Comparison of Deep Learning-based Models for Oral Disease Detection

BY

Peeranat Thongsakul

A PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF BACHELOR OF ENGINEERING IN BIOMEDICAL ENGINEER

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2023

SCHOOL OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

PROJECT CERTIFICATE

Project Title	Comparison of Deep Learning-based Models for Oral Disease Detection
Student Name	Mr. Peeranat Thongsakul Student ID. 63011249
Degree	Bachelor of Engineering in Biomedical Engineer

Project Advisor Signed: _____

(Dr. May Phu Paing)

Committee Signed: _____

(Assoc. Prof. Dr. Matthew Paul Gleeson)

Committee Signed: _____

(Asst. Prof. Dr. Jamie Alexander O'Reilly)

Committee Signed: _____

(Asst. Prof. Dr. Pimkhuan Hannanta-an)

Head of Department Signed: _____

(Assoc. Prof. Dr. Sarinporn Visitsattapongse)

Project Title	Comparison of Deep Learning-based Models for Oral Disease Detection
Student Name	Mr. Peeranat Thongsakul
Degree	Bachelor of Biomedical engineering
Project Advisor	Dr. May Phu Paing
Project Co-Advisor	-
Academic Years	2023

ABSTRACT

In contemporary dentistry, oral object detection is essential for a variety of uses, including automated dental caries diagnosis and orthodontic treatment planning. In order to find the best strategy for dental image analysis, this thesis provides a thorough comparison of several oral object detection techniques. Three cutting-edge deep learning models, namely You Only Look Once or YOLO (especially YOLO V8 and YOLO-NAS), Detection Transformer or DETR and Detectron2 were implemented, and their performances were compared and contrast aiming to select the most effective model for dental radiograph image datasets.

An opened dataset of 936 oral x-ray images along with the expert annotations were applied for input dataset. After that, each of the aforementioned detection models were trained and performance were evaluated in terms of precision, recall, F1-score, and inference speed. Experimental findings demonstrated that Detectron2 outperforms both YOLO and DETR in terms of detection accuracy, achieving an accuracy of 0.97 and total loss of 0.3716, while maintaining real-time inference capabilities. Furthermore, mobile apps also developed to port the model into it and test with android studio.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to all those who have supported and guided me throughout the journey of completing this thesis. I am immensely grateful to my thesis advisor, Dr. May Phu Paing, for her unwavering support, invaluable insights, and dedication to helping me develop this research. Her mentorship has been instrumental in shaping the direction and quality of this thesis.

To my friends and family, I extend my gratitude for their unwavering encouragement and understanding throughout this demanding process. Their emotional support was the driving force behind my determination to complete this work.

Last but not least, I want to thank all the participants in my study for their willingness to share their experiences and insights, which were critical to the success of this research.

Peeranat Thongsakul

TABLE OF CONTENTS

	Page
ABSTRACT	(i)
ACKNOWLEDGEMENTS	(ii)
LIST OF TABLES	(v)
LIST OF FIGURES	(vi)
LIST OF SYMBOLS/ABBREVIATIONS	(viii)
CHAPTER 1 INTRODUCTION	
1.1 Background and significant of the study	1
1.2 Objectives	2
1.3 Scope	2
1.4 Report outline	2
CHAPTER 2 REVIEW OF THEORY RELATED	3
2.1 Oral health	3
2.2 Deep learning	8
2.1.1 Object detection	8
2.3 YOLO	10
2.4 End-to-End Object DEtection with Transformers (DETR)	21
2.5 Detectron2	28
2.6 Android application	29
CHAPTER 3 METHODOLOGY	38
3.1 Introduction	38
3.2 Design Methodology	38

3.3 Interesting problems 42
3.4 Proposed Solution 43
3.5 Summary 43

CHAPTER 4 EXPERIMENTAL RESULT AND DISCUSSION 44
4.1 Introduction 44
4.2 Result 44
4.3 Discussion 50
4.4 Summary 50

CHAPTER 5 CONCLUSION 52
5.1 Introduction 52
5.2 Summary 52
5.3 Conclusions 52
5.4 Suggestions 53

REFERENCES 54

LIST OF TABLES

Tables	Page
3.1: Comparison of each model	41
3.2: Setting of YOLO V8	44
3.3: Setting of YOLO-NAS	46
3.4: Setting of DETR	47
3.5: Setting of Detectron2	48

LIST OF FIGURES

Figures	Page
2.1: Characteristics and Economic background	7
2.2: Object detection algorithm	9
2.3: YOLO Backbone network	11
2.4: YOLO V1 convolution network	12
2.5: Evaluate the performance of YOLOv2 trained on Visual Object Classes (VOC) 2012 challenge with different input dimensions against its counterpart detectors	13
2.6: Convolution network of YOLO v3	16
2.7: Overall structure of YOLOv4, including CSPDarknet (backbone), SPPnet, PANet and 3 YOLO heads.	17
2.8: YOLO-v6 model base architecture	19
2.9: Relative evaluation of YOLO-v6 vs. YOLO-V5	20
2.10: YOLO-v7 comparison vs. other object detectors	20
2.11: DETR directly predicts (in parallel) the final set of detections by combining a common CNN with a transformer architecture. During training, bipartite matching uniquely assigns predictions with ground truth boxes.	22
2.12: DETR uses a conventional CNN backbone to learn a 2D representation of an input image. The model flattens it and supplements it with a positional encoding before passing it into a transformer encoder. A transformer decoder then takes as input a small, fixed number of learned positional embeddings, which call object queries, and additionally attends to the encoder output. Then pass each output embedding of the decoder to a shared feed forward network (FFN) that predicts either a detection (class and bounding box) or a “no object” class.	24
2.13: Structure of android application	32
2.14: The position of Dalvik virtual machine in Android system	33
2.15: The lifecycle of an activity	35

3.1: Google-colab program and example of code for YOLO-nas 39
3.2: Google-colab ask permission to access to google drive 39
3.3: Example of color images from dataset 40
3.4: Example of mobile app on android studio with build-in model 42
4.1: Result and confusion matrix of YOLO V8 45
4.2: Test result of YOLO V8 45
4.3: Raw result from YOLO-NAS 46
4.4: Graph showing mAP and precision from YOLO-NAS 46
4.5: Test result from YOLO-NAS 47
4.6: Raw result from DETR 47
4.7: Test result from DETR 48
4.8: Raw and graph result from Detectron2 49
4.9: Test result from Android studio 50

LIST OF SYMBOLS/ABBREVIATIONS

Symbols/Abbreviations	Terms
YOLO	You Only Look Once
NAS	Neural Architecture Search
DETR	Detection Transformer
WHO	World Health Organization
R-CNN	Region-based Convolution Neural Networks
IOU	Intersection Over Union
GIOU	Generalized Intersection Over Union
SIOU	Scale-Invariant Intersection Over Union
mAP	mean Average Precision
SPP	Spatial Pyramid Pooling
PAN	Path Aggregation Network

CHAPTER 1
INTRODUCTION

This chapter introduces the report's main concepts and situates the motivation for the work. Following that, the rationale and goals for the project's investigation are discussed, followed by a summary of the overall project. Finally, a chapter-by-chapter overview of the dissertation is provided.

1.1 Background and significance of the study

The tooth, a hard bone structure in the mouth, being the organ involved in the digestion process by breaking down food into smaller ones via chewing and mixing with saliva, serve a major role in the digestion system. Moreover, teeth also act as impression of person as people smile and talk to other people. Unhealthy teeth can lead to a range of illnesses, such as gum infection, cavity, and oral cancer, all of which pose serious risks to digestion system or one's well-being.

Tooth decay results from many factors, conversion of sugar into acid by bacteria inside human's mouth is one of the examples and can occur by the consumable routine. According to the World Health Organization (WHO), approximately 3.5 billion people around the world suffer with oral disorder. Among these individuals, a significant portion resides in middle-income countries, primarily due to insufficient exposure to fluoride and excessive consumption of sugary foods [1].

In Thailand, a study indicated four key factors contributing to tooth loss among Thai individuals. Firstly, by Muay Thai, researcher shows that 23.5% of 260 Muay Thai athletes experienced injuries to their oral and jaw regions. Secondly, Thai people tend to not seek to dentist for dental services. Research shows that 63.9% of 1500 group sample did not visit dentist in the preceding year. Thirdly, tooth decay results from second factors, causing loss of tooth and reduce efficient of chewing and digestion system. Lastly, behavior like Betel Quid chewing, techniques, and smoking were identified as influencing tooth loss patterns [2].

Another important factor is cost, which is shared by all 658 dental undergraduates at Mahidol University in Thailand. The estimated adjusted cost of becoming a dentist range between 1.2 and 1.8 million Baht. Such education costs result in high treatment costs [3].

1.2 Objectives

The project's goal is to develop a suitable model that can detect oral disorder (pocket and cavity) from x-ray oral image with high accuracy and precision. Deep learning-based object detection models will be implemented and compare in order to select the best one. Furthermore, this project also aims to develop a mobile application which can be applicable via a smartphone to detect oral diseases for convenient, while still keeping high accurate and precision.

1.3 Scope of the study

The oral tooth x-ray images that show a cavity or pocket will be the main focus of this project. For added difficulty, a portion of the image will include both. The best model for object detection will be chosen. Finally, a suitable program that is compatible to the model and can be use on android smartphone.

1.4 Report outline

The remainder of this report is structured as follows:

Chapter 2 examines the fundamental knowledge about oral diseases, the state-of-the-art deep learning-based object detection models, and other related topics.

Chapter 3 discusses the design and implementation of object detection.

Chapter 4 demonstrates how accurate and precise each model as well as mobile application testing is.

Chapter 5 concludes the report by reviewing the work done and drawing conclusions about key aspects of the work done. Finally, future work is discussed, with an emphasis on improving the model and mobile application.

CHAPTER 2
REVIEW OF THEORY RELATED

This chapter reviews the theory and information related to the project, consisting of oral health, deep learning-based object detection models that are used for this project, and finally, the implementation of mobile application.

2.1 Oral health

Given its significant implications, the distinction made by separating oral health from the full range of physical well-being needs to be reconsidered. The effects of oral health go beyond simple dental issues; they affect general health by causing severe pain, distress, and changes in dietary practices, speech patterns, and general quality of life. Furthermore, it is important to remember that oral health is important in the context of many chronic diseases. The consequences of ignoring the social and material foundations that support oral health are highlighted by the persistence of persistent toothache and the resulting decreased quality of life. In light of this, a paradigm shift in health policies is encouraged, where oral health is assimilated into the framework of more extensive health promotion programs. Through the use of common risk factor methodology and occidental methodologies for needs assessment, which both serve as the cornerstones of comprehensive health promotion, this holistic incorporation can be achieved. Oral diseases are the most common chronic illnesses, as evidenced by their prevalence and societal impact. Their prominence as public health challenges stems from their extreme prevalence, consequences for the individual and the community, and high cost of care. The core issue is highlighted by the knowledge that the causes of oral diseases are related to risk factors that are frequently shared by a variety of chronic illnesses, including dietary and hygiene practices as well as behaviors like drinking alcohol, smoking, and engaging in injury-prone behavior. It is crucial to remember that humans have effective public health methodologies at human's disposal that can be used to reduce the prevalence of oral diseases and their related effects.

According to some countries' financial burden rankings, oral diseases are the fourth most expensive health conditions. To put this in perspective, consider that the projected cost to address caries, which comes to USD $3513 per 1000 children, exceeds the total health budget allotted for children in a significant number of low-income countries. For adults living in

developing countries, the situation takes on an aggravating complexion as they deal with the compounding effects of untreated oral diseases' unchecked progression. Unfortunately, there aren't many efficient dental care systems available that are ready to deal with this crisis, especially for people who live in low-resource areas. When such systems are present, the associated costs are frequently insurmountable obstacles for the majority of the population. Millions of people live with untreated caries and experience cavities and suppuration. However, planners continue to push oral diseases to the margins of their considerations despite the obvious effects on costs and the degradation of life quality. Even though it is unintentional, this neglect has the potential to start a cascading trajectory marked by increasing decay and the eventual need for expensive but ineffective clinical interventions. To stop this unfavorable course, a firm commitment must be made to correct this oversight.

Oral health has a wide range of effects on people, affecting both their physical and psychological well-being, and consequently having an impact on a wide range of aspects of life. Growth, personal satisfaction, appearance, speech, mastication, gustatory experiences, and social interactions are all impacted by oral health. Additionally, it encompasses feelings related to one's sense of community and belonging in the context of social well-being. When it comes to severe caries, its negative effects have a long-lasting impact on the way that kids live their lives. The price paid includes the lingering specter of acute and chronic infections as well as pain, discomfort, and disfigurement. This condition's symptoms include disturbed eating and sleeping habits, which increases the risk of hospitalization, increases the cost of treatment, and significantly lowers school attendance. As a result, one's ability to participate in efficient learning is compromised. Additionally, nutrition, growth, and weight gain are all affected by the cascading effects. According to research, children as young as three who had nursing caries weighed almost 1 kg less than their peers. This is because toothache and infection cause disruptions in eating habits, sleep schedules, dietary patterns, and metabolic processes. Additionally, the disruptions in sleep patterns have an impact on glucocorticoids production, adding to the complex chain of events. On a different axis, the effect also has hematological effects, with the suppression of hemoglobin levels resulting from a concurrent drop in erythrocyte production. The cumulative cost highlights the importance of oral health in the microcosm of individual experiences as well as in the larger context of public health considerations in the comprehensive assessment of these numerous repercussions.

Ninety percent of pre-adolescents expressed the possibility of consequences related to their oral health status. The prevalence of dental pain was estimated to be around 33% among Brazilian teenagers [4], with 9% of this group reporting the distressing manifestation of excruciating pain. A tangible marker of a child's general health status, toothaches emanating from such dental conditions lead to a trajectory that culminates in absenteeism from educational institutions. Notably, the burden continues to be significant even in environments with relatively lower caries rates, like the United States. The quantification of lost school time—117,000 hours per 100,000 kids—due to dental visits or other related issues serves as a stark reminder of the situation. In this context, it is crucial to emphasize the implication that school dental services, which are typically available during school hours, unintentionally worsen the situation. This has significant implications for marginalized groups in society who suffer disproportionately from amplified loss of schooling due to high caries rates. This portrayal emphasizes the relationship between oral health and wider socioeconomic inequalities, which is crucial to the conversation about public health issues.

In the Ban Phang district of Khon Kaen, Thailand [5], a comprehensive survey of 501 people, ages 35 to 44, living in 16 rural villages was conducted. The main goal was to evaluate how oral performance affected different aspects of performance in the areas of physical, psychological, and social domains. This evaluation covered the previous six months, and it was later supported by thorough oral examinations. When clinical and behavioral data were combined, interesting patterns emerged, including a low incidence of caries (DMFT = 2.7) and little use of dental services. A startling 73.6% of the population in the sample reported oral influences on their daily performances, which is particularly significant. The areas of eating (49.7%), emotional stability (46.5%), and smiling (26.1%) showed the highest frequencies of impact. A pronounced prevalence of impacts was contrasted with a low degree of severity within the scope of eating, emotional stability, and teeth cleaning. In contrast, performances with low frequency, like sleeping and participating in physical activities, were linked to higher severity ratings. For the majority of performances (40.1%), the perceptible causative factors underlying these impacts were primarily attributed to unpleasant feelings of pain. Notably, toothache emerged as the most common oral ailment (32.7%), resonating across almost all performance dimensions. Despite the low prevalence of caries, it was concluded from these observations that people in this demographic exhibit a parallel

frequency of oral impacts similar to populations located in industrialized contexts with elevated dental disease rates. The interaction of frequency and severity produced complex patterns that highlighted how impacts on various performances can take many different forms. This paradoxical interplay necessitates a careful analysis of both parameters in the thorough evaluation of overall impact. These results highlight the necessity of a comprehensive assessment that takes into account how oral health interacts with other aspects of wellbeing. For children, a significant 89.8% of the children in the cohort under study showed signs of one or more oral impacts. The median impact score for this phenomenon was 7.6, and the corresponding mean score was 8.8. It's noteworthy that nearly half (47.0%) of the kids who showed impacts said they happened at low or moderate levels of intensity. Out of a possible 8, a significant majority of those coping with impacts (84.8%) had 1-4 of their daily performances disrupted. The analysis of particular performances highlights a broad range of impacts. Notably, eating emerged as the performance area where children were most frequently negatively impacted, accounting for 72.9% of the children. When it comes to the severity of impacts, a dichotomy emerges: Study and social contact were marked by relatively low levels of impact, whereas eating and smiling emerged as performances with heightened severity. The primary causal factors for these impacts' etiology, which include a variety of clinical conditions, have clinical resonance. Among these, sensitive teeth (27.9%), oral ulcers (25.8%), toothaches (25.1%), and primary tooth exfoliation (23.4%) naturally occur are noteworthy. The multifaceted picture revealed by this evaluation highlights the complex interactions between oral health and various facets of childhood experiences, highlighting the holistic nature of oral health's influence on people in this demographic.

The problem of oral health also affects economy of Thailand, for example, over the past ten years, 18 dental schools have licensed between 500 and 700 new dentists annually in Thailand. The Dentistry Faculty of Mahidol University plays a significant role in this, producing about 20% of all dental school graduates each year [6]. However, a significant financial commitment was acquired in order to graduate. The figure below provided the data of 486 samples students from Mahidol University with 75.0% response rate.

	N	%
Total	486	100%
Gender		
Male	137	28.2%
Female	349	71.8%
Place of residence		
Bangkok Metropolitan Region	310	63.8%
Provinces (Municipality)	117	24.1%
Suburb	59	12.1%
Student year		
Year 1	74	15.2%
Year 2	126	21.8%
Year 3	71	14.6%
Year 4	77	15.8%
Year 5	49	10.1%
Year 6	109	22.4%
Source of financing		
Fully Family	453	93.2%
Family support with Loan	12	2.5%
Family support with Scholarship	16	3.3%
N/A	5	1%
Household income (THB)		
<10,000 (>424 USD)	3	0.6%
10,000–15,000 (424–635 USD)	3	0.6%
15,001–30,000 (645.1–1,270 USD)	25	5.1%
30,001–50,000 (1,270.1–2,118 USD)	73	15.0%
50,001–100,000 (2,118.1–4,235 USD)	214	43.9%
>100,000 (>4235 USD)	168	34.5%

Fig 2.1: Characteristics and Economic background [6]

The current health paradigms support a thorough redefinition of oral health that integrates it with the more general concepts of physical, psychological, and social well-being, all of which are reliant on an individual's oral status. In line with this holistic viewpoint, Cohen and Jago emphasize that improving quality of life is dentistry's primary function and its most significant contribution. As a result, the evaluation of oral health takes into account factors such as physical, psychological, and social disturbances. This triadic evaluation framework emphasizes how important these factors are when evaluating oral health. Conventional metrics for assessment primarily rely on clinical indices to assess oral health. However, there is a parallel approach that uses metrics based on the idea of oral health-related quality of life, seamlessly lining up with the sociodental approaches used for the need assessment. The evolving ethos that seeks to encompass the whole human experience within the scope of healthcare considerations finds resonance with this shift toward including the myriad facets of well-being in the evaluation of oral health.

In developing countries, the prevalence of chronic illnesses—including ailments like obesity, diabetes, and tooth decay—is rising. This developing environment suggests that not only the quality of life related to oral health, but also the wider scope of overall quality of life, may suffer decline. This new reality emphasizes the intricate relationship between oral health and a variety of chronic diseases, underscoring the need to give the common risk factor paradigm more weight. The future strategies that will shape the landscape of oral health rely heavily on integration with the common risk factor approach as a foundation. One of its notable benefits is that it makes a concerted effort to improve overall health conditions for

the population as a whole as well as the subgroups with higher susceptibility. This makes it possible to reduce the pervasive disparities that highlight social injustices. This point in the integration trajectory within the overarching health promotion strategies is crucial. Health planners have the power to significantly increase the impact on both the area of general health and the dimensions associated with oral well-being by incorporating oral health considerations into the larger framework of strategies aimed at advancing general well-being and by endorsing the sociodental methodologies for assessing oral needs. This integrated paradigm captures the essence of comprehensive healthcare planning, where a synergistic approach fosters a holistic improvement in health and well-being while also elevating individual domains.

2.2 Deep learning

Deep learning is a subset of artificial intelligence, which is an imitation of human brain's observe, think, learn and analyst to be a virtual brain for many purposes. It is an automating method that machine exaction of represent from raw data, like image, text or video, for more specific, feature from data [7]. Deep learning has many uses such as for object detection like mask detection on people or object classification, machine translation, autonomous driving and so on. Deep learning algorithms require a lot of supervised or unsupervised high-quality data to automatically extract complex features. The major purpose of this project, which is oral disease detection, is a type of object detection task, thus it will be mainly focused on the next sub sessions.

2.1.1 Objection detection

Objection detection, as a name suggest, is a method that uses computers program to detect a specific object, for instance, face detection, animal detection, traffic detection, etc. [8]. At present, object detection methods have two main categories of two, machine learning and deep learning.

When comparing, deep learning tends to have better performance than traditional machine learning. Object detection has prominently aligned itself with the paradigm shift towards deep learning approaches in the context of the quick and significant breakthroughs in deep learning over the last decade. A watershed moment in this evolution occurred in 2014, when Girchick R. and colleagues released the R-CNN

model [9], which marked the first use of deep learning techniques in object detection. As a result, the Fast R-CNN and Faster R-CNN models emerged in 2015 as modifications and enhancements to the R-CNN framework. These advancements significantly improved the R-CNN approach's computing efficiency and training efficacy. In the same year, Redmon J. and associates created YOLO paradigm [10], a ground-breaking feat in enabling real-time object monitoring. In 2017, YOLO version 2 was released, which was a significant revision aiming at refining the YOLO model and increasing its accuracy to new heights. Building on these developments, the year 2018 saw the release of the Mask R-CNN and YOLO version 3, which were enhanced variations of the Faster R-CNN and YOLO version 2 models, respectively. These advancements represent the unwavering pursuit of perfection in the field of object identification, driven by the ever-changing environment of deep learning approaches.

Object detection require raw data consist of a location of target object or object localization. Therefore, the stages of preparing data are mainly divide into three stages: informative region selection, feature extraction and classification [15].

Informative region selection: Each image has unique object position and has a difference aspect ratios or size; it is advised to scan a whole image using multi-scale sliding window. An insufficient of applying multi-scale sliding window can be result of unsatisfactory regions.

Feature extraction: Each object needs to extract features to representations that are both semantic and resilient.

Classification: For an image that has many objects, classification is required to categories each type of object for training.

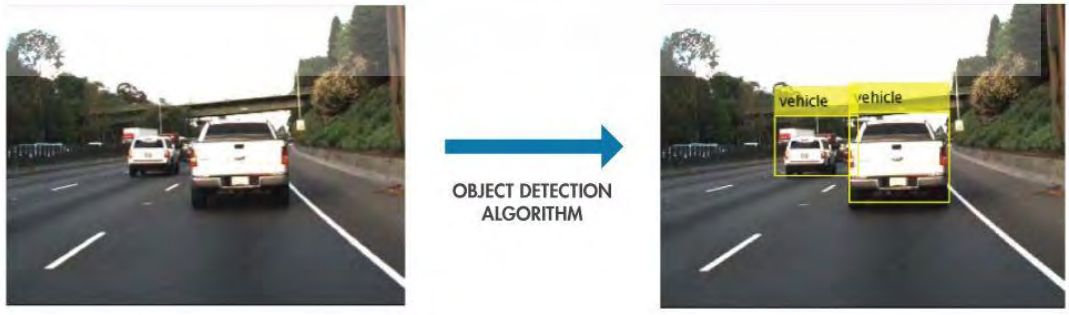


Fig 2.2: Object detection algorithm [15]

Object require process information namely “dataset” which include training dataset, validation dataset and testing dataset.

Train dataset: Text or image samples are available for the model to employ in its learning process.

Validation dataset: The model uses another set of samples for evaluation and hyperparameter adjustment; these samples must match the training dataset. The model cannot "learn" from this; instead, it must "develop" from it in order to become more effective.

Test dataset: As the name implies, these samples are used to assess the model's ability to fit once training is complete.

2.3 You Only Look Once (YOLO)

The inception of the initial YOLO version was introduced by Redmon et al [10]. in the year 2015. Over the preceding years, numerous iterations of YOLO have been disseminated in academic literature, denoted as YOLO V2, YOLO V3, YOLO V4, and YOLO V5. Additionally, there exist a handful of refined and constrained variants, exemplified by YOLO-LITE. It is pertinent to emphasize that this research study exclusively centers its attention on the five principal iterations of YOLO.

The forthcoming sections of this paper are dedicated to an exhaustive examination of the salient distinctions among these versions.

The YOLO target identification algorithm's essence lies in its small model size and quick processing capabilities. YOLO's architectural design is distinguished by its simplicity, as it is capable of directly providing positional coordinates and category labels of bounding boxes via its neural network. YOLO's outstanding speed is due to its capacity to process photos within the network and quickly produce the definite detection results. As a result, YOLO is capable of performing real-time video detection. Furthermore, YOLO takes a comprehensive approach to detection by using the complete image for identification, encompassing global contextual information, and reducing the tendency to misclassify background items as objects [10].

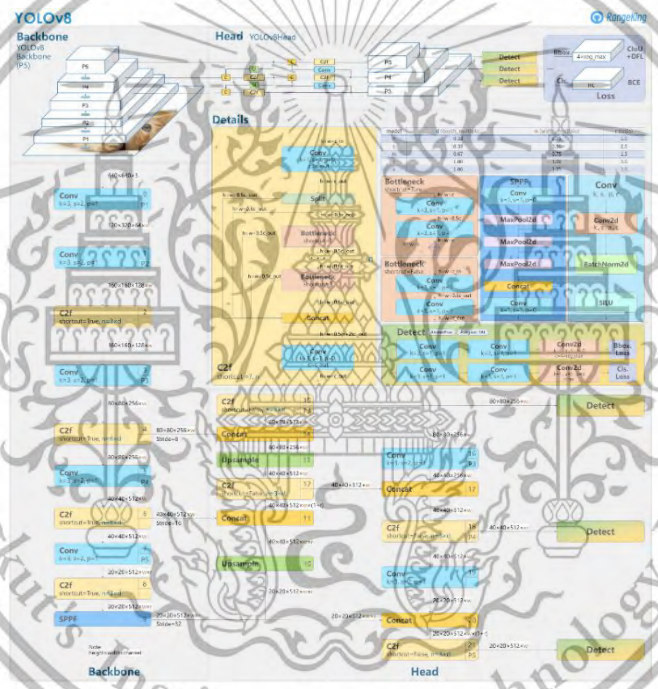


Fig 2.3: YOLO Backbone network [10]

YOLO's powerful generalization capacity is another distinguishing property, allowing it to gain highly generalized properties that can be extrapolated to varied domains. This revolutionary technique turns target detection into a regression problem. However, it is worth noting that there is still space for improvement in detection accuracy. When confronted with things in close proximity or grouped together, YOLO's performance tends to suffer. This decreased efficacy is mostly due to the prediction of only two boxes per grid and their assignment to a single category, resulting in abnormal aspect ratios and other restrictions, including generalization issues.

The improvement of detection efficiency is mostly dependent on fixing positional errors, with a focus on increasing detection of both large and tiny objects. The skillful design of the loss function, which must strike a delicate balance among these three key characteristics, is a critical part of this advancement. YOLO takes a multi-scale approach by using numerous subsampling layers. This intentional use of lower sampling layers aids in the capture of more complete target features from the network, resulting in an improvement in overall detection performance.

The original YOLO design is made up of 24 convolution layers that are followed by two completely connected layers. The YOLO method includes predicting numerous bounding boxes per grid cell and then picking the bounding boxes with the highest Intersection Over Union (IOU) with the ground truth. Non-maximum suppression is a method that serves to refine the final list of identified items.

Fig 2.4: YOLO V1 convolution network [10].

However, YOLO has two significant flaws: inaccurate item location and a lower recall rate when compared to techniques based on area-based recommendations. To address these flaws, YOLO V2 was largely aimed to improve in these two aspects. Importantly, YOLO V2 does this without deepening or enlarging the neural network, but rather by streamlining its architecture.

YOLO V2 is distinguished by two critical enhancements: increased accuracy and faster processing speed, which encompass the fundamental goals of this iteration.

Batch normalization is a technique used in YOLO V2 to standardize the input of each layer inside the network. This approach has various advantages, including faster convergence rates, lower loss, and a 2% increase in mean Average Precision (mAP).

Another notable improvement in YOLO V2 is the addition of a high-resolution classifier. The network initially trains images of size 224*224 pixels in the original YOLO architecture before shifting to 448*448 pixels during the detection phase. The switch from a classification-oriented to a detection-oriented approach ensures that the network responds correctly. This transition is carried out in two steps by YOLO V2: initially, it trains on 224*224 pixels images for 160 epochs, followed by an adjustment to 448*448 pixels images for an additional 10 epochs of training. This method improves network performance for object detection workloads.

A major enhancement to YOLO V2 is the addition of a vital layer meant to connect the previous layer's 2626 feature map with the current layer's 1313 feature map. This architectural change is motivated by the realization that the 1313 feature map is sufficient for anticipating huge objects. When it comes to anticipating smaller items, however, the traditional approach may not always be practical and clear-cut.

Fig 2.5: Evaluate the performance of YOLOv2 trained on Visual Object Classes (VOC) 2012 challenge with different input dimensions against its counterpart detectors [10].

The difficulty arises from the fact that smaller items may vanish when they pass through numerous layers of convolution and merge. As a result, it is critical to include the prior layer's broader contextual information into the current layer, essentially combining these features. This architecture change addresses the challenges associated with predicting smaller objects and contributes to the network's overall efficacy and interpretability.

The use of a multi-scale training approach in YOLO V2 enables the same network to recognize images of varied resolutions. While it is true that training with bigger input sizes results in a slower training tempo, the same strategy also results in faster training speeds when using smaller input sizes. Multi-scale training has been shown to improve detection accuracy over a range of input sizes. As a result, it strikes a respectable balance between accuracy and speed, meeting the different objectives of the detection task while maximizing network performance overall.

The training network in YOLO originally used GoogleNet as its underlying architecture. A quick comparison of GoogleNet and VGG16 demonstrates that GoogleNet has higher computational efficiency, requiring 8.25 billion operations compared to 30.69 billion operations for VGG16. However, when it comes to ImageNet classification accuracy, GoogleNet lags behind VGG16 somewhat, reaching 88% accuracy versus VGG16's 90%.

As the principal network of YOLO V2, a novel categorization model was employ known as Darknet-19. Its final network configuration, which has a comparatively low computing need of 5.58 billion operations. Darknet-19 is made up of 19 convolutional layers and 5 maximum pooling layers. Darknet-19 uses fewer convolutional and fully connected layers than YOLO V1, which used GoogleNet with 24 convolutional layers and two fully connected layers. This decrease in the number of convolutional processes within Darknet-19 is critical to reducing computational overhead in YOLO. Finally, the inclusion of an average pooling layer eliminates the requirement for an entire connection layer in the prediction process, contributing to the network's computational efficiency optimization.

The classification training process for YOLO V2 primarily involves two steps, both of which are pre-trained on the ImageNet dataset:

1. Initial Training: The network is trained for 160 epochs using the ImageNet dataset during this phase. The input image size is set to 224×224 pixels during training, and the initial learning rate is set to 0.1. To enrich the training data, standard data augmentation techniques such as random cropping, rotation, and chroma and brightness modifications are used.

2. Fine-tuning: After the initial training, fine-tuning is performed. The network is fine-tuned at this step with larger input images of size 448×448 pixels. Except for changes to the number of training epochs and the learning rate, all network parameters stay identical. The learning rate is dropped to 0.001, and the network is trained for an extra ten cycles.

According to the findings of this training process, the network obtains an accuracy of 76.5% for top-1 classification and 93.3% for top-5 classification after fine-tuning. These results are a significant improvement over the original training approach, in which Darknet-19 attained top-1 and top-5 accuracies of 72.9% and 91.2%, respectively.

For the training of YOLO V3 for object detection, a series of modifications and improvements were made compared to YOLO V2:

1. Layer Changes: In YOLO V3, the network's last convolution layer was eliminated, and three convolution layers with 1024 filters each were added. Each of these convolution layers is linked to the 11 convolution layers that came before it. The goal of this structural adjustment is to improve the network's feature extraction capabilities.

2. Category Probabilities per Box: Unlike YOLO V2, which assigned category probabilities to the entire cell and shared them with both bounding boxes inside that cell, YOLO V3 distributes category probabilities to individual boxes. This means that each box has its own set of category probabilities, which improves the model's ability to differentiate between distinct item classes.

3. YOLO V3 employs multi-scale detection techniques for object detection. It employs feature maps of three different scales, which correlate to input sizes of (416×416), (13×13), (26×26), and (52×52). YOLO V3 uses three prior boxes for each position in these feature maps, for a total of nine prior boxes. K-means clustering is used to determine these previous boxes, which are then distributed among the three size feature maps. For feature maps with bigger scales, smaller prior boxes are used, which improves the model's capacity to recognize objects of diverse sizes.

4. The residual model architecture is used by the feature extraction network in YOLO V3. This differs from YOLO V2, which used Darknet-19 with 19 convolutional layers. Darknet-53, the feature extraction network in YOLO V3, is made up of 53 convolutional layers. This more complicated network architecture is designed to collect more complex and hierarchical features, hence improving the model's overall detection capabilities.

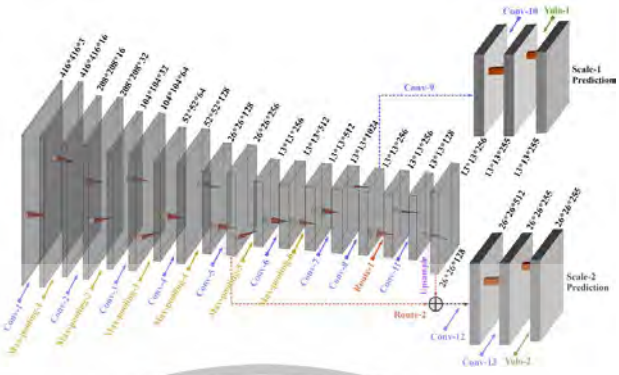


Fig 2.6: Convolution network of YOLO v3 [10].

These changes and improvements lead to YOLO V3's enhanced performance and accuracy in object identification tasks when compared to its predecessor, YOLO V2.

The YOLO V4 design is a dramatic divergence from its predecessors, with a focus on data comparison and significant performance increases. It can be summed up as YOLO V4 = CSP Darknet53 + SPP + PAN + YOLO V3. The following are YOLO V4's significant contributions and innovations:

1. Efficient Target identification Model: In YOLO V4, an efficient and highly effective target identification model is introduced. This model is intended to be simply trainable while having remarkable object detection speed and accuracy.

2. YOLO V4 extensively evaluates the state-of-the-art (SOTA) methodologies known as "bag-of-freebies" and "bag-of-specials" during the detector training process. These methods have been found to have a considerable impact on the model's performance.

3. SOTA Method Improvements: YOLO V4 improves SOTA approaches to make them more efficient and suited for training on a single GPU. Cross-Stage Partial Networks (CSP), Spatial Pyramid Pooling (SPP), Path Aggregation Network (PAN), and Spatial Attention Module (SAM) are examples of such approaches. These enhancements contribute to the model's overall efficiency and effectiveness.

The YOLO V4 target detector framework is fundamentally restructured, with four different components: input, backbone, neck, and head. One of the most significant differences between YOLO V4 and YOLO V3 is the shift in anchor point responsibilities. A single anchor point was responsible for one ground truth object in YOLO V3, whereas many anchor points jointly handle a single ground truth object in YOLO V4. Although the total

number of anchor points remains constant, this shift raises the percentage of positive sample selections, successfully reducing the issue of positive/negative sample imbalance.

This advancement has various advantages. For starters, it eliminates grid sensitivity as an open time interval since the range of the sigmoid function allows for more exact localization of object boundaries. Furthermore, YOLO V4 employs the CIOU (Complete Intersection over Union) loss function, which has a fast convergence rate and successfully addresses the problem of bounding boxes containing ground truth objects.

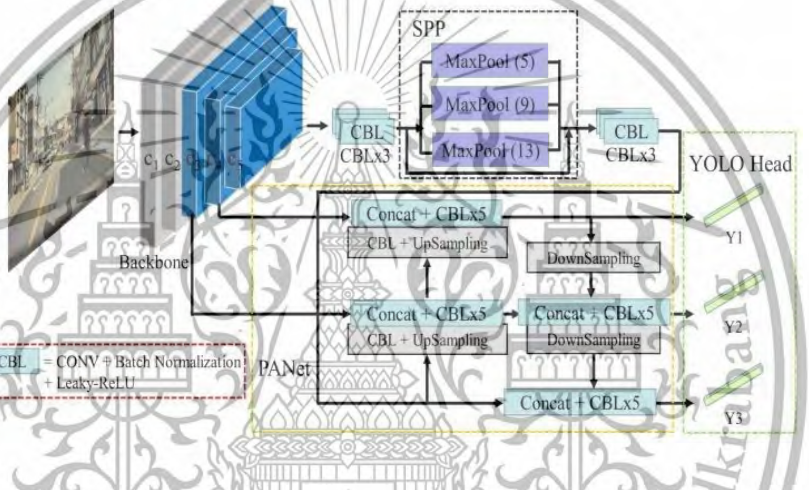


Fig 2.7: Overall structure of YOLOv4, including CSPDarknet (backbone), SPPnet, PANet and 3 YOLO heads [10].

Indeed, despite some worries regarding its level of innovation in comparison to YOLO V4, YOLO V5 has some features that make it a useful solution for certain computer vision jobs. Here are the main benefits of YOLO V5:

1. Model Flexibility and Lightweight Ness: YOLO V5 has a number of network architectures that allow for increased usage flexibility. These models are compact while maintaining competitive accuracy with the YOLO V4. This adaptability is useful for a wide variety of applications and deployment settings.

2. User-Friendly PyTorch [22] Framework: YOLO V5's use of the PyTorch framework simplifies and improves the training process. This ease of use is especially beneficial for individuals and teams who want to train models on custom datasets and incorporate them into production contexts.

3. Readable Code and Integration: The YOLO V5 codebase is well-known for its readability and simplicity. It combines a number of computer vision technologies, making it an invaluable resource for learning and reference. This makes it suitable for both skilled and inexperienced practitioners.

4. Efficient Environment setup: YOLO V5 streamlines the environment setup process, which is critical for model development and deployment efficiency. The model training is extremely quick, and the ability to execute batch inference enables the generation of real-time results, which is critical for many real-world applications.

In the context of YOLO V5, each batch of training data is enhanced using the data loader. These improvements are divided into three categories: scaling, color space modification, and mosaic enhancement.

The empirical evidence supports the efficacy of mosaic augmentation in addressing one of model training's most persistent challenges: reliable detection of small objects. This problem develops owing to the difference in detection accuracy between tiny and large objects. However, it is important to note that the YOLO V5's name has sparked debate. Furthermore, its implementation is dynamic and not yet complete, giving plenty of space for future development and modification.

From its initial version (V1) to its most recent iteration (V5), the YOLO (You Only Look Once) network has seen tremendous advances. The following is a summary of the major enhancement measures implemented in each version:

- YOLO V1: The first YOLO pioneered grid-based object detection and used confidence loss as part of its detection mechanism.
- YOLO V2: YOLO V2 featured a number of significant improvements, including the inclusion of anchor boxes selected by K-means clustering, a two-stage training method, and a full convolutional network design.
- YOLO V3: YOLO V3's capabilities were expanded by incorporating multi-scale detection via Feature Pyramid Networks (FPN). This enabled it to detect items of varying sizes within an image.
- YOLO V4: The introduction of Spatial Pyramid Pooling (SPP), the MISH activation function, advanced data augmentation techniques such as Mosaic and Mixup, and the use of the GIOU (Generalized Intersection over Union) loss function improved bounding box accuracy.

- YOLO V5: The advancement of YOLO V5 continues with features such as variable control over model size, the use of the Hardswish activation function, and further data augmentation approaches. These enhancements improve network performance and usability.

In June 2022, YOLO V6, the follow-up of YOLO V5, was made available by the Meituan Technical Team in China [11]. Based on the earlier YOLO model, this one was made to be extremely accurate and fast on a variety of hardware configurations, all the while preserving the familiar user experience. There are two variants available for this version of YOLO V6: YOLO V6-nano, which is faster, lighter, or has a smaller GPU or RAM, but has fewer parameters, and YOLO V6-large, which has higher accuracy and performance at the expense of slower speed.

YOLO V6 improvement include adopting an anchor-free approach, resulting in a 51% speed improvement compared to anchor-based approaches. Additionally, YOLO V6 also revised reparametrized backbone and neck proposed as EfficientRep backbone and Rep-PAN neck. As a result of this, the architecture has more layers that divide features from the final head, improving performance. Moreover, YOLO V6 import two loss function, Varifocal loss (VFL) for classification and distribution focal loss (DFL) with SIoU/GIoU for regression.

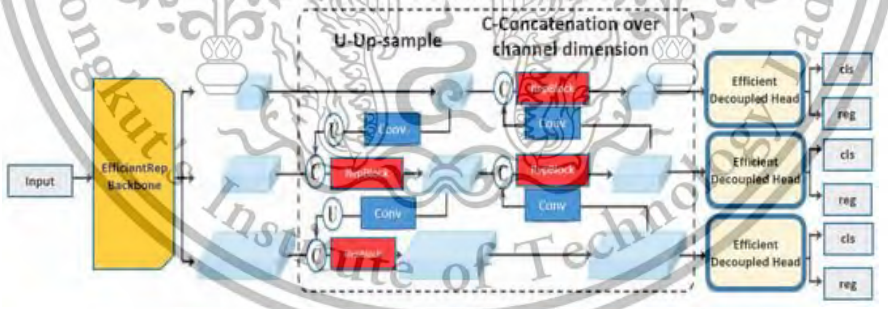


Fig 2.8: YOLO-v6 model base architecture [11]

When comparing to its precursor, YOLO V6 outperformed all, as shown in Fig 2.9.

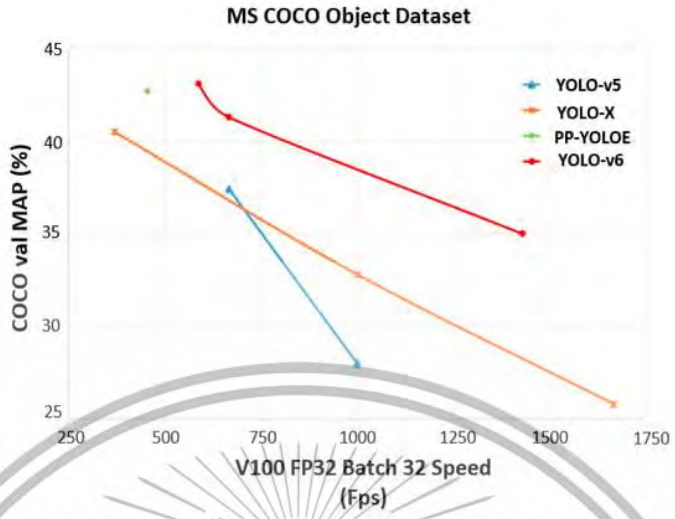


Fig 2.9: Relative evaluation of YOLO-v6 vs. YOLO-V5 [11]

A month after YOLO V6, YOLO V7 was released, with improved architecture without sacrificing speed or efficiency. By introducing a trainable bag-of-freebies—a set of strategies or methodologies that alter the training costs or approach to improve the model's accuracy—and reforming its architecture (by adding E-ELAN). Also called BOF. Compound model scaling was added to YOLO V7 in order to expand its spectrum of use, as some applications require high precision but poor recall. In an effort to improve performance, it also re-parameterizes. Furthermore, it was suggested to use an auxiliary head in the middle layers to help with training, however it performs poorly because it cannot access the complete network.

Additionally, YOLO V7 features a small version that is optimized for Edge AI and designed for mobile deployment.

As YOLO V7 is more effective than its successor, a comparison of YOLO V7 with other object detection models may be found in Fig. 2.10.

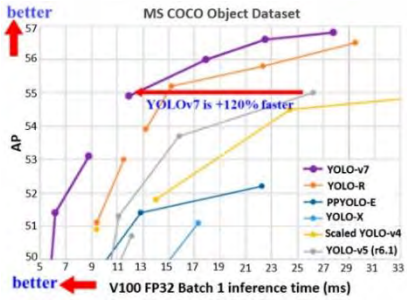


Fig 2.10: YOLO-v7 comparison vs. other object detectors [11]

In January 2023, Ultralytics, the same company that released YOLO V5 as well, developed and released YOLO V8 [12]. This model was created to be cutting edge, capable of performing numerous visual AI tasks like rapid segmentation, object detection, and classification. through improvements and new features meant to increase efficiency, flexibility, and overall system capabilities; these also include architectural changes and more hardware efficiency like multi-GPU support and auto dataset download.

In addition, Yolo V8 included an export option with multiple formats to utilize for any application—from mobile to IoT projects—while preserving its unique efficiency. As a cutting-edge model, YOLO V8 offers even more customizable options.

2.4 End-to-End Object DEtection with Transformers (DETR)

The goal of object detection is to predict a set of bounding boxes and their associated category labels for each object of interest in a picture. Surrogate issues requiring regression and classification are used by modern object detectors to approach this task indirectly. These surrogate challenges are often based on a large number of area suggestions, anchors, or window centers.

Such detectors' efficacy is heavily reliant on later postprocessing operations aimed at aggregating near-duplicate predictions. Furthermore, it is dependent on the careful design of anchor sets and the algorithms used to allocate target boxes to specific anchors. To improve and simplify this complex process, a direct set prediction strategy was proposed to eliminate the need for surrogate tasks. This method offers a more straightforward and comprehensive answer to the object detection problem.

End-to-end learning, which entails training a model to directly anticipate the intended output without relying on intermediate steps or prior knowledge, has resulted in significant advances in difficult structured prediction tasks such as machine translation and speech recognition. However, in the field of object detection, this all-in-one technique has not been as successful. Previous efforts either included extra forms of prior knowledge or failed to demonstrate competitiveness when compared to robust baseline models on difficult benchmark datasets [18].

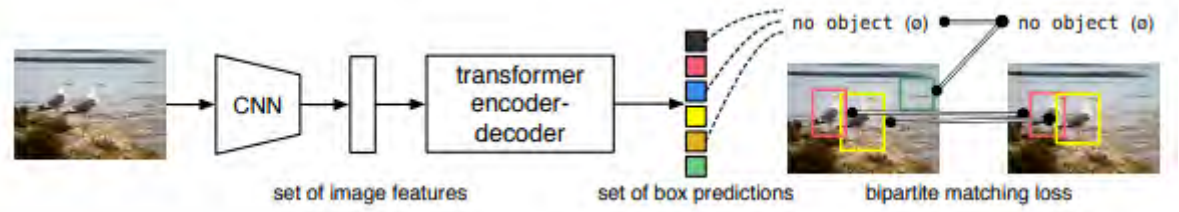


Fig. 2.11: DETR directly predicts (in parallel) the final set of detections by combining a common CNN with a transformer architecture. During training, bipartite matching uniquely assigns predictions with ground truth boxes [18].

DETR model recast object detection as a direct set prediction job to simplify the training procedure. Using an encoder-decoder architecture inspired by transformers in this technique, which is a frequently used design in sequence prediction problems. Because of its self-attention processes, which explicitly represent interactions between elements in a sequence, transformers are well-suited for this task. Because of this property, transformer-based architectures are particularly well suited to addressing specific issues in set prediction tasks, such as avoiding duplicate predictions.

Because of its self-attention processes, which explicitly represent interactions between elements in a sequence, transformers are well-suited for this task. Because of this property, transformer-based architectures are particularly well suited to addressing specific issues in set prediction tasks, such as avoiding duplicate predictions.

DEtection TRansformer, or DETR for short, is intended to anticipate all objects in a picture at the same time. It is trained from beginning to end using a specialized set loss function that conducts bipartite matching between predicted and ground-truth items. DETR simplifies the object detection process by reducing the requirement for many manually constructed components that encode past knowledge, such as spatial anchoring or non-maximal suppression.

DETR's simplicity is one of its unique features. DETR, unlike many other existing object identification algorithms, does not require the use of special layers. This implies it is simple to replicate in any framework that provides standard Convolutional Neural Network (CNN) and transformer classes. This ease of use enables DETR uptake and implementation in a variety of computer vision applications.

DETR distinguishes itself from the majority of previous efforts in direct set prediction by combining two crucial features: the use of bipartite matching loss and transformers with non-autoregressive parallel decoding. DETR offers a different strategy than previous techniques, which mostly used autoregressive decoding with Recurrent Neural Networks (RNNs).

DETR's matching loss function is intended to uniquely assign a prediction to each ground truth item. Importantly, it is insensitive to anticipated object permutations, allowing us to emit predictions in parallel rather than sequentially. This novel approach improves the efficiency and effectiveness of the object detection process, distinguishing DETR from earlier work in this sector.

DETR has been evaluated using the well-known COCO object detection dataset and compared its performance to a highly competitive Faster R-CNN baseline. Faster R-CNN has undergone multiple design revisions, resulting in significant performance increases since its initial publication.

Experimental results show that DETR model outperforms Faster R-CNN in terms of performance. DETR, in particular, performs substantially better at detecting huge objects, which is likely due to the transformer architecture's non-local computations. However, it performs poorly when it comes to identifying little objects. Prediction that future research will address and improve on this feature, similar to how the development of Feature Pyramid Networks (FPN) improved the performance of Faster R-CNN in handling small objects.

DETR's training settings differ significantly from those employed in ordinary object detectors in various areas. The new model demands a longer training duration, but it benefits from auxiliary decoding losses included in the transformer architecture. The investigation of the essential components that are critical in producing the observed high-performance results.

DETR's design philosophy is inherently adaptable to more complicated jobs. To demonstrate this extensibility in the experiments by training a simple segmentation head on a pre-trained DETR model. This expanded model not only performs well, but also surpasses rival baseline models in Panoptic Segmentation, a difficult pixel-level recognition challenge that has recently gained popularity. This demonstrates the adaptability and efficacy of DETR's design concepts in a variety of computer vision tasks.

Direct set predictions in object detection are dependent on two important components:

1. Set Prediction Defeat: It is critical to use a set prediction loss to ensure a one-to-one connection between anticipated and ground truth boxes. This loss function ensures that predicted items are uniquely matched to their associated ground truth objects.

2. Set Prediction Architecture: An effective architecture is required to forecast a set of objects and their interrelationships in a single pass. This architecture should be able to forecast the entire collection of items of interest immediately.

Figure 2.12 depicts the complete architecture of this technique, emphasizing how these critical factors are incorporated into the model to enable accurate and efficient direct set predictions in object identification.

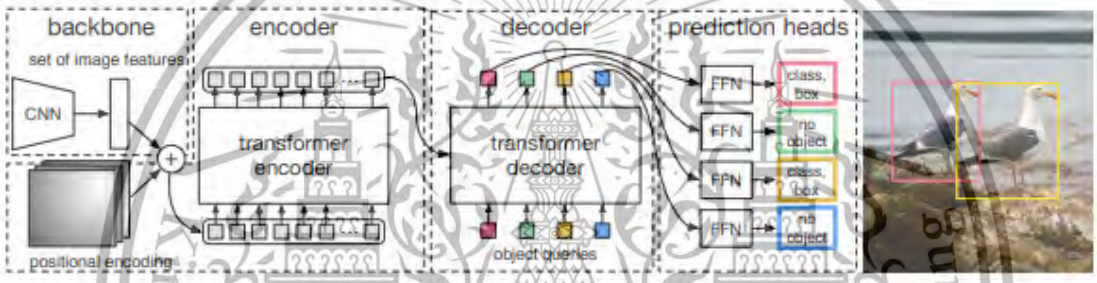


Fig. 2.12: DETR uses a conventional CNN backbone to learn a 2D representation of an input image. The model flattens it and supplements it with a positional encoding before passing it into a transformer encoder. A transformer decoder then takes as input a small, fixed number of learned positional embeddings, which call object queries, and additionally attends to the encoder output. Then pass each output embedding of the decoder to a shared feed forward network (FFN) that predicts either a detection (class and bounding box) or a "no object" class [18].

DETR takes a novel technique to prediction, inferring a fixed-size set of N predictions in a single run through the decoder. Notably, N is set to be substantially greater than the average number of objects in an image. One of the most difficult issues during training is determining how to score these predicted items based on their class, position, and size in relation to the ground truth objects.

The loss function is intended to find the best bipartite match between the anticipated and ground truth items. Then optimize object-specific (bounding box) losses. To demonstrate this procedure, consider y to be the ground truth set of objects and $\hat{y} = \{\hat{y}_i\}_{i=1}^{N}$ to be the set of N predictions. Given that N is chosen to be greater than the number of items in the image, y was express as a set of size N, padded ∅ with to indicate the absence of an object. The goal

is to find the lowest cost permutation of N items, abbreviated as SN, in this bipartite matching procedure.

$$\sigma = arg \min_{\sigma \in GN} \sum_{i}^{N} \mathcal{L}_{match}(y_i, \hat{y}_{\sigma(i)})$$

The Hungarian algorithm is used to determine the optimal assignment, which finds the best connection between ground truth items y_i and expected objects with index $\sigma(i)$. $\mathcal{L}_{match}(y_i, \hat{y}_{\sigma(i)})$ denotes the cost associated with this pairwise matching. As previously shown in prior studies, this computer procedure quickly selects the best suited pairing between ground truth and predicted items.

The matching cost function takes into account both class prediction and similarity between predicted and ground truth boxes. Each element i in the ground truth set can be written as $y_i = (c_i, b_i)$, where c_i is the target class label (which may be $\emptyset$, signaling no object), and $b_i \in [0,1]^4$ is a vector containing the center coordinates of the ground truth box as well as its height and width relative to the picture size.

The probability of class has been denoted c_i as $\hat{p}_{\sigma(i)}(c_i)$ and the predicted box as $b_{\sigma(i)}$ for the prediction with index (i). It was defined the matching cost $\mathcal{L}_{match}(y_i, \hat{y}_{\sigma(i)})$ using these notations as follows:

$$\mathcal{L}_{Hungarian}(y, \hat{y}) = \sum_{i=1}^{N} [-\log \hat{p}_{\hat{\sigma}(i)}(c_i) + \mathbb{1}_{\{c_i \neq \emptyset\}} \mathcal{L}_{box}(b_i, \hat{b}_{\hat{\sigma}}(i))]$$

In the case where σ reflects the optimal assignment derived in the first phase, practical implementation entails applying a down-weighting factor to the log-probability term when $c_i = \emptyset$. This 10-fold down-weighting is used to solve class imbalance concerns, mimicking the method used in the Faster R-CNN training procedure to balance positive and negative proposals via subsampling. It is worth noting that the matching cost between an object and $\emptyset$ remains independent of the prediction in such cases, making it a constant. Instead of log-probabilities, using probabilities indicated as $\hat{p}_{\sigma(i)}(c_i)$ in the context of the matching cost. This change ensures that the class prediction term and the $\mathcal{L}_{box}$ metric are comparable, as detailed later. This enhancement has been empirically shown to boost performance in practice. The second component of the matching cost and Hungarian loss is the bounding box

loss, represented as $\mathcal{L}_{box}(\cdot,\cdot)$. Unlike many conventional detectors, approaching includes direct box predictions rather than deltas (Δ) respect to baseline estimates. While this simplifies implementation, it creates a barrier due to the loss's relative scalability. Even though their relative errors are comparable, the generally used 1 loss has different scales for small and large boxes. To overcome this scaling issue, a scale-invariant linear combination of the 1 loss and the generalized Intersection over Union (IoU) loss were use, designated as $\mathcal{L}_{iou}(\cdot,\cdot)$.

In summary, the box loss is defined as $\mathcal{L}_{box}(b_i, \hat{b}_{\sigma(i)}) = \lambda_{iou}\mathcal{L}_{iou}(b_i, \hat{b}_{\sigma(i)}) + \lambda_{L1}\|b_i - \hat{b}_{\sigma(i)}\|_1$, where λ_{iou} and λ_{L1} are real number set (R) hyperparameters. Notably, these two loss components are normalized by the number of objects in the batch, assuring consistency and fairness in the loss computation across various batch sizes.

Figure 2.12 depicts the DETR architecture, which is made up of three main components: a CNN backbone for feature extraction, an encoder-decoder transformer for contextual comprehension, and a basic feed-forward network (FFN) for detection predictions. Surprisingly, DETR's design is remarkably simple, making it compatible with a variety of deep learning frameworks that include a standard CNN backbone and a transformer implementation. In actuality, its implementation requires only a few hundred lines of code, and the DETR inference code may be written in less than 50 lines using PyTorch [22]. This ease of use is intended to inspire fresh scholars to investigate and contribute to the subject of object detection.

Several critical activities are performed by the Transformer encoder module. To begin, a 1x1 convolutional layer is used to shrink the channel dimension of the high-level activation map, abbreviated as "f," from its original dimension "C" to a smaller dimension "d." This procedure generates a new feature map designated "z0," which is located in the space RdHW. Because the encoder anticipates sequential input, the spatial dimensions of "z0" are collapsed into a single dimension, yielding a feature map with dimensions "dHW.". Each encoder layer follows a standardized architecture, which includes a multi-head self-attention module and a feed-forward network (FFN). Using fixed positional encodings since the transformer design is permutation-invariant. These encodings are introduced to the input of each attention layer to ensure that the model takes into account the positioning information of the sequence's elements. Author recommends readers to the supplementary material for a

complete discussion of the encoder's architecture, which follows the framework previously defined in Figure 2.12.

The Transformer decoder in DETR model follows the traditional transformer architecture, processing N embeddings of size d using multi-headed self-attention and encoder-decoder attention techniques. Approach differs from the original transformer in that it can decode N items in parallel at each decoder layer. Vaswani et al., on the other hand, used an autoregressive technique to forecast the output sequence one element at a time. Readers who are unfamiliar with these concepts can find more information in the supplemental material. Because the decoder, like the encoder, is permutation-invariant, the N input embeddings must be distinct in order to provide separate outcomes. These input embeddings are simply learnt positional encodings known as "object queries," and supplement them, similarly to the encoder, by adding them to the input of each attention layer. Following that, the decoder transforms these N object requests into an output embedding. These embeddings are then decoded individually into box coordinates and class labels using a feed-forward network, as detailed in the following subsection. This procedure yields N final predictions. The model provides global reasoning about all objects in the scene by using pair-wise interactions between them by utilizing self- and encoder-decoder attention over these embeddings. Notably, it can use the full image as context, allowing for more thorough interpretation and context-aware predictions.

A three-layer perceptron with ReLU activation and a hidden dimension 'd,' complemented by a linear projection layer, produces the final predictions. This feed-forward network (FFN) predicts the normalized center coordinates, height, and breadth of bounding boxes relative to the input image while also giving class labels using a SoftMax function. Notably, this approach predicts a fixed-size collection of N bounding boxes, which frequently surpasses the number of objects in a picture. To deal with this, develop of a custom class label defined as ',' which indicates the lack of an object in a specific slot. This class label is smoothly integrated into the prediction process, ensuring complete handling of potential object scenarios. Discovering that incorporating auxiliary losses into the decoder throughout the training phase was beneficial. These auxiliary losses are crucial in directing the model to appropriately forecast the number of objects in each class. This accomplishes by employing prediction feed-forward networks (FFNs) and employing Hungarian loss after each decoder

layer. Importantly, all prediction FFNs use the same set of consistency parameters. Furthermore, perform an additional layer normalization step to standardize the input to ensure coherence in the input data for the prediction FFNs across different decoder layers. This judicious use of auxiliary decoding losses improves the model's ability to make accurate class predictions and item count estimates.

2.5 Detectron2

Detectron2, which began its transition from the maskrcnn-benchmark codebase to the PyTorch [22] framework, is a complete redesign of the original Detectron framework. The recently implemented modular architecture improves Detectron2's flexibility and expandability, making it capable of providing quick model training on one or more GPU servers. This improved version of Detectron2 includes state-of-the-art applications of novel object detection techniques, including prominent contributions like DensePose [21], panoptic feature pyramid networks, and variations of the widely recognized Mask R-CNN model lineage developed by FAIR. Because of its intrinsic modularity, cutting-edge research projects can be integrated with ease and there's no need to keep up with a separate codebase fork [20].

Detectron2 brings forth notable enhancements in several key aspects:

- Framework Transition: Detectron2 adopts the PyTorch framework, departing from its predecessor, which was created in Caffe2. This change provides a more imperative programming model that is more intuitive, allowing practitioners and researchers to iterate quickly when designing and testing models. The switch to PyTorch gives users access to a vibrant and active community that continuously improves the framework, in addition to providing its deep learning paradigm.
- Detectron2's architecture is both extensible and modular, allowing users to easily incorporate custom modules into almost any aspect of an object detection system. The separation of the core Detectron2 library from the individual research components is facilitated by this modular architecture, which also makes it easier to implement new research projects. This design is continuously improved to support new models and increase the adaptability of the system.

- Diverse Model Portfolio: Detectron2 integrates the Faster R-CNN, Mask R-CNN, RetinaNet [24], and DensePose legacy models that were part of the first Detectron. It also presents a number of new models, including TensorMask, Panoptic FPN, and Cascade R-CNN. The goal of adding more algorithms to the toolkit is not going away. Additional features of Detectron2 include support for datasets such as LVIS and synchronous Batch Normalization.
- Diversity of Tasks: Detectron2 is capable of handling a large range of object detection tasks. Along with human pose prediction, it continues to support object detection with bounding boxes and instance segmentation masks. Moreover, Detectron2 expands its functionality to include panoptic segmentation, which combines semantic and instance segmentation techniques.
- Speed and Scalability: Detectron2 is faster than its predecessor across multiple standard models thanks to the complete training pipeline migration to GPU. Additionally, the framework streamlines the scalability of training processes, especially for large datasets, by making it easier to distribute training tasks across several GPU servers.
- Detectron2go: Facebook AI's computer vision engineers have unveiled Detectron2go, an extra software layer designed to make the deployment of sophisticated models in real-world settings easier. This part includes common training procedures that use internal datasets, network quantization, and model conversion into mobile and cloud-friendly formats.

2.6 Android application

This approach requires the creation of an Android application for object identification, with the smartphone's camera serving as the principal sensor. The application makes use of TensorFlow's object detection API [23] to recognize items in the user's immediate proximity. Following that, it plays an instructive audio message to the user, including information such as the object's name and position. This location data includes both the direction and distance of the identified object from the user's current position. The audio message is sent using audio output devices such as headphones or the smartphone's built-in speaker to enable accessibility for visually impaired users. Importantly, the system is self-contained and does not require the usage of an external camera because it uses the capabilities of the smartphone's integrated

camera to perform the aforementioned functions. This app intends to empower visually impaired people by giving them with real-time information about their surroundings, thereby improving their overall accessibility and mobility. Computer vision-based techniques provide an appropriate answer for tackling the issues that visually impaired people face. Computer vision, unlike RFID (Radio-Frequency Identification) systems, does not necessitate any changes to the environment. RFID-based aids require the installation of several hardware components, such as transmitters and receivers, in advance to enable the system's functionality, making it an expensive system. In contrast, computer vision-based aids can be accessed using a standard smartphone. There is no need to carry along extra hardware, such as a specialized camera, because almost all smartphones include built-in cameras. Because of its ease of use and accessibility, computer vision-based systems are a practical and cost-effective option for supporting visually impaired people, allowing them to navigate and interact with their surroundings more efficiently.

To facilitate functioning, the Android operating system is divided into layers. The Applications layer is the platform for Android applications, including features such as an email client, SMS program, maps, web browser, contacts, and more. The Java programming language is used to create these applications. The Application Framework layer, which sits beneath the Applications layer, is critical in designing the Android application framework. This framework, which includes several critical components, serves as the foundation for all Android applications.

- Rich and Extensible Views: This is a versatile collection of Views that includes items like as lists, grids, text boxes, buttons, and even an embedded web browser, allowing developers to construct aesthetically appealing user interfaces.
- Content Providers: These allow applications to retrieve data from other applications (for example, Contacts) or exchange their own data, allowing for smooth data sharing and integration.
- Resource Manager: It provides non-code resources such as localized strings, images, and layout files, easing the resource management process in programs.
- Notification Manager: This allows all programs to display bespoke alerts in the device's status bar, allowing for increased user involvement and engagement.

- Activity Manager: The Activity Manager is in charge of managing the lifetime of applications, ensuring effective resource allocation and providing a common navigation back stack for a smooth user experience and application management.

The Application Framework layer serves as the foundation for Android apps, providing a set of tools and services that improve development, increase interactivity, and maintain a uniform user experience across the Android ecosystem.

The libraries layer contains a collection of C/C++ libraries utilized by various Android system components and offers support to the application framework. Google's Android Runtime consists of a group of core libraries and a Java virtual engine (Dalvik virtual machine) that have been rewritten and optimized for the Android platform.

The Linux kernel is the core layer of the Android system, located at the bottom of the software stack. It is essential as an abstraction layer, bridging the gap between the device's hardware and higher-level software components. The Linux kernel provides a variety of core system services, such as, but not limited to:

- Security procedures and access controls are enforced, protecting the device from illegal access and potential threats.
- Memory Management: The kernel controls memory resource allocation and deallocation, guaranteeing efficient utilization and preventing memory-related problems.
- Process Management is in charge of overseeing the development, execution, and termination of processes, allowing for multitasking and efficient resource allocation.
- Network Stack: The kernel provides networking features, allowing the device to communicate with external networks such as the internet.
- It provides a framework for device drivers, allowing for the smooth integration of hardware components into the Android system.

In addition, the Linux kernel is crucial in supporting the Dalvik virtual machine, which is in charge of running Android applications. It aids in thread and concurrency management, assuring the appropriate execution of Android programs.

In essence, the Linux kernel is the foundation of the Android operating system, providing vital system functions and maintaining the smooth interplay of hardware and software components, allowing Android devices to run and perform [19].

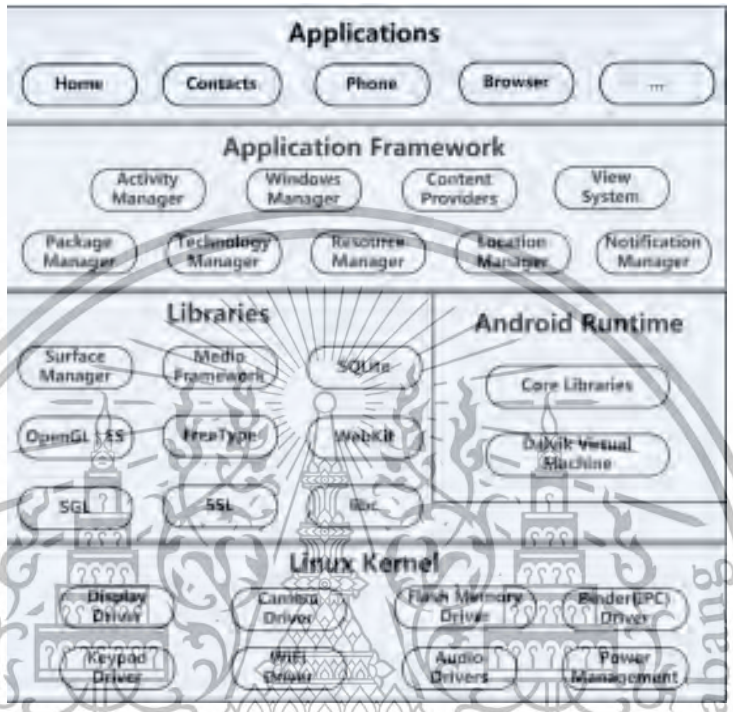


Fig. 2.13: Structure of android application [19].

As was already said, Android runs on a Linux kernel, and the majority of its applications are created using the Java programming language. These Android programs are executed by the Dalvik virtual machine, a Java virtual machine. The Dalvik virtual machine has undergone substantial redesign and optimization by Google to conform to the unique hardware requirements and limitations of mobile devices.

The Android Software Development Kit (SDK) includes a tool named "dx," which is essential to the Android operating system. This program converts Java Class files, which are usually produced by a common Java compiler, into files in the specialist ".dex" (Dalvik Executable) format. The ".dex" format is intended to group all Java class files together while deleting any extraneous data from each individual Java class file. This optimization method improves the overall efficiency and functionality of Android applications and is essential for effective execution on mobile devices with limited resources.

Numerous essential characteristics of the Dalvik virtual machine contribute to its effectiveness and operation inside the Android ecosystem. First off, it allows for the presence

of numerous instances running simultaneously within separate isolated Linux processes on a single hardware. This configuration guarantees that each Android application runs in a separate instance of the Dalvik virtual computer, improving security and reducing application interference.

Second, Dalvik heavily depends on the underlying operating system, usually the Linux kernel, for essential features like process separation, memory management, and multithreading support. The correct operation and resource management of Android applications are guaranteed by this reliance on the operating system's capabilities.

Last but not least, Dalvik has a register-based design, in which CPU registers are directly used for operations. This design makes bytecode instructions easier to execute, making it especially suitable for mobile devices with constrained computational and processing capability. These characteristics work together to make the Dalvik virtual machine, which powers the Android platform, more reliable and effective.

Fig.2.14: the position of Dalvik virtual machine in Android system [19].

Android application component

The ability of the Android operating system to let one application use component pieces belonged to another application, provided the component's usage is allowed, is a vital feature. The Android system is intended to launch the target application whenever any of its components are requested in order to enable such inter-application capability. Java objects are instantiated throughout this procedure when necessary to provide the specified functionality.

In particular, Android differs from many other operating systems in that it does not support a single-entry point, such as the conventional "main ()" function, within an Android

program. As opposed to this, Android apps are built around a set of components, and the system dynamically activates the right ones in response to user interactions or outside demands. The modular and adaptable application structure made possible by this component-based approach enables smooth interaction between various components of an application and across various applications found throughout the Android ecosystem.

In fact, each component in Android acts as a unique entry point via which the system can access and independently instantiate component objects. There are four main categories of application components, each of which is developed for a specific function and has a distinct lifespan that specifies how it is ended. These elements are crucial to the Android application architecture because they offer a structured and modular foundation for creating many applications of different types and enabling fluid communication between them.

The user experience within an application is mostly determined by the Android activity, which is a distinct screen with a user interface. While an application's activities work together to provide a seamless user journey, it's important to remember that each activity runs separately from the others. Therefore, a different application can start any of these processes as necessary [13].

An activity is often implemented as a subclass of the "Activity" class in Android when it comes to implementation. The developer's design and goals determine the specific form and content that an activity offers to users as well as the quantity of activities in an application. One activity is typically chosen as the "main" activity in applications with several activities; this activity serves as the first screen the user sees when they run the application for the first time. Each activity can then trigger other activities to support different actions or functions. It's significant to note that when a new activity is started, the previous activity is paused, but the Android system keeps track of these activities in a stack known as the "back stack" [13].

The user can travel back via previously visited activities by using this stack, which records the user's navigation history.

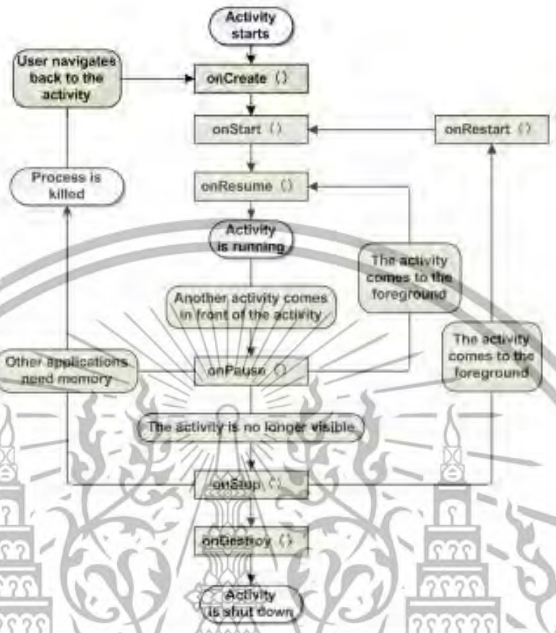


Fig 2.15: The lifecycle of an activity [13]

An essential part of Android called a service is made to carry out operations in the background, making it appropriate for long-running operations or supporting remote processes. It's important to note that services lack a user interface. An activity can connect to a service that is already operating or bind to one that is already running in the context of an Android application. The action can start the service's launch if it isn't already running. Once a service is linked, an activity is given the opportunity to interact with it using the interface made available by the service component.

It's important to keep in mind that service component' often run on an application's main thread, just like other application components do. Therefore, if a service performs time-consuming or blocking actions, it may interfere with the execution of the activity that started it. It is advised to begin a new thread within the service in order to address this problem and guarantee that the activity is responsive. With this method, resource-intensive tasks can be handled by the service without degrading the efficiency of the related activity as a whole.

The exchange of data stored in diverse places, such as the file system, SQLite databases, or other persistent storage spaces accessible by an application, is made possible

via content providers, which act as a crucial data-sharing mechanism among Android applications. Developers construct a subclass of the “Content Provider” class to implement a content provider. This content provider specifies the kind of data it will accept and provides a variety of ways for other programs to interface with and work with that data. It’s vital to remember that apps don’t use the content provider’s APIs directly. Instead, they interact with the content source through a middle object called a "Content Resolver". The Content Resolver is a flexible component that can connect to any Android system content provider and establish communication with them. In order to ensure secure and controlled access to shared data resources, the Content Resolver works along with the Content Provider to manage Inter-Process Communication (IPC). The security and dependability of data sharing across Android applications are improved by this architecture.

In order to handle system-wide broadcasts and carry out replies specific to the data supplied via these broadcasts, broadcast receivers are essential. The Android operating system itself generates a lot of broadcasts for a variety of functions, such as warning users when the screen has been switched off or when the battery is becoming low. Applications can also start their own broadcasts for particular notifications or events. Each Broadcast Receiver is normally built as a subclass of the “Broadcast Receiver” class, and an application may include any number of Broadcast Receivers. Although they lack a user interface, broadcast receivers can create status bar notifications to inform users when a broadcast event happens. A Broadcast Receiver is often designed to conduct minimal work on its own and acts as a “gateway” to other application components. Instead, it serves mostly as a coordinator, sending the broadcast data to the proper parts for processing. This simplified architecture guarantees effective handling of broadcast events and enables Android applications to efficiently react to notifications that are both system-wide and application-specific.

Activities, services, and broadcast receivers—three out of Android’s four main component types—are started using an asynchronous message known as a “intent.” Regardless of whether they are part of the same application, intents work as dynamic connectors between separate components, enabling them to communicate with one another during runtime. The “Intent” object, which specifies the messages that can activate either a specific component or a particular type of component, is used by developers to create an intent. An intent, which describes the action to be taken for activities and services, may also

contain extra information, such as the Uniform Resource Identifier (URI) of the data that will be acted upon. In the case of broadcast receivers, the announcement or message being broadcast to the system is simply defined by the intent. However, content providers function differently because intents do not directly activate them. Rather, they are activated when they are the intended 37recipient of a request from a "Content Resolver." The management of data interchange between applications, contacting content providers, and querying or altering data are all responsibilities of content resolvers. The controlled and secure sharing of data among applications in the Android ecosystem is ensured by this unique method to activation.

CHAPTER 3
METHODOLOGY

3.1 Introduction

Chapter 2 provides an overview of deep learning, identifies important object detection features, describes the theory of the model used in this project, and provides an overview of oral disease and its effects. This chapter describes the development of object detection, a mobile application. First, the chapter describes the project's design methodology (section 3.2), and system requirements (section 3.3). A proposed solution is then discussed (section 3.4), followed by a summary of methodology.

3.2 Design Methodology

In an effort to design and develop the proposed oral pocket and cavity detection, a number of deep-learning based object detection models including YOLOv8, DETR, etc. will be implemented. Then their performance will be accessed and compared in order to select the best model, which can give high detection accuracy, mPA, recall and precision. After that a mobile application will be created using the optimal detection model. Then compare and find the best one with high accuracy, mPA, high recall and precision. While also suitable to port it into mobile application.

Coding

Google-Colab Pro was used to code most of the models (DETR use Jupyter notebook), and it has a GPU that can be used to speed up and save time. Furthermore, every model was written in Python. Which offer extensive use of the high-level language and support libraries.

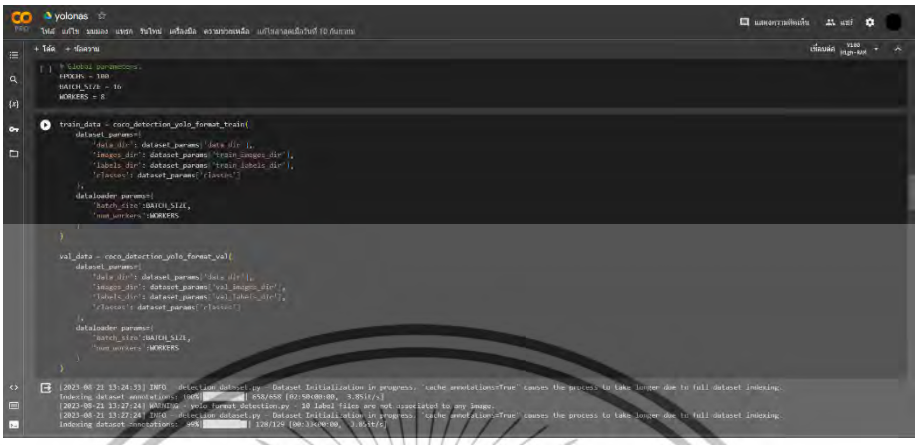


Fig 3.1: google-colab program and example of code for YOLO-nas

However, google-Colab has a soft-lock security that it can't access to file on local computer. Therefore, all of data (libraries, dataset) must upload to google drive and connect it with google-colab to access it.

Fig 3.2: google-colab ask permission to access to google drive

Models

Many models that can be used for object detection have been found through research. However, chosen models are limited by numerous requirements. Such are:

- Using a medical image, this project has a medical subject. As a result, output needs to meet strict requirements to guarantee that there won't be any serious problems when applied in practice.
- This project also aims to import it into android mobile application and not all of model for object detection can be imported or convert to mobile-friendly due to size of model or the model incompatible to mobile.
- The majority of the x-ray pictures in the datasets are in grayscale color. Certain models don't work well with this kind of color.

Dataset

The dataset that obtained includes both color and x-ray images. Since working with color images will be more efficient in the future, first want to explore with them. But there are only ninety-one color pictures. As a result, x-ray images had been employed instead.

There are some duplicate images or images in the collection without an annotation file. After removing such images, only 936 images are functional. The dataset also provides XML annotation file, which can be used for YOLO model. However, the other two (DETR and Detectron2) work COCO-format, so annotation format was change before training those two models.

Fig 3.3: Example of color images from dataset [17]

Dataset preprocessing

Every image in the collection is the same size, measuring 748 pixels in width and 512 pixels in height. As a result, resizing is not required. By subjecting the model to a greater

variety of data fluctuations during data augmentation, such as rotation and scaling to increase the model's robustness and generalization.

Training the model

This project will set every train model (epochs, learning rate, worker, etc.) to be as similar as possible because the aim for this project is to compare each model on the same dataset. However, some models have problems or errors that come up during training. To remedy the issues, some modifications of the settings have to occur. Each model's setting will be revealed in Chapter 4.

The selecting models are YOLO V8, YOLO-NAS, DETR and Detectron2 as those meet all requirement condition, the comparison of each model will show at below:

YOLO V8

Pros.	Cons.
Highly performance	Training takes a lot of time.
Highly optimized	Require specific annotation data
Automatic generate result graph	
Can export to many format	

YOLO-NAS

Pros.	Cons.
Better performance than YOLO V8	Not optimization as YOLO V8
Come with 3 variants to choose for convenient	Doesn’t generate graph
Can detect very small object	Large weight size compares to others

DETR

Pros.	Cons.
User-friendly	Training takes a lot of time.
Better performance than YOLO on real-time detection	Cannot be train on Google-Colab
	Low optimization

Detectron2

Pros.	Cons.
Very high performance	Require more GPU and memory
Training takes less time	Doesn't generate graph
Can be port into mobile-friendly model	

Table 3.1: Comparison of each model

Converting into mobile-friendly model

Detectron2 was selected for convert as it has the most suitable one. The model then converts into Detectron2Go which is release by Facebook research AI build for mobile device. Then use example code for Detectron2Go in android studio and replace its model with this one.

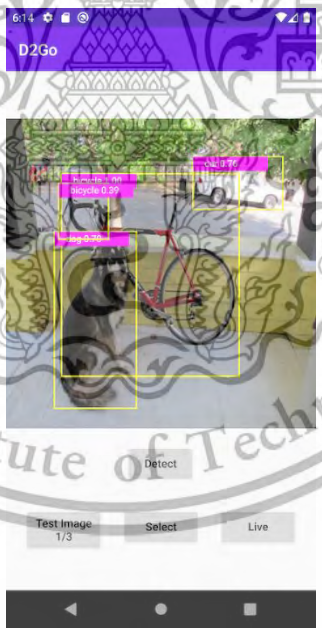


Fig 3.4: example of mobile app on android studio with build-in model

3.3 Interesting Problems

The main purpose of this project is to develop a mobile application for detection, as such, various problems have been encountered.

- Object detection require a lot of time to train and validation, as it requires both images and annotation for training. Also, some models require other types of datasets. Therefore, other types of datasets need to be prepared too.

- Sometimes the "data leak" problem occur which cause the performance result to decrease.

- Mobile applications can only be done in android OS, as IOS require a complicated process and have high private security.

- The datasets have only the patient oral disease, advisor suggest that it should also have both patient and healthy dataset at it will be more realistic. Because in real life, it will be both of dataset to detection.

3.4 Proposed Solution

- For time required to train, instead of using notebook's GPU to train, buying google-colab pro and use its GPU to train can save time and faster than local GPU.

- To prevent data-leaks, preprocess and augment are applied to dataset before training.

- Use auto-convert dataset (Roboflow [14]) to speed the experiment.

3.5 Summary

This chapter indicate both problems and solutions that occur, as well as design methodology including Dataset Collection and preprocessing (random rotations and contrast adjustments), model selection (YOLO V8, YOLO-NAS, DETR and Detectron2), model training, and convert into mobile-friendly model on android studio.

CHAPTER 4
EXPERIMENTAL RESULT AND DISCUSSION

4.1 Introduction

In this experiment, four models were trained by using datasets from Umer Rashid [17]. These datasets contain 936 x-ray images with annotations. All of each contain at least 1 type of disease, pocket, cavity or both. Moreover, some images contain diseases in more than 1 spot. The dataset was split as 70% for training, 20% for validation and 10% for testing.

The models that were selected for this project are YOLO V8, YOLO-NAS, DETR and Detectron2. Google-Colab and Jypyter notebook were used to train with python-language. Finally, choose the most suitable model to convert it into mobile-friendly model (D2Go) to use it on android smartphone by us android studio to write a code and optimize it.

4.2 Result

Setting and result of each model are show below:

YOLO V8

YOLO V8 model setting is show as below:

Epoch	100
Learning rate	0.001
worker	8
Batch size	16
Optimizer	Adam

Table 3.2: setting of YOLO V8

YOLO V8 model got mAP of 84.6% with precision of 86.4% and recall of 86.65%:

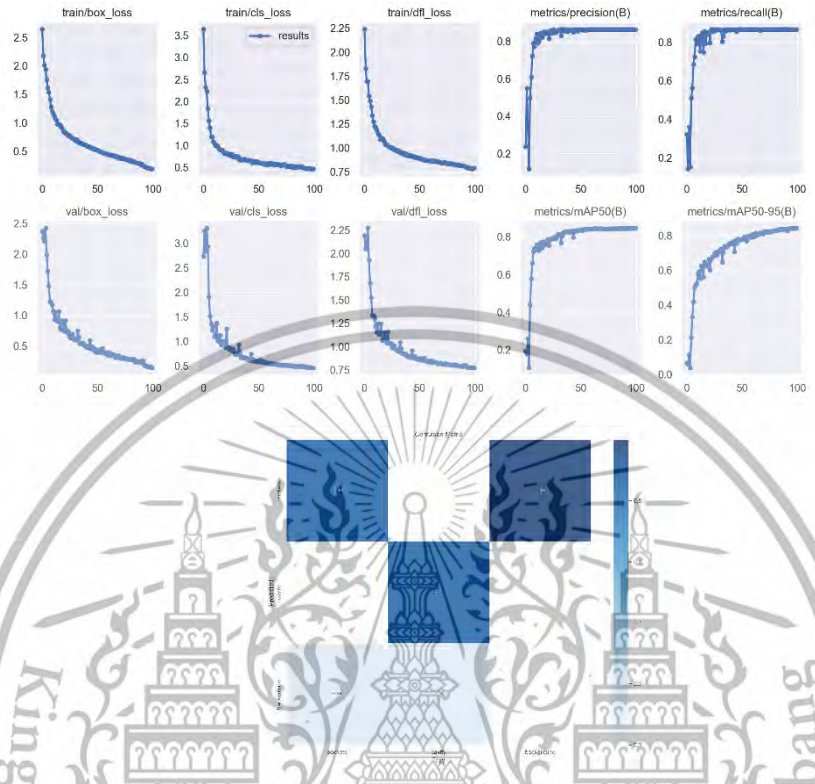


Fig 4.1: Result and confusion matrix of YOLO V8

Test images are show below (red: from YOLO V8 model, blue: from dataset):

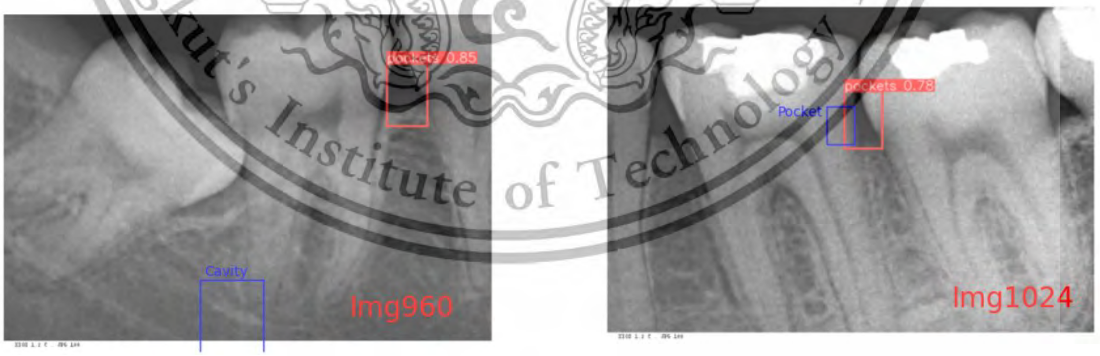


Fig 4.2: Test result of YOLO V8

YOLO-NAS

YOLO-NAS_L model setting is show as below:

Epoch	100
Learning rate	0.001
Iterations per epoch	41
Batch size	16

Table 3.3: setting of YOLO-NAS

YOLO-NAS_L model have mAP at 84.3% with validation_precision of 40.22% and validation_recall at 88.02%. Raw result and graph are show below:

```
└─ Validation
   ├─ Ppyoloeloss/loss_cls = 0.4818
   │  ├─ Epoch N-1      = 0.4889 (▼[32m -0.0071[0m)
   │  └─ Best until now = 0.4508 (▼[31m 0.031[0m)
   ├─ Ppyoloeloss/loss_iou = 0.0964
   │  ├─ Epoch N-1      = 0.1028 (▼[32m -0.0064[0m)
   │  └─ Best until now = 0.0837 (▼[31m 0.0127[0m)
   ├─ Ppyoloeloss/loss_dfl = 0.7349
   │  ├─ Epoch N-1      = 0.7511 (▼[32m -0.0163[0m)
   │  └─ Best until now = 0.6944 (▼[31m 0.0405[0m)
   ├─ Ppyoloeloss/loss = 1.0903
   │  ├─ Epoch N-1      = 1.1214 (▼[32m -0.0312[0m)
   │  └─ Best until now = 1.0117 (▼[31m 0.0785[0m)
   ├─ Precision@0.50 = 0.3903
   │  ├─ Epoch N-1      = 0.4237 (▼[31m -0.0334[0m)
   │  └─ Best until now = 0.4819 (▼[31m -0.0916[0m)
   ├─ Recall@0.50 = 1.0
   │  ├─ Epoch N-1      = 1.0    (▲[37m= 0.0[0m)
   │  └─ Best until now = 1.0    (▲[37m= 0.0[0m)
   ├─ Map@0.50 = 1.0
   │  ├─ Epoch N-1      = 1.0    (▲[37m= 0.0[0m)
   │  └─ Best until now = 1.0    (▲[37m= 0.0[0m)
   ├─ F1@0.50 = 0.5575
   │  ├─ Epoch N-1      = 0.584  (▼[31m -0.0265[0m)
   │  └─ Best until now = 0.6499 (▼[31m -0.0924[0m)
   ├─ Precision@0.50:0.95 = 0.334
   │  ├─ Epoch N-1      = 0.3585 (▼[31m -0.0245[0m)
   │  └─ Best until now = 0.4022 (▼[31m -0.0682[0m)
   ├─ Recall@0.50:0.95 = 0.8517
   │  ├─ Epoch N-1      = 0.8405 (▲[32m 0.0112[0m)
   │  └─ Best until now = 0.8802 (▼[31m -0.0285[0m)
   └─ Map@0.50:0.95 = 0.8128
      ├─ Epoch N-1      = 0.8078 (▲[32m 0.005[0m)
      └─ Best until now = 0.843  (▼[31m -0.0302[0m)
```

Fig 4.3: Raw result from YOLO-NAS

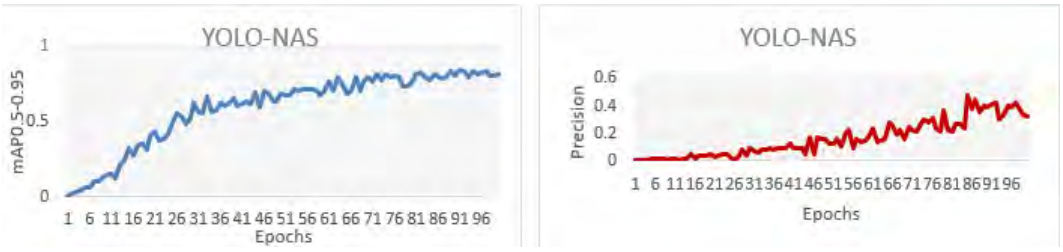


Fig 4.4: Graph showing mAP and precision from YOLO-NAS

Test images result is show below (blue: from YOLO-NAS model, red: from dataset):

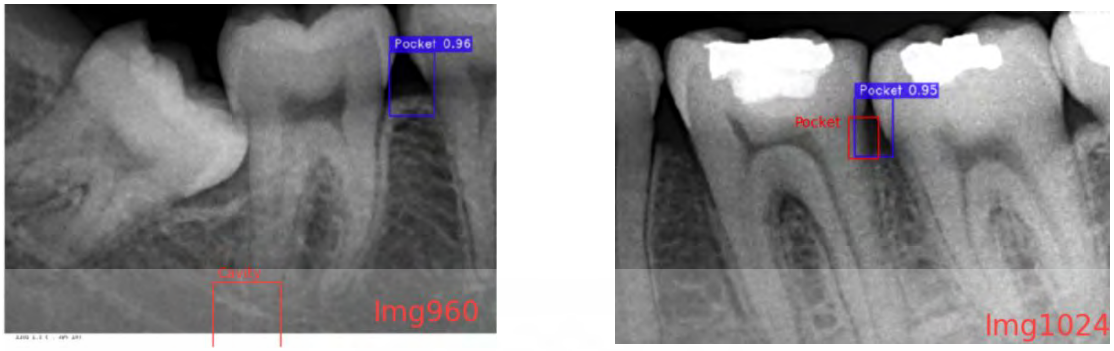


Fig 4.5: Test result from YOLO-NAS

DETR

DETR model setting is show as below:

Epoch	100
Learning rate	0.001
Batch size	16

Table 3.4: setting of DETR

DETR model has train_loss_giou of 62.84% with train_loss_bbox of 3.51%. Raw result and test image result are show as below:

457	2.233427	0.552728	0.048227	0.719784	37.75	45	2054					
458	2.118945	0.3644	0.058017	0.73723	30.75	45	2059					
459	2.255291	0.411843	0.053389	0.788252	36.75	45	2064					
460	2.459908	0.338782	0.081671	0.85643	46	45	2069					
461						45	2069	2.466042	0.522931	0.064129	0.811234	36.10799
462	2.66095	0.453812	0.088812	0.88154	48.75	46	2074					
463	2.25366	0.511432	0.041487	0.767397	31.75	46	2079					
464	2.245675	0.515402	0.044625	0.753576	37.5	46	2084					
465	2.247343	0.414674	0.052374	0.7854	42.25	46	2089					
466	2.023301	0.467999	0.03611	0.687375	29	46	2094					
467	2.720887	0.498173	0.09627	0.870682	35.75	46	2099					
468	2.26418	0.368063	0.063413	0.789525	33.5	46	2104					
469	2.286626	0.435374	0.06387	0.765952	28.25	46	2109					
470	2.638117	0.603084	0.0952	0.779516	49	46	2114					
471						46	2114	2.420109	0.52185	0.060408	0.798111	34.35205
472	2.460495	0.410019	0.073292	0.842008	39.5	47	2119					
473	2.149699	0.440132	0.04918	0.731834	36.75	47	2124					
474	2.567641	0.485164	0.07758	0.847288	34.25	47	2129					
475	2.295516	0.472065	0.056329	0.770903	41	47	2134					
476	2.24687	0.418055	0.053353	0.781025	36.75	47	2139					
477	2.543206	0.48795	0.077556	0.833737	33	47	2144					
478	2.174276	0.377222	0.045933	0.783694	33.5	47	2149					
479	2.375056	0.475672	0.065857	0.785048	30.75	47	2154					
480	2.614965	0.479285	0.066189	0.902367	32	47	2159					
481						47	2159	2.441877	0.532193	0.059939	0.804995	34.26026
482	2.255608	0.469519	0.050392	0.767066	41.25	48	2164					
483	1.746698	0.36055	0.036015	0.603036	32.75	48	2169					
484	2.586862	0.479034	0.073601	0.869912	27	48	2174					
485	2.301202	0.555141	0.050848	0.745911	33.5	48	2179					
486	2.407002	0.516094	0.065254	0.782319	38.5	48	2184					
487	2.399701	0.331249	0.071082	0.856521	36.5	48	2189					
488	2.186194	0.528612	0.037778	0.734345	35	48	2194					
489	2.174761	0.53604	0.036627	0.727793	34.25	48	2199					
490	1.783686	0.377527	0.037519	0.609282	34	48	2204					

Fig 4.6: Raw result from DETR

Fig 4.7: Test result from DETR

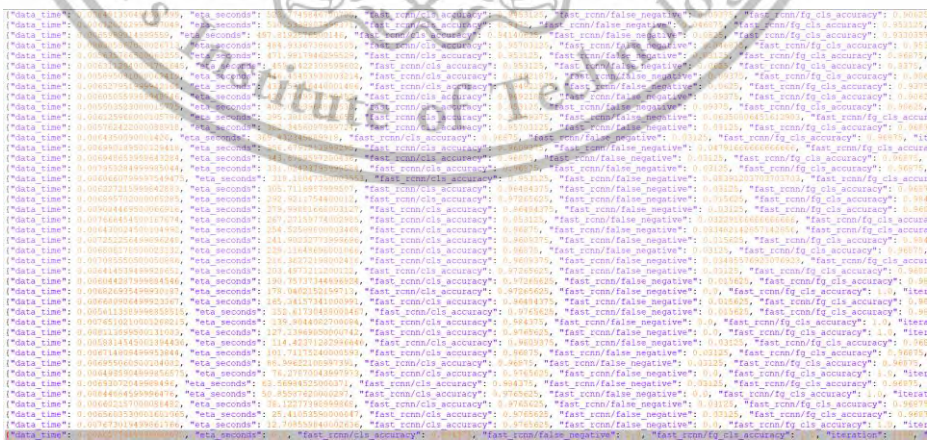
Detectron2

Detectron2 setting is show as below:

Epoch	100
Learning rate	0.001
Batch size	8
Worker	8

Table 3.5: setting of Detectron2

Detectron2 have the highest class-accuracy of 98.44% with total loss of 37.16%.



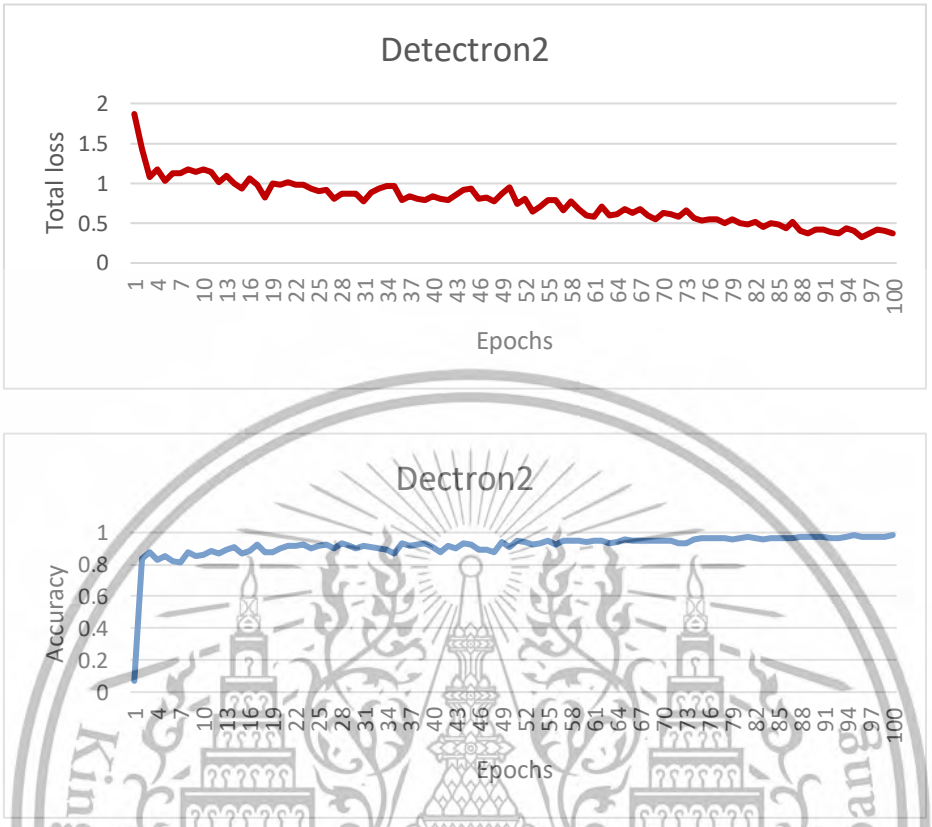


Fig 4.8: Raw and graph result from Detectron2

Mobile application

Results of output from android studio (left) comparing with actual results from dataset (right) are show below:

Image 112.jpg

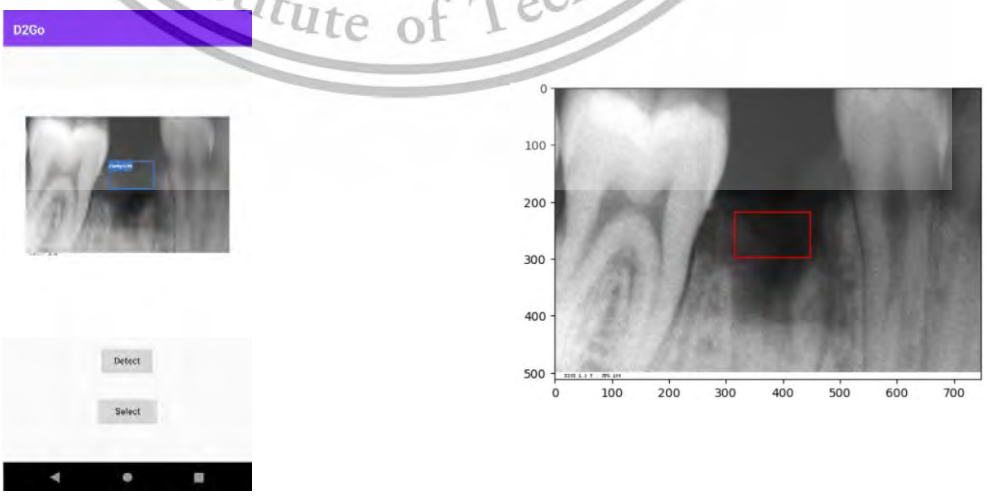


Image 209.jpg

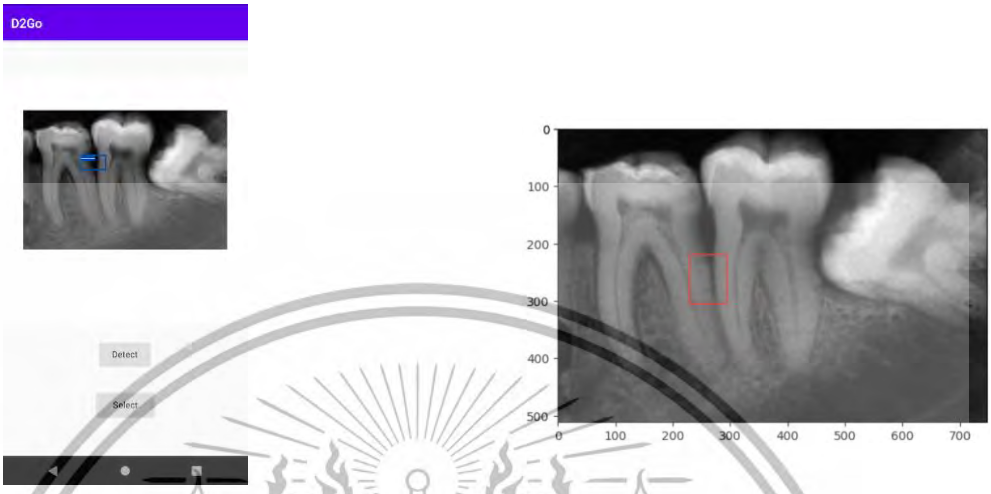


Fig 4.9: Test result from Android studio

4.3 Discussion

- The difference between YOLO V8 and YOLO-NAS is YOLO-Nas excel as detect small object and has better use for real-time edge-device applications than YOLO V8 due to its efficiency, accuracy, and performance-per-compute ratio.
- The reason of Detrctron2 has high accuracy and precision is because it uses fast-RCNN for training. However, YOLO V8 and YOLO-NAS still use less time to train.
- DETR performs the worst, mostly due to the dataset's poor compatibility with the model. As it often raises the error message "image XXX not found" indicates that the model is unable to properly train the dataset.
- When converting models into mobile-compatible model, performance and efficient had been reduce, this is because model itself need to be compressed and optimized. Resulting to loss some of performance. Another Factor to be consider is hardware limitation, as it has been tested on visual mobile smartphone at android studio.

4.4 Summary

This section represents the results of object detection experiments using the YOLO V8, YOLO-NAS, DETR and Detectron2 models on a dataset of x-ray images. The dataset contains a type of oral disease.

Our results indicate that Detectron2 achieved the highest average precision (mAP) across all object classes, with an impressive 97.6% accuracy. This performance surpasses that of YOLO V8 (84.28%) and YOLO-NAS (84.3%), highlighting Detectron2's superior object detection capability.

CHAPTER 5
CONCLUSION

5.1 Introduction

This chapter contains a summary of every experiment chapter comparing deep learning-based models for the diagnosis of oral diseases. It concludes with an analysis of how well each model performs in terms of objections from pockets and cavities on x-ray oral pictures, as well as the theory and technique behind each model. Followed by result, discussion, and conclusion.

5.2 Summary

Chapter 1 outlines the oral disease problem and the effectiveness of object detection.

In Chapter 2, the latest advancements in deep learning and oral health were examined, along with an explanation of the theory behind each model used in this experiment (YOLO V8, YOLO-NAS, DETR, and Detectron2). Finally, introduce the Android application.

The experiment's design methodology, along with the challenge and solution, are covered in Chapter 3.

In Chapter 4, each model's output is displayed as raw data, a graph, or text, along with how it was used for training. Observe the outcome on the mobile app. This chapter also discusses the experiment's results and the performance of some models.

Chapter 5 presented a conclusion of each chapter as described above.

5.3 Conclusions

The aim of this project is described as follow.

- Compare each select model that use for object detection on custom oral dataset.
- Port the suitable model to mobile app and use it for object detection.

In conclusion, oral object detection on a customized dataset is a difficult but interesting task in computer vision and AI. This project investigated the complexities of training a model to identify and locate objects in python language during this process, an area that presents special opportunities and challenges.

Custom datasets are essential to this project because they let us customize the model for use in particular fields and applications. The goal is to improve the model's performance and make it more applicable to real-world scenarios by carefully selecting and annotating a dataset that accurately represents the variety and intricacy of the target objects and spoken descriptions.

For result, Detectron2 show the best for all 4 models, which then use it for mobile application develop on android studio.

5.4 Suggestions

- There are many more models that can be trained using custom-dataset but did not experiment it due to time-limitation. If possible, providing more time will increase the efficiency of outcome on this experiment.
- Because oral datasets are highly confidential and are referred to as "patient data," they are challenging to access. Furthermore, some datasets are not very useful for object detection. Find the best dataset to boost the model's efficiency, as suggested.

REFERENCES

[1] S. Shah, A. Abaza, A. Ross, and H. Ammar, "Automatic Tooth Segmentation Using Active Contour Without Edges," in *2006 Biometrics Symposium: Special Session on Research at the Biometric Consortium Conference*, Sep. 2006, pp. 1–6. doi: 10.1109/BCC.2006.4341636.

[2] Reported by Khon Kaen University Staffs during 1984 to 2020. In *Oral Health Care—An Important Issue of the Modern Society*. IntechOpen.

[3] L. Alzubaidi *et al.*, "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions," *J. Big Data*, vol. 8, no. 1, p. 53, Mar. 2021, doi: 10.1186/s40537-021-00444-8.

[4] S. Gherunpong, G. Tsakos, and A. Sheiham, "The prevalence and severity of oral impacts on daily performances in Thai primary school children," *Health Qual. Life Outcomes*, vol. 2, no. 1, p. 57, 2004, doi: 10.1186/1477-7525-2-57.

[5] S. Adulyanon, J. Vourapukjaru, and A. Sheiham, "Oral impacts affecting daily performance in a low dental disease Thai population," *Community Dent. Oral Epidemiol.*, vol. 24, no. 6, pp. 385–389, Dec. 1996, doi: 10.1111/j.1600-0528.1996.tb00884.x.

[6] T. Tussanapirom, P. Siribal, P. Trirattanaphinthusorn, W. Kengtong, and P. Gaewkhiew, "Economic burden of becoming a dentist in Thailand," *BDJ Open*, vol. 9, no. 1, p. 5, Feb. 2023, doi: 10.1038/s41405-023-00131-1.

[7] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, "Deep learning applications and challenges in big data analytics," *J. Big Data*, vol. 2, no. 1, p. 1, Dec. 2015, doi: 10.1186/s40537-014-0007-7.

[8] X. Zou, "A Review of Object Detection Techniques," in *2019 International Conference on Smart Grid and Electrical Automation (ICSGEA)*, Xiangtan, China: IEEE, Aug. 2019, pp. 251–254. doi: 10.1109/ICSGEA.2019.00065.

[9] R. Girshick, "Fast R-CNN," 2015, doi: 10.48550/ARXIV.1504.08083.

[10] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2015, doi: 10.48550/ARXIV.1506.02640.

[11] M. Hussain, “YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection,” *Machines*, vol. 11, no. 7, p. 677, Jun. 2023, doi: 10.3390/machines11070677.

[12] G. Jocher *et al.*, “ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation.” Zenodo, Nov. 22, 2022. doi: 10.5281/ZENODO.7347926.

[13] Developers, Android. "What is android." Dosegljivo: http://www. academia. edu/download/30551848/andoid--tech. pdf (2011).

[14] Dwyer, B., Nelson, J. (2022), Solawetz, J., et. al. Roboflow (Version 1.0) [Software]. Available from https://roboflow.com. computer vision.

[15] Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, “Object Detection With Deep Learning: A Review,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 11, pp. 3212–3232, Nov.[18] 2019, doi: 10.1109/TNNLS.2018.2876865.

[16] A. Paul and S. Venkatasubramanian, “Why does Deep Learning work? - A perspective from Group Theory,” 2014, doi: 10.48550/ARXIV.1412.6621.

[17] Umer Rashid, “Dental Caries Dataset.” Zenodo, Jun. 07, 2021. doi: 10.5281/ZENODO.4907879.

[18] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, “End-to-End Object Detection with Transformers,” in *Computer Vision – ECCV 2020*, vol. 12346, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds., in Lecture Notes in Computer Science, vol. 12346. , Cham: Springer International Publishing, 2020, pp. 213–229. doi: 10.1007/978-3-030-58452-8_13.

[19] J. Liu and J. Yu, “Research on Development of Android Applications,” in *2011 4th International Conference on Intelligent Networks and Intelligent Systems*, Kuming, China: IEEE, Nov. 2011, pp. 69–72. doi: 10.1109/ICINIS.2011.40.

[20] Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y. & Girshick, R. (2019). Detectron2 url{https://github.com/facebookresearch/detectron2}.

[21] R. A. Güler, N. Neverova, and I. Kokkinos, “DensePose: Dense Human Pose Estimation In The Wild,” 2018, doi: 10.48550/ARXIV.1802.00434.

[22] A. Paszke *et al.*, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” 2019, doi: 10.48550/ARXIV.1912.01703.

[23] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," 2016, doi: 10.48550/ARXIV.1605.08695.

[24] R. Del Prete, M. D. Graziano, and A. Renga, "RetinaNet: A deep learning architecture to achieve a robust wake detector in SAR images," in *2021 IEEE 6th International Forum on Research and Technology for Society and Industry (RTSI)*, Naples, Italy: IEEE, Sep. 2021, pp. 171–176. doi: 10.1109/RTSI50628.2021.9597297.