

A project report on

THEFT DETECTION USING MACHINE LEARNING

Submitted in partial fulfillment for the award of the degree of

B.Tech- Computer Science and Engineering Specialization in Data Analytics

by

MAJJIGA JASWANTH (20BCD7171)

MULLAPUDI SRI SURYA TARUN (20BCD7030)



**SCHOOL OF COMPUTER SCIENCE AND
ENGINEERING (SCOPE)**

May, 2024

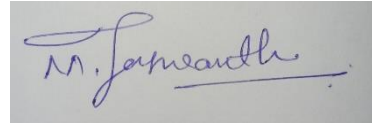
DECLARATION

I hereby declare that the thesis entitled “THEFT DETECTION USING MACHINE LEARNING” submitted by me, for the award of the degree of Specify the name of the degree VIT is a record of bonafide work carried out by me under the supervision of Dr. E. Ajith Jubilson

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Amaravati

Date: 24/05/2024

A handwritten signature in blue ink, appearing to read "M. Jayanthi", is written on a light-colored rectangular background.

Signature of the Candidate

CERTIFICATE

This is to certify that the Senior Design Project titled “**THEFT DETECTION USING MACHINE LEARNING**” that is being submitted by **M. JASWANTH (20BCD7171), M. SRI SURYA TARUN (20BCD7030)** is in partial fulfilment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.



Dr. Ajith Jubilson, PhD
Phone: +91-8632370164(O)
Associate Professor
+91- 9994514352 (M)
Deputy Director, Software Development Cell (SDC)
E-mail: ajith.jubilson@vitap.ac.in

Project Guide

Dr. Ajith Jubilson

The thesis is satisfactory



Internal Examiner

Dr. Chittipireddi Koteswararao



External Examiner

Dr. Suresh Dara

Approved by

HoD, Department of Data Science and Engineering
School of Computer Science and Engineering

THEFT DETECTION

USING

MACHINE LEARNING

ABSTRACT

Using the YOLOv8 object identification algorithm, our project's main goal is to create a theft detection system and integrate it into the Flask web application allowing real-time monitoring. We carefully classified objects of interest, such as axes, knives, heavy weapons, handguns, people, masks, and helmets, by using a large dataset of 13,382 photos that were gathered and annotated using Roboflow. To increase the robustness of the model, preprocessing operations like flipping, rotating, and resizing were applied to the dataset. Using Google Colab Pro to train the YOLOv8 model for 150 epochs produced a precision-recall curve of 0.935 at mAP @ 0.5, which shows good detection accuracy in all classes. Users can upload photos and videos for quick theft detection thanks to the integration with Flask, which enables a smooth deployment process. Our system seeks to offer a practical solution for real-time security and surveillance applications by integrating cutting-edge computer vision techniques and intuitive web interfaces. It has the potential to be widely implemented in a variety of security-sensitive settings.

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to Dr. E. Ajith Jubilson, Associate Professor Grade-2 , SCOPE, VIT-AP, for her constant guidance, continual encouragement, understanding; more than all, he taught me patience in my endeavor. My association with her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Artificial intelligence and machine learning . I would like to express my gratitude to Dr. G. Viswanathan, Dr. G. V. Selvam, Mr. Sankar Viswanathan, Dr. Sekar Viswanathan, Dr. S. V. Kota Reddy, and Dr. CH. Pradeep Reddy School of Computer Science and Engineering(SCOPE), for providing with an environment to work in and for his inspiration during the tenure of the course.

In jubilant mood I express ingeniously my whole-hearted thanks to Dr.G.Muneeswari . HOD / Department of Data Science and Engineering, all teaching staff and members working as limbs of our university for their not-self-centered enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Amaravati

Date: 24/05/24

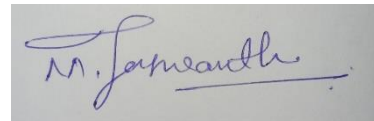
A handwritten signature in blue ink, reading "M. Jayanthi", with a horizontal line underneath the name.

TABLE OF CONTENTS

CONTENTS.....	Vii
LIST OF FIGURES.....	X

CHAPTER 1

INTRODUCTION

1.1	GENERAL OVERVIEW OF PROJECT.....	1
1.2	LITERATURE SURVEY.....	1
1.3	INTRODUCTION TO PYTHON.....	3
1.4	PYTHON LIBRARIES IN PROJECT.....	4
1.5	ESSENCE OF DEEP LEARNING.....	6
1.6	IMPORTANCE OF COMPUTER VISION.....	7
1.7	BRIEF VIEW OF ULTRALYTICS.....	7
1.8	HISTORY OF YOLO.....	9
1.9	OBJECTIVES.....	19

CHAPTER 2

DATA COLLECTION AND PRE PROCESSING

2.1	WHAT IS ROBOFLOW?.....	21
2.2	WHAT IS ROBOFLOW UNIVERSE AND DATASETS?.....	22
2.3	HOW WE ANNOTATE IMAGE?.....	23
2.4	DATA PRE-PROCESSING METRICS.....	24

CHAPTER-3

IMPLEMENTATION AND TESTING OF THE THEFT DETECTION SYSTEM

3.1 INTRODUCTION TO HARDWARE AND SOFTWARE SETUP.....	26
3.2 DATA COLLECTION AND PRE-PROCESSING.....	27
3.3 MODEL TRAINING AND EVALUATION.....	28
3.4 DEPLOYMENT AND INTEGRATION.....	29
3.5 PERFORMANCE EVALUATION AND FUTURE IMPROVEMENTS.....	30

CHAPTER - 4

PROTOTYPE DEPLOYMENT

4.1 CHALLENGES FACED IN THE PROJECT.....	32
4.2 IMPLEMENTATION OF TRAINED MODEL IN FLASK AND DEPLOYMENT OF WEIGHTS IN ROBOFLOW.....	33
4.3 INTEGRATION OF `BEST.PT` FILE AND API KEY FROM ROBOFLOW.....	35
4.4 ACTIVITY DIAGRAM.....	39

CHAPTER – 5

USER INTERACTION AND FEEDBACK

5.1 USER INTERACTION.....	41
5.2 USER SUGGESTIONS.....	42
5.3 ADVANTAGES AND DISADVANTAGES OF OUR PROJECT.....	42

CHAPTER-6

RESULTS

6.1 F1-CURVE.....	44
6.2 PRECISION- CONFIDENCE CURVE.....	45
6.3 RECALL CONFIDENCE CURVE.....	46
6.4 PRECISION-RECALL CURVE.....	48

CHAPTER-7

CONCLUSION & FUTURE WORKS.....	50
--------------------------------	----

APPENDICES.....	52
-----------------	----

REFERENCES.....	60
-----------------	----

LIST OF FIGURES

1.0	BASIC ARCHITECTURE OF YOLO.....	9
1.1	HOW DOES YOLO WORK.....	11
1.2	MAX POOLING LAYERS OF YOLO V2.....	12
1.3	YOU ONLY LEARN ONE REPRESENTATION (YOLOR)	14
1.4	YOLOX.....	15
1.5	ARCHITECTURE OF YOLO V6.....	16
1.6	ARCHITECTURE OF YOLO V7.....	17
1.7	ARCHITECTURE OF YOLOv8.....	18
1.8	EVOLUTION OF YOLO.....	19
3.0	HARD WARE CONFIGURATION.....	27
4.0	ACTIVITY DIAGRAM.....	39
6.0	F1-CURVE.....	44
6.1	PRECISION- CONFIDENCE CURVE.....	45
6.2	RECALL CONFIDENCE CURVE.....	46
6.3	PRECISION-RECALL CURVE.....	48

CHAPTER:1

INTRODUCTION

1.1 GENERAL OVERVIEW OF THE PROJECT

The goal of this project is to create a sophisticated theft detection system that can instantly recognize weapons of mass destruction like knives, rifles, axes, and hammers. This system seeks to improve security measures by using the cutting-edge YOLOv8 (You Only Look Once) algorithm to provide instant alerts and notifications, when possible, risks are identified. The project aims to improve personal and asset security by combining deep learning and computer vision techniques to produce a reliable and effective solution that can be used in a variety of monitoring scenarios.

Because it offers a faster and more precise detection mechanism than existing security systems, this initiative holds great potential to change the industry. Manual monitoring, which takes time and is prone to human error, is a common component of traditional surveillance systems. The suggested solution, on the other hand, uses artificial intelligence to autonomously and precisely identify threats, speeding up response times and lowering the possibility of theft-related occurrences. By protecting assets and guaranteeing a safer environment, the successful completion of the following endeavour will not only strengthen the safety systems but also give people and companies piece of mind.

1.2 LITERATURE SURVEY:

Over the past ten years, there have been notable breakthroughs in the field of computer vision and deep learning-based theft detection. Different object identification methods have been studied extensively; each has advantages and disadvantages. Conventional techniques mostly depended on manually designed elements and rule-based methodologies, which frequently had limitations when it came to managing a wide range of intricate situations. Convolutional neural networks (CNNs), in particular, represented a revolution in deep learning by providing more reliable and precise detection capabilities. The foundation for later advancements was laid by early work

using Alex Net by researchers Krichevsky et al., which showed the promise of deep learning in object identification tasks.

Girshick et al.'s R-CNN family of algorithms is one of the most important contributions to object detection. Gradually, the effectiveness and precision of object detection in images were enhanced by R-CNN, Fast R-CNN, and Faster R-CNN. These models greatly improved detection performance by generating possible object locations using region proposal networks (RPNs). Real-time applications faced difficulties because these multi-stage models required a lot of computing power. Despite these drawbacks, R-CNN and its variations created new standards and sparked interest in more effective designs.

Introduced YOLO (You Only Look Once), which became a ground-breaking method for object recognition. In contrast to earlier models, YOLO treated object recognition as a single regression issue, using entire images as the basis for direct prediction of bounding boxes and class probabilities in a single evaluation. Real-time object detection became possible and speed was significantly increased with this end-to-end training method. YOLOv2, YOLOv3, and YOLOv4 were some of the variations that came after the original YOLO model, and they all brought little gains in accuracy and functionality. Yolo is a well-liked option for applications needing quick and precise detection because of its effectiveness and simplicity.

Concurrently provided another effective option for real-time object detection in the form of the SSD (Single Shot Multi Box Detector). YOLO and R-CNN elements were integrated by SSD to predict bounding boxes and class scores from several feature maps at various scales. SSD is appropriate for a wide range of applications because of its multi-scale approach, which improved its ability to detect objects of different sizes. SSD typically required more processing power than YOLO, even with its advantages especially in situations involving high-resolution photographs.

The development of Ultralytics YOLOv5 and later iterations carried on the tradition of improving object detecting capabilities. YOLOv5 achieved state-of-the-art performance on multiple benchmarks by integrating numerous advances in data augmentation, architectural design, and training methodologies. Notably, YOLOv5's accessibility as an open-source project and its simplicity of usage encouraged broad acceptance in academic and professional contexts. YOLOv5 has been used by

researchers and practitioners in a variety of fields, including autonomous driving and medical imaging.

The most recent development in this family is YOLOv8, which incorporates new ideas while enhancing the qualities of its predecessors. It is especially well-suited for situations involving real-time theft detection since it provides increased accuracy, speed, and resilience. Research has demonstrated that YOLOv8 routinely outperforms competing models, including a quicker R-CNN and SSD, in terms of both precision in detection and processing performance. Due to its ability to identify different robbery objects in real-time and its ability to ensure increased security and prompt response to potential threats, it is the perfect option for establishing theft detection systems.

1.3 INTRODUCTION TO PYTHON:

One of the most widely used programming languages for software development is Python, a high-level language valued for both its ease of use and adaptability. Python was developed by Guido van Rossum and was made available in 1991 with a focus on user-friendliness and readable code. Because of its simple syntax, developers are able to generate more productive and error-free code that is easily understood. Because of its emphasis on readability, Python is a great option for programmers of all skill levels.

Python is known for its large standard library, which contains functions and modules for a wide range of activities like file I/O, system operations, and internet protocols. With the help of this extensive library, developers may do a variety of programming jobs without having to start from scratch. Furthermore, Python offers developers freedom in their approach to problem-solving by supporting a variety of programming paradigms, such as procedural, object-oriented, and functional programming. Because of its versatility, Python has become widely used in a variety of sectors and domains.

Python's robust libraries and frameworks have made it especially popular in the fields of artificial intelligence, machine learning, and data science. Machine learning models are commonly constructed using TensorFlow, Keras, and PyTorch, while libraries like NumPy and pandas facilitate data processing and analysis. OpenCV is a computer vision package that demonstrates the usefulness of Python in specific fields. Since these libraries, Python is the language of choice for academics and professionals

working on sophisticated projects since it has the capabilities needed to perform intricate calculations and model building.

Python's adaptability applies to developing websites, digitization, scientific computing, and other fields beyond data science and AI. While Selenium and Pretty Soup are commonly utilized for web scraping along with automation operations, frameworks such as Flask as well as Django are prominent for building web apps. Libraries like SciPy and Matplotlib facilitate sophisticated mathematical calculations and data visualization in scientific computing. Python's broad range of applications, along with its easy-to-use and strong features, guarantee its popularity and relevance in the rapidly changing software development and technological scene.

1.4 PYTHON LIBRARIES IN PROJECT:

1. Ultralytics: An advanced implementation of the YOLO (You Only Look Once) object identification technique is the Ultralytics YOLOv8 package. This library is essential to our endeavour to create a reliable theft detection system. To make training and deploying YOLOv8 models easier, it offers pre-trained models, training scripts, and tools. Because of its great speed optimization, the Ultralytics library is perfect for jobs involving the real-time detection of objects.
2. OpenCV: A strong library for computer vision applications is called OpenCV (Open Source Computer Vision Library). Real-time image and video processing is done in our project using OpenCV. It makes visual data capture, processing, and analysis possible, which makes it easier to integrate live camera feeds with the YOLOv8 model for real-time object detection. OpenCV is a vital tool for the project because of its wide range of features for feature identification, picture modification, and image preprocessing.
3. TensorFlow and Keras: Google created the open-source TensorFlow deep learning framework, and Keras is an API for high-level neural networks that runs on top of TensorFlow. In order to construct and train the YOLOv8 model, these libraries are essential. Deep learning model construction and training is made easier by Keras's intuitive interface, while TensorFlow handles the

backend support for the computational operations required for model training. Combining them allows the YOLOv8 architecture to be implemented effectively and the model to be optimized for precise theft detection.

4. NumPy and pandas: A core package for computational mathematics in Python, NumPy offers support for matrices and arrays as well as a number of mathematical operations that may be performed on these types of data structures. NumPy is utilized in our project for preprocessing and data manipulation, making sure that the provided data is formatted correctly for model development and evaluation. Conversely, Pandas is an effective library for working with and analysing data. It is utilized to manage structured data, making data transformation, cleaning, and analysis chores more effective. These tasks are essential for getting the dataset ready for YOLOv8 model training.
5. Matplotlib and Seaborn: A complete Python visualization toolkit for static, animate, and interactive visualizations is called Matplotlib. Seaborn offers an advanced interface for creating visually appealing statistical visuals and is developed on top of Matplotlib. The model training and assessment outcomes are visualized in this project using Matplotlib and Seaborn. Plotting performance metrics like accuracy, precision, recall, and loss curves with the aid of these libraries makes it easier to comprehend the model's performance and speeds up the fine-tuning procedure.
6. Flask: We utilize Flask, a lightweight Python web framework, to run the privilege of YOLOv8-based theft detection model as an online application. With Flask, we can build an intuitive user interface that lets users input videos or images for object identification in real time. It offers the infrastructure required to manage requests made via HTTP, process inputs, and show the outcomes, enabling end users to access and interact with the implementation of our detection system.

1.5 ESSENCE OF DEEP LEARNING :

Deep learning completely transforms security management, especially when used to theft detection systems. This project exemplifies the use of deep learning to improve security and surveillance by utilizing Roboflow's preprocessing capabilities and YOLOv8, an enhanced version of the YOLO method. The large dataset of 13,382 photos, which was painstakingly pre-processed using methods like auto-orientation, 640x640 scaling, and augmentations including flipping, rotation, shearing, and saturation tweaks, demonstrates the capacity of deep learning to handle enormous volumes of heterogeneous data.

This research demonstrates one of deep learning's primary advantages: the technology can automatically identify and extract valuable features from unprocessed data. The deep layers and convolutional neural networks (CNNs) of the YOLOv8 architecture enable it to recognize and localize objects in photos with exceptional accuracy. The theft detection system is more reliable and flexible in different surveillance circumstances because of this autonomous feature extraction, which lessens the need for manual feature engineering.

Furthermore, deep learning models with good generalization, such as YOLOv8, are trained to identify items linked to theft, such as axes, hammers, knives, and firearms, in a variety of settings and circumstances. In real-world applications, where surveillance film may change in light, angles, and object orientations, this sort of generalization capability is essential. Roboflow's preprocessing techniques that improve data quality and YOLOv8's resilience guarantee dependable and consistent results in theft task detection.

The continuous learning process is another feature of this project that highlights the basic principles of deep learning. Large datasets are used to train deep learning models, such as YOLOv8, iteratively modifying internal variables (weights and biases) to reduce prediction errors. The model is skilled at identifying changing theft strategies and items thanks to this iterative learning technique, which enables the model to continuously increase its level of precision and adapt to new data.

YOLOv8's real-time functionality highlights the importance underlying deep learning with theft detection systems even more. YOLOv8 greatly improves security measures

by enabling immediate detection and reaction to possible threats through the real-time processing of streaming video or live camera feeds. In high-security settings where prompt action is necessary to prevent stealing or unauthorized access, this real-time capacity is essential.

1.6 IMPORTANCE OF COMPUTER VISION:

The project that uses Roboflow and YOLOv8 for theft detection heavily relies on computer vision (CV). Real-time analysis and interpretation of visual data from security cameras or picture streams is made possible by the integration of CV algorithms into the YOLOv8 model. By immediately alerting security people or automated systems, this capacity is essential for quickly identifying theft-related items such as knives, rifles, axes, and hammers and improving security procedures.

The capacity of CV to extract relevant information from visual inputs is crucial to our study since it enables the YOLOv8 model to precisely detect and localize objects associated to theft. By combining deep learning approaches with CV algorithms, the system can identify theft with robust and dependable skills, even in complicated situations with shifting lighting conditions and object orientations. Overall, CV is a crucial element for guaranteeing security in a variety of settings since it improves the efficacy and efficiency of the theft detection system.

1.7 BRIEF VIEW OF ULTRALYTICS:

Modern object identification technology is led by Ultralytics, which is especially well-known for using the YOLO (You Only Look Once) algorithm. Developers and researchers studying computer vision are turning to this ground-breaking platform because of its remarkable speed optimization and intuitive interface. Ultralytics is a key player in the YOLOv8 and Roboflow theft detection project since it provides an extensive toolkit and features that improve security and monitoring.

YOLO-based model training, deployment, and fine-tuning are made easier with Ultralytics' focus on speed and efficiency. It makes the creation of unique object detection systems easier for even individuals with little experience with deep learning,

thanks to its pre-trained models, extensive training scripts, and integrated data augmentation approaches. Furthermore, Ultralytics' real-time processing powers allow for the prompt identification and handling of possible theft situations, greatly enhancing security protocols across a range of contexts.

Discovering more about Ultralytics' features and advantages makes it clear that this platform is more than simply a tool—rather, it's a revolutionary advancement in the field of object identification and monitoring. It is an essential tool for developing strong theft detection systems that can adjust to changing security threats due to its exceptional accuracy, dependability, and future-proofing qualities.

Applications Of Ultralytics:

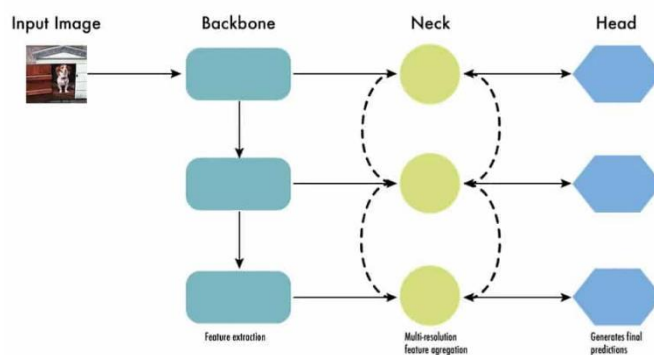
Real-time object detection and analysis find significant applications in a variety of sectors where ultralytics is useful. It is used in surveillance and security to find and follow objects of interest, strengthening security protocols in public areas, businesses, and residential buildings. Furthermore, ultralytics is essential to the automotive sector because it allows self-driving cars to recognize and avoid obstacles, pedestrians, and other cars in real time, which promotes safe road navigation. By recognizing product placements, monitoring inventory movement, and spotting theft-related activity, it helps with inventory management, theft prevention, and customer behaviour analysis in retail settings. Additionally, Ultralytics is used in medical imaging analysis, patient monitoring, and surgical support settings in healthcare settings, which enhances surgical precision and patient care.

Features Of Ultralytics:

Ultralytics has several characteristics that help it be both versatile and effective in object detection jobs. The availability of pre-trained models, such as YOLOv5 and YOLOv8, which enable users to begin object identification tasks without requiring substantial training, is one of its primary benefits. Furthermore, Ultralytics offers training scripts that handle data preparation, model construction, and training iterations automatically for creating bespoke YOLOv8 models using user-defined datasets. The integrated data augmentation methods such as flipping, rotation, and shearing which improve the

robustness and diversity of training data and provide more accurate and dependable object detection models, are another noteworthy aspect. Additionally, real-time analysis of video streams or picture feeds is made possible by Ultralytics streamlined design, which qualifies it for applications like theft detection systems that call for prompt detection and response.

1.8 HISTORY OF YOLO:



The architecture of object detectors is divided into three parts: the backbone, the neck, and the head.

In order to extract useful characteristics from input photos, a convolutional neural network (CNN) that has been trained on extensive image classification tasks such as ImageNet is usually used as the backbone. At different scales, the backbone records hierarchical features. Higher-level characteristics (such object pieces and semantic information) are eliminated in the deeper levels, while lower-level features (like edges and textures) are extracted in the preceding layers. The neck serves as a transitional structure that joins the head and backbone.

Measuring the performance of object detection models:

metrics and non-maximum suppression (NMS)

The statistic used to assess object detection models is called mean Average Precision, or mAP for short. It gives a single number to compare several models by calculating the average precision across all categories.

How does mAP work?

$$mAP = \frac{1}{|Classes|} \sum_{c \in Classes} \frac{|TPc|}{|FPc| + |TPc|}$$

Eq 1.0

Intersection over Union (IoU) is used to define a positive prediction and handle many object categories. These three factors form the foundation of the mAP metric.

YOLO: You Only Look Once

The story of YOLO starts on a sweltering Monday afternoon in June 2016, during the CVPR Conference, which was hosted at the Caesar's Palace Conference Center in Las Vegas, Nevada.

Presenting his article You Only Look Once: Unified, Real-Time Object Detection, Joseph Redmon took the podium. This work presented a novel end-to-end method for object identification that facilitated real-time processing. For the field of computer vision, this was an important breakthrough.

Yolo is an effective object detection architecture that requires only one network pass, in contrast to earlier approaches. It gets rid of the necessity for two-step processes or several runs. On the PASCAL VOC2007 dataset, this model obtained an astounding mAP of 63.4.

How Does YOLO Work?

The input image is divided into a grid by the YOLO object detection algorithm, which then predicts B bounding boxes with confidence scores for C classes per grid element to identify all the bounding boxes simultaneously.

The accuracy and confidence of the model are shown by P_c , which is included in every bounding box prediction. The box's height and breadth in relation to the entire image are represented by the values b_h and b_w , while the box's centers in relation to the grid cell are denoted by the box and by coordinates.

After non-maximum suppression, which can be used to eliminate duplicate detections, the output is a tensor of $S \times S \times (B \times 5 + C)$.

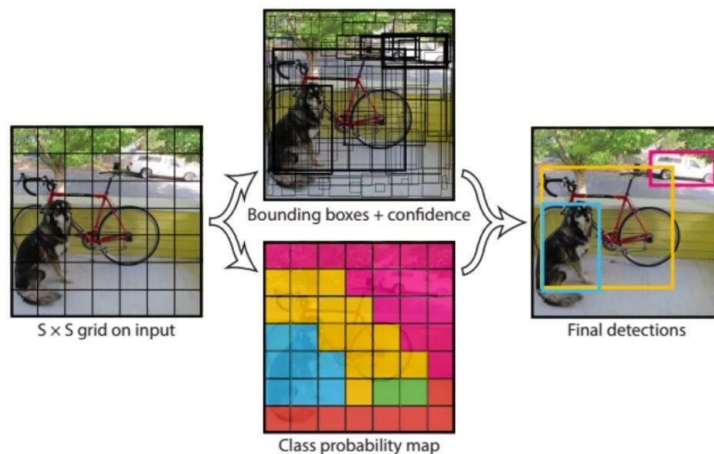


Fig 1.1

YOLOv2: Better, Faster, and Stronger

Redmon took the stage once more on a steamy Tuesday afternoon in July 2017.

During the talk, Redmon presented an object detection system that can recognize more than 9000 categories and shared a paper titled Yolo9000: Better, Faster, Stronger. On the PASCAL VOC2007 dataset, the model known as YOLOv2 had an amazing average accuracy (AP) of 78.6%, surpassing the performance of its predecessor, YOLOv1, which only managed to obtain 63.4%. These findings show how much better YOLOv2 is at identifying and recognising objects, providing new directions for future computer vision research.

The system has been improved in several ways. These improvements include using batch normalization for convolutional layers to improve convergence and reduce overfitting. A high-resolution classifier was added, resulting in better performance for higher-resolution inputs. The architecture was changed to fully convolutional layers – including a backbone called Darknet, which contained 19 convolutional layers and 5 max-pooling layers – and anchor boxes are now used to predict bounding boxes.

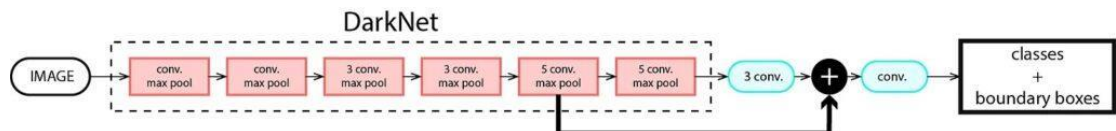


Fig 1.2

YOLOv3:

An incremental improvement to Yologuv, released on arXiv on April 8, 2018, was written by Joseph Redmon and Ali Farhadi.

The object detection benchmark was moved from PASCAL VOC to Microsoft COCO with the release of YOLOv3. Moving on, the MS COCO dataset is used to evaluate each and every YOLO. With a mAP of 60.6% at 20 frames per second, the YOLOv3-spp variant presented in the paper achieved state-of-the-art performance two times quicker. This ends up becoming Joseph Redmon's final iteration of YOLO. To maintain real-time performance and keep up with the state-of-the-art, this article includes massive architectural alterations and considerable modifications. To determine the objectness scores of anchor boxes, YOLOv3 employs logistic regression. Multiple labels can be applied to the same box by training binary classifiers. By using three prior boxes for three sizes, the authors employ k-means to calculate bounding box priors.

YOLOv4:

Joseph Redmon departed from computer vision research due to worries about privacy issues and military uses. Thus, for two years, everyone anxiously anticipated the release of a new iteration of YOLO. Optimal Speed and Accuracy of Object Detection, or

Yolov4, was a study published on arXiv in April 2020 by Alexey Bochkovskiy and colleagues, continuing where Redmon left off.

The community quickly accepted YOLOv4 as the official version, despite its advancements being so great that it retained the core YOLO principles of real-time, open source, single shot, and Darknet infrastructure. By testing a variety of adjustments dubbed "bag-of-freebies" and "bag-of-specials," YOLOv4 attempted to determine the ideal ratio.

Data augmentation is a popular example of a bag-of-freebies technique, which just modifies the training approach and raises training costs without lengthening inference times. However, bag-of-specials techniques result in a modest increase in inference cost but a large improvement in accuracy.

Among these techniques are those for post-processing, merging features, and expanding the receptive field. Mosaic augmentation is a new picture correction that combines four images to improve object detection without needing a bigger mini-batch size. They utilized Drop Block, which is a Dropout substitute for convolutional neural networks with class label smoothing, for regularization.

YOLOv5:

Glenn Jocher of Ultralytics launched YOLOv5 in June 2020, a mere two months after the release of YOLOv4. Although we won't go into that here, there was some debate on the name. Though PyTorch was used in development rather than Darknet, YOLOv5 retains many of the enhancements from YOLOv4.

YOLOv5 is an open-source project that is actively maintained by Ultralytics, with over 250 contributors and regular updates. It is easy to use for implementation, deployment, and training. YOLOv5x, with a picture size of 640 pixels, earned an excellent AP of 50.7% in the MS COCO dataset test-dev 2017. On an NVIDIA V100, it can get 200 FPS with a batch size of 32. An even higher AP of 55.8% can be obtained with YOLOv5 by using a larger input size of 1536 pixels.

Scaled-YOLOv4:

Scanned-YOLOv4 was developed with Pytorch rather than DarkNet, as the YOLOv4 authors revealed in CVPR 2021.

The application of scaling-up and scaling-down strategies was the primary innovation of Scaled-YOLOv4. A quicker model with lesser accuracy is produced by scaling down, whereas an accurate model with higher speed is produced by scaling up. The YOLOv4-tiny architecture was optimized for low-end GPUs and ran at 46 frames per second on a Jetson TX2 or 440 frames per second on an RTX2080Ti, resulting in a 22% mAP on MS COCO. YOLOv4-large was the name of the scaled-up model architecture, which came in three sizes: P5, P6, and P7. With 56% mAP on MS COCO, this architecture—which was created for cloud GPU—achieved state-of-the-art performance, outperforming all earlier models.

YOLOR

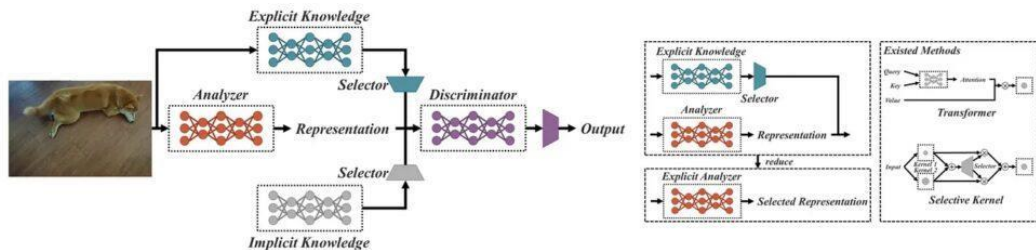


Fig 1.3

You Only Learn One Representation: Unified Network for Multiple Tasks (YOLOR) was published in ArXiv in May 2021 by the same research team of YOLOv4.

The writers adopted a different strategy. In order to construct a single model for many tasks, including classification, detection, and pose estimation, they devised a multi-task learning strategy. By first learning a broad representation, they were able to create task-specific representations by utilizing sub-networks.

YOLOv3 was created to encode neural networks' implicit knowledge so that it may be applied to a variety of tasks, much as how people approach new challenges by drawing on their prior knowledge.

With an NVIDIA V100, YOLOv3 achieved mAPs of 55.4% and mAP50s of 73.3% at 30 frames per second when tested on the MS COCO dataset test-dev 2017.

YOLOX:

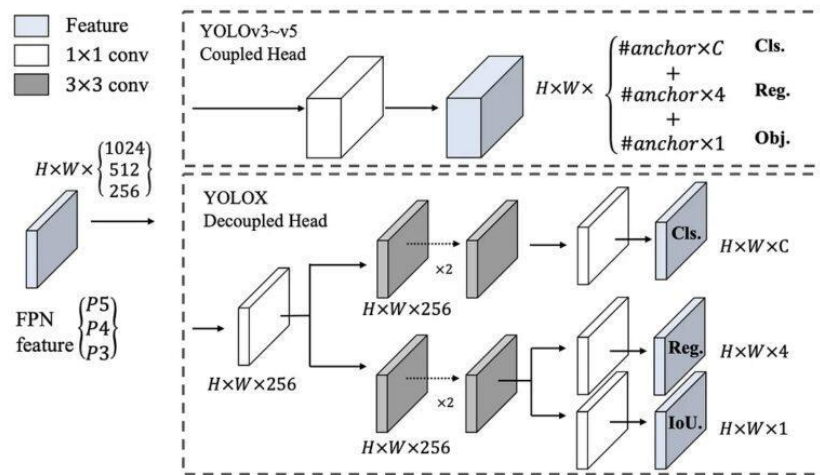


Fig 1.4

Megvii Technology researchers used YOLOv3 from Ultralytics as a starting point to develop YOLOX: Exceeding YOLO Series in 2021, which was published in ArXiv in July 2021.

YOLOX made several changes to its predecessor.

Its lack of anchors makes training and decoding easier, which is a big change. YoloX employs center sampling, designating the center 3×3 region as positive, to make up for the absence of anchors. In order to prevent misalignments, it additionally divides the duties of classification and regression into two heads.

YOLOX also uses Mix UP and Mosaic augmentations, which have been shown to be more advantageous than ImageNet pretraining. YOLOX uses 50.1% mAP on MS COCO to strike a balance between speed and precision.

YOLOv6:

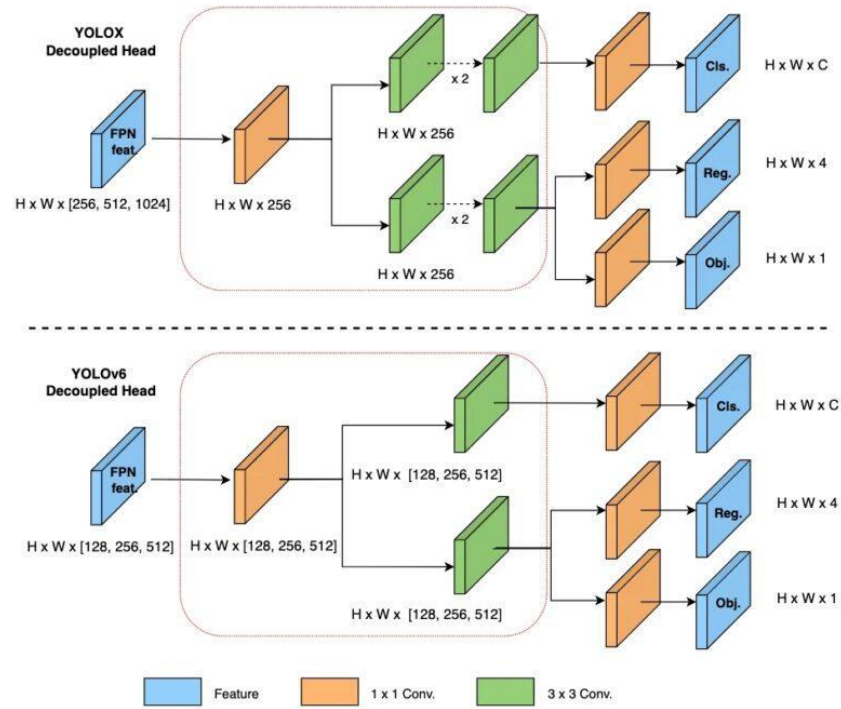


Fig 1.5

YOLOv6: The Meituan Vision AI Department released A Single-Stage Object Detection Framework for Industrial Applications on ArXiv in September 2022. For industrial uses, it offers multiple variants in varying sizes, much as YOLOv4 and YOLOv5.

YOLOv6 implemented an anchor-free detector, following the trend of anchor point-based techniques. With an NVIDIA Tesla T4, YOLOv6-L was able to attain an AP of 52.5% and an AP50 of 70% at about 50 frames per second.

YOLOv7:

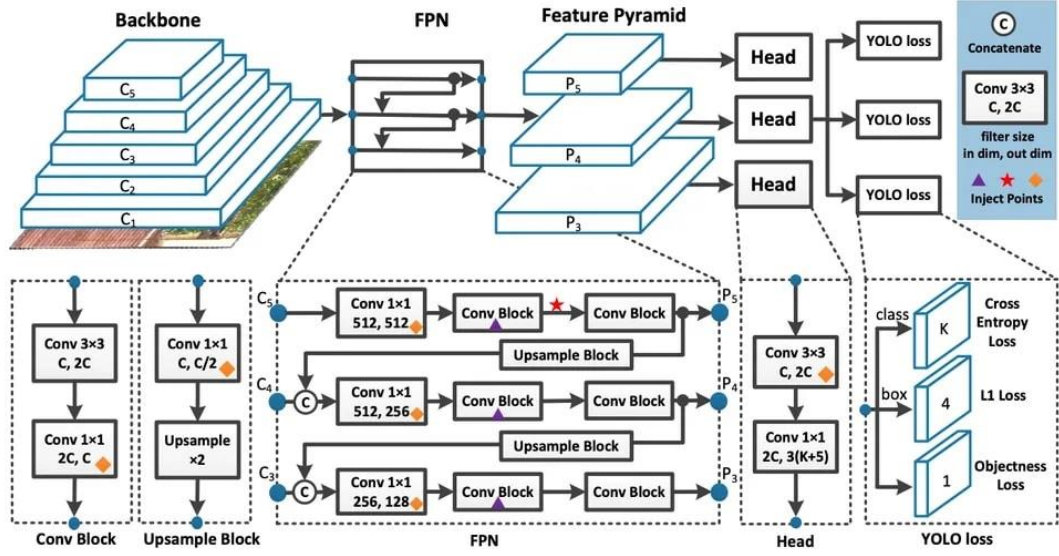


Fig 1.6

The same authors of YOLOv4 and YOLOR released YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors in ArXiv in July 2022. From 5 FPS to 160 FPS, this object detector outperformed all others in terms of speed and accuracy. Similar to YOLOv4, YOLOv7 used no pre-trained backbones and was trained only on the MS COCO dataset.

The three primary components of the YOLOv7 architecture are the "bag-of-freebies" method for accuracy and efficiency, model scaling for different sizes, and E-ELAN for efficient learning. While model scaling modifies features for hardware use, E-ELAN manages gradient routes for deep models. The "bag-of-freebies" strategy involves re-parametrization, which enhances the functionality of the model.

By reducing parameters by 75% and computation by 36%, and increasing average precision by 1.5%, the most recent YOLOv7 model outperformed YOLOv4. Without sacrificing the mAP, YOLOv7-tiny also lowered computation and parameters by 49% and 39%, respectively. YOLOv7 considerably decreased the amount of parameters and computation when compared to YOLOR, by 43% and 15%, respectively, with a minor 0.4% gain in AP.

With an input size of 1280 pixels, YOLOv7-E6 attained an AP of 55.9% and an AP50 of 73.5% on the MS COCO dataset test-dev 2017. Its rapid processing speed of 50 FPS on an NVIDIA V100.

Backbone
YOLOv8 Backbone (P5)

Head YOLOv8Head

Details

Model Architecture

Model Parameters

model	d (depth, multiple)	w (width, multiple)	r (ratio)
n	0.33	0.25	2.0
s	0.33	0.50	2.0
m	0.67	0.75	1.5
l	1.00	1.00	1.0
x	1.00	1.25	1.0

Block Details

Bottleneck (shortcut=True)

SPFP (shortcut=False)

Conv (k, s, p, c)

Conv2d (k, s, p, c)

BatchNorm2d

SILU

Concat

MaxPool2d

AnchorFree

Assigner TAL

Detect

Model Architecture

Model Parameters

model	d (depth, multiple)	w (width, multiple)	r (ratio)
n	0.33	0.25	2.0
s	0.33	0.50	2.0
m	0.67	0.75	1.5
l	1.00	1.00	1.0
x	1.00	1.25	1.0

Block Details

Bottleneck (shortcut=True)

SPFP (shortcut=False)

Conv (k, s, p, c)

Conv2d (k, s, p, c)

BatchNorm2d

SILU

Concat

MaxPool2d

AnchorFree

Assigner TAL

Detect

Model Architecture

Model Parameters

model	d (depth, multiple)	w (width, multiple)	r (ratio)
n	0.33	0.25	2.0
s	0.33	0.50	2.0
m	0.67	0.75	1.5
l	1.00	1.00	1.0
x	1.00	1.25	1.0

Block Details

Bottleneck (shortcut=True)

SPFP (shortcut=False)

Conv (k, s, p, c)

Conv2d (k, s, p, c)

BatchNorm2d

SILU

Concat

MaxPool2d

AnchorFree

Assigner TAL

Detect

Model Architecture

Model Parameters

model	d (depth, multiple)	w (width, multiple)	r (ratio)
n	0.33	0.25	2.0
s	0.33	0.50	2.0
m	0.67	0.75	1.5
l	1.00	1.00	1.0
x	1.00	1.25	1.0

Block Details

Bottleneck (shortcut=True)

SPFP (shortcut=False)

Conv (k, s, p, c)

Conv2d (k, s, p, c)

BatchNorm2d

SILU

Concat

MaxPool2d

AnchorFree

Assigner TAL

Detect

Model Architecture

Model Parameters

model	d (depth, multiple)	w (width, multiple)	r (ratio)
n	0.33	0.25	2.0
s	0.33	0.50	2.0
m	0.67	0.75	1.5
l	1.00	1.00	1.0
x	1.00	1.25	1.0

Block Details

Bottleneck (shortcut=True)

SPFP (shortcut=False)

Conv (k, s, p, c)

Conv2d (k, s, p, c)

BatchNorm2d

SILU

Concat

MaxPool2d

AnchorFree

Assigner TAL

Detect

Model Architecture

Model Parameters

model	d (depth, multiple)	w (width, multiple)	r (ratio)
n	0.33	0.25	2.0
s	0.33	0.50	2.0
m	0.67	0.75	1.5
l	1.00	1.00	1.0
x	1.00	1.25	1.0

Block Details

Bottleneck (shortcut=True)

SPFP (shortcut=False)

Conv (k, s, p, c)

Conv2d (k, s, p, c)

BatchNorm2d

SILU

Concat

MaxPool2d

AnchorFree

Assigner TAL

Detect

Model Architecture

Model Parameters

model	d (depth, multiple)	w (width, multiple)	r (ratio)
n	0.33	0.25	2.0
s	0.33	0.50	2.0
m	0.67	0.75	1.5
l	1.00	1.00	1.0
x	1.00	1.25	1.0

Block Details

Bottleneck (shortcut=True)

SPFP (shortcut=False)

Conv (k, s, p, c)

Conv2d (k, s, p, c)

BatchNorm2d

SILU

Concat

MaxPool2d

AnchorFree

Assigner TAL

Detect

Model Architecture

Model Parameters

model	d (depth, multiple)	w (width, multiple)	r (ratio)
n	0.33	0.25	2.0
s	0.33	0.50	2.0
m	0.67	0.75	1.5
l	1.00	1.00	1.0
x	1.00	1.25	1.0

Block Details

Bottleneck (shortcut=True)

SPFP (shortcut=False)

Conv (k, s, p, c)

Conv2d (k, s, p, c)

BatchNorm2d

SILU

Concat

MaxPool2d

AnchorFree

Assigner TAL

Detect

Ultralytics, the company behind YOLOv5, released YOLOv8 in January 2023.

There are five scaled versions available for YOLOv8: YOLOv8n (nano), YOLOv8s (small), YOLOv8m (medium), YOLOv8l (large), and YOLOv8x (extra-large). Using an image size of 640 pixels, YOLOv8x produced an excellent AP of 53.9% on the MS

COCO dataset test-dev 2017, while YOLOv5 only managed 50.7% on the same input size. It also moves quickly.

Evolution of YOLO:

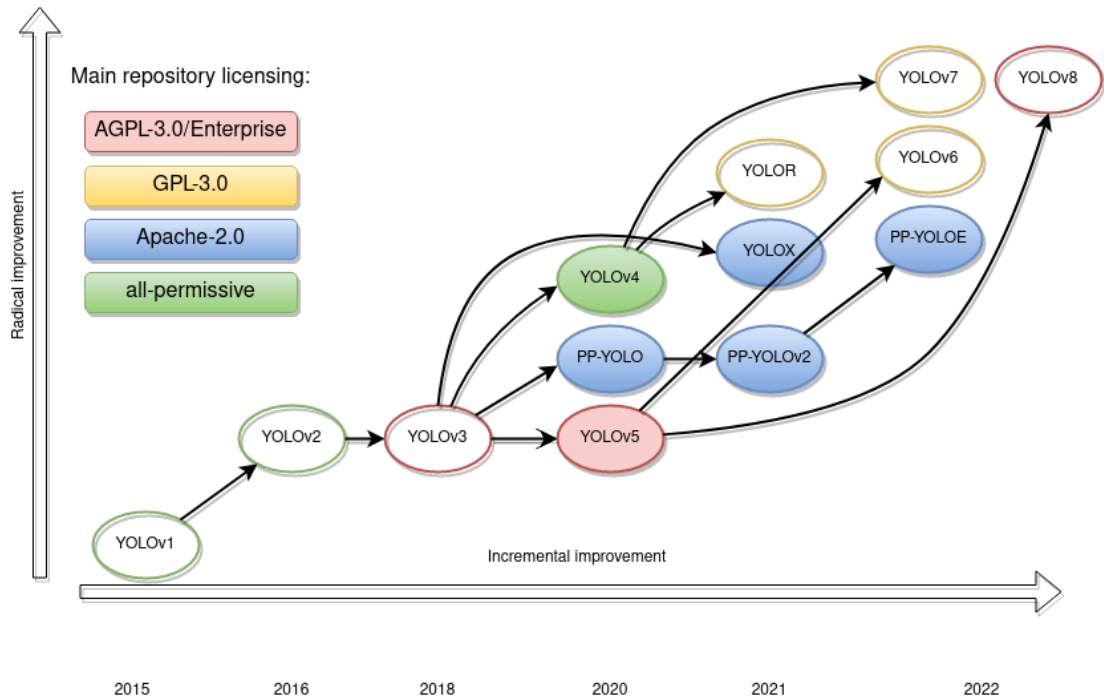


Fig 1.8

1.9 OBJECTIVES:

1. **Design a Reliable Detection System:** The main goal is to build a sophisticated theft detection system that can quickly and precisely identify items linked to theft, like axes, hammers, knives, and firearms. By using sophisticated object detecting algorithms, the system seeks to strengthen security safeguards.
2. **Boost Real-Time Response:** By putting YOLOv8, which is designed for rapid and accurate object recognition, into practice, the project aims to improve real-time response capabilities. In situations involving theft, this goal is essential for guaranteeing prompt warnings and alarms.

3. **Optimize Detection Accuracy:** Improving object detection accuracy is a further goal, especially in demanding and dynamic situations. To increase the detecting system's dependability, variables including changing lighting, item orientations, and crowded backgrounds must be taken into consideration.
4. **Integrate with Surveillance Systems:** The project's goal is to smoothly incorporate the theft detection system into any security frameworks or surveillance systems that are already in place. This integration combines object detection with additional surveillance features to ensure a thorough security strategy.
5. **Customize for Specific Requirements:** The goal of customization is to fit the theft detection system to needs, including differing security configurations or kinds of items linked to theft. This modification guarantees that the system may be adjusted to various security situations.
6. **Ensure Scalability and Robustness:** To guarantee the fact that theft detection system can manage a range of workloads, grow with the demands of security, and continue to function reliably in a variety of operational scenarios, robustness and scalability are essential goals.

CHAPTER: 2

DATA COLLECTION AND PRE PROCESSING

2.1 WHAT IS ROBOFLOW?

A platform called [Roboflow](#) is intended to make the process of creating, honing, and implementing computer vision models more efficient. In order to help developers, researchers, and companies create high-quality computer vision applications, it offers an extensive array of tools and services. Among Roboflow's salient characteristics are:

1. **Dataset Management:** Databases may be organized, imported, and preprocessed with ease using Roboflow. Apart from handling a broad range of labelling tasks like object identification, image segmentation, and classification, it supports multiple image formats.
2. **Data Augmentation:** To improve the variety and caliber of the training data, the platform provides strong data augmentation methods. To enhance model performance, users can execute transformations including flips, rotations, color modifications, and more.
3. **Annotation Tools:** Roboflow offers simple-to-use annotation tools for picture labeling. Preparing datasets for various computer vision applications is made easier by these tools, which enable a variety of annotation types.
4. **Model Training:** Users can use predefined scenarios on the platform to train their models directly. Roboflow facilitates the use of well-known machine learning frameworks, including PyTorch, TensorFlow, and YOLO (You Only Look Once).
5. **Model Deployment:** Roboflow offers hosting solutions and APIs to make the deployment process simpler. This lets users use their trained models as independent services or incorporate them into apps.
6. **Collaboration and Sharing:** By enabling users to share datasets, models, and findings with other team members or the larger community, the platform promotes cooperation.

2.2 WHAT IS ROBOFLOW UNIVERSE AND DATASETS?

Roboflow Universe:

A community-driven portal, [Roboflow Universe](#) allows users to explore, discuss, and work together on open-source computer vision projects. It acts as a storehouse for pre-trained models and datasets provided by the Roboflow community. Among Roboflow Universe's salient features are:

1. **Public Datasets:** For a variety of computer vision applications, including object detection, image segmentation, and classification, users can peruse an extensive assortment of publicly accessible datasets. Numerous industries are covered by these datasets, including transportation, agriculture, healthcare, and more.
2. **Pre-trained Models:** A selection of pre-trained models are available on Roboflow Universe and can be used as a basis for additional customisation and training or for rapid prototyping.
3. **Community Contributions:** The Roboflow Universe allows organizations, developers, and researchers to contribute their models and datasets, promoting knowledge sharing and cooperation among computer vision experts.
4. **Project Showcase:** Contributors can provide information about their datasets, techniques, and results, as well as showcase their computer vision projects. This feature boosts visibility and encourages feedback and teamwork.

Roboflow Datasets:

Computer vision models are trained, validated, and tested using sets of labeled images known as "[roboflow datasets](#)". For organizing and preparing these datasets, the platform offers a wide range of tools:

1. **Importing Data:** Images can be imported by users from a variety of locations, such as other platforms, cloud storage, and local storage. Roboflow is capable of efficiently managing huge datasets and supports a variety of image formats.
2. **Annotation Tools:** Annotation tools supported by Roboflow are easy to use and can accommodate several annotation types, including polygons, bounding

boxes, and segmentation masks. These instruments make it easier to precisely and accurately classify photos.

3. **Data Preprocessing:** A variety of preprocessing options are available on the platform to get datasets ready for training. Users can guarantee that the photos are appropriate for their particular computer vision applications by applying transformations including scaling, cropping, and normalization.
4. **Data Augmentation:** Roboflow offers a variety of augmentation approaches to improve the training data's robustness and diversity. These consist of noise addition, color modifications, and geometric transformations (such as rotations and flips), among others.
5. **Versioning and Management:** Roboflow facilitates the management of many dataset versions, hence facilitating the tracking of modifications and enhancements over an extended period. This feature is very helpful for experimentation and iterative development.
6. **Integration with Training Frameworks:** When datasets are ready, they may be easily combined with well-known machine learning frameworks like PyTorch, TensorFlow, and YOLO. To help with this integration, Roboflow offers export choices in a variety of formats.

2.3 HOW WE ANNOTATE IMAGE?

Our methodical approach of annotating our photos comprised gathering them from Roboflow and grouping them into multiple categories. In order to start our theft detection research, we first gathered a wide range of pertinent photos. Through the cloning of pre-existing datasets that suited the needs of our research, Roboflow provided these photographs. Our picture acquisition process was made easier in the beginning thanks to Roboflow's stable platform and pre-annotated datasets.

Following the acquisition of the photographs, we went ahead and annotated them according to the particular objects that we were trying to find. This required classifying the pictures into multiple groups, including axes, hammers, knives, and firearms. Bounding boxes were created around the objects of interest in each carefully scrutinized photograph. The associated class names were then written on these enclosing boxes.

For accurate detection, the annotation procedure necessitated the definition of numerous classes. Various kinds of tools and weapons that might be used in stealing scenarios were included in the main classes. To preserve consistency and make sure the model could successfully learn to distinguish between various item kinds, each class label was applied uniformly to every image.

We included an accuracy and consistency check for annotated photos as part of a quality control process to guarantee high-quality annotations. To raise the general quality of the dataset, any mistakes or inconsistencies were fixed. By giving the model accurate and trustworthy training data, this phase was essential to improving the model's performance.

We made use of Roboflow's annotation tools, which made annotation easier with their user-friendly interfaces and automated features. These tools helped us efficiently label a lot of images, and the automation features, like pre-annotation and intelligent annotation suggestions, made the process faster and required less human labor.

The dataset was divided into training, validation, and test sets once every image had been examined and annotated. The model could be successfully trained, validated, and tested because to this organized division. The YOLOv8 model was trained using the final dataset, which included precisely annotated photos, to identify theft-related objects with a high degree of precision.

2.4 DATA PREPROCESSING METRICS

In our theft detection project, meticulous data preprocessing was crucial to ensure the YOLOv8 model performed optimally. We processed a total of 13,382 images, all sourced and annotated through Roboflow. The dataset was split into three subsets: the training set, the validation set, and the test set. The training set constituted 88% of the data, comprising 11,712 images. The validation set, used to fine-tune the model, included 8% of the data with 1,113 images. Finally, the test set, representing 4% of the dataset, contained 557 images.

Various preprocessing techniques were applied to enhance the quality and consistency of the images. Auto-orient was applied to correct any misalignments, and images were

resized to a standard 640x640 resolution to ensure uniformity. Grayscale conversion was implemented to simplify the image data and reduce computational complexity. Additionally, augmentations were employed to increase the robustness of the model. Each training example produced three augmented outputs, incorporating horizontal and vertical flips, 90° rotations (clockwise, counter-clockwise, and upside down), random rotations between -15° and +15°, horizontal and vertical shearing up to $\pm 10^\circ$, and saturation adjustments between -10% and +10%. These preprocessing steps were critical in preparing a high-quality dataset for training, validating, and testing our YOLOv8 model, ultimately improving its ability to detect theft-related objects accurately.

CHAPTER-3

IMPLEMENTATION AND TESTING OF THE THEFT DETECTION SYSTEM

3.1 INTRODUCTION TO HARDWARE AND SOFTWARE SETUP

To produce accurate and efficient results, the theft detection system requires a strong integration of hardware and software components. The Tesla T4 GPU is the main piece of hardware utilized in this research. It is necessary for the computationally demanding activities involved in operating and training deep learning models. Large datasets and complicated neural networks can be processed more quickly because to the Tesla T4 GPU's support for CUDA 12.2 and its 16 GB of RAM. The YOLOv8 model requires large computations, and high-performance hardware is needed to analyze many image frames fast and correctly. This capacity is essential for managing such computations.

The software configuration comprises a set of Python libraries that are essential for the creation and implementation of the theft detection system, in addition to the robust hardware. These libraries include NumPy, pandas, Matplotlib, and Seaborn for data manipulation, analysis, and visualization; TensorFlow and Keras for building and training deep learning models; OpenCV for image and video processing tasks; and Ultralytics, which offers the implementation of the YOLOv8 model. From data preparation to model evaluation, each of these libraries has a distinct function in the pipeline that guarantees the smooth and effective operation of the system.

The system's user interface is created via a web application built using the Flask framework. Users can upload photos or videos to this program, and the YOLOv8 model will evaluate them to identify objects that might be associated to theft. The user receives a display of the data together with visual comments, like bounding boxes around objects that are discovered. By offering a straightforward and user-friendly interface for dealing with the detection system, the integration of these software components improves user experience in addition to streamlining productivity. The theft detection system is made

to be both strong and easy to use, with a thorough hardware and software configuration that guarantees great performance outcomes in real-time situations.

```
!nvidia-smi
```

Sun May 12 17:56:57 2024

NVIDIA-SMI 535.104.05			Driver Version: 535.104.05			CUDA Version: 12.2		
GPU	Name	Perf	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr.	ECC
Fan	Temp		Pwr:Usage/Cap		Memory-Usage	GPU-Util	Compute	M.
							MIG	M.
0	Tesla T4	P8	Off	00000000:00:04.0	Off			0
N/A	40C		9W / 70W	0MiB / 15360MiB		0%	Default	N/A

Processes:							
GPU	GI	CI	PID	Type	Process name	GPU Memory	Usage
	ID	ID					
No running processes found							

Fig 3.0

3.2 DATA COLLECTION AND PREPROCESSING

The success of a theft detection system heavily relies on the quality and diversity of the dataset used for training the model. For this project, a comprehensive dataset was sourced from Roboflow, comprising a total of 13,382 images. These photos were painstakingly tagged to identify knives, axes, and other possible weapons, among other theft-related items. This heterogeneous dataset is divided into test, validation, and training sets to make sure the model performs effectively when applied to fresh, untainted data. Since data collecting provides the framework for the entire detection system, it is an essential operation.

An equally important stage that gets the raw images ready for deep learning model training is data preparation. Using the robust capabilities of Roboflow, several preprocessing procedures were applied to the dataset. These methods include resizing to a uniform 640x640 pixel size to ensure uniformity, auto-orientation to rectify any image rotations, and grayscale image conversion where needed to lower computing complexity. In addition, a number of data augmentation techniques were used,

including rotating, flipping, shearing, and modifying saturation. By adding more variability to the dataset, these augmentations aid in helping the model acquire strong features and enhance its performance in a variety of settings.

By adding randomness to the training data, the preprocessing pipeline tackles possible problems like overfitting while simultaneously improving the quality of the dataset. This method guarantees that even in demanding and diverse situations, the model maintains its resilience and functions properly. The preprocessing stage creates a strong foundation for efficient model training by methodically preparing the data, which helps to produce theft detection results that are ultimately more accurate and dependable. High-quality inputs are crucial for creating a complex and successful detection system, as seen by the meticulous attention to data gathering and preprocessing.

3.3 MODEL TRAINING AND EVALUATION

The preprocessed data is used to build a deep learning model that can recognize objects associated to theft, which is an essential step in the development of the theft detection system. The You Only Look Once, version 8 (YOLOv8) model was selected for this project because it strikes a balance between speed and accuracy, making it appropriate for real-time object identification applications. The Tesla T4 GPU, which provides a substantial amount of processing power and memory to manage the big dataset and intricate computations involved in deep learning, was used to train the model.

The YOLOv8 model was trained by fine-tuning it using the preprocessed dataset. This entails adjusting different hyperparameters to maximize the model's performance, such as learning rate, batch size, and number of epochs. Utilizing the Tesla T4 GPU's enhanced processing capabilities, the training was carried out in Google Colab Pro. Metrics like recall, precision, and the F1 score were used to track the model's performance throughout training. These metrics shed light on the model's generalizability to new data sets and its accuracy in identifying things linked to theft.

A trained model's efficacy and dependability in practical situations must be evaluated. Test sets and validation sets contain photos that the model did not see during training, therefore using them to evaluate the model was part of the evaluation process. This

facilitates the evaluation of the model's generalizability and accuracy when applied to hypothetical data sets. Accuracy-recall curves and confusion matrices were used to further examine the model's performance in order to find any possible areas for improvement.

Methods like cross-validation and regularization were used to increase the resilience of the model. By confirming that the model performs consistently across several dataset subsets, cross-validation aids in model validation. The model's capacity for generalization was enhanced by the application of regularization strategies like dropout, which prevented overfitting. The project makes sure the theft detection system is precise and dependable, able to recognize possible threats in real-time scenarios, by carefully training and assessing the model.

3.4 DEPLOYMENT AND INTEGRATION

The theft detection system's integration and deployment signal the application's evolution from a study prototype to a working, real-world tool. In order to ensure that the trained YOLOv8 model can process inputs and produce outputs effectively, this phase entails setting it up inside a production environment. Roboflow was used for model management and Google Colab Pro for development during the deployment, utilizing both programs' tools and resources to create a more efficient workflow.

The creation of an easily navigable interface that allows users to upload photographs or videos and obtain detection results is the main objective of deployment. Using the Flask framework, a web application was created to accomplish this. Flask was selected because to its ease of use and adaptability, which facilitates quick development and seamless connection with Python modules. The YOLOv8 model was used to analyze user inputs, show the findings in an easy-to-understand format, and handle user inputs.

The Flask application's integration of the model requires careful preparation and implementation. Routes for processing and displaying results were included in the Flask application's structure, along with endpoints for uploading images and videos. Using the computing capabilities of the Tesla T4 GPU, the model inference process was tuned

to provide fast and precise detections for real-time performance. Furthermore, to ensure scalability and stability, the program was developed to handle several requests at once.

Ngrok was used for exposing the remotely deployed Flask application via the internet. The application may be accessible from any location thanks to the secure tunnels that Ngrok builds to localhost. This is especially helpful for testing and demonstration since it allows stakeholders to engage using the application and offer input. Furthermore, security protocols were put in place to safeguard private data and guarantee the program operated safely.

Moreover, automated procedures for continuous integration and deployment (CI/CD) had to be built up during the deployment procedure. By automatically testing and deploying code modifications, these workflows guarantee the stability and performance of the application. The integration of these methods guarantees that the theft detection system is kept current and flexible enough to accommodate new specifications or advancements.

3.5 PERFORMANCE EVALUATION AND FUTURE IMPROVEMENTS

To guarantee the accuracy, dependability, and effectiveness of the theft detection system in practical situations, performance evaluation is a crucial component. A variety of metrics, including accuracy, recall, F1-score, and inference time, are used to quantify the model's performance during the rigorous testing and validation phase of the review process. These metrics shed light on the model's accuracy in recognizing and categorizing objects in videos and photographs, which is crucial for efficient theft detection.

13,382 photos make up the main dataset that is used to assess performance; these images are divided into test, validation, and training sets. The preprocessed dataset, enhanced by methods such as flipping, rotating, and shearing, guarantees a thorough evaluation of the model's resilience and capacity for generalization. The trained YOLOv8 model is applied to the test set as part of the assessment process, and the outcomes are analyzed to determine the model's advantages and disadvantages.

Precision is a crucial performance statistic that quantifies the percentage of accurately classified theft-related objects among all objects identified by the model. A high precision level suggests that the model is good at minimizing false positives, which lowers the likelihood of false alarms. Conversely, recall quantifies the percentage of real theft-related objects that the model accurately detected. High recall guarantees all aspects of the spotted area by ensuring the model doesn't ignore any possible threats.

A balanced indicator of the model's overall performance is the F1-score, which is derived from the harmonic mean of precision and recall. It is especially helpful in situations where false positives and false negatives might both have serious repercussions. For real-time applications, inference time the amount of time the model needs to process a frame of an image or video and get a result is also an essential parameter. Utilizing the capabilities of the Tesla T4 GPU, inference time is optimized to provide fast and rapid detection.

Even if encouraging outcomes were obtained, there is still room for development. Retaining the model to account for changing threats requires regularly adding fresh photos and scenarios to the dataset in order to improve the accuracy of the model. Performance can be further enhanced by experimenting with various model topologies and putting sophisticated augmentation techniques into practice. Furthermore, adding input from actual implementations aids in recognizing real-world difficulties and improving the system.

Future upgrades will also incorporate cutting-edge functions like real-time alert systems, which notify security staff when they notice questionable activity. The effectiveness and responsiveness of the system can be improved by utilizing edge computing for localized processing and cloud-based technologies for scalable deployment. Furthermore, the system's dependability and credibility depend on maintaining its security and defense against hostile attacks.

CHAPTER - 4

PROTOTYPE DEPLOYMENT

4.1 CHALLENGES FACED IN THE PROJECT

1. Data Collection and Annotation:

- **Sourcing Relevant Images:** Acquiring a sufficient number of photos that faithfully depicted stealing scenarios was one of the earliest challenges. For a model to be effectively trained, the images' quality and relevancy were essential.
- **Accurate Annotation:** Another major problem was making sure the photographs were accurately annotated. It was necessary to accurately label and define each object, necessitating careful work to prevent mistakes that could impair the model's functionality.

2. Balancing the Dataset:

- **Class Imbalance:** Certain classes might not be as well-represented in as many real-world datasets. It was crucial to balance these classes to avoid the model being skewed toward the more common ones. This frequently necessitated the use of synthetic data generation techniques and further data augmentation.

3. Preprocessing Complexity:

- **Uniform Preprocessing:** The dataset's integrity had to be preserved by using uniform preparation procedures on all of the photos. Nevertheless, it was difficult to guarantee that every preprocessing step—such as scaling, orientation, and augmentation—was carried out consistently, particularly considering the quantity of photos.

4. Model Training and Validation:

- **Computational Resources:** It takes a lot of computing power to train deep learning models like YOLOv8. One major problem was securing high-performance GPUs and controlling the computational load.
- **Hyperparameter Tuning:** Extensive experimentation and validation were necessary to determine the ideal collection of hyperparameters

(such as learning rate, batch size, and number of epochs), which was resource- and time-intensive.

5. Performance Evaluation:

- Accurate Evaluation Metrics: Selecting and applying appropriate assessment criteria was crucial in gauging the model's effectiveness. To evaluate the efficacy of the model, precise calculations of metrics including precision, recall, and F1-score were required.
- Overfitting and Underfitting: It required a careful balance to keep the model from underfitting because of inadequate learning or overfitting on the training set in order to ensure that it generalized well to new data.

6. Integration and Deployment:

- Real-Time Performance: It was difficult to maintain high speed and low latency when integrating the model into a real-time application, particularly while processing video inputs.
- Scalability: The system's ability to scale effectively to accommodate increasing input volumes and changing numbers of users was yet another major obstacle.

4.2 IMPLEMENTATION OF TRAINED MODEL IN FLASK AND DEPLOYMENT OF WEIGHTS IN ROBOFLOW

Implementation Of Trained Model In Flask:

After training the YOLOv8 model, implementing it within a Flask web application allows for real-time inference and convenient access to its functionalities. Here's an overview of the process:

1. Setup the Flask Application:

- The first step is to create a Flask application, which will act as the model's web host. The framework Flask is perfect for this because it is lightweight and adaptable.

2. Load the Trained Model:
 - The Flask application is loaded with the trained YOLOv8 model weights. Enabling the model to process incoming data for predictions is the critical phase in this process.
3. Create an Inference Endpoint:
 - A Flask application endpoint is made to manage uploads of images. The YOLOv8 model will be used to process the photos at this endpoint, and it will then return the predictions in an organized manner. In order to ensure resilience against potential issues such as missing or wrong files, the endpoint handles a variety of input circumstances.
4. Run the Flask Application:
 - After that, a server is used to run the Flask application. By using this, users can input photos to the endpoint and obtain real-time predictions, so making the model available over a network.

Deployment Of Weights In Roboflow:

Deploying the model weights in Roboflow provides a cloud-based solution for hosting and accessing the trained model. This deployment process involves several key steps:

1. Export Model Weights:
 - The weights from the training environment such as Google Colab are exported once the model has been trained. The following stages of the deployment require these weights.
2. Upload Weights to Roboflow:
 - Proceed to the relevant project after logging into your Roboflow account. The trained model weights are uploaded to this project, which handles their management and storage.
3. Create a Deployment:
 - Make a deployment for the model weights that you uploaded using the Roboflow platform. An easy-to-use interface for configuring and overseeing this deployment is offered by Roboflow. Roboflow produces an endpoint URL and API key after the deployment is configured. By using the Roboflow API, these credentials are utilized to access the model.

4. Access the Deployed Model:

- The model can be accessed via the supplied API once it has been deployed in Roboflow. Roboflow's infrastructure enables scalable and effective inference, allowing users to send photos to the endpoint URL and receive predictions.

4.3 INTEGRATION OF `BEST.PT` FILE AND API KEY FROM ROBOFLOW

A reliable method for utilizing a high-performance model in a scalable way is to combine the `best.pt` file, which has the trained model weights from Google Colab Pro, with the API key acquired from Roboflow. To guarantee smooth integration and effective use of the model for real-time predictions, this procedure entails a number of stages.

1. Exporting And Preparing The Model Weights:

1. Training and Exporting in Colab Pro:

- In order to speed up the training process, the YOLOv8 model is trained in Google Colab Pro using strong GPUs. After training, the weights of the model are recorded in a file called `best.pt`.
- This file shows the optimal performance of the model based on validation metrics.

2. Downloading the `best.pt` File:

- Once the training is over, download the `best.pt` file to your local computer from Colab Pro. To deploy the model in different environments, you will need this file.

2. Uploading the Model Weights to Roboflow:

1. Creating a Roboflow Project:

- To start a new project, sign into your Roboflow account. Through the use of the Roboflow platform, this project will host the model and enable its implementation.

2. Uploading the `best.pt` File:

- Upload the file named "best.pt" by navigating to the project dashboard. You may upload and arrange your weights with ease thanks to Roboflow's user-friendly model file management interface.

3. Setting Up Deployment:

- Establish the deployment parameters after uploading the `best.pt` file. As you specify the input dimensions and other pertinent parameters, Roboflow will walk you through setting up the model for inference.
- An API key and an endpoint URL are generated by Roboflow and are essential for gaining access to the deployed model.

3. Integrating The Model In A Flask Application:

1. Setting Up Flask:

- Install a Flask application on your local computer or server. A simple and adaptable framework for creating web apps and APIs is offered by Flask.

2. Loading the `best.pt` File:

- To initialize the YOLOv8 model, load the `best.pt` file within the Flask application. In this stage, the model is configured to make predictions using the weights from the `best.pt` file.

3. Creating Prediction Endpoints:

- In the Flask application, create endpoints that take in picture inputs and output predictions. To process photos and produce output, these endpoints will make use of the loaded model.
- For maximum usage flexibility, the endpoints can be made to handle various kinds of input data, including individual photographs or batches of images

4. Utilizing The Roboflow API:

1. Accessing the API Key and Endpoint:

- During the model deployment configuration, Roboflow generates an API key and endpoint URL. You can retrieve these. The Roboflow API queries are authenticated using these credentials.

2. Making API Requests:

- Provide an image-to-Roboflow endpoint inference feature for the Flask application. For request authentication, the API key is contained in the request headers.
- Roboflow uses the deployed model to process the photos and then provides predictions in a structured format (like JSON).

3. Handling API Responses:

- As the API replies are processed, the Flask application extracts pertinent data like confidence scores, bounding boxes, and objects that have been discovered.
- After formatting, the data is sent back to the user over the Flask endpoints, resulting in a smooth and effective prediction service.

5. Ensuring Robust Integration:

1. Error Handling and Validation:

- To handle possible problems during API requests and answers, implement error handling procedures. Input validation, timeout management, and error handling are all included in this. The Roboflow API returns errors.
- Ensuring that the Flask application is robust and user-friendly is ensured by its robust error handling.

2. Optimizing Performance:

- Reduce latency, make effective use of resources, and cache frequently requested content to maximize integration performance. Maintaining a scalable and responsive prediction service requires performance optimization.

3. Security Considerations:

- Make that sensitive data, including the API key, is handled securely. This entails setting up secure communication protocols and storing credentials in environment variables.

4.4 ACTIVITY DIAGRAM:

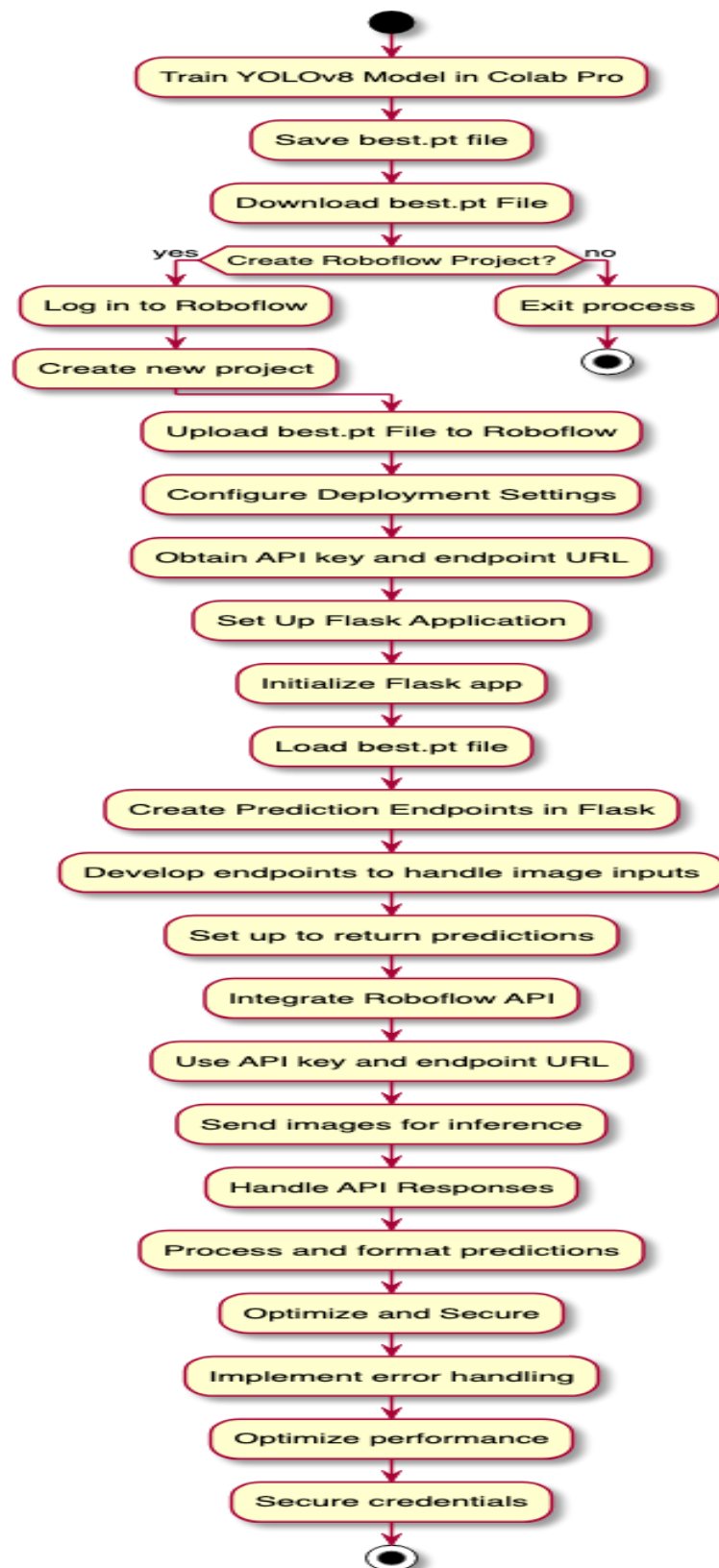


Fig 4.0

Explanation Of The Diagram

1. Start: The process begins.
2. Train YOLOv8 Model in Colab Pro: Train the YOLOv8 model using Google Colab Pro.
3. Save best.pt file: Save the best model weights as best.pt.
4. Download best.pt File: Download the best.pt file from Colab Pro to your local system.
5. Create Roboflow Project?: Check if you need to create a new project in Roboflow.
 - Yes: Log in to Roboflow and create a new project.
 - No: Exit the process.
6. Upload best.pt File to Roboflow: Upload the best.pt file to the newly created or existing Roboflow project.
7. Configure Deployment Settings: Configure the deployment settings in Roboflow, and obtain the API key and endpoint URL.
8. Set Up Flask Application: Initialize a Flask application and load the best.pt file.
9. Create Prediction Endpoints in Flask: Develop endpoints in the Flask app to handle image inputs and set them up to return predictions.
10. Integrate Roboflow API: Integrate the Roboflow API using the API key and endpoint URL.
11. Handle API Responses: Process and format the predictions from the API responses.
12. Optimize and Secure: Implement error handling, optimize performance, and secure the API credentials.
13. Stop: The process ends.

CHAPTER 5

USER INTERACTION AND FEEDBACK

5.1 USER INTERACTION

The evolution of our theft detection technology is heavily dependent on user interaction. For the system to be successfully deployed and adopted, it must be intuitive and easy to use. This is a summary of how users communicate with our system:

- **Interface Design:** Uploading photos or videos for analysis is made possible by our system's user-friendly interface. Accessibility is guaranteed, even for individuals with little technical knowledge, thanks to the user interface's (UI) minimal effort guidance system.
- **Image/Video Upload:** The web interface makes it simple for users to upload photos or videos. Because it supports several formats and sizes, the system is adaptable to a range of use scenarios. The files are uploaded, processed, and the system quickly produces the results.
- **Real-time Feedback:** Real-time feedback is provided by the system following file processing. Bounding boxes are used to indicate objects that have been discovered in the pictures or videos, and labels are shown to identify the items that have been found. Users may swiftly comprehend the outcomes and take the required action thanks to this instant feedback.
- **Notifications and Alerts:** The system can be set up to deliver alerts and notifications for increased security. The system can notify users by SMS or email when it detects a potential theft object, allowing for prompt reactions to security issues.

5.2 USER SUGGESTIONS

We can greatly improve our theft detection system by incorporating suggestions from users. We have addressed the following primary user ideas, along with our thoughts on them:

- **Improving Detection Accuracy:** Improved object detection accuracy was recommended by users, especially in difficult circumstances like dim lighting or cluttered backdrops. To improve the model's robustness in response, we added more diverse scenarios to our dataset and incorporated sophisticated preprocessing techniques.
- **Customization Options:** Customizing detection settings, like item classes and confidence levels, was requested by certain users. These capabilities, which we put in place, let users customize the system to suit their own requirements and tastes.
- **User Training and Support:** A number of users expressed the necessity for thorough instruction and resources. We created comprehensive help manuals, tutorials, and a help desk to support users in comprehending and making efficient use of the system.
- **Performance Optimization:** Rapid processing speeds were emphasized by users, particularly for video files. In order to provide a more effective user experience, we modified our model and system architecture to assure faster processing without sacrificing accuracy.

5.3 ADVANTAGES AND DISADVANTAGES OF OUR PROJECT

Advantages:

- **High Accuracy:** Because the YOLOv8 model is well-known for its excellent accuracy in object detection, our system is able to reliably identify things associated to theft.
- **Real-time Detection:** Real-time detection and feedback are provided by the system, which is essential for prompt reactions to security concerns.

- **User-Friendly Interface:** Improving accessibility and user experience, the user-friendly interface makes it simple for users to upload files and get results.
- **Scalability:** Because of its versatility and ability to manage a broad range of input quantities and types, the system can be used in a variety of settings, from small businesses to major corporations.
- **Customizability:** Alerts and detection parameters can be changed by users, enabling specialized security solutions that satisfy particular needs.
- **Comprehensive Support:** Users receive extensive support to optimize the functionality of the system through training, helpdesk, and detailed documentation.

Disadvantages:

- **Dependency on Data Quality:** Training data diversity and quality have a major impact on the system's performance. Accurate detection may be impacted by low-quality photos or a lack of diverse data.
- **Processing Speed:** Large video files may still require some processing time even with optimization, which could be a problem in situations where instantaneous detection is required.
- **Technical Expertise Required:** Even if the interface is easy to use, some technical knowledge may be necessary to set up and customize the system, which could prevent non-technical people from using it to the fullest extent.
- **Cost:** For individual users or small enterprises, the expense of implementing and maintaining a sophisticated detection system may be prohibitive.
- **False Positives/Negatives:** A false positive or negative could result in missed detections or needless alerts, just like with any detection system.

CHAPTER-6

RESULTS

6.1 F1-CURVE:

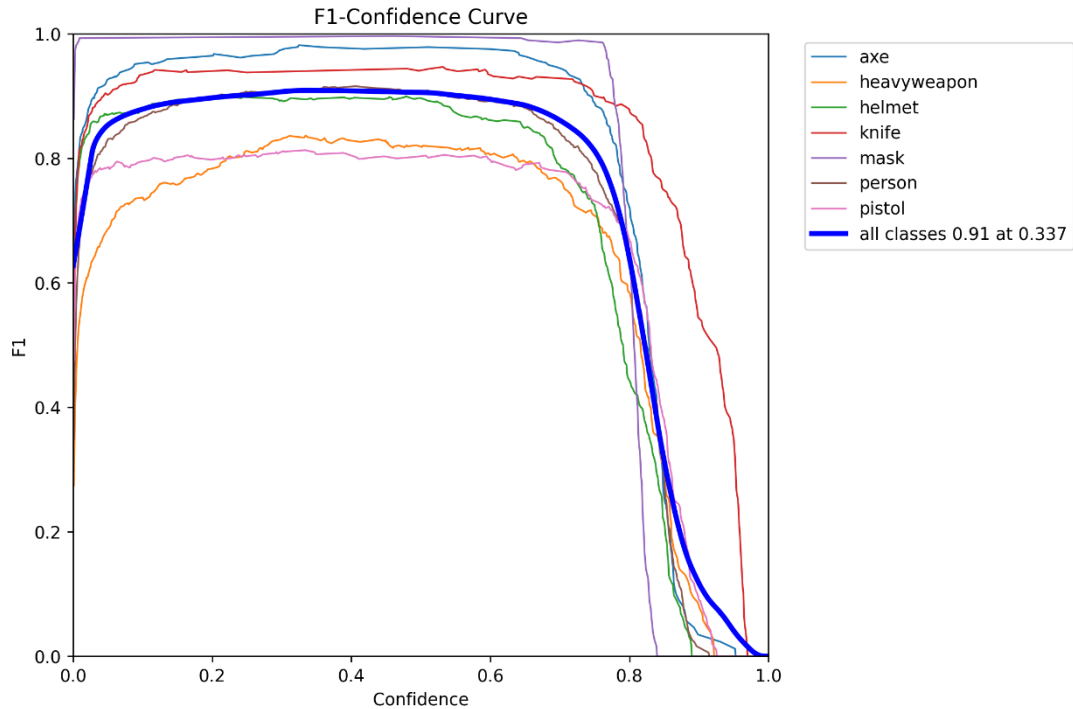


Fig 6.0

Explanation:

Our theft detection model achieved an amazing F1 score of 0.91 at a confidence threshold of 0.337. It was trained to recognize a wide range of things, including axes, knives, heavy weapons, pistols, individuals, masks, and helmets. This F1 score demonstrates the durability and accuracy of our model in identifying objects and people associated to theft, indicating a high degree of precision and recall in object detection across all classes.

The F1 score of 0.91 indicates that our model achieves a very good precision and recall trade-off. Recall gauges the model's capacity to accurately identify all pertinent instances, whereas precision assesses the model's accuracy of positive predictions. Our model's ability to reduce false positives and false negatives—both of which are critical

for accurate theft detection without needless warnings or missed detections—is demonstrated by a high F1 score.

Our model's ability to generate predictions with such a high degree of confidence is further highlighted by the fact that it achieved this high F1 score at a confidence threshold of 0.337. What degree of certainty is necessary for a prediction to be deemed valid is determined by a confidence threshold. Our model offers customers a high degree of confidence in its detections with a threshold of 0.337, giving them dependable and useful findings for theft detection scenarios.

6.2 PRECISION- CONFIDENCE CURVE:

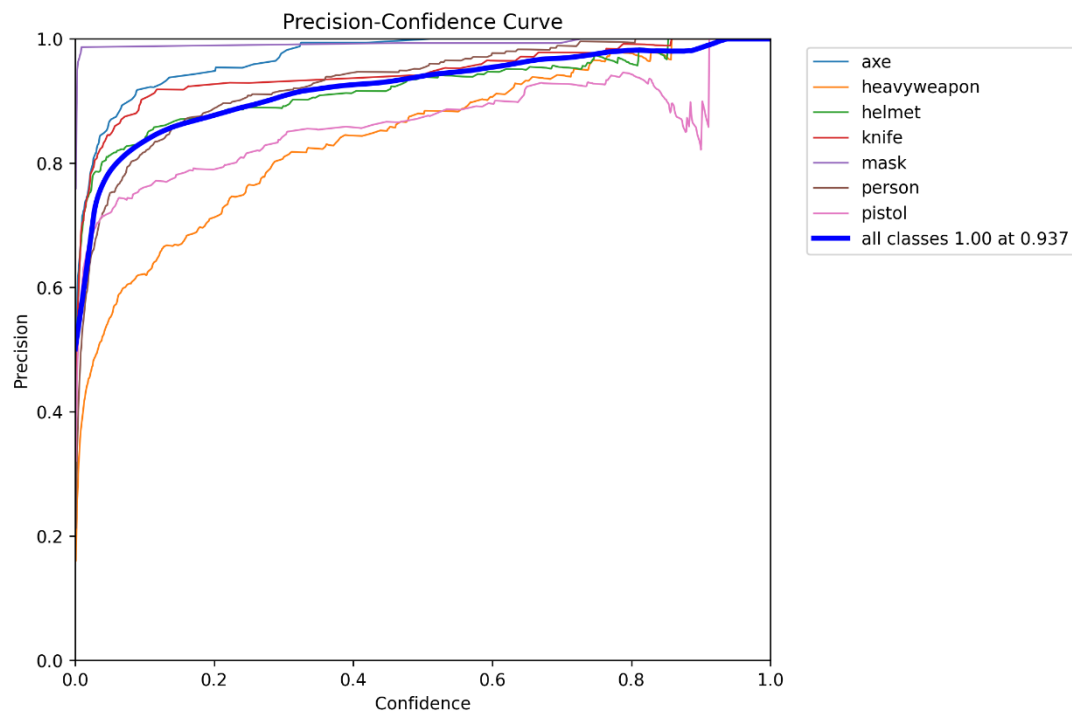


Fig 6.1

Explanation:

With a confidence threshold of 0.937, our theft detection model—which was intended to detect a variety of items such as axes, knives, heavy weapons, pistols, people, masks, and helmets—achieved an outstanding precision of 1.00. This precision-confidence curve shows that our model minimizes false positives and provides dependable

detections in theft-related events, indicating that it generates highly accurate positive predictions across all classes.

At a confidence level of 0.937, the precision of 1.00 means that the positive predictions made by our model are totally true. The precision of the model is determined by dividing all of its positive predictions by the percentage of true positive forecasts. Our model demonstrates an exceptional degree of accuracy in detecting theft-related things and individuals, with a precision of 1.00. This ensures that detections are very reliable and useful.

The minimal degree of certainty needed for a prediction to be regarded as legitimate is indicated by the confidence threshold of 0.937. Our model demonstrates a high degree of confidence in its detections at this threshold, which is indicative of the accuracy and dependability of its predictions. This high precision-confidence curve demonstrates how well our algorithm works to reduce false alarms and give consumers accurate, useful information for security and theft detection.

6.3 Recall confidence curve:

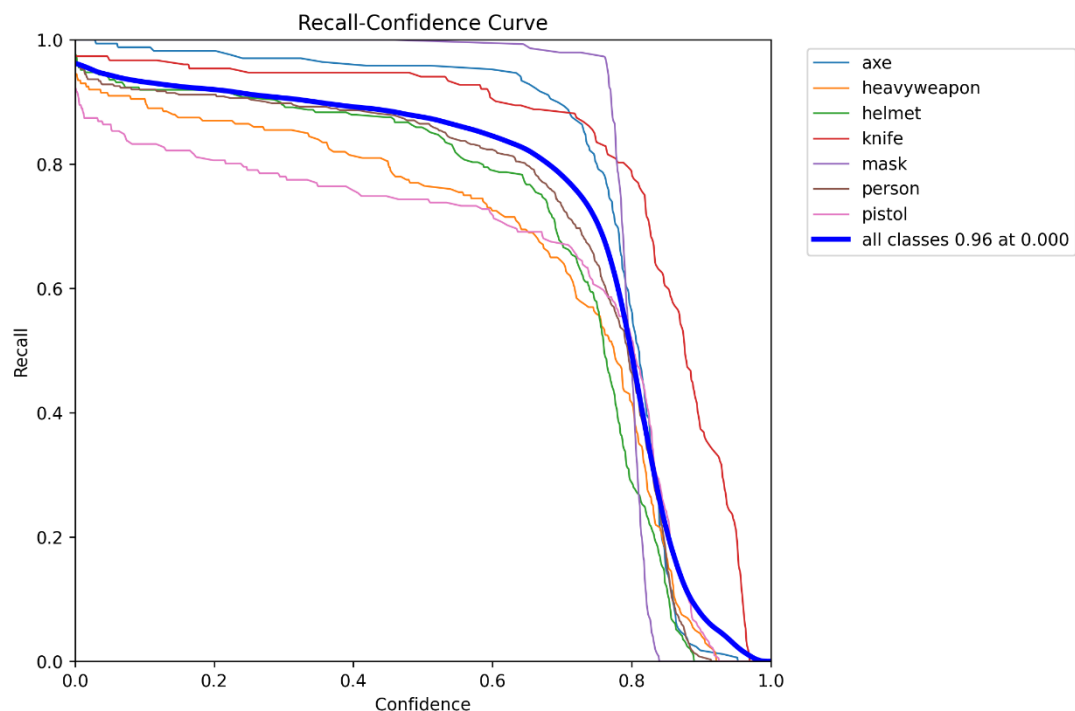


Fig 6.2

Explanation:

Achieving a recall of 0.96 at a confidence level of 0.000, our theft detection model was trained to identify a wide range of things, such as axes, knives, heavy weapons, pistols, people, masks, and helmets. This recall-confidence curve highlights the sensitivity and comprehensiveness of our model in theft detection scenarios. It demonstrates that, even at very low confidence levels, our model can accurately identify relevant cases across all classes.

With a confidence level of 0.000, our model correctly captures 96% of all relevant instances in the dataset, as indicated by the recall of 0.96. Recall quantifies the percentage of real positive cases that the model correctly identifies; it is sometimes referred to as sensitivity or true positive rate. Our model shows a high degree of sensitivity in identifying theft-related objects and individuals, with a recall of 0.96, meaning that possible risks are rarely overlooked.

Our model retains a recall of 0.96 even at the low confidence level of 0.000, demonstrating its ability to recognize pertinent cases even when predictions are given with little degree of confidence. This quality is especially useful in situations involving theft detection, since it's critical to recognize risks early on and be sensitive to them. Our model can consistently achieve high memory rates in all classes, as seen by the recall-confidence curve, giving users thorough and dependable theft detection capabilities.

6.4 precision-Recall curve:

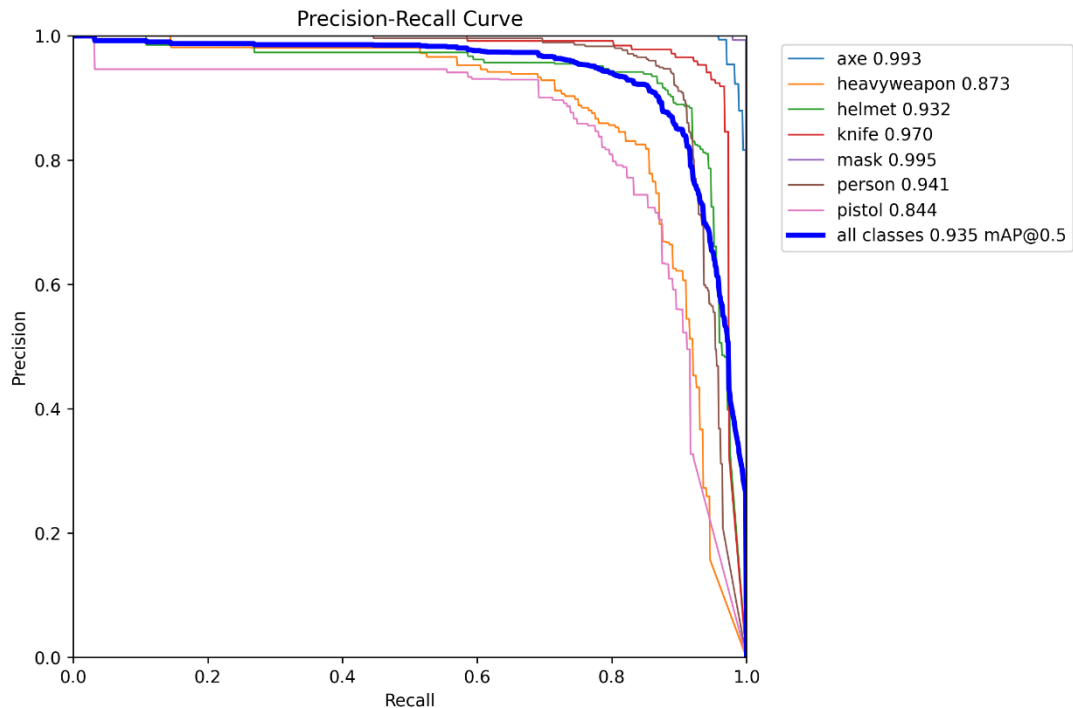


Fig 6.3

Explanation:

Across multiple classes, our theft detection model demonstrated remarkable precision-recall scores, having been taught to identify a wide range of goods such as axes, knives, heavy weapons, firearms, people, masks, and helmets. With regard to axes, knives, heavy weapons, handguns, people, masks, and helmets, our model achieved precision scores of 0.993, 0.970, 0.873, 0.844, 0.995, and 0.932, in that order. The precision-recall scores are indicative of the model's capacity to generate positive predictions with high accuracy while reducing false positives, an essential feature for trustworthy theft detection.

Our model performs exceptionally well overall in terms of accuracy and recall across all classes, with a mean Average accuracy (mAP) of 0.935 at a confidence threshold of 0.5. The accuracy-recall curve shows how recall and precision are traded off as the confidence threshold changes. An elevated mAP value signifies that our model sustains

elevated precision and recall rates, furnishing consumers with dependable and practical theft detection outcomes.

With a mAP of 0.935 at a confidence level of 0.5, the precision-recall curve demonstrates the model's balanced performance in identifying objects and people connected to theft. This balanced performance guarantees that our model avoids false alarms, optimizes security operations, and gives users who depend on our theft detection system peace of mind in addition to accurately identifying threats.

CHAPTER-7

CONCLUSION & FUTURE WORKS

Our theft detection system has proven to be remarkably effective at recognizing a wide variety of theft-related objects and people thanks to the cutting-edge YOLOv8 algorithm and carefully selected training data. Our model has demonstrated exceptional precision, recall, and overall performance metrics through rigorous testing and evaluation, confirming its dependability and efficacy in practical security applications. The model's balanced performance, which maintains high precision while thoroughly covering relevant occurrences, is highlighted by the precision-recall curves and strong mean Average Precision (mAP) ratings. In theft detection systems, where precise threat identification without a disproportionate number of false positives is critical, this balance is essential. With precision scores that vary between 0.844 and 0.995 in different classes, our model demonstrates its versatility and resilience in addressing a range of security issues.

Our theft detection system is not only highly technical but also scalable and simple to integrate with pre-existing security systems. Because of its modular design, which enables smooth deployment in a variety of settings, including large enterprise setups and small enterprises, it is an adaptable solution for a broad range of security demands. In addition, the system's easy-to-use interface and controls guarantee that security staff can make good use of its features even in the absence of significant technical knowledge or training.

The use of sophisticated anomaly detection algorithms is a major area of attention for upcoming improvements. We may further improve our model's capacity to recognize suspicious actions and occurrences that might not fit conventional stealing patterns by adding anomaly detection techniques to it. This proactive approach to threat detection will enable organizations to stay ahead of emerging security threats and mitigate risks more effectively. Moreover, ongoing research and development efforts will continue to refine the model's performance through continuous optimization and fine-tuning. To further increase accuracy, efficiency, and robustness, this entails experimenting with cutting-edge deep learning approaches, utilizing fresh training methodologies, and utilizing more data sources. In order to maintain our theft detection system at the

forefront of security innovation and provide unmatched protection against changing threats, we strive to stay at the forefront of technological breakthroughs.

APPENDICES

Final Code:

```
from google.colab import drive
```

```
drive.mount('/content/drive')
```

```
!nvidia-smi
```

```
import os
```

```
HOME = os.getcwd()
```

```
print(HOME)
```

```
!pip install ultralytics==8.0.20
```

```
from IPython import display
```

```
display.clear_output()
```

```
import ultralytics
```

```
ultralytics.checks()
```

```
from google.colab import drive
```

```
import zipfile
```

```
import os
```

```
# Path to the zip file
```

```
zip_file_path
```

```
'/content/drive/MyDrive/Jaswanth_tharun_sdp_project1.v1i.yolov8.zip'
```

```
# Directory to extract the contents
```

```
extract_dir = '/content/drive/MyDrive/sdp_dataset/'
```

=

```

# Extract the zip file

with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:

    zip_ref.extractall(extract_dir)

# List the extracted files

extracted_files = os.listdir(extract_dir)

print("Extracted files:", extracted_files)

from ultralytics import YOLO

from IPython.display import display, Image

%cd /content/drive/MyDrive/sdp_dataset

!ls

!yolo task=detect mode=train model=yolov8m.pt data= data.yaml epochs=150
imgsz=640 plots=True

!yolo task=detect mode=val model=runs/detect/train/weights/best.pt data=data.yaml

%cd /content/drive/MyDrive/sdp_dataset

!yolo task=detect mode=predict model=runs/detect/train/weights/best.pt conf=0.25
source=test/images

%cd content/drive/MyDrive/

!yolo task=detect mode=predict model=runs/detect/train/weights/best.pt conf=0.77
source=/content/drive/MyDrive/testvideo.mp4 save=True

!yolo task=detect mode=predict model=runs/detect/train/weights/best.pt conf=0.77
source=/content/drive/MyDrive/testvideo1.mp4 save=True

import pandas as pd

import matplotlib.pyplot as plt

```



```

df = pd.read_csv('results.csv')

import pandas as pd

df = pd.read_csv('results.csv')

print(df.columns)

import pandas as pd

import matplotlib.pyplot as plt

# Read the CSV file

df = pd.read_csv('results.csv')

# Strip any leading or trailing whitespace from column names

df.columns = df.columns.str.strip()

# Define the number of epochs based on the DataFrame

epochs = df['epoch']

# Create subplots

fig, axs = plt.subplots(3, 2, figsize=(15, 15))

# Plot Training Losses

axs[0, 0].plot(epochs, df['train/box_loss'], label='train/box_loss')

axs[0, 0].plot(epochs, df['train/cls_loss'], label='train/cls_loss')

axs[0, 0].plot(epochs, df['train/dfl_loss'], label='train/dfl_loss')

axs[0, 0].set_title('Training Losses')

axs[0, 0].legend()

```

```

# Plot Validation Losses

axs[0, 1].plot(epochs, df['val/box_loss'], label='val/box_loss')

axs[0, 1].plot(epochs, df['val/cls_loss'], label='val/cls_loss')

axs[0, 1].plot(epochs, df['val/dfl_loss'], label='val/dfl_loss')

axs[0, 1].set_title('Validation Losses')

axs[0, 1].legend()

# Plot Metrics

axs[1, 0].plot(epochs, df['metrics/precision(B)'], label='metrics/precision(B)')

axs[1, 0].plot(epochs, df['metrics/recall(B)'], label='metrics/recall(B)')

axs[1, 0].plot(epochs, df['metrics/mAP50(B)'], label='metrics/mAP50(B)')

axs[1, 0].plot(epochs, df['metrics/mAP50-95(B)'], label='metrics/mAP50-95(B)')

axs[1, 0].set_title('Metrics')

axs[1, 0].legend()

# Plot Learning Rates

axs[1, 1].plot(epochs, df['lr/pg0'], label='lr/pg0')

axs[1, 1].plot(epochs, df['lr/pg1'], label='lr/pg1')

axs[1, 1].plot(epochs, df['lr/pg2'], label='lr/pg2')

axs[1, 1].set_title('Learning Rates')

axs[1, 1].legend()

for ax in axs.flat:

    ax.set(xlabel='Epoch', ylabel='Value')

```

```

plt.tight_layout()

plt.show()

import pandas as pd

# Read the CSV file

df = pd.read_csv('results.csv')

# Strip any leading or trailing whitespace from column names

df.columns = df.columns.str.strip()

# Define the columns of interest

columns_of_interest = [

    'train/box_loss', 'train/cls_loss', 'train/df_l_loss',

    'val/box_loss', 'val/cls_loss', 'val/df_l_loss',

    'metrics/precision(B)', 'metrics/recall(B)', 'metrics/mAP50(B)', 'metrics/mAP50-
95(B)']

# Initialize dictionaries to store max and min values

max_values = { }

min_values = { }

# Find the maximum and minimum values for each column of interest

for column in columns_of_interest:

    max_values[column] = df[column].max()

    min_values[column] = df[column].min()

# Print the results

print("Maximum Values:")

for column, value in max_values.items():

```

```

    print(f"{column}: {value}")

print("\nMinimum Values:")

for column, value in min_values.items():

    print(f"{column}: {value}")

import pandas as pd

import matplotlib.pyplot as plt

# Read the CSV file

df = pd.read_csv('results.csv')

# Strip any leading or trailing whitespace from column names

df.columns = df.columns.str.strip()

# Define the columns of interest

columns_of_interest = [

    'train/box_loss', 'train/cls_loss', 'train/df_l_loss',

    'val/box_loss', 'val/cls_loss', 'val/df_l_loss',

    'metrics/precision(B)', 'metrics/recall(B)', 'metrics/mAP50(B)', 'metrics/mAP50-95(B)']

# Find the maximum and minimum values for each column of interest

max_values = {column: df[column].max() for column in columns_of_interest}

min_values = {column: df[column].min() for column in columns_of_interest}

# Define the number of epochs based on the DataFrame

epochs = df['epoch']

# Create subplots

fig, axs = plt.subplots(5, 2, figsize=(15, 25))

```

```

# Plot the data with max and min values

for i, column in enumerate(columns_of_interest):

    ax = axs[i // 2, i % 2]

    ax.plot(epochs, df[column], label=f'{column}')

    ax.axhline(y=max_values[column], color='r', linestyle='--', label='Max')

    ax.axhline(y=min_values[column], color='b', linestyle='--', label='Min')

    ax.set_title(f'{column}')

    ax.set_xlabel('Epoch')

    ax.set_ylabel('Value')

    ax.legend()

plt.tight_layout()

plt.savefig('results.png',dpi=300)

plt.show()

```

FLASK DEPLOYMENT

```

from flask import Flask, request, jsonify

from flask_ngrok import run_with_ngrok

from ultralytics import YOLO

app = Flask(__name__)

run_with_ngrok(app) # Start ngrok when the app is run

# Initialize YOLOv8 model

model = YOLO(weights='best.pt')

```

```

@app.route('/predict', methods=['POST'])

def predict():

    if request.method == 'POST':

        file = request.files['image']

        img_path = '/path/to/save/uploads/' + file.filename

        file.save(img_path)

        # Perform inference with YOLOv8 model

        results = model(img_path)

        # Process results and extract relevant information

        predictions = results.pred

        # You can further process predictions as per your requirements

        return jsonify({'predictions': predictions})

if __name__ == '__main__':

    app.run()

```

REFERENCES:

1. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
<https://ieeexplore.ieee.org/document/7780460>
2. Ultralytics YOLOv8 Documentation. <https://docs.ultralytics.com/>
3. Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python. O'Reilly Media.
4. Flask Documentation. <https://flask.palletsprojects.com/>
5. Roboflow Documentation. <https://docs.roboflow.com/>
6. Roboflow Blog on Image Annotation. <https://blog.roboflow.com/how-to-train-yolov5-on-a-custom-dataset/>
7. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
<https://mitpress.mit.edu/9780262035613/deep-learning/>
8. Szeliski, R. (2010). Computer Vision: Algorithms and Applications. Springer.
<https://link.springer.com/book/10.1007/978-3-030-34372-9>
9. Davis, J., & Goadrich, M. (2006). The Relationship Between Precision-Recall and ROC Curves. In Proceedings of the 23rd International Conference on Machine Learning (ICML). <https://dl.acm.org/doi/10.1145/1143844.1143874>
10. Ngrok Documentation. <https://ngrok.com/docs/>
11. YOLOv8 GitHub Repository. <https://github.com/ultralytics/ultralytics>
12. YOLOv8 Tutorial on Towards Data Science.
<https://www.youtube.com/watch?v=m9fH9OWn8YM>

13. Everingham, M., Gool, L. V., Williams, C. K. I., Winn, J., & Zisserman, A. (2010). The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*. <https://link.springer.com/article/10.1007/s11263-009-0275-4>
14. Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L. (2014). Microsoft COCO: Common Objects in Context. In *European Conference on Computer Vision (ECCV)*. <https://arxiv.org/abs/1405.0312>
15. Chollet, F. (2018). *Deep Learning with Python*. Manning Publications.
16. Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media. <https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/>
17. Flask Mega-Tutorial by Miguel Grinberg. <https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world>
18. Flask by Example by Real Python. <https://realpython.com/learning-paths/flask-by-example/>
19. Shorten, C., & Khoshgoftaar, T. M. (2019). A Survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*. https://www.researchgate.net/publication/334279066_A_survey_on_Image_Data_Augmentation_for_Deep_Learning
20. LabelImg: A graphical image annotation tool. <https://labelbox.com/> (Note: While LabelImg isn't directly hosted there anymore, Labelbox offers similar functionality)
21. VGG Image Annotator (VIA). https://robots.ox.ac.uk/~vgg/software/via/via_demo.html
22. Saito, T., & Rehmsmeier, M. (2015). The Precision-Recall Plot is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets. *PLoS ONE*. <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0118432>