of Maryland proposed the G-CNN algorithm. G-CNN is a kind of object detection algorithm independent of proposal algorithms. The G-CNN algorithm models the object detection problem as finding a path from a fixed grid to a dense object box. This strategy eliminates the generation stage of candidate boxes and reduces the number of candidate boxes that need to be processed, making object 21 detection faster (Najibi, Rastegari & Davis, 2016).

YOLO In 2015, Yolo algorithm proposed by Joseph Redmon of Washington University inherits the idea of regression in overfeat algorithm, and its speed can reach 45 frames per second. YOLO algorithm is based on the global information of the image to make predictions. Its overall structure is simple. The input image is reconstructed to a fixed size of 448×448 pixels, and the image is divided into 7×7 grid area.

The feature training is extracted by CNN to directly predict the border coordinates in each grid and the confidence of each category. P-ReLU activation function is adopted during the training. However, it also has problems such as inaccurate positioning and unsatisfactory recall rate, poor detection effect for very close objects and very small objects, and relatively weak generalization ability (Redmon, Divvala, Girshick & Farhadi, 2016).

After improvement, YOLOv2 (Redmon & Farhadi, 2017) and YOLOv3 (Redmon & Farhadi, 2018) algorithms were proposed on CVPR 2017 and were nominated for the best paper, focusing on solving the poor recall rate and positioning accuracy. It uses Darknet19 as a feature extraction network and adds Batch Normalization as

pretreatment. The original YOLO uses the full connection layer to directly predict the coordinates of the bounding box, while YOLOv2 refers to the idea of Faster R-CNN, introduces anchor mechanism, and calculates a better anchor template in the training set through k-means clustering.

The operation of Anchor Boxes is used in the convolutional layer to improve the prediction of the candidate box. Meanwhile, the positioning method with strong constraints is adopted to greatly improve the recall rate of the algorithm. Combining with the image fine grain feature, the shallow feature and the deep feature are connected, which is helpful to the detection of the small size target. In 2020, Alexey Bochkovskiy released YOLO V4 (Bochkovskiy, Wang & Liao, H. Y. M. 2020).

YOLO V4 is a significant update to the YOLO family, with an increase in 22 average accuracy (AP) and frame rate accuracy (FPS) of 10% and 12%, respectively, on COCO data sets. While computer vision practitioners are working on YOLO V4, On June 25, 2020, Ultralytics released YOLOV5 on Github, which has the same performance as YOLO V4 and is faster at reasoning. YOLO V5 does perform very well in object detection, especially the reasoning speed of YOLO V5s model 140FPS is amazing. The authors of YOLO V5 have yet to publish a paper.

SSD Aiming at the poor positioning accuracy of the initial YOLO v1 algorithm, Liu et al proposed SSD algorithm in 2016 and combined the regression idea of YOLO with the Anchor box mechanism of Faster R-CNN (Liu et al., 2016).

It adopts the border regression of local feature map of multi-scale region in every position of the whole image, which keeps the fast speed of YOLO algorithm and ensures that the border positioning effect is similar to that of Faster R-CNN. However, due to its use of multi-level feature classification, it is difficult to detect small objects, and the receptive field of the last convolutional layer becomes large, making the features of small objects not obvious.

Chapter – 3

AIM AND SCOPE OF THE PRESENT INVESTIGATION

The main aim of this study is to understand the names and details of the teeth and to predict the name of the each tooth, numbering them according to their number system in the dentistry and position of the root canal along with their dimensions from an Panoramic X-ray image using the CNN based YOLOv5 architecture. To get to this aim we employed the following objectives,

- Collecting Panoramic dental X-ray images from various sources.
- Thereby annotating them with the help of Labelimg.

- From the literature review analysis, the dataset must be annotated initially , trained with python- deep learning methods, splitting the dataset into train and validation dataset respectively and moves to the detection with YOLOv5 architecture.
- Then calculating the results of the model such as precision, recall, Map@_0.5 accuracy values
- The study is to be a first step in understanding the position of the tooth thereby understanding the name of the tooth, position of root canal and determining the dimensions of each teeth that are presented in an dental X-ray image.

Chapter-4

MATERIALS AND METHODS

4.1 Materials used

4.1.1 Hardware Specifications

PROCESSOR: Intel core i5 8th gen octacore

GPU : Nvidia geforce MX150

RAM : 8GB

HARD DISK : 256GB SSD, 1TB HDD

4.1.2 Software Applications

PLATFORM: Windows 11

Python : Being the most advanced and prominent programming language for deep learning techniques, I have used the python 3.7 version software for programming.

Pytorch: PyTorch is an optimized Deep Learning tensor library based on Python and Torch and is mainly used for **applications using GPUs and CPUs**. PyTorch is favored over other Deep Learning frameworks like TensorFlow and Keras since it uses dynamic computation graphs and is completely Pythonic. I have used PyTorch to implement an object detector based on YOLO v5, one of the faster object detection algorithms.

Tensorboard: TensorBoard is a tool for providing the measurements and visualizations needed during the machine learning and deep learning workflow. It enables tracking experiment metrics like loss and accuracy, visualizing the model graph, projecting embeddings to a lower dimensional space, and much more.

OpenCV: OpenCV is a great tool for image processing and performing computer vision tasks. It is an open-source library that can be used to perform tasks like face detection, objection tracking, landmark detection, and much more. It supports multiple languages including python, java C++.

Cuda: UDA (or Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) that allows software to use certain types of graphics processing unit (GPU) for general purpose processing, an approach called general-purpose computing on GPUs (GPGPU).

YOLO: You Only Look Once, Version is a real-time object detection algorithm that identifies specific objects in videos, live feeds, or images. YOLO uses features learned by a deep convolutional neural network to detect an object.

4.2 METHODOLOGY

This section outlines the data collection and modification process inorder to train the dental X-ray images identification. The block diagram in below figure illustrates the

work flow of the methodology followed for this project. The block diagram consists of the initial data preparation and the rest of the algorithmic process for the prediction.

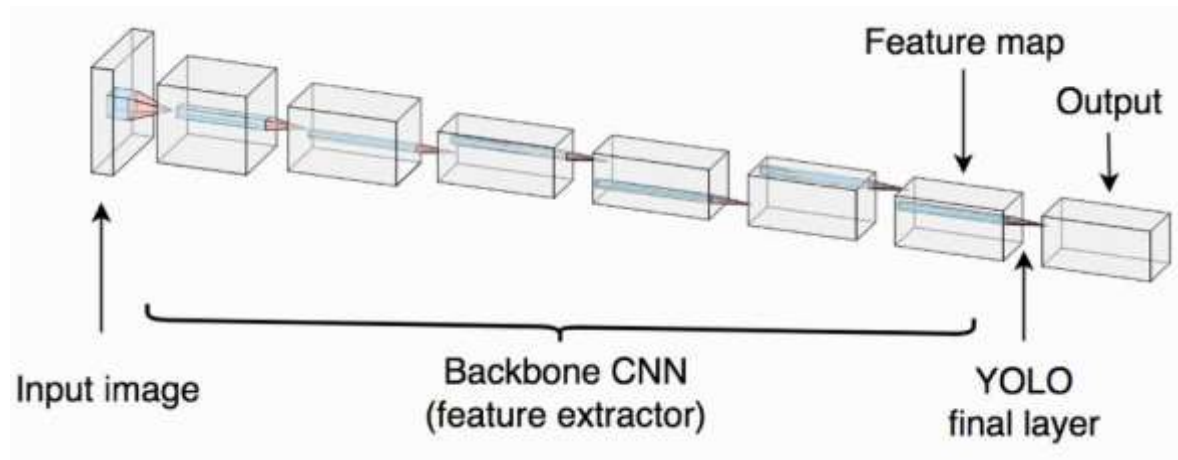


Fig 4.2 :Block diagram

4.2.1Data Collection

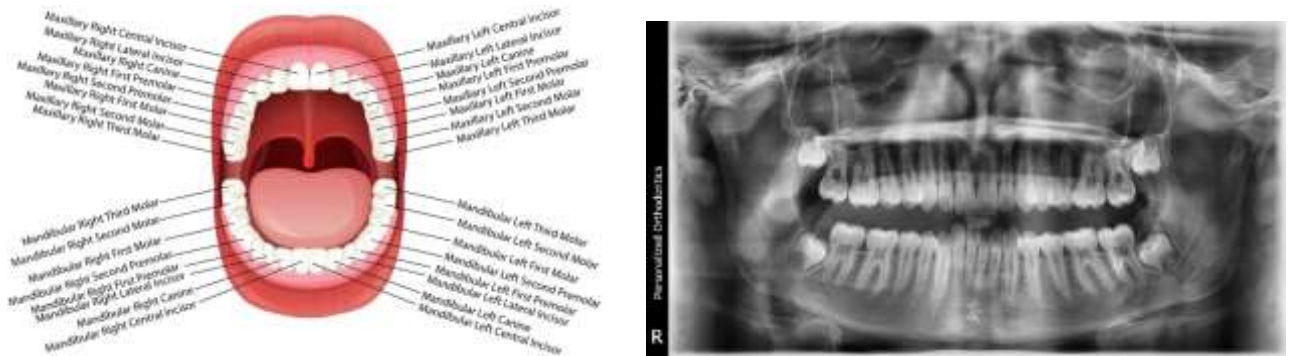
Generally, we are in need of many dental panoramic X-ray images to achieve best results. Owing to the availability of the data, we did the process with around 1000 images of dataset containing various persons dental images.

4.3 Panoramic dental X-ray Dataset preparation

The basic dataset contains different persons panoramic dental X-ray images collected from two different machines, each image having about 32 types of teeth or classes namely 1_MLTM, 2_MLSM, 3_MLFM, 4_MLSP, 5_MLFP, 6_MLLI, 7_MLC, 8_MLCI, 9_MRCI, 10_MRC, 11_MRLI, 12_MRFP, 13_MRSP, 14_MRFM, 15_MRSM, 16_MRTM, 17_MaRTM, 18_MaRSM, 19_MaRFM, 20_MaRSP,

21_MaRFP, 22_MaRC, 23_MaRLI, 24_MaRCI, 25_MaLCI, 26_MaLLI, 27_MaLC, 28_MaLFP, 29_MaLSP, 30_MaLFM, 31_MaLSM, 32_MaLTM.

4.4 Dataset Annotation



We use labelme, a data annotation software, to label the images. The method of labelling is to label each class by manually selecting a rectangular box. In the process of labelling , different types teeth abbreviations are represented by the combination of their number followed by the respective tooth code. '1_MLTM' stands for Mandibular Left Third Molar, '7_MLC' stands for Mandibular Left Canine, '24_MaRCI' stands for Maxillary Right Central Incisor.

The annotated data is saved as a txt format file, the specific contents include, class serial number, horizontal coordinate of the upper left corner of the rectangular box, width of the rectangular box, length of the rectangular box and each value is the ratio of the edge of the rectangle to the entire size of the image.



Fig 4.4 : annotation

4.5 Data Augmentation

Given the relatively small number of overall dataset, data augmentation is necessary. The specific operations on a dataset includes flipping, zoom in and out, clipping and combining. The 330 images collected were expanded 3 times to 990 images. Then the data is divided into training set and validation set according to the ratio of 8:2. The final training set has 776 images and 194 validation sets.

4.6 YOLO

The YOLO model was first described by J. Redmon in the paper titled “You Only Look Once: Unified, Real- Time Object Detection” in 2015.

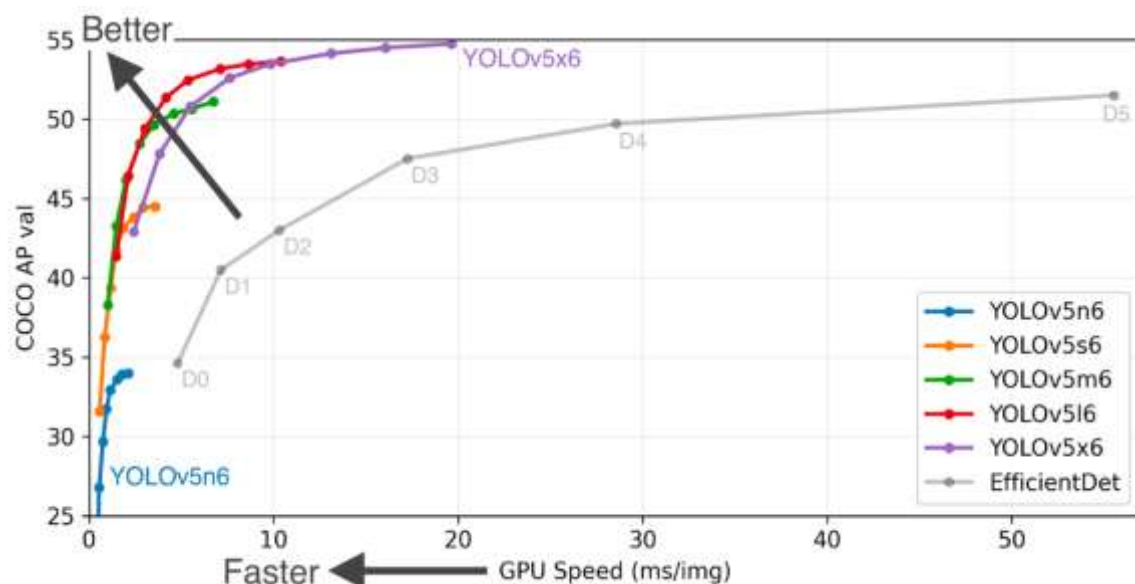
YOLO, a representative of the one-stage algorithms was considered in this project to detect tooth details and analyzed the performance. At First, we described the working principle of YOLO then introduces the specific process of environment implementation, data set preparation and model training, and finally explains various performances in the area of object detection, criteria for evaluating the performance of the proposed model.

4.6.1 YOLO Architecture

Yolo was used as an object detection tool with the help of research work published under IEEE, the pre trained yolo is substantially better in detecting the most of the common objects like vehicles including Teeth in images.

Since its development, various versions of Yolo algorithms have been released with the most recent one being Yolo-v5 and its sub versions. Yolo-v2 was the most promising version of the first release but Yolo-v3 was the final version released by the original author, later introduced with the capability of changing the processing speed by changing the model size without the need for any retraining.

Yolo-v4 was later introduced by the deep learning community as an enhanced version of Yolo-v3 and was the final darknet based version. Thereafter, Yolo-v5 was introduced and was claimed to be better than other versions released so far. There has been several controversies and failing to draw a fine distinct line between these versions and hence it was decided to fully explore Yolo-v5 and hypertune it, thus carrying out most of the work on fitting it to our dataset and then use the same parameterizations along with the last two stages of Multiple Identities dataset on Yolo-v5 to check the results and performance.



• Fig 4.6.1 : GPU speed of yolov5 versions

4.7 Working principle of YOLOv5

In the official code of YOLOv5, four versions of object detection algorithms are given, namely YOLOv5s, YOLOv5m, YOLOv5l and YOLOv5x. The YOLOv5s model has been employed in this project. YOLOv5s network is the algorithm with the minimum depth and the minimum width of feature map in YOLOv5 series. The YOLOv5s is the fastest, but the AP is also the least accurate. However, this model is also a good choice if the detection is focused on larger targets and less complex scenarios, which are more speed oriented. The other three are continuously improved, but the speed will become slower and slower. YOLO Model operates at 45 frames per second upto 155 frames per second. The model works by splitting the input image into a grid that is having responsible cells for bounding box prediction.

The YOLO Network consists of three main parts:

4.7.1 Backbone

The function of backbone is to aggregate and combine different annotated images to form a convolutional neural network of image features. The backbone part of YOLOv5s adopts ResNet- "cross stage partial network". ResNet solves the problem of repeated annotated information in other large CNN framework backbone. The change of annotation is integrated into the feature map from the beginning to the end, so the parameters and FLOPS values of the model are reduced, which not only ensures the reasoning speed and accuracy, but also reduces the model size. YOLOv5s network as an example takes two structures for an image, it is "CSP1_X" structure applied to the Backbone, while another "CSP2_X" structure is applied to the Neck to strengthen the ability of network characteristic fusion

4.7.2 Neck

Neck is a series of network layers that mix and combine image features and transfer them to the prediction layer, also known as Head. The "Neck" part of YOLOv5s adopts PANET(Wang, Liew, Zou, Zhou & Feng, 2019). Neck is mainly used to generate feature pyramids.

The feature pyramid can enhance the detection of objects with different scales, so it can recognize the same object with different sizes and scales. In the research of YOLOv5, PANET is considered to be the most suitable feature fusion network for YOLO (Bochkovskiy, Wang & Liao, H. Y. M. 2020).

4.7.3 Head

The Head of the model is mainly used for final inspection. It applies anchor box to feature map and generates final output vector with class probability, object score and bounding box. These heads with different scaling scales are used to detect objects of different sizes. Each head has a total of (32 classes + 1 probability + 1 28 coordinates) * 3 anchor frames, a total of 255 channels.

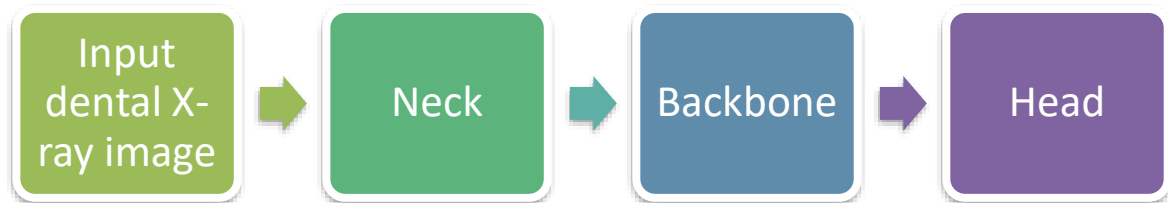


Fig 4.7 network

4.8 COCO Weights

With the limited available training data constraint, transfer learning was used by adapting the weights of a YOLOv5 model trained on COCO128 dataset for, which is a huge corpus of images, containing numerous dental images.

The ResNet backbone weights were initialized to the ones pre-trained on COCO. Initial layers of the backbone were frozen because the backbone has already learnt how to extract the low-level features very well. Fine-tuning them would improve the accuracy of the feature maps hence that of the entire model. The element of transfer learning helps model generalize on the global features enabling these to perform accurate predictions on the images sets not included

In the training set. Although Yolo does not come with a backbone provided its single-shot nature, COCO weights were also used for Yolo for enhanced performance without having to spend additional resources.

4.9 Yolo Limitations

While Yolo seems to suffice the task, there are certain limitations. One such limitation that one needs to be know is called multiple objects within a grid box. It is possible that a grid cell would contain the center of bounding box of multiple objects with similar shapes in an image. Yolo atmost supports only 3 bounding boxes per grid cell. Having more than three objects in the same grid cell would not be handled by Yolo, in such a case because there are only 3 anchor boxes per grid cell.

Just like when objects are put closed together as seen above, another limitation of Yolo in this special scenario revolves around the execution of a non-max suppression algorithm because it could potentially remove some bounding boxes whose IoU might appear greater than threshold due to their position despite these being valid distinct objects.

4.10 Environment developed

Since the model training and validation need a lot of computing and computing power, I have used Google colab platform for training and validation. Colab is a free deep learning cloud platform provided by Google Based on Jupiter notebook. It provides a free Tesla P100 GPU for deep learning research. The colab environ ment is as follows

NAME	CONFIGURATION
GPU	Tesla P100
Yolov5 framework	Pytorch
Language	Python 3.7
Operation platform	Colab(Linux)

Table 4.10 : Colab environment Configuration

The computing power of dataset preparation and annotation is small, so I have used local environment to make data set faster and more convenient. After that, the data set is put into the colab platform for training and validation.

In the local windows environment, I have used Lableme software to label the collected images. The specific configuration and environment are shown below

Table 4.10.1 CPU Configuration

NAME	CONFIGURATION
CPU	Intel(R) Core™ i5-8100U
Annotation tool	Labelme v4.5.6
Language	Python v3.7
Operation system	Windows 11 pro

4.11 Implementation

After the annotation of the dataset, the actual object detection process has been implemented.

4.11.1 Source codes

4.11.2 Training Pipeline

The model used in this project `ssd_inception_v2_coco` model, since it provides a relatively good trade-off between performance and speed

*.Config file

```
import torch
```

```
def _create(name, pretrained=True, channels=3, classes=80, autoshape=True, verbose=True,
device=None):
```

```
    from pathlib import Path
```

```
    from models.common import AutoShape, DetectMultiBackend
```

```
    from models.yolo import Model
```

```
    from utils.downloads import attempt_download
```

```
    from utils.general import LOGGER, check_requirements, intersect_dicts, logging
```

```
    from utils.torch_utils import select_device
```

```
    if not verbose:
```

```
        LOGGER.setLevel(logging.WARNING)
```

```
    check_requirements(exclude=('tensorboard', 'thop', 'opencv-python'))
```

```
    name = Path(name)
```

```
    path = name.with_suffix('.pt') if name.suffix == "" else name # checkpoint path
```

```
    try:
```

```
        device = select_device(('0' if torch.cuda.is_available() else 'cpu') if device is None else device)
    except:
```

```

if pretrained and channels == 3 and classes == 80:
    model = DetectMultiBackend(path, device=device) # download/load FP32 model
    # model = models.experimental.attempt_load(path, map_location=device) # download
/load FP32 model
else:
    cfg = list((Path(__file__).parent / 'models').rglob(f'{path.stem}.yaml'))[0] # model.yaml
path
    model = Model(cfg, channels, classes) # create model
    if pretrained:
        ckpt = torch.load(attempt_download(path), map_location=device) # load
        csd = ckpt['model'].float().state_dict() # checkpoint state_dict as FP32
        csd = intersect_dicts(csd, model.state_dict(), exclude=['anchors']) # intersect
        model.load_state_dict(csd, strict=False) # load
        if len(ckpt['model'].names) == classes:
            model.names = ckpt['model'].names # set class names attribute
    if autoshape:
        model = AutoShape(model) # for file/URI/PIL/cv2/np inputs and NMS
    return model.to(device)

except Exception as e:
    help_url = 'https://github.com/ultralytics/yolov5/issues/36'
    s = f'{e}. Cache may be out of date, try `force_reload=True` or see {help_url} for help.'
    raise Exception(s) from e

def custom(path='path/to/model.pt', autoshape=True, verbose=True, device=None):
    # YOLOv5 custom or local model
    return _create(path, autoshape=autoshape, verbose=verbose, device=device)

def yolov5n(pretrained=True, channels=3, classes=80, autoshape=True, verbose=True, device=None):
    # YOLOv5-nano model https://github.com/ultralytics/yolov5
    return _create('yolov5n', pretrained, channels, classes, autoshape, verbose, device)

def yolov5s(pretrained=True, channels=3, classes=80, autoshape=True, verbose=True, device=None):
    # YOLOv5-small model https://github.com/ultralytics/yolov5
    return _create('yolov5s', pretrained, channels, classes, autoshape, verbose, device)

def yolov5m(pretrained=True, channels=3, classes=80, autoshape=True, verbose=True, device=None):
    # YOLOv5-medium model https://github.com/ultralytics/yolov5
    return _create('yolov5m', pretrained, channels, classes, autoshape, verbose, device)

```

```

def yolov5l(pretrained=True, channels=3, classes=80, autoshape=True, verbose=True, device=None):
    # YOLOv5-large model https://github.com/ultralytics/yolov5
    return _create('yolov5l', pretrained, channels, classes, autoshape, verbose, device)

def yolov5x(pretrained=True, channels=3, classes=80, autoshape=True, verbose=True, device=None):
    # YOLOv5-xlarge model https://github.com/ultralytics/yolov5
    return _create('yolov5x', pretrained, channels, classes, autoshape, verbose, device)

def yolov5n6(pretrained=True, channels=3, classes=80, autoshape=True, verbose=True, device=None):
    # YOLOv5-nano-P6 model https://github.com/ultralytics/yolov5
    return _create('yolov5n6', pretrained, channels, classes, autoshape, verbose, device)

def yolov5s6(pretrained=True, channels=3, classes=80, autoshape=True, verbose=True, device=None):
    # YOLOv5-small-P6 model https://github.com/ultralytics/yolov5
    return _create('yolov5s6', pretrained, channels, classes, autoshape, verbose, device)

def yolov5m6(pretrained=True, channels=3, classes=80, autoshape=True, verbose=True, device=None):
    # YOLOv5-medium-P6 model https://github.com/ultralytics/yolov5
    return _create('yolov5m6', pretrained, channels, classes, autoshape, verbose, device)

def yolov5l6(pretrained=True, channels=3, classes=80, autoshape=True, verbose=True, device=None):
    # YOLOv5-large-P6 model https://github.com/ultralytics/yolov5
    return _create('yolov5l6', pretrained, channels, classes, autoshape, verbose, device)

def yolov5x6(pretrained=True, channels=3, classes=80, autoshape=True, verbose=True, device=None):
    # YOLOv5-xlarge-P6 model https://github.com/ultralytics/yolov5
    return _create('yolov5x6', pretrained, channels, classes, autoshape, verbose, device)

if __name__ == '__main__':
    model = _create(name='yolov5s', pretrained=True, channels=3, classes=80, autoshape=True, verbose=True) # pretrained
    # model = custom(path='path/to/model.pt') # custom

    # Verify inference
    from pathlib import Path

```

```

import cv2
import numpy as np
from PIL import Image

imgs = ['data/images/zidane.jpg', # filename
        Path('data/images/zidane.jpg'), # Path
        'https://ultralytics.com/images/zidane.jpg', # URI
        cv2.imread('data/images/bus.jpg')[::-1, ::-1], # OpenCV
        Image.open('data/images/bus.jpg'), # PIL
        np.zeros((320, 640, 3))] # numpy

results = model(imgs, size=320) # batched inference
results.print()
results.save()

```

4.11.2 Training the model

Train.py

```

import argparse
import math
import os
import random
import sys
import time
from copy import deepcopy
from datetime import datetime
from pathlib import Path

import numpy as np
import torch
import torch.distributed as dist
import torch.nn as nn
import yaml
from torch.cuda import amp
from torch.nn.parallel import DistributedDataParallel as DDP
from torch.optim import SGD, Adam, AdamW, lr_scheduler
from tqdm import tqdm

FILE = Path(__file__).resolve()
ROOT = FILE.parents[0] # YOLOv5 root directory
if str(ROOT) not in sys.path:
    sys.path.append(str(ROOT)) # add ROOT to PATH
ROOT = Path(os.path.relpath(ROOT, Path.cwd())) # relative

import val # for end-of-epoch mAP
from models.experimental import attempt_load
from models.yolo import Model

```

```

from utils.autoanchor import check_anchors
from utils.autobatch import check_train_batch_size
from utils.callbacks import Callbacks
from utils.datasets import create_dataloader
from utils.downloads import attempt_download
from utils.general import (LOGGER, check_dataset, check_file, check_git_status, check_img_
size, check_requirements,
                           check_suffix, check_yaml, colorstr, get_latest_run, increment_path, init_seed
s,
                           intersect_dicts, labels_to_class_weights, labels_to_image_weights, methods,
one_cycle,
                           print_args, print_mutation, strip_optimizer)
from utils.loggers import Loggers
from utils.loggers.wandb.wandb_utils import check_wandb_resume
from utils.loss import ComputeLoss
from utils.metrics import fitness
from utils.plots import plot_evolve, plot_labels
from utils.torch_utils import EarlyStopping, ModelEMA, de_parallel, select_device, torch_distri
buted_zero_first

LOCAL_RANK = int(os.getenv('LOCAL_RANK', -
1)) # https://pytorch.org/docs/stable/elastic/run.html
RANK = int(os.getenv('RANK', -1))
WORLD_SIZE = int(os.getenv('WORLD_SIZE', 1))

def train(hyp, # path/to/hyp.yaml or hyp dictionary
          opt,
          device,
          callbacks
          ):
    save_dir, epochs, batch_size, weights, single_cls, evolve, data, cfg, resume, noval, nosave,
workers, freeze = \
        Path(opt.save_dir), opt.epochs, opt.batch_size, opt.weights, opt.single_cls, opt.evolve, o
pt.data, opt.cfg, \
        opt.resume, opt.noval, opt.nosave, opt.workers, opt.freeze

    # Directories
    w = save_dir / 'weights' # weights dir
    (w.parent if evolve else w).mkdir(parents=True, exist_ok=True) # make dir
    last, best = w / 'last.pt', w / 'best.pt'

    # Hyperparameters
    if isinstance(hyp, str):
        with open(hyp, errors='ignore') as f:
            hyp = yaml.safe_load(f) # load hyps dict
    LOGGER.info(colorstr('hyperparameters: ') + ', '.join(f'{k}={v}' for k, v in hyp.items()))

    # Save run settings

```

```

if not evolve:
    with open(save_dir / 'hyp.yaml', 'w') as f:
        yaml.safe_dump(hyp, f, sort_keys=False)
    with open(save_dir / 'opt.yaml', 'w') as f:
        yaml.safe_dump(vars(opt), f, sort_keys=False)

# Loggers
data_dict = None
if RANK in [-1, 0]:
    loggers = Loggers(save_dir, weights, opt, hyp, LOGGER) # loggers instance
    if loggers.wandb:
        data_dict = loggers.wandb.data_dict
        if resume:
            weights, epochs, hyp = opt.weights, opt.epochs, opt.hyp

# Register actions
for k in methods(loggers):
    callbacks.register_action(k, callback=getattr(loggers, k))

# Config
plots = not evolve # create plots
cuda = device.type != 'cpu'
init_seeds(1 + RANK)
with torch_distributed_zero_first(LOCAL_RANK):
    data_dict = data_dict or check_dataset(data) # check if None
train_path, val_path = data_dict['train'], data_dict['val']
nc = 1 if single_cls else int(data_dict['nc']) # number of classes
names = ['item'] if single_cls and len(data_dict['names']) != 1 else data_dict['names'] # class
names
assert len(names) == nc, f'{len(names)} names found for nc={nc} dataset in {data}' # check
is_coco = isinstance(val_path, str) and val_path.endswith('coco/val2017.txt') # COCO datas
et

# Model
check_suffix(weights, '.pt') # check weights
pretrained = weights.endswith('.pt')
if pretrained:
    with torch_distributed_zero_first(LOCAL_RANK):
        weights = attempt_download(weights) # download if not found locally
        ckpt = torch.load(weights, map_location=device) # load checkpoint
        model = Model(cfg or ckpt['model'].yaml, ch=3, nc=nc, anchors=hyp.get('anchors')).to(dev
ice) # create
        exclude = ['anchor'] if (cfg or hyp.get('anchors')) and not resume else [] # exclude keys
        csd = ckpt['model'].float().state_dict() # checkpoint state_dict as FP32
        csd = intersect_dicts(csd, model.state_dict(), exclude=exclude) # intersect
        model.load_state_dict(csd, strict=False) # load
        LOGGER.info(f'Transferred {len(csd)}/{len(model.state_dict())} items from {weights}') # re
port
else:

```

```

model = Model(cfg, ch=3, nc=nc, anchors=hyp.get('anchors')).to(device) # create

# Freeze
freeze = [f'model.{x}:' for x in (freeze if len(freeze) > 1 else range(freeze[0]))] # layers to freeze
for k, v in model.named_parameters():
    v.requires_grad = True # train all layers
    if any(x in k for x in freeze):
        LOGGER.info(f'freezing {k}')
        v.requires_grad = False

# Image size
gs = max(int(model.stride.max()), 32) # grid size (max stride)
imgsz = check_img_size(opt.imgsz, gs, floor=gs * 2) # verify imgsz is gs-multiple

# Batch size
if RANK == -1 and batch_size == -1: # single-GPU only, estimate best batch size
    batch_size = check_train_batch_size(model, imgsz)
    loggers.on_params_update({"batch_size": batch_size})

# Optimizer
nbs = 64 # nominal batch size
accumulate = max(round(nbs / batch_size), 1) # accumulate loss before optimizing
hyp['weight_decay'] *= batch_size * accumulate / nbs # scale weight_decay
LOGGER.info(f"Scaled weight_decay = {hyp['weight_decay']}")

g0, g1, g2 = [], [], [] # optimizer parameter groups
for v in model.modules():
    if hasattr(v, 'bias') and isinstance(v.bias, nn.Parameter): # bias
        g2.append(v.bias)
    if isinstance(v, nn.BatchNorm2d): # weight (no decay)
        g0.append(v.weight)
    elif hasattr(v, 'weight') and isinstance(v.weight, nn.Parameter): # weight (with decay)
        g1.append(v.weight)

if opt.optimizer == 'Adam':
    optimizer = Adam(g0, lr=hyp['lr0'], betas=(hyp['momentum'], 0.999)) # adjust beta1 to momentum
elif opt.optimizer == 'AdamW':
    optimizer = AdamW(g0, lr=hyp['lr0'], betas=(hyp['momentum'], 0.999)) # adjust beta1 to momentum
else:
    optimizer = SGD(g0, lr=hyp['lr0'], momentum=hyp['momentum'], nesterov=True)

optimizer.add_param_group({'params': g1, 'weight_decay': hyp['weight_decay']}) # add g1 with weight_decay
optimizer.add_param_group({'params': g2}) # add g2 (biases)
LOGGER.info(f"{colorstr('optimizer:')} {type(optimizer).__name__} with parameter groups "
            f"{len(g0)} weight (no decay), {len(g1)} weight, {len(g2)} bias")

```

```

del g0, g1, g2

# Scheduler
if opt.linear_lr:
    lf = lambda x: (1 - x / (epochs - 1)) * (1.0 - hyp['lrf']) + hyp['lrf'] # linear
else:
    lf = one_cycle(1, hyp['lrf'], epochs) # cosine 1->hyp['lrf']
    scheduler = lr_scheduler.LambdaLR(optimizer, lr_lambda=lf) # plot_lr_scheduler(optimizer,
scheduler, epochs)

# EMA
ema = ModelEMA(model) if RANK in [-1, 0] else None

# Resume
start_epoch, best_fitness = 0, 0.0
if pretrained:
    # Optimizer
    if ckpt['optimizer'] is not None:
        optimizer.load_state_dict(ckpt['optimizer'])
        best_fitness = ckpt['best_fitness']

    # EMA
    if ema and ckpt.get('ema'):
        ema.ema.load_state_dict(ckpt['ema'].float().state_dict())
        ema.updates = ckpt['updates']

    # Epochs
    start_epoch = ckpt['epoch'] + 1
    if resume:
        assert start_epoch > 0, f'{weights} training to {epochs} epochs is finished, nothing to re
sume.'
    if epochs < start_epoch:
        LOGGER.info(f'{weights} has been trained for {ckpt['epoch']} epochs. Fine-
tuning for {epochs} more epochs.')
        epochs += ckpt['epoch'] # finetune additional epochs

del ckpt, csd

# DP mode
if cuda and RANK == -1 and torch.cuda.device_count() > 1:
    LOGGER.warning('WARNING: DP not recommended, use torch.distributed.run for best D
DP Multi-GPU results.\n'
'See Multi-
GPU Tutorial at https://github.com/ultralytics/yolov5/issues/475 to get started.')
    model = torch.nn.DataParallel(model)

# SyncBatchNorm
if opt.sync_bn and cuda and RANK != -1:
    model = torch.nn.SyncBatchNorm.convert_sync_batchnorm(model).to(device)

```

```

    LOGGER.info('Using SyncBatchNorm()')

    # Trainloader
    train_loader, dataset = create_dataloader(train_path, imsz, batch_size // WORLD_SIZE, g
s, single_cls,
    hyp=hyp, augment=True, cache=opt.cache, rect=opt.rect, rank=LOCAL_RANK,
        workers=workers, image_weights=opt.image_weights, quad=opt.quad,
        prefix=colorstr('train: '), shuffle=True)
    mlc = int(np.concatenate(dataset.labels, 0)[: , 0].max()) # max label class
    nb = len(train_loader) # number of batches
    assert mlc < nc, f'Label class {mlc} exceeds nc={nc} in {data}. Possible class labels are 0-
{nc - 1}'

    # Process 0
    if RANK in [-1, 0]:
        val_loader = create_dataloader(val_path, imsz, batch_size // WORLD_SIZE * 2, gs, sing
le_cls,
            hyp=hyp, cache=None if noval else opt.cache, rect=True, rank=-1,
            workers=workers, pad=0.5,
            prefix=colorstr('val: '))[0]

    if not resume:
        labels = np.concatenate(dataset.labels, 0)
        # c = torch.tensor(labels[:, 0]) # classes
        # cf = torch.bincount(c.long(), minlength=nc) + 1. # frequency
        # model._initialize_biases(cf.to(device))
        if plots:
            plot_labels(labels, names, save_dir)

    # Anchors
    if not opt.noautoanchor:
        check_anchors(dataset, model=model, thr=hyp['anchor_t'], imsz=imsz)
        model.half().float() # pre-reduce anchor precision

    callbacks.run('on_pretrain_routine_end')

    # DDP mode
    if cuda and RANK != -1:
        model = DDP(model, device_ids=[LOCAL_RANK], output_device=LOCAL_RANK)

    # Model attributes
    nl = de_parallel(model).model[-1].nl # number of detection layers (to scale hyps)
    hyp['box'] *= 3 / nl # scale to layers
    hyp['cls'] *= nc / 80 * 3 / nl # scale to classes and layers
    hyp['obj'] *= (imsz / 640) ** 2 * 3 / nl # scale to image size and layers
    hyp['label_smoothing'] = opt.label_smoothing
    model.nc = nc # attach number of classes to model
    model.hyp = hyp # attach hyperparameters to model

```

```

model.class_weights = labels_to_class_weights(dataset.labels, nc).to(device) * nc # attach
class_weights
model.names = names

# Start training
t0 = time.time()
nw = max(round(hyp['warmup_epochs'] * nb), 1000) # number of warmup iterations, max(3
epochs, 1k iterations)
# nw = min(nw, (epochs - start_epoch) / 2 * nb) # limit warmup to < 1/2 of training
last_opt_step = -1
maps = np.zeros(nc) # mAP per class
results = (0, 0, 0, 0, 0, 0, 0) # P, R, mAP@.5, mAP@.5-.95, val_loss(box, obj, cls)
scheduler.last_epoch = start_epoch - 1 # do not move
scaler = amp.GradScaler(enabled=cuda)
stopper = EarlyStopping(patience=opt.patience)
compute_loss = ComputeLoss(model) # init loss class
LOGGER.info(f'Image sizes {imgsz} train, {imgsz} val\n'
            f'Using {train_loader.num_workers * WORLD_SIZE} dataloader workers\n'
            f'Logging results to {colorstr('bold', save_dir)}\n'
            f'Starting training for {epochs} epochs...')
for epoch in range(start_epoch, epochs): # epoch -----
    -----
    model.train()

    # Update image weights (optional, single-GPU only)
    if opt.image_weights:
        cw = model.class_weights.cpu().numpy() * (1 - maps) ** 2 / nc # class weights
        iw = labels_to_image_weights(dataset.labels, nc=nc, class_weights=cw) # image weig
hts
        dataset.indices = random.choices(range(dataset.n), weights=iw, k=dataset.n) # rand w
eighted idx

    # Update mosaic border (optional)
    # b = int(random.uniform(0.25 * imgsz, 0.75 * imgsz + gs) // gs * gs)
    # dataset.mosaic_border = [b - imgsz, -b] # height, width borders

    mloss = torch.zeros(3, device=device) # mean losses
    if RANK != -1:
        train_loader.sampler.set_epoch(epoch)
        pbar = enumerate(train_loader)
        LOGGER.info(('\n' + '%10s' * 7) % ('Epoch', 'gpu_mem', 'box', 'obj', 'cls', 'labels', 'img_size
'))
    if RANK in [-1, 0]:
        pbar = tqdm(pbar, total=nb, bar_format='{l_bar}{bar:10}{r_bar}{bar:-
10b}') # progress bar
        optimizer.zero_grad()
        for i, (imgs, targets, paths, _) in pbar: # batch -----
            ----
            ni = i + nb * epoch # number integrated batches (since train start)

```

```

imgs = imgs.to(device, non_blocking=True).float() / 255 # uint8 to float32, 0-
255 to 0.0-1.0

# Warmup
if ni <= nw:
    xi = [0, nw] # x interp
    # compute_loss.gr = np.interp(ni, xi, [0.0, 1.0]) # iou loss ratio (obj_loss = 1.0 or iou)
    accumulate = max(1, np.interp(ni, xi, [1, nbs / batch_size])).round()
    for j, x in enumerate(optimizer.param_groups):
        # bias lr falls from 0.1 to lr0, all other lrs rise from 0.0 to lr0
        x['lr'] = np.interp(ni, xi, [hyp['warmup_bias_lr'] if j == 2 else 0.0, x['initial_lr'] * lf(epo
ch)])

        if 'momentum' in x:
            x['momentum'] = np.interp(ni, xi, [hyp['warmup_momentum'], hyp['momentum']])

# Multi-scale
if opt.multi_scale:
    sz = random.randrange(imgsz * 0.5, imgsz * 1.5 + gs) // gs * gs # size
    sf = sz / max(imgs.shape[2:]) # scale factor
    if sf != 1:
        ns = [math.ceil(x * sf / gs) * gs for x in imgs.shape[2:]] # new shape (stretched to
gs-multiple)
        imgs = nn.functional.interpolate(imgs, size=ns, mode='bilinear', align_corners=False)

# Forward
with amp.autocast(enabled=cuda):
    pred = model(imgs) # forward
    loss, loss_items = compute_loss(pred, targets.to(device)) # loss scaled by batch_si
ze

    if RANK != -1:
        loss *= WORLD_SIZE # gradient averaged between devices in DDP mode
    if opt.quad:
        loss *= 4.

# Backward
scaler.scale(loss).backward()

# Optimize
if ni - last_opt_step >= accumulate:
    scaler.step(optimizer) # optimizer.step
    scaler.update()
    optimizer.zero_grad()
    if ema:
        ema.update(model)
    last_opt_step = ni

# Log
if RANK in [-1, 0]:

```

```

        mloss = (mloss * i + loss_items) / (i + 1) # update mean losses
        mem = f'{torch.cuda.memory_reserved() / 1E9 if torch.cuda.is_available() else 0:.3g}
G' # (GB)
        pbar.set_description((' %10s' * 2 + '%10.4g' * 5) % (
            f'{epoch}/{epochs - 1}', mem, *mloss, targets.shape[0], imgs.shape[-1]))
        callbacks.run('on_train_batch_end', ni, model, imgs, targets, paths, plots, opt.sync_bn)
        if callbacks.stop_training:
            return
        # end batch -----

# Scheduler
lr = [x['lr'] for x in optimizer.param_groups] # for loggers
scheduler.step()

if RANK in [-1, 0]:
    # mAP
    callbacks.run('on_train_epoch_end', epoch=epoch)
    ema.update_attr(model, include=['yaml', 'nc', 'hyp', 'names', 'stride', 'class_weights'])
    final_epoch = (epoch + 1 == epochs) or stopper.possible_stop
    if not noval or final_epoch: # Calculate mAP
        results, maps, _ = val.run(data_dict,
                                   batch_size=batch_size // WORLD_SIZE * 2,
                                   imgsz=imgsz,
                                   model=ema.ema,
                                   single_cls=single_cls,
                                   dataloader=val_loader,
                                   save_dir=save_dir,
                                   plots=False,
                                   callbacks=callbacks,
                                   compute_loss=compute_loss)

    # Update best mAP
    fi = fitness(np.array(results).reshape(1, 1)) # weighted combination of [P, R, mAP@.5,
mAP@.5-.95]
    if fi > best_fitness:
        best_fitness = fi
    log_vals = list(mloss) + list(results) + lr
    callbacks.run('on_fit_epoch_end', log_vals, epoch, best_fitness, fi)

# Save model
if (not nosave) or (final_epoch and not evolve): # if save
    ckpt = {'epoch': epoch,
            'best_fitness': best_fitness,
            'model': deepcopy(de_parallel(model)).half(),
            'ema': deepcopy(ema.ema).half(),
            'updates': ema.updates,
            'optimizer': optimizer.state_dict(),
            'wandb_id': loggers.wandb.wandb_run.id if loggers.wandb else None,
            'date': datetime.now().isoformat()}

```

```

# Save last, best and delete
torch.save(ckpt, last)
if best_fitness == fi:
    torch.save(ckpt, best)
if (epoch > 0) and (opt.save_period > 0) and (epoch % opt.save_period == 0):
    torch.save(ckpt, w / f'epoch{epoch}.pt')
del ckpt
callbacks.run('on_model_save', last, epoch, final_epoch, best_fitness, fi)

# Stop Single-GPU
if RANK == -1 and stopper(epoch=epoch, fitness=fi):
    break

# Stop DDP TODO: known issues https://github.com/ultralytics/yolov5/pull/4576
# stop = stopper(epoch=epoch, fitness=fi)
# if RANK == 0:
#     dist.broadcast_object_list([stop], 0) # broadcast 'stop' to all ranks

# Stop DPP
# with torch_distributed_zero_first(RANK):
# if stop:
#     break # must break all DDP ranks

# end epoch -----
# end training -----
if RANK in [-1, 0]:
    LOGGER.info(f'\n{epoch - start_epoch + 1} epochs completed in {(time.time() - t0) / 3600:.3f} hours.')
    for f in last, best:
        if f.exists():
            strip_optimizer(f) # strip optimizers
            if f is best:
                LOGGER.info(f'\nValidating {f}...')
                results, _, _ = val.run(data_dict,
                                        batch_size=batch_size // WORLD_SIZE * 2,
                                        imgsz=imgsz,
                                        model=attempt_load(f, device).half(),
                                        iou_thres=0.65 if is_coco else 0.60, # best pycocotools results at 0.65
                                        single_cls=single_cls,
                                        dataloader=val_loader,
                                        save_dir=save_dir,
                                        save_json=is_coco,
                                        verbose=True,
                                        plots=True,
                                        callbacks=callbacks,
                                        compute_loss=compute_loss) # val best model with plots
                if is_coco:
                    callbacks.run('on_fit_epoch_end', list(mloss) + list(results) + lr, epoch, best_fitness, fi)

```

```

callbacks.run('on_train_end', last, best, plots, epoch, results)
LOGGER.info(f"Results saved to {colorstr('bold', save_dir)}")

torch.cuda.empty_cache()
return results

def parse_opt(known=False):
    parser = argparse.ArgumentParser()
    parser.add_argument('--
weights', type=str, default=ROOT / 'yolov5s.pt', help='initial weights path')
    parser.add_argument('--cfg', type=str, default="", help='model.yaml path')
    parser.add_argument('--
data', type=str, default=ROOT / 'data/coco128.yaml', help='dataset.yaml path')
    parser.add_argument('--
hyp', type=str, default=ROOT / 'data/hyps/hyp.scratch.yaml', help='hyperparameters path')
    parser.add_argument('--epochs', type=int, default=300)
    parser.add_argument('--batch-
size', type=int, default=16, help='total batch size for all GPUs, -1 for autobatch')
    parser.add_argument('--imgsz', '--img', '--img-
size', type=int, default=640, help='train, val image size (pixels)')
    parser.add_argument('--rect', action='store_true', help='rectangular training')
    parser.add_argument('--
resume', nargs='?', const=True, default=False, help='resume most recent training')
    parser.add_argument('--nosave', action='store_true', help='only save final checkpoint')
    parser.add_argument('--noval', action='store_true', help='only validate final epoch')
    parser.add_argument('--noautoanchor', action='store_true', help='disable AutoAnchor')
    parser.add_argument('--
evolve', type=int, nargs='?', const=300, help='evolve hyperparameters for x generations')
    parser.add_argument('--bucket', type=str, default="", help='gsutil bucket')
    parser.add_argument('--cache', type=str, nargs='?', const='ram', help='--
cache images in "ram" (default) or "disk")
    parser.add_argument('--image-
weights', action='store_true', help='use weighted image selection for training')
    parser.add_argument('--device', default="", help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--multi-scale', action='store_true', help='vary img-size +/- 50%%')
    parser.add_argument('--single-cls', action='store_true', help='train multi-
class data as single-class')
    parser.add_argument('--
optimizer', type=str, choices=['SGD', 'Adam', 'AdamW'], default='SGD', help='optimizer')
    parser.add_argument('--sync-
bn', action='store_true', help='use SyncBatchNorm, only available in DDP mode')
    parser.add_argument('--
workers', type=int, default=8, help='max dataloader workers (per RANK in DDP mode)')
    parser.add_argument('--project', default=ROOT / 'runs/train', help='save to project/name')
    parser.add_argument('--name', default='exp', help='save to project/name')
    parser.add_argument('--exist-
ok', action='store_true', help='existing project/name ok, do not increment')

```

```

parser.add_argument('--quad', action='store_true', help='quad dataloader')
parser.add_argument('--linear-lr', action='store_true', help='linear LR')
parser.add_argument('--label-
smoothing', type=float, default=0.0, help='Label smoothing epsilon')
parser.add_argument('--
patience', type=int, default=100, help='EarlyStopping patience (epochs without improvement)')
parser.add_argument('--
freeze', nargs='+', type=int, default=[0], help='Freeze layers: backbone=10, first3=0 1 2')
parser.add_argument('--save-period', type=int, default=-
1, help='Save checkpoint every x epochs (disabled if < 1)')
parser.add_argument('--local_rank', type=int, default=-
1, help='DDP parameter, do not modify')

# Weights & Biases arguments
parser.add_argument('--entity', default=None, help='W&B: Entity')
parser.add_argument('--
upload_dataset', nargs='?', const=True, default=False, help='W&B: Upload data, "val" option')
parser.add_argument('--bbox_interval', type=int, default=-1, help='W&B: Set bounding-
box image logging interval')
parser.add_argument('--
artifact_alias', type=str, default='latest', help='W&B: Version of dataset artifact to use')

opt = parser.parse_known_args()[0] if known else parser.parse_args()
return opt

def main(opt, callbacks=Callbacks()):
    # Checks
    if RANK in [-1, 0]:
        print_args(FILE.stem, opt)
        check_git_status()
        check_requirements(exclude=['thop'])

    # Resume
    if opt.resume and not check_wandb_resume(opt) and not opt.evolve: # resume an interrupt
ed run
        ckpt = opt.resume if isinstance(opt.resume, str) else get_latest_run() # specified or most
recent path
        assert os.path.isfile(ckpt), 'ERROR: --resume checkpoint does not exist'
        with open(Path(ckpt).parent.parent / 'opt.yaml', errors='ignore') as f:
            opt = argparse.Namespace(**yaml.safe_load(f)) # replace
        opt.cfg, opt.weights, opt.resume = "", ckpt, True # reinstate
        LOGGER.info(f'Resuming training from {ckpt}')
    else:
        opt.data, opt.cfg, opt.hyp, opt.weights, opt.project = \
            check_file(opt.data), check_yaml(opt.cfg), check_yaml(opt.hyp), str(opt.weights), str(o
pt.project) # checks
        assert len(opt.cfg) or len(opt.weights), 'either --cfg or --weights must be specified'
        if opt.evolve:

```

```

    opt.project = str(ROOT / 'runs/evolve')
    opt.exist_ok, opt.resume = opt.resume, False # pass resume to exist_ok and disable r
esume
    opt.save_dir = str(increment_path(Path(opt.project) / opt.name, exist_ok=opt.exist_ok))

# DDP mode
device = select_device(opt.device, batch_size=opt.batch_size)
if LOCAL_RANK != -1:
    assert torch.cuda.device_count() > LOCAL_RANK, 'insufficient CUDA devices for DDP c
ommand'
    assert opt.batch_size % WORLD_SIZE == 0, '--batch-
size must be multiple of CUDA device count'
    assert not opt.image_weights, '--image-
weights argument is not compatible with DDP training'
    assert not opt.evolve, '--evolve argument is not compatible with DDP training'
    torch.cuda.set_device(LOCAL_RANK)
    device = torch.device('cuda', LOCAL_RANK)
    dist.init_process_group(backend="nccl" if dist.is_nccl_available() else "gloo")

# Train
if not opt.evolve:
    train(opt.hyp, opt, device, callbacks)
    if WORLD_SIZE > 1 and RANK == 0:
        LOGGER.info('Destroying process group... ')
        dist.destroy_process_group()

# Evolve hyperparameters (optional)
else:
    # Hyperparameter evolution metadata (mutation scale 0-1, lower_limit, upper_limit)
    meta = {'lr0': (1, 1e-5, 1e-1), # initial learning rate (SGD=1E-2, Adam=1E-3)
            'lrf': (1, 0.01, 1.0), # final OneCycleLR learning rate (lr0 * lrf)
            'momentum': (0.3, 0.6, 0.98), # SGD momentum/Adam beta1
            'weight_decay': (1, 0.0, 0.001), # optimizer weight decay
            'warmup_epochs': (1, 0.0, 5.0), # warmup epochs (fractions ok)
            'warmup_momentum': (1, 0.0, 0.95), # warmup initial momentum
            'warmup_bias_lr': (1, 0.0, 0.2), # warmup initial bias lr
            'box': (1, 0.02, 0.2), # box loss gain
            'cls': (1, 0.2, 4.0), # cls loss gain
            'cls_pw': (1, 0.5, 2.0), # cls BCELoss positive_weight
            'obj': (1, 0.2, 4.0), # obj loss gain (scale with pixels)
            'obj_pw': (1, 0.5, 2.0), # obj BCELoss positive_weight
            'iou_t': (0, 0.1, 0.7), # IoU training threshold
            'anchor_t': (1, 2.0, 8.0), # anchor-multiple threshold
            'anchors': (2, 2.0, 10.0), # anchors per output grid (0 to ignore)
            'fl_gamma': (0, 0.0, 2.0), # focal loss gamma (efficientDet default gamma=1.5)
            'hsv_h': (1, 0.0, 0.1), # image HSV-Hue augmentation (fraction)
            'hsv_s': (1, 0.0, 0.9), # image HSV-Saturation augmentation (fraction)
            'hsv_v': (1, 0.0, 0.9), # image HSV-Value augmentation (fraction)
            'degrees': (1, 0.0, 45.0), # image rotation (+/- deg)

```

```

'translate': (1, 0.0, 0.9), # image translation (+/- fraction)
'scale': (1, 0.0, 0.9), # image scale (+/- gain)
'shear': (1, 0.0, 10.0), # image shear (+/- deg)
'perspective': (0, 0.0, 0.001), # image perspective (+/- fraction), range 0-0.001
'flipud': (1, 0.0, 1.0), # image flip up-down (probability)
'fliplr': (0, 0.0, 1.0), # image flip left-right (probability)
'mosaic': (1, 0.0, 1.0), # image mixup (probability)
'mixup': (1, 0.0, 1.0), # image mixup (probability)
'copy_paste': (1, 0.0, 1.0)} # segment copy-paste (probability)

with open(opt.hyp, errors='ignore') as f:
    hyp = yaml.safe_load(f) # load hyps dict
    if 'anchors' not in hyp: # anchors commented in hyp.yaml
        hyp['anchors'] = 3
opt.noval, opt.nosave, save_dir = True, True, Path(opt.save_dir) # only val/save final epo
ch
# ei = [isinstance(x, (int, float)) for x in hyp.values()] # evolvable indices
evolve_yaml, evolve_csv = save_dir / 'hyp_evolve.yaml', save_dir / 'evolve.csv'
if opt.bucket:
    os.system(f'gsutil cp gs://{opt.bucket}/evolve.csv {save_dir}') # download evolve.csv if
exists

for _ in range(opt.evolve): # generations to evolve
    if evolve_csv.exists(): # if evolve.csv exists: select best hyps and mutate
        # Select parent(s)
        parent = 'single' # parent selection method: 'single' or 'weighted'
        x = np.loadtxt(evolve_csv, ndmin=2, delimiter=',', skiprows=1)
        n = min(5, len(x)) # number of previous results to consider
        x = x[np.argsort(-fitness(x))][-n] # top n mutations
        w = fitness(x) - fitness(x).min() + 1E-6 # weights (sum > 0)
        if parent == 'single' or len(x) == 1:
            # x = x[random.randint(0, n - 1)] # random selection
            x = x[random.choices(range(n), weights=w)[0]] # weighted selection
        elif parent == 'weighted':
            x = (x * w.reshape(n, 1)).sum(0) / w.sum() # weighted combination

        # Mutate
        mp, s = 0.8, 0.2 # mutation probability, sigma
        npr = np.random
        npr.seed(int(time.time()))
        g = np.array([meta[k][0] for k in hyp.keys()]) # gains 0-1
        ng = len(meta)
        v = np.ones(ng)
        while all(v == 1): # mutate until a change occurs (prevent duplicates)
            v = (g * (npr.random(ng) < mp) * npr.randn(ng) * npr.random() * s + 1).clip(0.3, 3.0
)

        for i, k in enumerate(hyp.keys()): # plt.hist(v.ravel(), 300)
            hyp[k] = float(x[i + 7] * v[i]) # mutate

```

```

# Constrain to limits
for k, v in meta.items():
    hyp[k] = max(hyp[k], v[1]) # lower limit
    hyp[k] = min(hyp[k], v[2]) # upper limit
    hyp[k] = round(hyp[k], 5) # significant digits

# Train mutation
results = train(hyp.copy(), opt, device, callbacks)
callbacks = Callbacks()
# Write mutation results
print_mutation(results, hyp.copy(), save_dir, opt.bucket)

# Plot results
plot_evolve(evolve_csv)
LOGGER.info(f'Hyperparameter evolution finished\n'
            f'Results saved to {colorstr('bold', save_dir)}\n'
            f'Use best hyperparameters example: $ python train.py --hyp {evolve_yaml}')

def run(**kwargs):
    # Usage: import train; train.run(data='coco128.yaml', imgsz=320, weights='yolov5m.pt')
    opt = parse_opt(True)
    for k, v in kwargs.items():
        setattr(opt, k, v)
    main(opt)

if __name__ == "__main__":
    opt = parse_opt()
    main(opt)

```

4.11.3 Detecting the object

Detect.py

```

import argparse
import os
import sys
from pathlib import Path

import cv2
import torch
import torch.backends.cudnn as cudnn

FILE = Path(__file__).resolve()
ROOT = FILE.parents[0] # YOLOv5 root directory
if str(ROOT) not in sys.path:
    sys.path.append(str(ROOT)) # add ROOT to PATH

```

```

ROOT = Path(os.path.relpath(ROOT, Path.cwd())) # relative

from models.common import DetectMultiBackend
from utils.datasets import IMG_FORMATS, VID_FORMATS, LoadImages, LoadStreams
from utils.general import (LOGGER, check_file, check_img_size, check_imshow, check_requirements, colorstr,
                           increment_path, non_max_suppression, print_args, scale_coords, strip_optimizer, xyxy2xywh)
from utils.plots import Annotator, colors, save_one_box
from utils.torch_utils import select_device, time_sync

@torch.no_grad()
def run(weights=ROOT / 'yolov5s.pt', # model.pt path(s)
        source=ROOT / 'data/images', # file/dir/URL/glob, 0 for webcam
        data=ROOT / 'data/coco128.yaml', # dataset.yaml path
        imgsz=(640, 640), # inference size (height, width)
        conf_thres=0.25, # confidence threshold
        iou_thres=0.45, # NMS IOU threshold
        max_det=1000, # maximum detections per image
        device="", # cuda device, i.e. 0 or 0,1,2,3 or cpu
        view_img=False, # show results
        save_txt=False, # save results to *.txt
        save_conf=False, # save confidences in --save-txt labels
        save_crop=False, # save cropped prediction boxes
        nosave=False, # do not save images/videos
        classes=None, # filter by class: --class 0, or --class 0 2 3
        agnostic_nms=False, # class-agnostic NMS
        augment=False, # augmented inference
        visualize=False, # visualize features
        update=False, # update all models
        project=ROOT / 'runs/detect', # save results to project/name
        name='exp', # save results to project/name
        exist_ok=False, # existing project/name ok, do not increment
        line_thickness=3, # bounding box thickness (pixels)
        hide_labels=False, # hide labels
        hide_conf=False, # hide confidences
        half=False, # use FP16 half-precision inference
        dnn=False, # use OpenCV DNN for ONNX inference
        ):
    source = str(source)
    save_img = not nosave and not source.endswith('.txt') # save inference images
    is_file = Path(source).suffix[1:] in (IMG_FORMATS + VID_FORMATS)
    is_url = source.lower().startswith(('rtsp://', 'rtmp://', 'http://', 'https://'))
    webcam = source.isnumeric() or source.endswith('.txt') or (is_url and not is_file)
    if is_url and is_file:
        source = check_file(source) # download

# Directories

```

```

save_dir = increment_path(Path(project) / name, exist_ok=exist_ok) # increment run
(save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True, exist_ok=True) # make dir

# Load model
device = select_device(device)
model = DetectMultiBackend(weights, device=device, dnn=dnn, data=data)
stride, names, pt, jit, onnx, engine = model.stride, model.names, model.pt, model.jit, model.
onnx, model.engine
imgsz = check_img_size(imgsz, s=stride) # check image size

# Half
half &= (pt or jit or onnx or engine) and device.type != 'cpu' # FP16 supported on limited ba
ckends with CUDA
if pt or jit:
    model.model.half() if half else model.model.float()

# Dataloader
if webcam:
    view_img = check_imshow()
    cudnn.benchmark = True # set True to speed up constant image size inference
    dataset = LoadStreams(source, img_size=imgsz, stride=stride, auto=pt)
    bs = len(dataset) # batch_size
else:
    dataset = LoadImages(source, img_size=imgsz, stride=stride, auto=pt)
    bs = 1 # batch_size
vid_path, vid_writer = [None] * bs, [None] * bs

# Run inference
model.warmup(imgsz=(1, 3, *imgsz), half=half) # warmup
dt, seen = [0.0, 0.0, 0.0], 0
for path, im, im0s, vid_cap, s in dataset:
    t1 = time_sync()
    im = torch.from_numpy(im).to(device)
    im = im.half() if half else im.float() # uint8 to fp16/32
    im /= 255 # 0 - 255 to 0.0 - 1.0
    if len(im.shape) == 3:
        im = im[None] # expand for batch dim
    t2 = time_sync()
    dt[0] += t2 - t1

# Inference
visualize = increment_path(save_dir / Path(path).stem, mkdir=True) if visualize else Fals
e
pred = model(im, augment=augment, visualize=visualize)
t3 = time_sync()
dt[1] += t3 - t2

# NMS

```

```

    pred = non_max_suppression(pred, conf_thres, iou_thres, classes, agnostic_nms, max_det=max_det)
    dt[2] += time_sync() - t3

    # Second-stage classifier (optional)
    # pred = utils.general.apply_classifier(pred, classifier_model, im, im0s)

    # Process predictions
    for i, det in enumerate(pred): # per image
        seen += 1
        if webcam: # batch_size >= 1
            p, im0, frame = path[i], im0s[i].copy(), dataset.count
            s += f'{i}: '
        else:
            p, im0, frame = path, im0s.copy(), getattr(dataset, 'frame', 0)

        p = Path(p) # to Path
        save_path = str(save_dir / p.name) # im.jpg
        txt_path = str(save_dir / 'labels' / p.stem) + (' ' if dataset.mode == 'image' else f'_{frame}')
    # im.txt
        s += '%gx%g ' % im.shape[2:] # print string
        gn = torch.tensor(im0.shape)[[1, 0, 1, 0]] # normalization gain whwh
        imc = im0.copy() if save_crop else im0 # for save_crop
        annotator = Annotator(im0, line_width=line_thickness, example=str(names))
        if len(det):
            # Rescale boxes from img_size to im0 size
            det[:, :4] = scale_coords(im.shape[2:], det[:, :4], im0.shape).round()

            # Print results
            for c in det[:, -1].unique():
                n = (det[:, -1] == c).sum() # detections per class
                s += f'{n} {names[int(c)]}' + ('s' * (n > 1)), " # add to string

            # Write results
            for *xyxy, conf, cls in reversed(det):
                if save_txt: # Write to file
                    xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist() # normalized xywh
                    line = (cls, *xywh, conf) if save_conf else (cls, *xywh) # label format
                    with open(txt_path + '.txt', 'a') as f:
                        f.write('%g ' * len(line)).rstrip() % line + '\n')

                if save_img or save_crop or view_img: # Add bbox to image
                    c = int(cls) # integer class
                    label = None if hide_labels else (names[c] if hide_conf else f'{names[c]} {conf:.2f}')
                    annotator.box_label(xyxy, label, color=colors(c, True))
                if save_crop:

```

```

        save_one_box(xyxy, imc, file=save_dir / 'crops' / names[c] / f'{p.stem}.jpg', B
GR=True)

    # Print time (inference-only)
    LOGGER.info(f'{s}Done. ({t3 - t2:.3f}s)')

    # Stream results
    im0 = annotator.result()
    if view_img:
        cv2.imshow(str(p), im0)
        cv2.waitKey(1) # 1 millisecond

    # Save results (image with detections)
    if save_img:
        if dataset.mode == 'image':
            cv2.imwrite(save_path, im0)
        else: # 'video' or 'stream'
            if vid_path[i] != save_path: # new video
                vid_path[i] = save_path
            if isinstance(vid_writer[i], cv2.VideoWriter):
                vid_writer[i].release() # release previous video writer
            if vid_cap: # video
                fps = vid_cap.get(cv2.CAP_PROP_FPS)
                w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
            else: # stream
                fps, w, h = 30, im0.shape[1], im0.shape[0]
            save_path = str(Path(save_path).with_suffix('.mp4')) # force *.mp4 suffix on re
sults videos
            vid_writer[i] = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*'mp4v'), fps,
(w, h))
            vid_writer[i].write(im0)

    # Print results
    t = tuple(x / seen * 1E3 for x in dt) # speeds per image
    LOGGER.info(f'Speed: %.1fms pre-
process, %.1fms inference, %.1fms NMS per image at shape {(1, 3, *imgsz)}' % t)
    if save_txt or save_img:
        s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to {save_dir / 'labels'}" if save_t
xt else "
    LOGGER.info(f'Results saved to {colorstr('bold', save_dir)}{s}')
    if update:
        strip_optimizer(weights) # update model (to fix SourceChangeWarning)

def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--
weights', nargs='+', type=str, default=ROOT / 'yolov5s.pt', help='model path(s)')

```

```

    parser.add_argument('--
source', type=str, default=ROOT / 'data/images', help='file/dir/URL/glob, 0 for webcam')
    parser.add_argument('--
data', type=str, default=ROOT / 'data/coco128.yaml', help='(optional) dataset.yaml path')
    parser.add_argument('--imgsz', '--img', '--img-
size', nargs='+', type=int, default=[640], help='inference size h,w')
    parser.add_argument('--conf-thres', type=float, default=0.25, help='confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.45, help='NMS IoU threshold')
    parser.add_argument('--max-
det', type=int, default=1000, help='maximum detections per image')
    parser.add_argument('--device', default="", help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='show results')
    parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-
txt labels')
    parser.add_argument('--save-
crop', action='store_true', help='save cropped prediction boxes')
    parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --classes 0, or --
classes 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--visualize', action='store_true', help='visualize features')
    parser.add_argument('--update', action='store_true', help='update all models')
    parser.add_argument('--
project', default=ROOT / 'runs/detect', help='save results to project/name')
    parser.add_argument('--name', default='exp', help='save results to project/name')
    parser.add_argument('--exist-
ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--line-
thickness', default=3, type=int, help='bounding box thickness (pixels)')
    parser.add_argument('--hide-labels', default=False, action='store_true', help='hide labels')
    parser.add_argument('--hide-
conf', default=False, action='store_true', help='hide confidences')
    parser.add_argument('--half', action='store_true', help='use FP16 half-precision inference')
    parser.add_argument('--
dnn', action='store_true', help='use OpenCV DNN for ONNX inference')
    opt = parser.parse_args()
    opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1 # expand
    print_args(FILE.stem, opt)
    return opt

def main(opt):
    check_requirements(exclude=('tensorboard', 'thop'))
    run(**vars(opt))

if __name__ == "__main__":

```

```
opt = parse_opt()
main(opt)
```

4.11.4 exporting the detected process

Export.py

```
import argparse
import json
import os
import platform
import subprocess
import sys
import time
from pathlib import Path

import torch
import torch.nn as nn
from torch.utils.mobile_optimizer import optimize_for_mobile

FILE = Path(__file__).resolve()
ROOT = FILE.parents[0] # YOLOv5 root directory
if str(ROOT) not in sys.path:
    sys.path.append(str(ROOT)) # add ROOT to PATH
ROOT = Path(os.path.relpath(ROOT, Path.cwd())) # relative

from models.common import Conv
from models.experimental import attempt_load
from models.yolo import Detect
from utils.activations import SiLU
from utils.datasets import LoadImages
from utils.general import (LOGGER, check_dataset, check_img_size, check_requirements, check_version, colorstr,
                           file_size, print_args, url2file)
from utils.torch_utils import select_device

def export_torchscript(model, im, file, optimize, prefix=colorstr('TorchScript:')):
    # YOLOv5 TorchScript model export
    try:
        LOGGER.info(f'\n{prefix} starting export with torch {torch.__version__}...')
        f = file.with_suffix('.torchscript')

        ts = torch.jit.trace(model, im, strict=False)
        d = {"shape": im.shape, "stride": int(max(model.stride)), "names": model.names}
        extra_files = {'config.txt': json.dumps(d)} # torch._C.ExtraFilesMap()

        ts.save(f)
```

```

if optimize: # https://pytorch.org/tutorials/recipes/mobile\_interpreter.html
    optimize_for_mobile(ts)._save_for_lite_interpreter(str(f), _extra_files=extra_files)
else:
    ts.save(str(f), _extra_files=extra_files)

LOGGER.info(f'{prefix} export success, saved as {f} ({file_size(f):.1f} MB)')
return f
except Exception as e:
    LOGGER.info(f'{prefix} export failure: {e}')

def export_onnx(model, im, file, opset, train, dynamic, simplify, prefix=colorstr('ONNX:')):
    # YOLOv5 ONNX export
    try:
        check_requirements(('onnx',))
        import onnx

        LOGGER.info(f'\n{prefix} starting export with onnx {onnx.__version__}...')
        f = file.with_suffix('.onnx')

        torch.onnx.export(model, im, f, verbose=False, opset_version=opset,
                          training=torch.onnx.TrainingMode.TRAINING if train else torch.onnx.Training
Mode.EVAL,
                          do_constant_folding=not train,
                          input_names=['images'],
                          output_names=['output'],
                          dynamic_axes={'images': {0: 'batch', 2: 'height', 3: 'width'}, # shape(1,3,640,6
40)
                                'output': {0: 'batch', 1: 'anchors'} # shape(1,25200,85)
                                } if dynamic else None)

        # Checks
        model_onnx = onnx.load(f) # load onnx model
        onnx.checker.check_model(model_onnx) # check onnx model
        # LOGGER.info(onnx.helper.printable_graph(model_onnx.graph)) # print

        # Simplify
        if simplify:
            try:
                check_requirements(('onnx-simplifier',))
                import onnxsim

                LOGGER.info(f'{prefix} simplifying with onnx-simplifier {onnxsim.__version__}...')
                model_onnx, check = onnxsim.simplify(
                    model_onnx,
                    dynamic_input_shape=dynamic,
                    input_shapes={'images': list(im.shape)} if dynamic else None)
                assert check, 'assert check failed'
                onnx.save(model_onnx, f)

```

```

        except Exception as e:
            LOGGER.info(f'{prefix} simplifier failure: {e}')
        LOGGER.info(f'{prefix} export success, saved as {f} ({file_size(f):.1f} MB)')
        return f
    except Exception as e:
        LOGGER.info(f'{prefix} export failure: {e}')

def export_openvino(model, im, file, prefix=colorstr('OpenVINO:')):
    # YOLOv5 OpenVINO export
    try:
        check_requirements(('openvino-dev',)) # requires openvino-
dev: https://pypi.org/project/openvino-dev/
        import openvino.inference_engine as ie

        LOGGER.info(f'\n{prefix} starting export with openvino {ie.__version__}...')
        f = str(file).replace('.pt', '_openvino_model' + os.sep)

        cmd = f"mo --input_model {file.with_suffix('.onnx')} --output_dir {f}"
        subprocess.check_output(cmd, shell=True)

        LOGGER.info(f'{prefix} export success, saved as {f} ({file_size(f):.1f} MB)')
        return f
    except Exception as e:
        LOGGER.info(f'\n{prefix} export failure: {e}')

def export_coreml(model, im, file, prefix=colorstr('CoreML:')):
    # YOLOv5 CoreML export
    try:
        check_requirements(('coremltools',))
        import coremltools as ct

        LOGGER.info(f'\n{prefix} starting export with coremltools {ct.__version__}...')
        f = file.with_suffix('.mlmodel')

        ts = torch.jit.trace(model, im, strict=False) # TorchScript model
        ct_model = ct.convert(ts, inputs=[ct.ImageType('image', shape=im.shape, scale=1 / 255,
bias=[0, 0, 0])])
        ct_model.save(f)

        LOGGER.info(f'{prefix} export success, saved as {f} ({file_size(f):.1f} MB)')
        return ct_model, f
    except Exception as e:
        LOGGER.info(f'\n{prefix} export failure: {e}')
        return None, None

```

```

def export_engine(model, im, file, train, half, simplify, workspace=4, verbose=False, prefix=col
orstr('TensorRT:')):
    # YOLOv5 TensorRT export https://developer.nvidia.com/tensorrt
    try:
        check_requirements(('tensorrt',))
        import tensorrt as trt

        if trt.__version__[0] == '7': # TensorRT 7 handling https://github.com/ultralytics/yolov5/iss
ues/6012
            grid = model.model[-1].anchor_grid
            model.model[-1].anchor_grid = [a[...,:1,:1,:] for a in grid]
            export_onnx(model, im, file, 12, train, False, simplify) # opset 12
            model.model[-1].anchor_grid = grid
        else: # TensorRT >= 8
            check_version(trt.__version__, '8.0.0', hard=True) # require tensorrt>=8.0.0
            export_onnx(model, im, file, 13, train, False, simplify) # opset 13
            onnx = file.with_suffix('.onnx')

        LOGGER.info(f'\n{prefix} starting export with TensorRT {trt.__version__}...')
        assert im.device.type != 'cpu', 'export running on CPU but must be on GPU, i.e. `python e
xport.py --device 0`'
        assert onnx.exists(), f'failed to export ONNX file: {onnx}'
        f = file.with_suffix('.engine') # TensorRT engine file
        logger = trt.Logger(trt.Logger.INFO)
        if verbose:
            logger.min_severity = trt.Logger.Severity.VERBOSE

        builder = trt.Builder(logger)
        config = builder.create_builder_config()
        config.max_workspace_size = workspace * 1 << 30

        flag = (1 << int(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))
        network = builder.create_network(flag)
        parser = trt.OnnxParser(network, logger)
        if not parser.parse_from_file(str(onnx)):
            raise RuntimeError(f'failed to load ONNX file: {onnx}')

        inputs = [network.get_input(i) for i in range(network.num_inputs)]
        outputs = [network.get_output(i) for i in range(network.num_outputs)]
        LOGGER.info(f'{prefix} Network Description:')
        for inp in inputs:
            LOGGER.info(f'{prefix}\tinput "{inp.name}" with shape {inp.shape} and dtype {inp.dtype
}')
        for out in outputs:
            LOGGER.info(f'{prefix}\toutput "{out.name}" with shape {out.shape} and dtype {out.dtyp
e}')

        half &= builder.platform_has_fast_fp16
        LOGGER.info(f'{prefix} building FP{16 if half else 32} engine in {f}')

```

```

if half:
    config.set_flag(trt.BuilderFlag.FP16)
    with builder.build_engine(network, config) as engine, open(f, 'wb') as t:
        t.write(engine.serialize())
    LOGGER.info(f'{prefix} export success, saved as {f} ({file_size(f):.1f} MB)')
    return f
except Exception as e:
    LOGGER.info(f'\n{prefix} export failure: {e}')

def export_saved_model(model, im, file, dynamic,
                      tf_nms=False, agnostic_nms=False, topk_per_class=100, topk_all=100, iou_thr
es=0.45,
                      conf_thres=0.25, prefix=colorstr('TensorFlow SavedModel:')):
    # YOLOv5 TensorFlow SavedModel export
    try:
        import tensorflow as tf
        from tensorflow import keras

        from models.tf import TFDetect, TFModel

        LOGGER.info(f'\n{prefix} starting export with tensorflow {tf.__version__}...')
        f = str(file).replace('.pt', '_saved_model')
        batch_size, ch, *imgsz = list(im.shape) # BCHW

        tf_model = TFModel(cfg=model.yaml, model=model, nc=model.nc, imgsz=imgsz)
        im = tf.zeros((batch_size, *imgsz, 3)) # BHWC order for TensorFlow
        y = tf_model.predict(im, tf_nms, agnostic_nms, topk_per_class, topk_all, iou_thres, conf_
thres)
        inputs = keras.Input(shape=(imgsz, 3), batch_size=None if dynamic else batch_size)
        outputs = tf_model.predict(inputs, tf_nms, agnostic_nms, topk_per_class, topk_all, iou_th
res, conf_thres)
        keras_model = keras.Model(inputs=inputs, outputs=outputs)
        keras_model.trainable = False
        keras_model.summary()
        keras_model.save(f, save_format='tf')

        LOGGER.info(f'{prefix} export success, saved as {f} ({file_size(f):.1f} MB)')
        return keras_model, f
    except Exception as e:
        LOGGER.info(f'\n{prefix} export failure: {e}')
        return None, None

def export_pb(keras_model, im, file, prefix=colorstr('TensorFlow GraphDef:')):
    # YOLOv5 TensorFlow GraphDef *.pb export https://github.com/leimao/Frozen_Graph_Ten
sorFlow
    try:
        import tensorflow as tf

```

```

from tensorflow.python.framework.convert_to_constants import convert_variables_to_constants_v2

```

```

    LOGGER.info(f'\n{prefix} starting export with tensorflow {tf.__version__}...')
    f = file.with_suffix('.pb')

```

```

    m = tf.function(lambda x: keras_model(x)) # full model
    m = m.get_concrete_function(tf.TensorSpec(keras_model.inputs[0].shape, keras_model.inputs[0].dtype))
    frozen_func = convert_variables_to_constants_v2(m)
    frozen_func.graph.as_graph_def()
    tf.io.write_graph(graph_or_graph_def=frozen_func.graph, logdir=str(f.parent), name=f.name, as_text=False)

```

```

    LOGGER.info(f'{prefix} export success, saved as {f} ({file_size(f):.1f} MB)')
    return f

```

```

except Exception as e:

```

```

    LOGGER.info(f'\n{prefix} export failure: {e}')

```

```

def export_tflite(keras_model, im, file, int8, data, ncalib, prefix=colorstr('TensorFlow Lite:')):
    # YOLOv5 TensorFlow Lite export

```

```

    try:

```

```

        import tensorflow as tf

```

```

        LOGGER.info(f'\n{prefix} starting export with tensorflow {tf.__version__}...')
        batch_size, ch, *imgsz = list(im.shape) # BCHW
        f = str(file).replace('.pt', '-fp16.tflite')

```

```

        converter = tf.lite.TFLiteConverter.from_keras_model(keras_model)
        converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS]
        converter.target_spec.supported_types = [tf.float16]
        converter.optimizations = [tf.lite.Optimize.DEFAULT]

```

```

        if int8:

```

```

            from models.tf import representative_dataset_gen
            dataset = LoadImages(check_dataset(data)['train'], img_size=imgsz, auto=False) # representative data

```

```

        representative_data

```

```

            converter.representative_dataset = lambda: representative_dataset_gen(dataset, ncalib)

```

```

        b)

```

```

            converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
            converter.target_spec.supported_types = []
            converter.inference_input_type = tf.uint8 # or tf.int8
            converter.inference_output_type = tf.uint8 # or tf.int8
            converter.experimental_new_quantizer = False
            f = str(file).replace('.pt', '-int8.tflite')

```

```

        tflite_model = converter.convert()

```

```

        open(f, "wb").write(tflite_model)

```

```

        LOGGER.info(f'{prefix} export success, saved as {f} ({file_size(f):.1f} MB)')

```

```

    return f
except Exception as e:
    LOGGER.info(f'\n{prefix} export failure: {e}')

def export_edgetpu(keras_model, im, file, prefix=colorstr('Edge TPU:')):
    # YOLOv5 Edge TPU export https://coral.ai/docs/edgetpu/models-intro/
    try:
        cmd = 'edgetpu_compiler --version'
        help_url = 'https://coral.ai/docs/edgetpu/compiler/'
        assert platform.system() == 'Linux', f'export only supported on Linux. See {help_url}'
        if subprocess.run(cmd, shell=True).returncode != 0:
            LOGGER.info(f'\n{prefix} export requires Edge TPU compiler. Attempting install from {h
elp_url}')
            for c in ['curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-
key add -',
                    'echo "deb https://packages.cloud.google.com/apt coral-edgetpu-
stable main" | sudo tee /etc/apt/sources.list.d/coral-edgetpu.list',
                    'sudo apt-get update',
                    'sudo apt-get install edgetpu-compiler']:
                subprocess.run(c, shell=True, check=True)
            ver = subprocess.run(cmd, shell=True, capture_output=True, check=True).stdout.decode
().split()[-1]

            LOGGER.info(f'\n{prefix} starting export with Edge TPU compiler {ver}...')
            f = str(file).replace('.pt', '-int8_edgetpu.tflite') # Edge TPU model
            f_tfl = str(file).replace('.pt', '-int8.tflite') # TFLite model

            cmd = f"edgetpu_compiler -s {f_tfl}"
            subprocess.run(cmd, shell=True, check=True)

            LOGGER.info(f'\n{prefix} export success, saved as {f} ({file_size(f):.1f} MB)')
            return f
    except Exception as e:
        LOGGER.info(f'\n{prefix} export failure: {e}')

def export_tfjs(keras_model, im, file, prefix=colorstr('TensorFlow.js:')):
    # YOLOv5 TensorFlow.js export
    try:
        check_requirements(('tensorflowjs',))
        import re

        import tensorflowjs as tfjs

        LOGGER.info(f'\n{prefix} starting export with tensorflowjs {tfjs.__version__}...')
        f = str(file).replace('.pt', '_web_model') # js dir
        f_pb = file.with_suffix('.pb') # *.pb path
        f_json = f + '/model.json' # *.json path

```

```

cmd = f'tensorflowjs_converter --input_format=tf_frozen_model '\
    f'--output_node_names="Identity,Identity_1,Identity_2,Identity_3" {f_pb} {f}'
subprocess.run(cmd, shell=True)

json = open(f_json).read()
with open(f_json, 'w') as j: # sort JSON Identity_* in ascending order
    subst = re.sub(
        r'{"outputs": {"Identity.?.?": {"name": "Identity.?.?"}, '
        r'""Identity.?.?": {"name": "Identity.?.?"}, '
        r'""Identity.?.?": {"name": "Identity.?.?"}, '
        r'""Identity.?.?": {"name": "Identity.?.?"}}, '
        r'{"outputs": {"Identity": {"name": "Identity"}, '
        r'""Identity_1": {"name": "Identity_1"}, '
        r'""Identity_2": {"name": "Identity_2"}, '
        r'""Identity_3": {"name": "Identity_3"}}, '
        json)
    j.write(subst)

LOGGER.info(f'{prefix} export success, saved as {f} ({file_size(f):.1f} MB)')
return f
except Exception as e:
    LOGGER.info(f'\n{prefix} export failure: {e}')

```

```

@torch.no_grad()
def run(data=ROOT / 'data/coco128.yaml', # 'dataset.yaml path'
        weights=ROOT / 'yolov5s.pt', # weights path
        imgsz=(640, 640), # image (height, width)
        batch_size=1, # batch size
        device='cpu', # cuda device, i.e. 0 or 0,1,2,3 or cpu
        include=('torchscript', 'onnx'), # include formats
        half=False, # FP16 half-precision export
        inplace=False, # set YOLOv5 Detect() inplace=True
        train=False, # model.train() mode
        optimize=False, # TorchScript: optimize for mobile
        int8=False, # CoreML/TF INT8 quantization
        dynamic=False, # ONNX/TF: dynamic axes
        simplify=False, # ONNX: simplify model
        opset=12, # ONNX: opset version
        verbose=False, # TensorRT: verbose log
        workspace=4, # TensorRT: workspace size (GB)
        nms=False, # TF: add NMS to model
        agnostic_nms=False, # TF: add agnostic NMS to model
        topk_per_class=100, # TF.js NMS: topk per class to keep
        topk_all=100, # TF.js NMS: topk for all classes to keep
        iou_thres=0.45, # TF.js NMS: IoU threshold
        conf_thres=0.25 # TF.js NMS: confidence threshold
    ):

```

```

t = time.time()
include = [x.lower() for x in include]
tf_exports = list(x in include for x in ('saved_model', 'pb', 'tf_lite', 'edgetpu', 'tfjs')) # TensorFlow
w exports
file = Path(url2file(weights) if str(weights).startswith(('http:', 'https:')) else weights)

# Checks
imgsz *= 2 if len(imgsz) == 1 else 1 # expand
opset = 12 if ('openvino' in include) else opset # OpenVINO requires opset <= 12

# Load PyTorch model
device = select_device(device)
assert not (device.type == 'cpu' and half), '--half only compatible with GPU export, i.e. use --
device 0'
model = attempt_load(weights, map_location=device, inplace=True, fuse=True) # load FP3
2 model
nc, names = model.nc, model.names # number of classes, class names

# Input
gs = int(max(model.stride)) # grid size (max stride)
imgsz = [check_img_size(x, gs) for x in imgsz] # verify img_size are gs-multiples
im = torch.zeros(batch_size, 3, *imgsz).to(device) # image size(1,3,320,192) BCHW iDetec
tion

# Update model
if half:
    im, model = im.half(), model.half() # to FP16
model.train() if train else model.eval() # training mode = no Detect() layer grid construction
for k, m in model.named_modules():
    if isinstance(m, Conv): # assign export-friendly activations
        if isinstance(m.act, nn.SiLU):
            m.act = SiLU()
    elif isinstance(m, Detect):
        m.inplace = inplace
        m.onnx_dynamic = dynamic
    if hasattr(m, 'forward_export'):
        m.forward = m.forward_export # assign custom forward (optional)

for _ in range(2):
    y = model(im) # dry runs
LOGGER.info(f"\n{colorstr('PyTorch:')} starting from {file} ({file_size(file):.1f} MB)")

# Exports
f = [] * 10 # exported filenames
if 'torchscript' in include:
    f[0] = export_torchscript(model, im, file, optimize)
if 'engine' in include: # TensorRT required before ONNX
    f[1] = export_engine(model, im, file, train, half, simplify, workspace, verbose)
if ('onnx' in include) or ('openvino' in include): # OpenVINO requires ONNX

```

```

    f[2] = export_onnx(model, im, file, opset, train, dynamic, simplify)
    if 'openvino' in include:
        f[3] = export_openvino(model, im, file)
    if 'coreml' in include:
        _, f[4] = export_coreml(model, im, file)

# TensorFlow Exports
if any(tf_exports):
    pb, tflite, edgetpu, tfjs = tf_exports[1:]
    if int8 or edgetpu: # TFLite --int8 bug https://github.com/ultralytics/yolov5/issues/5707
        check_requirements(('flatbuffers==1.12',)) # required before `import tensorflow`
    assert not (tflite and tfjs), 'TFLite and TF.js models must be exported separately, please p
ass only one type.'
    model, f[5] = export_saved_model(model, im, file, dynamic, tf_nms=nms or agnostic_nms
or tfjs,
                                   agnostic_nms=agnostic_nms or tfjs, topk_per_class=topk_per_class
,
                                   topk_all=topk_all, conf_thres=conf_thres, iou_thres=iou_thres) # ker
as model
    if pb or tfjs: # pb prerequisite to tfjs
        f[6] = export_pb(model, im, file)
    if tflite or edgetpu:
        f[7] = export_tflite(model, im, file, int8=int8 or edgetpu, data=data, ncalib=100)
    if edgetpu:
        f[8] = export_edgetpu(model, im, file)
    if tfjs:
        f[9] = export_tfjs(model, im, file)

# Finish
f = [str(x) for x in f if x] # filter out " and None
LOGGER.info(f'\nExport complete ({time.time() - t:.2f}s)'
            f'\nResults saved to {colorstr('bold', file.parent.resolve())}'
            f'\nVisualize with https://netron.app'
            f'\nDetect with `python detect.py --weights {f[-1]}`'
            f' or `model = torch.hub.load('ultralytics/yolov5', 'custom', '{f[-1]}')`'
            f'\nValidate with `python val.py --weights {f[-1]}`')
return f # return list of exported files/dirs

def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--
data', type=str, default=ROOT / 'data/coco128.yaml', help='dataset.yaml path')
    parser.add_argument('--
weights', nargs='+', type=str, default=ROOT / 'yolov5s.pt', help='model.pt path(s)')
    parser.add_argument('--imgsz', '--img', '--img-
size', nargs='+', type=int, default=[640, 640], help='image (h, w)')
    parser.add_argument('--batch-size', type=int, default=1, help='batch size')
    parser.add_argument('--device', default='cpu', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')

```

```

    parser.add_argument('--half', action='store_true', help='FP16 half-precision export')
    parser.add_argument('--
inplace', action='store_true', help='set YOLOv5 Detect() inplace=True')
    parser.add_argument('--train', action='store_true', help='model.train() mode')
    parser.add_argument('--
optimize', action='store_true', help='TorchScript: optimize for mobile')
    parser.add_argument('--int8', action='store_true', help='CoreML/TF INT8 quantization')
    parser.add_argument('--dynamic', action='store_true', help='ONNX/TF: dynamic axes')
    parser.add_argument('--simplify', action='store_true', help='ONNX: simplify model')
    parser.add_argument('--opset', type=int, default=12, help='ONNX: opset version')
    parser.add_argument('--verbose', action='store_true', help='TensorRT: verbose log')
    parser.add_argument('--
workspace', type=int, default=4, help='TensorRT: workspace size (GB)')
    parser.add_argument('--nms', action='store_true', help='TF: add NMS to model')
    parser.add_argument('--agnostic-
nms', action='store_true', help='TF: add agnostic NMS to model')
    parser.add_argument('--topk-per-
class', type=int, default=100, help='TF.js NMS: topk per class to keep')
    parser.add_argument('--topk-
all', type=int, default=100, help='TF.js NMS: topk for all classes to keep')
    parser.add_argument('--iou-
thres', type=float, default=0.45, help='TF.js NMS: IoU threshold')
    parser.add_argument('--conf-
thres', type=float, default=0.25, help='TF.js NMS: confidence threshold')
    parser.add_argument('--include', nargs='+',
                        default=['torchscript', 'onnx'],
                        help='torchscript, onnx, openvino, engine, coreml, saved_model, pb, tflite, edge
tpu, tfjs')
    opt = parser.parse_args()
    print_args(FILE.stem, opt)
    return opt

def main(opt):
    for opt.weights in (opt.weights if isinstance(opt.weights, list) else [opt.weights]):
        run(**vars(opt))

if __name__ == "__main__":
    opt = parse_opt()
    main(opt)

```

4.11.5 Validating the process

Val.py

```
import argparse
import json
import os
import sys
from pathlib import Path
from threading import Thread

import numpy as np
import torch
from tqdm import tqdm

FILE = Path(__file__).resolve()
ROOT = FILE.parents[0] # YOLOv5 root directory
if str(ROOT) not in sys.path:
    sys.path.append(str(ROOT)) # add ROOT to PATH
ROOT = Path(os.path.relpath(ROOT, Path.cwd())) # relative

from models.common import DetectMultiBackend
from utils.callbacks import Callbacks
from utils.datasets import create_dataloader
from utils.general import (LOGGER, box_iou, check_dataset, check_img_size, check_requirements,
                           coco80_to_coco91_class, colorstr, increment_path, non_max_suppression, print_args,
                           scale_coords, xywh2xyxy, xyxy2xywh)
from utils.metrics import ConfusionMatrix, ap_per_class
from utils.plots import output_to_target, plot_images, plot_val_study
from utils.torch_utils import select_device, time_sync

def save_one_txt(predn, save_conf, shape, file):
    # Save one txt result
    gn = torch.tensor(shape)[[1, 0, 1, 0]] # normalization gain whwh
    for *xyxy, conf, cls in predn.tolist():
        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist() # normalized xywh
        line = (cls, *xywh, conf) if save_conf else (cls, *xywh) # label format
        with open(file, 'a') as f:
```

```
f.write('%g ' * len(line)).rstrip() % line + '\n')
```

```
def save_one_json(predn, jdict, path, class_map):
    # Save one JSON result {"image_id": 42, "category_id": 18, "bbox": [258.15, 41.29, 348.26,
    243.78], "score": 0.236}
    image_id = int(path.stem) if path.stem.isnumeric() else path.stem
    box = xyxy2xywh(predn[:, :4]) # xywh
    box[:, :2] -= box[:, 2:] / 2 # xy center to top-left corner
    for p, b in zip(predn.tolist(), box.tolist()):
        jdict.append({'image_id': image_id,
                      'category_id': class_map[int(p[5])],
                      'bbox': [round(x, 3) for x in b],
                      'score': round(p[4], 5)})
```

```
def process_batch(detections, labels, iouv):
```

```
    """
```

Return correct predictions matrix. Both sets of boxes are in (x1, y1, x2, y2) format.

Arguments:

detections (Array[N, 6]), x1, y1, x2, y2, conf, class

labels (Array[M, 5]), class, x1, y1, x2, y2

Returns:

correct (Array[N, 10]), for 10 IoU levels

```
    """
```

```
    correct = torch.zeros(detections.shape[0], iouv.shape[0], dtype=torch.bool, device=iouv.device)
```

```
    iou = box_iou(labels[:, 1:], detections[:, :4])
```

```
    x = torch.where((iou >= iouv[0]) & (labels[:, 0:1] == detections[:, 5])) # IoU above threshold
    and classes match
```

```
    if x[0].shape[0]:
```

```
        matches = torch.cat((torch.stack(x, 1), iou[x[0], x[1]][:, None]), 1).cpu().numpy() # [label,
    detection, iou]
```

```
        if x[0].shape[0] > 1:
```

```
            matches = matches[matches[:, 2].argsort()[::-1]]
```

```
            matches = matches[np.unique(matches[:, 1], return_index=True)[1]]
```

```
            # matches = matches[matches[:, 2].argsort()[::-1]]
```

```
            matches = matches[np.unique(matches[:, 0], return_index=True)[1]]
```

```
            matches = torch.Tensor(matches).to(iouv.device)
```

```
            correct[matches[:, 1].long()] = matches[:, 2:3] >= iouv
```

```
    return correct
```

```
@torch.no_grad()
```

```
def run(data,
```

```
    weights=None, # model.pt path(s)
```

```
    batch_size=32, # batch size
```

```
    imgsz=640, # inference size (pixels)
```

```
    conf_thres=0.001, # confidence threshold
```

```

iou_thres=0.6, # NMS IoU threshold
task='val', # train, val, test, speed or study
device="", # cuda device, i.e. 0 or 0,1,2,3 or cpu
workers=8, # max dataloader workers (per RANK in DDP mode)
single_cls=False, # treat as single-class dataset
augment=False, # augmented inference
verbose=False, # verbose output
save_txt=False, # save results to *.txt
save_hybrid=False, # save label+prediction hybrid results to *.txt
save_conf=False, # save confidences in --save-txt labels
save_json=False, # save a COCO-JSON results file
project=ROOT / 'runs/val', # save to project/name
name='exp', # save to project/name
exist_ok=False, # existing project/name ok, do not increment
half=True, # use FP16 half-precision inference
dnn=False, # use OpenCV DNN for ONNX inference
model=None,
dataloader=None,
save_dir=Path(""),
plots=True,
callbacks=Callbacks(),
compute_loss=None,
):
# Initialize/load model and set device
training = model is not None
if training: # called by train.py
    device, pt, jit, engine = next(model.parameters()).device, True, False, False # get model
device, PyTorch model

    half &= device.type != 'cpu' # half precision only supported on CUDA
    model.half() if half else model.float()
else: # called directly
    device = select_device(device, batch_size=batch_size)

# Directories
save_dir = increment_path(Path(project) / name, exist_ok=exist_ok) # increment run
(save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True, exist_ok=True) # make
dir

# Load model
model = DetectMultiBackend(weights, device=device, dnn=dnn, data=data)
stride, pt, jit, onnx, engine = model.stride, model.pt, model.jit, model.onnx, model.engine
imgsz = check_img_size(imgsz, s=stride) # check image size
half &= (pt or jit or onnx or engine) and device.type != 'cpu' # FP16 supported on limited
backends with CUDA
if pt or jit:
    model.model.half() if half else model.model.float()
elif engine:
    batch_size = model.batch_size

```

```

else:
    half = False
    batch_size = 1 # export.py models default to batch-size 1
    device = torch.device('cpu')
    LOGGER.info(f'Forcing --batch-
size 1 square inference shape(1,3,{imgsz},{imgsz}) for non-PyTorch backends')

    # Data
    data = check_dataset(data) # check

    # Configure
    model.eval()
    is_coco = isinstance(data.get('val'), str) and data['val'].endswith('coco/val2017.txt') # COCO
dataset
    nc = 1 if single_cls else int(data['nc']) # number of classes
    iouv = torch.linspace(0.5, 0.95, 10).to(device) # iou vector for mAP@0.5:0.95
    niou = iouv.numel()

    # Dataloader
    if not training:
        model.warmup(imgsz=(1, 3, imgsz, imgsz), half=half) # warmup
        pad = 0.0 if task == 'speed' else 0.5
        task = task if task in ('train', 'val', 'test') else 'val' # path to train/val/test images
        dataloader = create_dataloader(data[task], imgsz, batch_size, stride, single_cls, pad=pad
, rect=pt,
                                workers=workers, prefix=colorstr(f'{task}: '))[0]

    seen = 0
    confusion_matrix = ConfusionMatrix(nc=nc)
    names = {k: v for k, v in enumerate(model.names if hasattr(model, 'names') else model.mod
ule.names)}
    class_map = coco80_to_coco91_class() if is_coco else list(range(1000))
    s = ('%20s' + '%11s' * 6) % ('Class', 'Images', 'Labels', 'P', 'R', 'mAP@.5', 'mAP@.5:.95')
    dt, p, r, f1, mp, mr, map50, map = [0.0, 0.0, 0.0], 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
    loss = torch.zeros(3, device=device)
    jdict, stats, ap, ap_class = [], [], [], []
    pbar = tqdm(dataloader, desc=s, bar_format='{l_bar}{bar:10}{r_bar}{bar:-
10b}') # progress bar
    for batch_i, (im, targets, paths, shapes) in enumerate(pbar):
        t1 = time_sync()
        if pt or jit or engine:
            im = im.to(device, non_blocking=True)
            targets = targets.to(device)
            im = im.half() if half else im.float() # uint8 to fp16/32
            im /= 255 # 0 - 255 to 0.0 - 1.0
            nb, _, height, width = im.shape # batch size, channels, height, width
            t2 = time_sync()
            dt[0] += t2 - t1

```

```

# Inference
out, train_out = model(im) if training else model(im, augment=augment, val=True) # inference, loss outputs
dt[1] += time_sync() - t2

# Loss
if compute_loss:
    loss += compute_loss([x.float() for x in train_out], targets)[1] # box, obj, cls

# NMS
targets[:, 2:] *= torch.Tensor([width, height, width, height]).to(device) # to pixels
lb = [targets[targets[:, 0] == i, 1:] for i in range(nb)] if save_hybrid else [] # for autolabelling
g
t3 = time_sync()
out = non_max_suppression(out, conf_thres, iou_thres, labels=lb, multi_label=True, agnostic=single_cls)
dt[2] += time_sync() - t3

# Metrics
for si, pred in enumerate(out):
    labels = targets[targets[:, 0] == si, 1:]
    nl = len(labels)
    tcls = labels[:, 0].tolist() if nl else [] # target class
    path, shape = Path(paths[si]), shapes[si][0]
    seen += 1

    if len(pred) == 0:
        if nl:
            stats.append((torch.zeros(0, niou, dtype=torch.bool), torch.Tensor(), torch.Tensor(), tcls))
        continue

    # Predictions
    if single_cls:
        pred[:, 5] = 0
    predn = pred.clone()
    scale_coords(im[si].shape[1:], predn[:, :4], shape, shapes[si][1]) # native-space pred

    # Evaluate
    if nl:
        tbox = xywh2xyxy(labels[:, 1:5]) # target boxes
        scale_coords(im[si].shape[1:], tbox, shape, shapes[si][1]) # native-space labels
        labelsn = torch.cat((labels[:, 0:1], tbox), 1) # native-space labels
        correct = process_batch(predn, labelsn, iouv)
        if plots:
            confusion_matrix.process_batch(predn, labelsn)
    else:
        correct = torch.zeros(pred.shape[0], niou, dtype=torch.bool)

```

```

stats.append((correct.cpu(), pred[:, 4].cpu(), pred[:, 5].cpu(), tcls)) # (correct, conf, pcls
, tcls)

# Save/log
if save_txt:
    save_one_txt(predn, save_conf, shape, file=save_dir / 'labels' / (path.stem + '.txt'))
if save_json:
    save_one_json(predn, jdict, path, class_map) # append to COCO-JSON dictionary
callbacks.run('on_val_image_end', pred, predn, path, names, im[si])

# Plot images
if plots and batch_i < 3:
    f = save_dir / f'val_batch{batch_i}_labels.jpg' # labels
    Thread(target=plot_images, args=(im, targets, paths, f, names), daemon=True).start()
    f = save_dir / f'val_batch{batch_i}_pred.jpg' # predictions
    Thread(target=plot_images, args=(im, output_to_target(out), paths, f, names), daemon
=True).start()

# Compute metrics
stats = [np.concatenate(x, 0) for x in zip(*stats)] # to numpy
if len(stats) and stats[0].any():
    tp, fp, p, r, f1, ap, ap_class = ap_per_class(*stats, plot=plots, save_dir=save_dir, names=
names)
    ap50, ap = ap[:, 0], ap.mean(1) # AP@0.5, AP@0.5:0.95
    mp, mr, map50, map = p.mean(), r.mean(), ap50.mean(), ap.mean()
    nt = np.bincount(stats[3].astype(np.int64), minlength=nc) # number of targets per class
else:
    nt = torch.zeros(1)

# Print results
pf = '%20s' + '%11i' * 2 + '%11.3g' * 4 # print format
LOGGER.info(pf % ('all', seen, nt.sum(), mp, mr, map50, map))

# Print results per class
if (verbose or (nc < 50 and not training)) and nc > 1 and len(stats):
    for i, c in enumerate(ap_class):
        LOGGER.info(pf % (names[c], seen, nt[c], p[i], r[i], ap50[i], ap[i]))

# Print speeds
t = tuple(x / seen * 1E3 for x in dt) # speeds per image
if not training:
    shape = (batch_size, 3, imsz, imsz)
    LOGGER.info(f'Speed: %.1fms pre-
process, %.1fms inference, %.1fms NMS per image at shape {shape}' % t)

# Plots
if plots:
    confusion_matrix.plot(save_dir=save_dir, names=list(names.values()))
callbacks.run('on_val_end')

```

```

# Save JSON
if save_json and len(jdict):
    w = Path(weights[0] if isinstance(weights, list) else weights).stem if weights is not None else " # weights
    anno_json = str(Path(data.get('path', '../coco')) / 'annotations/instances_val2017.json') #
    annotations json
    pred_json = str(save_dir / f'{w}_predictions.json') # predictions json
    LOGGER.info(f'\nEvaluating pycocotools mAP... saving {pred_json}...')
    with open(pred_json, 'w') as f:
        json.dump(jdict, f)

try: # https://github.com/cocodataset/cocoapi/blob/master/PythonAPI/pycocoEvalDemo.ipynb
    check_requirements(['pycocotools'])
    from pycocotools.coco import COCO
    from pycocotools.cocoeval import COCOeval

    anno = COCO(anno_json) # init annotations api
    pred = anno.loadRes(pred_json) # init predictions api
    eval = COCOeval(anno, pred, 'bbox')
    if is_coco:
        eval.params.imgIds = [int(Path(x).stem) for x in dataloader.dataset.img_files] # image IDs to evaluate
        eval.evaluate()
        eval.accumulate()
        eval.summarize()
        map, map50 = eval.stats[:2] # update results (mAP@0.5:0.95, mAP@0.5)
    except Exception as e:
        LOGGER.info(f'pycocotools unable to run: {e}')

# Return results
model.float() # for training
if not training:
    s = f'\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to {save_dir / 'labels'}' if save_txt else "
    LOGGER.info(f"Results saved to {colorstr('bold', save_dir)}{s}")
    maps = np.zeros(nc) + map
    for i, c in enumerate(ap_class):
        maps[c] = ap[i]
    return (mp, mr, map50, map, *(loss.cpu() / len(dataloader)).tolist()), maps, t

def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--
data', type=str, default=ROOT / 'data/coco128.yaml', help='dataset.yaml path')
    parser.add_argument('--
weights', nargs='+', type=str, default=ROOT / 'yolov5s.pt', help='model.pt path(s)')

```

```

parser.add_argument('--batch-size', type=int, default=32, help='batch size')
parser.add_argument('--imgsz', '--img', '--img-
size', type=int, default=640, help='inference size (pixels)')
parser.add_argument('--conf-thres', type=float, default=0.001, help='confidence threshold')
parser.add_argument('--iou-thres', type=float, default=0.6, help='NMS IoU threshold')
parser.add_argument('--task', default='val', help='train, val, test, speed or study')
parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
parser.add_argument('--
workers', type=int, default=8, help='max dataloader workers (per RANK in DDP mode)')
parser.add_argument('--single-cls', action='store_true', help='treat as single-class dataset')
parser.add_argument('--augment', action='store_true', help='augmented inference')
parser.add_argument('--verbose', action='store_true', help='report mAP by class')
parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
parser.add_argument('--save-
hybrid', action='store_true', help='save label+prediction hybrid results to *.txt')
parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-
txt labels')
parser.add_argument('--save-json', action='store_true', help='save a COCO-
JSON results file')
parser.add_argument('--project', default=ROOT / 'runs/val', help='save to project/name')
parser.add_argument('--name', default='exp', help='save to project/name')
parser.add_argument('--exist-
ok', action='store_true', help='existing project/name ok, do not increment')
parser.add_argument('--half', action='store_true', help='use FP16 half-precision inference')
parser.add_argument('--
dnn', action='store_true', help='use OpenCV DNN for ONNX inference')
opt = parser.parse_args()
opt.data = check_yaml(opt.data) # check YAML
opt.save_json |= opt.data.endswith('coco.yaml')
opt.save_txt |= opt.save_hybrid
print_args(FILE.stem, opt)
return opt

def main(opt):
    check_requirements(requirements=ROOT / 'requirements.txt', exclude=('tensorboard', 'thop'
))

    if opt.task in ('train', 'val', 'test'): # run normally
        if opt.conf_thres > 0.001: # https://github.com/ultralytics/yolov5/issues/1466
            LOGGER.info(f'WARNING: confidence threshold {opt.conf_thres} >> 0.001 will produc
e invalid mAP values.')
        run(**vars(opt))

    else:
        weights = opt.weights if isinstance(opt.weights, list) else [opt.weights]
        opt.half = True # FP16 for fastest results
        if opt.task == 'speed': # speed benchmarks

```

```

# python val.py --task speed --data coco.yaml --batch 1 --
weights yolov5n.pt yolov5s.pt...
opt.conf_thres, opt.iou_thres, opt.save_json = 0.25, 0.45, False
for opt.weights in weights:
    run(**vars(opt), plots=False)

elif opt.task == 'study': # speed vs mAP benchmarks
    # python val.py --task study --data coco.yaml --iou 0.7 --
    weights yolov5n.pt yolov5s.pt...
    for opt.weights in weights:
        f = f'study_{Path(opt.data).stem}_{Path(opt.weights).stem}.txt' # filename to save to
        x, y = list(range(256, 1536 + 128, 128)), [] # x axis (image sizes), y axis
        for opt.imgsz in x: # img-size
            LOGGER.info(f'\nRunning {f} --imgsz {opt.imgsz}...')
            r, _, t = run(**vars(opt), plots=False)
            y.append(r + t) # results and times
        np.savetxt(f, y, fmt='%10.4g') # save
        os.system('zip -r study.zip study_*.txt')
        plot_val_study(x=x) # plot

if __name__ == "__main__":
    opt = parse_opt()
    main(opt)

```

4.12 Process of Training the Yolov5 model

Firstly, we put the data in a specified location, as shown in fig

```

%cd /content/drive/MyDrive/Tooorth_detect
Location of the dataset

```

Then we configure the parameter file to determine the training data path and validation data path, as shown in fig

```

trainimagePath = '/content/drive/MyDrive/Tooorth_detect/dataset/images/train'
valimagePath = '/content/drive/MyDrive/Tooorth_detect/dataset/images/val'

```

4.13 YOLOv5 parameter configuration

After the model and data were ready, I set the hyperparameters as shown below

Table: 4.13 Hyperparameter configuration

Name	Value
Optimizer	SGD
Initial learning rate	0.001
Batch size	16
Image size for the Input	640x480 pixels

After setting the hyperparameter, we start the training process. A part of the training process is show in fig

Epoch	gpu mem	box	obj	cls	labels	img size	
3/199	1.7G	0.09095	0.1162	0.03492	252	640: 100%	49/49
[00:07<00:00, 6.54it/s]							
Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95:	100% 37/37
[00:05<00:00, 6.56it/s]							
all	1164	5703	0.34	0.729	0.681	0.442	
Epoch	gpu mem	box	obj	cls	labels	img size	
4/199	1.7G	0.08413	0.1144	0.03451	271	640: 100%	49/49 [00:07<00:00, 6.98it/s]
Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95:	100% 37/37
[00:05<00:00, 6.96it/s]							
all	1164	5703	0.47	0.713	0.615	0.546	

After training the model, the process will be stored to path which we have given earlier, and the detecting or testing is done as part of validation.

CHAPTER – 5

RESULTS AND DISCUSSION, PERFORMANCE ANALYSIS

5.1 Results and Discussion

In order to determine the details of the tooth and the root canal one has to do parameter tuning thereafter training the model accordingly. As a result we got several visualizations of the loss values, F1 scores and m@p metrics of the both train and the test dataset. The results are discussed below

5.1 Confusion Matrix

The confusion matrix describes the classification accuracy of a classifier. Assuming that there are only two categories of classification objectives, positive and negative, the meanings of TP, FP, FN and TN are as follows:

- (1) True positions (TP): the number of positive cases correctly divided, that is, the number of instances that are actually positive cases and are classified as positive cases by the classifier;
- (2) False positions (FP): the number of instances wrongly divided into positive cases, that is, the number of instances that are actually negative but are classified as positive cases by the classifier;
- (3) False negatives (FN): the number of instances wrongly divided into negative cases, that is, the number of instances that are actually positive but classified as negative by the classifier;
- (4) True negatives (TN): the number of cases correctly divided into negative cases, that is, the number of instances that are actually negative cases and are classified as negative cases by the classifier.

where P stands for precision, and the calculation formula is the actual number of positive samples in the predicted samples / the number of all positive samples, that

is, $\text{precision} = \text{TP} / (\text{TP} + \text{FP})$ (3.1) where R stands for recall, and the calculation formula is the actual number of positive 35 samples in the forecast samples / the number of predicted samples, i.e., $\text{recall} = \text{TP} / (\text{TP} + \text{FN})$

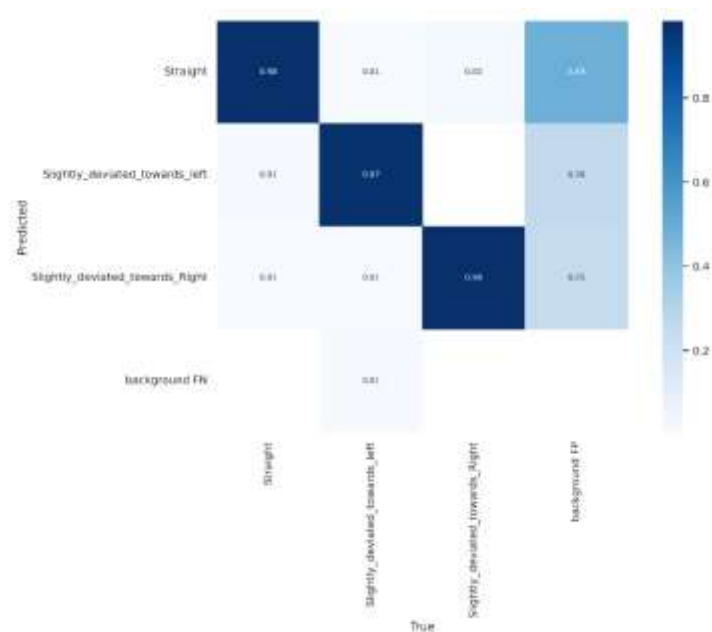


Fig 5.1 : confusion matrix for the root canals

5.2 M@P

mAP is the mean average precision, which is the average AP value of multiple categories of AP. The size of mAP must be in the interval [0,1], the larger the better. This index is a key index to measure the detection accuracy in object detection. We got an average of 0.93 m@p.

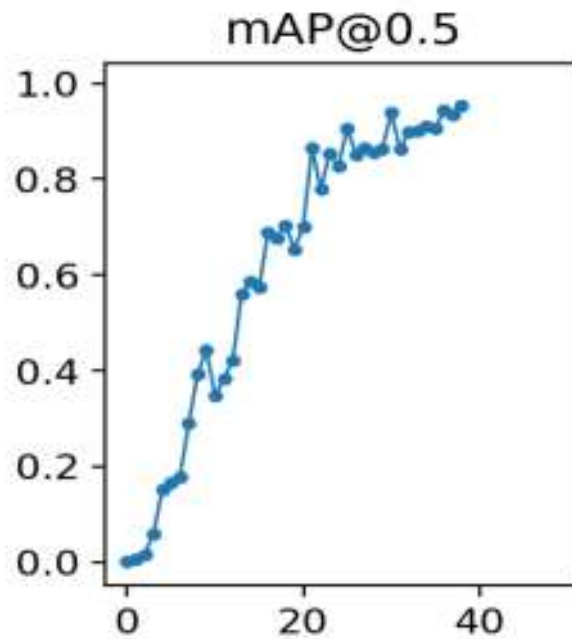


Fig 5.2 : m@p curve

5.3 Precision recall curve

By changing the recognition threshold, the system can recognize the first k images in turn. The change of the threshold value will cause the change of precision and recall values. The PR curve is given as follow. Figure 3.8 Precision-Recall curve In the multiclass classification task, we need to know the precision and recall corresponding to top-1 to top- N (n is the number of all test samples). Obviously, with more and more samples selected, recall will be higher and higher, and precision will show a downward trend. The change of recognition threshold enables the system to recognize the first k images in turn. The change of threshold value will lead to the change of precision and recall value, and then the curve can be obtained. If the performance of a classifier is better, it should have the following performance: while the recall value increases, the precision value remains at a high level. The classifier with poor performance may lose a lot of precision values in exchange for the 36 improvement of recall value.

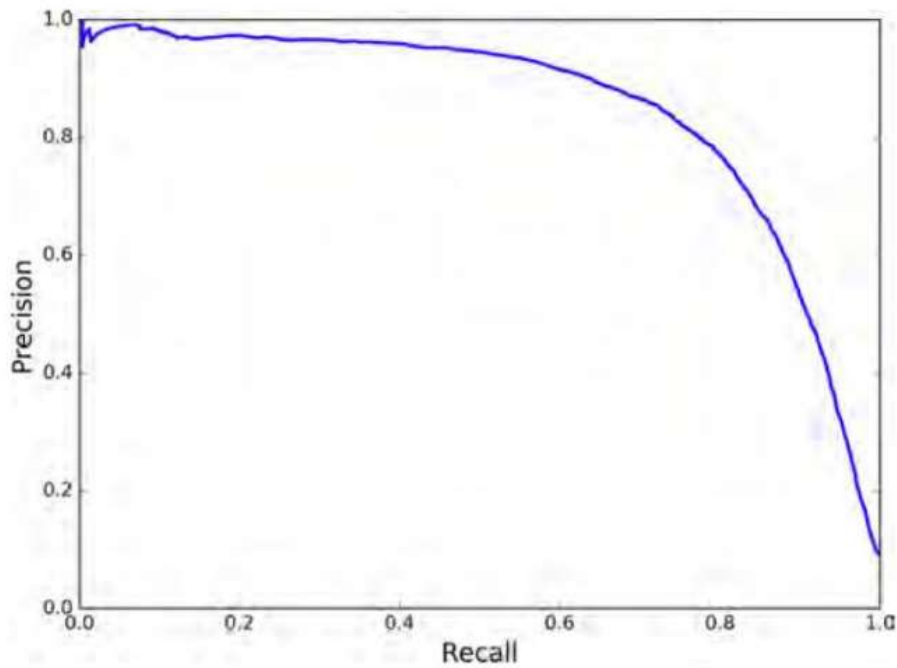


Fig 5.3: Precision -Recall curve

5.4 Loss Functions

The appropriate loss function is one of the most important steps to ensuring that the model works as expected. The main function of object detection is to locate and identify objects, while the function of loss function is to make positioning more accurate and recognition accuracy higher. Training Loss reflects the overall situation of whether different models are suitable for the selection of loss function and whether the setting of hyperparameters is reasonable. Below, we will make a comparative analysis of the results of classification loss, regression loss and overall loss of the two algorithms

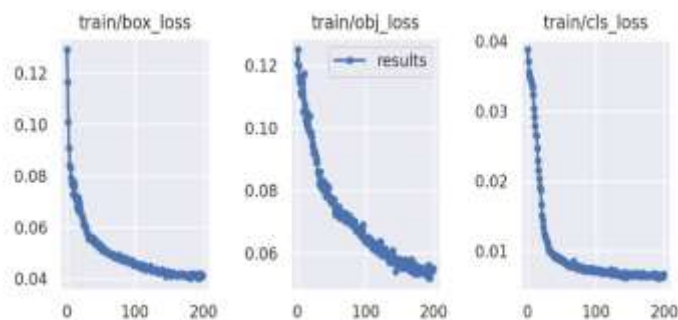


Fig 5.4: Loss functions

5.5 Discussion

From the experimental results of this project, we see that the YOLOv5 model represents the first stage algorithm is superior to the remaining two stage algorithms in most cases. The YOLOv5 model performs better in rectangular box regression. YOLO, especially the latest YOLOv5 algorithm, takes up less memory and executes faster, so it can be adapted to more devices and scenarios.

Type	YOLOv5
Training speed	26 ms/step
Execution speed	0.11s
Memory usage	14MB
mAP	0.93
Total Loss	0.032

Table 5.1 Experimental Results

YOLOv5 uses Mosaic data as input, by augmentation for input data. Each time four training images are read and flipped, scaled, changed color, gamut, etc. Data augmentation enables the model to learn to recognize objects in smaller scope, and its advantages is to enrich the background of the object detection, and the data of four images will be calculated at a time, so that the mini-batch size doesnot need to be very large, and even as single GPU can achieve better results.

In the feature extraction process, YOLOv5 adopts CSP+FPN+PAN structure. Labelimg can effectively enhance the learning ability of CNN, brings a relatively large performance improvement and also reduced the amount of computation.

Loss calculations are based on objectness score, class probability score, and bounding box regression score in the YOLO model. YOLOv5 adopts GIoU loss as bounding box loss. It calculates the loss of class probabilities and object scores using the 51 balanced_sigmoid_cross_entropy and Logits loss functions. Its advantage is that it can balance the positive and negative samples through the balance parameter, so as to get a better effect. To sum up, YOLOv5 has made various optimizations in data augmentation, feature extraction, loss function and other aspects, making it excellent in both accuracy and speed, and more suitable for more complex and diverse application scenarios.

5.7 Detection of the tooth details

The output of the detected image from an input is given as below, it detects the name of each tooth with the dimensions. The below picture shows the details of all the teeth we can able to detect accordingly by changing the number and details of the classes.

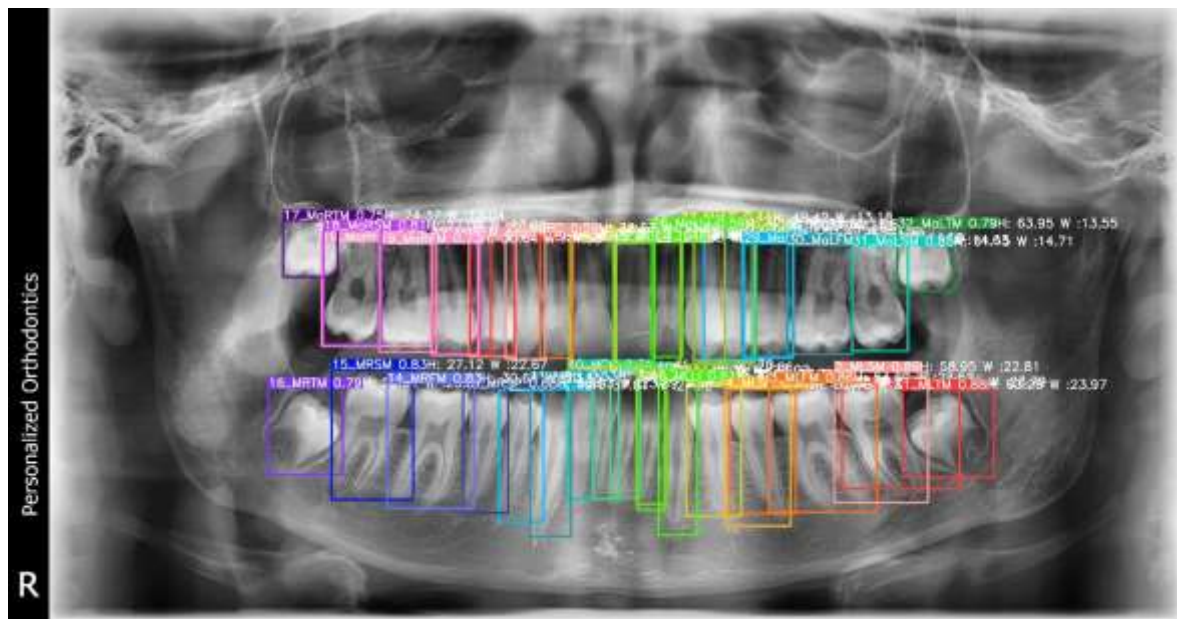


Fig 5.7 : Detected image

CHAPTER 6

SUMMARY AND CONCLUSIONS

In the field of tooth detection or recognition, compared with the traditional classification problem, object detection is obviously more in line with the practical needs. However, many of the two stage object detection algorithms with high accuracy are relatively slow and cannot meet the demand of the industry for the real time process. After five versions of iteration, the YOLO algorithm, the representative of the one stage algorithm, has gradually improved in speed and accuracy, especially in terms of accuracy, which has also surpassed some two stage algorithms.

This object detection system processed with panoramic X-ray dental images mainly aims for detecting the name of the tooth along with its properties. To achieve this, name and properties of each tooth were extracted by annotating and testing them with YOLOv5s. It uses large image database for correct object detection and recognition process. This process

References

- ultralytics. Yolo-V5. Retrieved from <https://github.com/ultralytics/yolov5>
- Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M. S., ... & Asari, V. K. (2018). The history began from AlexNet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*
- Aziz, L., bin Haji Salam, S., & Ayub, S. (2020). Exploring deep learning-based architecture, strategies, applications and current trends in generic object detection: A comprehensive review. *IEEE Access*
- Backes, A. R., Casanova, D., & Bruno, O. M. (2009). Plant leaf identification based on volumetric fractal dimension. *International Journal of Pattern Recognition and Artificial Intelligence*, 23(06), 1145-1160
- Berman, M., & Blaschko, M. B. (2017). Optimization of the Jaccard index for image segmentation with the lovász hinge. *CoRR*, *abs/1705.08790*, 5
- Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). YOLOv4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*
- Cai, Z., Fan, Q., Feris, R. S., & Vasconcelos, N. (2016). A unified multi-scale deep convolutional neural network for fast object detection. In *European Conference on Computer Vision* (pp. 354-370). Springer
- Dai, J., Li, Y., He, K., & Sun, J. (2016). R-FCN: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems* (pp. 379-387)
- Dai, K. J., & R-FCN, Y. L. (2016). Object detection via region-based fully convolutional networks. *arxiv preprint*. In *arXiv preprint*
- Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'05)* (Vol. 1, pp. 886-893)
- Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large- 55 scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition* (pp. 248-255)
- He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 770–778.

- Hoiem, D., Efros, A. A., and Hebert, M. Automatic photo popup. ACM transactions on graphics (TOG) 24, 3 (2005), 577–584.
- Hoiem, D., Efros, A. A., and Hebert, M. Geometric context from a single image. In Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on (2005), vol. 1, IEEE, pp. 654–661.
- Hoiem, D., Efros, A. A., and Hebert, M. Putting objects in perspective. International Journal of Computer Vision 80, 1 (2008), 3–15.
- Hornik, K. Approximation capabilities of multilayer feedforward networks. Neural networks 4, 2 (1991), 251–257.
- Huang, T. Computer vision: Evolution and promise. CERN EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH-REPORTSCERN (1996), 21– 26.
- Hubel, D. H., and Wiesel, T. N. Receptive fields and functional architecture of monkey striate cortex. The Journal of Physiology 195, 1 (1968), 215–243.
- Ioffe, S., and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. CoRR abs/1502.03167 (2015).
- Kaiming He, G. G. (2018, January 24). Mask R-CNN. Retrieved from arxiv.org: <https://arxiv.org/pdf/1703.06870.pdf>
- Supeshala, C. (2020, August 23). YOLO v4 or YOLO v5 or PP-YOLO? Retrieved from <https://towardsdatascience.com/yolo-v4-or-yolo-v5-or-pp-yolo-dad8e40f7109>.
- Fu, C. Y., Liu, W., Ranga, A., Tyagi, A., & Berg, A. C. (2017). DSSD: Deconvolutional single shot detector. arXiv preprint arXiv:1701.06659
- Ghazi, M. M., Yanikoglu, B., & Aptoula, E. (2017). Plant identification using deep neural networks via optimization of transfer learning parameters. Neurocomputing, 235, 228-235 .
- Ghiasi, G., Lin, T. Y., & Le, Q. V. (2019). NAS-FPN: Learning scalable feature pyramid architecture for object detection. In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (pp. 7036-7045) .
- Gidaris, S., & Komodakis, N. (2015). Object detection via a multi-region and semantic segmentation-aware CNN model. In Proceedings of IEEE International Conference on Computer Vision (pp. 1134-1142) .