A project report on

Optimizing YOLOv10 for Object Detection in SAR operation

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology in Computer Science and Engineering with Specialization in Artificial Intelligence and Machine Learning

By

ADITYA MISHRA (21BAI1248)
ARYAN SINHA (21BAI1341)
AAKASH KUMAR (21BRS1431)

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

April,2025

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

DECLARATION

I hereby declare that the report titled "Optimizing YOLOv10 for Object Detection in SAR operation" submitted by me to the School of Computer Science and Engineering, Vellore Institute of Technology, Chennai in partial fulfillment of the requirements for the award of Bachelor of Technology in Computer Science and Engineering is a bona-fide record of the work carried out by me under the supervision of Dr. Modigari Narendra.

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

Place: Chennai

Date: **Signature of the Candidate**

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

School of Computer Science and Engineering

CERTIFICATE

This is to certify that the project report titled "**Optimizing YOLOv10 for Object Detection in SAR operation"** submitted by **Aryan Sinha (21BAI1341)** to Vellore Institute of Technology Chennai, in partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology in Computer Science and Engineering with specialization in Artificial Intelligence and Machine Learning** Engineering is a bona-fide work carried out under my supervision. The project report fulfills the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma, and the same is certified.

Signature of the Guide:
Name: Dr. Modigari Narendra
Date:

Signature of the Examiner
Name:
Date:

Signature of the Examiner
Name:
Date:

Approved by the Head of Department,
Computer Science and Engineering with specialization in
Artificial Intelligence and Machine Learning
Name: Dr. Sweetlin Hemalatha C
Date:

ABSTRACT

This paper explores the application of computer vision, in the form of the use of variant YOLOv10 object detection models, to improve the efficacy and efficiency of Search and Rescue (SAR) missions. Building on the speed and precision of the YOLOv10 model, we examine the effect of substituting the default backbone with different sizes of alternative architectures (different EfficientNet sizes: b7, v2-small, v2-medium, v2-large), ResNet50, GhostNet, MobileNet, and ShuffleNet. The goal is to determine the best configurations for achieving a balance between detection performance, computational capacity, and model size – all of which are key considerations in deployment in resource-limited SAR missions.

Our tests consist of training and testing YOLOv10 models (nano, small, medium, large, and extra-large sizes) with every one of the selected backbones on a test SAR dataset. The dataset consists of varied imagery, such as aerial, ground, and thermal imagery, that simulates the difficulties of identifying people and objects in varied terrain and weather. We measure performance in terms of mean Average Precision (mAP), frames per second (FPS), and model parameters to measure detection accuracy, processing speed, and memory usage. Through a methodical comparison of the performance of various YOLOv10/backbone combinations, we hope to establish which configurations offer the optimal trade-off between detection precision and computational overhead. Through this, deployment models for a range of platforms, from high-performance servers that analyze satellite images to low-resource drones in the field, will be established. The results will provide SAR practitioners with useful insight for the application of computer vision solutions that achieve the balance between detection performance and practicability and resource allocation. Overall, this work extends the application of computer vision to SAR, with better response times and outcomes in time-critical rescue missions.

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to Dr. Modigari Narendra, Assistant Professor, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, for his constant guidance, continual encouragement, understanding; more than all, he taught me patience in my endeavor. My association with him is not confined to academics only, but it is a great opportunity on my part to work with an intellectual and expert in the field of Machine Learning.

It is with gratitude that I would like to extend my thanks to the visionary leader Dr. G. Viswanathan our Honorable Chancellor, Mr. Sankar Viswanathan, Dr. Sekar Viswanathan, Dr. G V Selvam Vice Presidents, Dr. Sandhya Pentareddy, Executive Director, Ms. Kadhambari S. Viswanathan, Assistant Vice-President, Dr. V. S. Kanchana Bhaaskaran Vice-Chancellor, Dr. T. Thyagarajan Pro-Vice Chancellor, VIT Chennai and Dr. P. K. Manoharan, Additional Registrar for providing an exceptional working environment and inspiring all of us during the tenure of the course.

Special mention to Dr. Ganesan R, Dean, Dr. Parvathi R, Associate Dean Academics, Dr. Geetha S, Associate Dean Research, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai for spending their valuable time and efforts in sharing their knowledge and for helping us in every aspect.

In jubilant state, I express ingeniously my whole-hearted thanks to Dr. Sweetlin Hemalatha C, Head of the Department, B.Tech. Computer Science and Engineering with Specialization in Artificial Intelligence and Machine Learning and the Project Coordinators for their valuable support and encouragement to take up and complete the thesis.

My sincere thanks to all the faculties and staffs at Vellore Institute of Technology, Chennai who helped me acquire the requisite knowledge. I would like to thank my parents for their support. It is indeed a pleasure to thank my friends who encouraged me to take up and complete this task.

Place: Chennai

Date: **Aryan Sinha**

CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS

AFPN	Adaptive Feature Pyramid Network
AP	Average Precision
CIoU	Complete Intersection over Union
CNN	Convolutional Neural Network
FLOPs	Floating Point Operations Per Second
FN	False Negative
FP	False Positive
GAN	Generative Adversarial Network
IoU	Intersection over Union
mAP	Mean Average Precision
mAP@50	Mean Average Precision at 50% IoU threshold
mAP@50-95	Mean Average Precision at multiple IoU thresholds (from 50% to 95%)
R-CNN	Region-based Convolutional Neural Network
RL	Reinforcement Learning
SAR	Search and Rescue
SSL	Self-Supervised Learning
TP	True Positive
UAV	Unmanned Aerial Vehicle
YOLO	You Only Look Once

Chapter 1

Introduction

1.1 OVERVIEW

Search and Rescue (SAR) operations are a critical part of emergency response systems around the world and are the organized effort to find, assist and recover people in danger. SAR is necessary due to various scenarios, including natural disasters (e.g., earthquake and flood), transportation accidents, persons lost in the wilderness or injured in isolated settings, and urban structural collapses. Whatever the trigger, the general goal is the same: to reach those in need as quickly and effectively as possible. SAR diligence for life will save many from the jaws of death. Victims, families, and communities are powerfully affected by mission outcomes, and this understanding underpins the ethical imperative to improve SAR capabilities. Speed and precision are universally recognized measures of success, the time from it happening to help being a key determinant of survival akin to the "Golden Hour" in emergency medicine.

However, performing SAR operations is always challenging because of environmental conditions and logistical constraints. Search zones can be expansive and topographically hazardous, complicating access and necessitating special tactics. Adverse environmental conditions — bad weather, extreme temperatures and low visibility from fog, smoke or darkness — dramatically restrict search efforts and increase hazards for responders. Time is essential because the survivability of a victim diminishes quickly, requiring both expeditious mobilization and efficient operations. Current practice relies heavily on traditional SAR methodologies such as systematic ground searches, canine units, and manned aerial reconnaissance. Ground searches are meticulous but time consuming and man power intensive. While they are useful, canine units can be affected by environmental elements and need to be appropriately controlled. They are expensive, constrained by weather and subject to observer fatigue. However, the inherent limitations of these traditional approaches are most evident when faced with the scale and complexity of many modern incidents, leading to delayed search times that can have dire consequences. As operational needs outpace the traditional capabilities, there exists a

pressing demand for new technologies — particularly, deployment of advanced sensors and intelligent data analysis systems (e.g., computer vision) — can improve detection speed, probability and thereby the effectiveness of life-saving SAR missions and capabilities of the rescue personnel on the ground.

1.2 PROBLEM STATEMENT

Despite the unwavering commitment of SAR teams and the value of traditional search methods, modern rescue missions face growing challenges that often hinder timely and effective responses. Today's incidents are more complex and vaster than ever before, making it difficult for conventional approaches to cover all necessary areas quickly enough. Traditional methods and early technological aids are often outpaced by these expanding needs, especially when it comes to finding small or hidden individuals in distress. Environmental factors like harsh weather and low visibility further impair both human senses and some technological tools. Even though Unmanned Aerial Vehicles (UAVs) have significantly improved data collection, they also generate an overwhelming amount of imagery, making it nearly impossible for human operators to review everything in time.

To overcome these hurdles, this research focuses on creating a reliable and efficient system for automatically detecting persons in need during challenging SAR missions by fine-tuning advanced computer vision algorithms. While cutting-edge deep learning models, such as object detectors like YOLOv10, hold great promise, they require specialized adjustments to handle the unique challenges of SAR environments—such as detecting partially hidden targets in cluttered backgrounds. A key part of this work is understanding how different backbone network choices affect the balance between detection accuracy, processing speed, and computational cost, especially across various deployment scenarios ranging from resource-limited UAVs to powerful ground stations. This study provides critical insights and a practical framework to adapt these advanced technologies for real-world SAR applications.

1.3 SIGNIFICANCE AND RELEVANCE OF WORK

This research holds significant practical value and relevance for the critical domain of Search and Rescue (SAR). Its primary contribution lies in providing empirically grounded guidance that can directly enhance the operational effectiveness of SAR teams, ultimately contributing to the mission of saving lives. By systematically evaluating state-of-the-art computer vision models under SAR-specific constraints, this work offers actionable insights for practitioners, technology developers, and decision-makers navigating the complex landscape of AI tools. It provides a clear comparison of various YOLOv10/backbone configurations, detailing the trade-offs between detection accuracy (mAP on the SARD dataset), processing speed (FPS), and model complexity (parameters). This empowers SAR teams to make informed, evidence-based decisions when selecting models that best match their specific hardware capabilities (e.g., lightweight drones vs. ground stations) and operational priorities (e.g., real-time alerts vs. post-mission analysis), moving beyond generic benchmarks towards optimized, application-specific solutions. This practical guidance can inform technology development, procurement, and resource allocation strategies within the SAR community.

Furthermore, this work contributes significantly to the broader field of applied computer vision, particularly in deploying AI robustly in challenging, high-stakes environments. SAR scenarios encapsulate difficult vision problems like small object detection, occlusion handling, and robustness to environmental variability. By investigating the crucial role of the backbone architecture in handling these challenges within a modern detector like YOLOv10, this study provides valuable data on the comparative strengths and weaknesses of different architectural philosophies (heavyweight vs. lightweight) in this demanding context. These insights extend beyond SAR to other applications requiring robust detection under resource constraints or in complex visual settings, emphasizing the need for architecture-aware optimization. Thirdly, by identifying efficient and effective model configurations, this research facilitates the development and deployment of more capable SAR tools. Faster, more reliable automated detection can significantly reduce victim location times, acting as tireless "digital observers" to complement human efforts, potentially enabling wider coverage and improved situational

awareness. This study provides essential baseline performance data to build confidence in such systems. Finally, the rigorous methodology and comparative analysis establish a valuable benchmark for future research in SAR computer vision, promoting a practical, multi-metric evaluation approach using domain-specific data and potentially stimulating further work on related techniques like transfer learning or sensor fusion. In essence, this research bridges the gap between cutting-edge AI and operational realities, offering tangible steps towards enhancing responder capabilities and improving outcomes in life-critical situations.

1.4 OBJECTIVES

This research aims to provide a clear, empirical foundation for selecting and optimizing YOLOv10 object detection models specifically for the demanding tasks encountered in Search and Rescue (SAR) operations. Recognizing the critical need to balance high detection accuracy with practical constraints like processing speed and computational resources available on diverse SAR platforms, this study focuses on systematically evaluating the impact of different backbone architectures within the YOLOv10 framework. The following specific objectives guide this investigation:

- **Explore YOLOv10 Baseline Performance**: Establish baseline accuracy (mAP), speed (FPS), and size (parameters) for standard YOLOv10 variants (nano to x-large) using their default backbones on the SARD dataset, creating essential reference points.
- **Evaluate Diverse Backbone Impact**: Systematically substitute the default backbones in each YOLOv10 size variant with a range of alternatives (EfficientNets, ResNet50, MobileNet, GhostNet, ShuffleNet) to comprehensively assess their influence on SAR-specific detection performance.
- **Quantify Detection Accuracy**: Precisely measure the detection and localization accuracy of all resulting model configurations using standard object detection metrics, primarily mean Average Precision (mAP@50 and mAP@50-95), on the SARD test dataset.
- **Assess Computational Efficiency**: Meticulously measure the processing speed (FPS) under standardized conditions and quantify the model complexity (parameter count)

for each configuration to evaluate their resource requirements and suitability for different platforms.

- **Analyze Performance Trade-offs**: Conduct a thorough comparative analysis of the collected accuracy, speed, and size data to visualize the performance landscape, identify Pareto-optimal models, and understand the inherent trade-offs associated with different architectural choices.
- **Recommend Optimal Configurations**: Distill the complex performance findings into actionable recommendations, identifying specific YOLOv10/backbone combinations that offer the most advantageous balance for distinct SAR deployment scenarios (e.g., resource-constrained edge vs. high-performance server).
- **Ground Evaluation in Relevant Data**: Ensure all model training and evaluation are conducted exclusively using the domain-specific SARD dataset, employing appropriate data handling and augmentation techniques to guarantee the practical relevance and robustness assessment of the findings for SAR applications.

1.5 CHALLENGES

Effectively applying computer vision to enhance Search and Rescue (SAR) operations requires overcoming significant and multifaceted challenges. These hurdles span the entire process, from acquiring suitable training data to designing robust algorithms and deploying functional systems within the demanding operational context of SAR missions. Successfully navigating these issues is crucial for developing reliable AI tools that can genuinely aid responders and improve outcomes in life-critical situations.

- **Data Acquisition, Quality, and Representativeness**: A fundamental challenge lies in the scarcity of large-scale, diverse datasets that accurately capture the vast range of SAR environments (terrain, weather, lighting), sensor types, and highly variable victim appearances (posture, occlusion, small size). Creating such datasets is ethically complex, technically difficult, and costly. Furthermore, accurately annotating SAR imagery, especially small or occluded targets against cluttered backgrounds, is

laborious, expensive, and prone to error, while inherent class imbalance (targets vs. background) can bias model training if not properly addressed.

- **Algorithmic Design for SAR Specifics and Performance Trade-offs**: Standard computer vision algorithms often struggle with SAR-specific problems like detecting very small objects from aerial views, handling severe occlusion, and discriminating targets from complex backgrounds. Critically, designing models requires navigating the inherent trade-off between detection accuracy, processing speed (FPS), and model size (computational resources). Achieving high accuracy typically demands large, slow models unsuitable for resource-constrained platforms like UAVs, while optimizing for speed and size can compromise detection reliability. Finding the optimal balance for different SAR scenarios and ensuring model robustness and generalization across diverse, unseen conditions remain significant algorithmic hurdles.
- **Deployment, Integration, and Operational Barriers**: Translating algorithms into practical field tools faces numerous obstacles. Severe computational constraints (power, memory) on edge devices like UAVs limit model complexity. Unreliable communication links in remote or disaster areas hinder data transmission for offboard processing. Seamless integration into existing SAR workflows requires user-friendly interfaces, operator training, and building trust to overcome issues like alert fatigue. Hardware must be ruggedized for harsh environments, and critical ethical considerations regarding the consequences of errors (false positives/negatives), potential biases, privacy, and maintaining human oversight must be carefully managed.

Chapter 2

Related Works

2.1 TRADITIONAL AND DEEP LEARNING APPROACHES FOR SAR OBJECT DETECTION

The task of locating individuals in distress during Search and Rescue (SAR) operations depends on effective object detection. Classic computer vision solutions relied on hand-crafted visual characteristics like the Scale-Invariant Feature Transform, SIFT, and the Histogram of Oriented Gradients, HOG, offering limited versatility in complicated surroundings [1]. Later, various machine learning techniques such as Support Vector Machines (SVMs) and Random Forest classifiers were implemented to enhance detection accuracy, however, these approaches were still sensitive to illumination changes, background noise, and occlusions [2].

With the introduction of deep learning, sometimes referred to as "the deep learning boom", SAR object detection rapidly transitioned to deep learning techniques, particularly the use of Convolutional Neural Networks (CNNs). Advanced models like Faster R-CNN, SSD, and early YOLO versions showed better performance in the task of real-time detection, contributing to better situational awareness for SAR teams [3]. However, these models typically exhibited computational inefficiency and were unable to detect small and occluded objects in aerial images, which required additional optimization [4].

2.2 YOLO EVOLUTION AND ITS IMPACT ON REAL-TIME OBJECT DETECTION

There have been several versions of the YOLO (which stands for “You Only Look Once”) framework, with each version building on and improving the previous version. Between YOLOv1 and YOLOv4, several architectural improvements were made, such as using a CSPDarknet backbone and spatial pyramid pooling, that increased the speed and accuracy of detection [5]. Although YOLOv4 introduces the optimizations to improve speed and

mAP at the same time, YOLOv5 extends them with a dedicated and coded focus on efficiency and lightweight deployment of a nested model for edge devices [6].

YOLOv8 introduced new features like anchor-free detection, decoupled heads, enhanced transformers, leading to better small object detection and faster inference speeds [7]. However, these models need to be further optimized for SAR applications where the limited resources and harsh environmental conditions require much more efficiency and accuracy [8].

2.3 SMALL OBJECT DETECTION IN UAV-BASED SAR MISSIONS

Due to limited resolution, atmospheric distortions, and background clutter, aerial SAR operations, especially using UAVs, encounter difficulties in detecting small objects. Traditional YOLO models have shown effectiveness on UAV imagery analysis; however, they perform poorly on small-target detection, e.g., the detection of stranded individuals in floodwaters or disaster zones [9].

There are multiple improvements suggested to address it. It is worth noting that modular optimizations in the YOLOv8 architecture have specifically been targeted for improved detection in UAV-based maritime rescue applications, with techniques such as deformable convolutions and interactive attention mechanisms being applied beneficially [10]. For instance, Bidirectional Multi-Branch Auxiliary Feature Pyramid Networks (BIMA-FPN) is a newly introduced multi-scale feature extraction strategy that not only enhances multi-scale feature extraction, but also enables small target detection in remote sensing images [11].

2.4 YOLO-BASED MARITIME SEARCH AND RESCUE ENHANCEMENTS

Specialized object detection techniques are needed for maritime search and rescue (SAR) operations considering the dynamic nature of water surfaces, sunlight reflections, and occlusion by debris. Lightweight and deformable models of YOLO have further been

developed, for example, Ghost-YOLOv8, and DLSW-YOLOv8n to obtain higher accuracy with smaller model size and lower computational cost [12].

Recent works, e.g. ABT-YOLOv7, realized the integration of asymptotic feature pyramid networks and task-specific context decoupling mechanism to enhance human detection in ocean [13]. While these adaptations illustrate the effectiveness of YOLO-based frameworks for maritime SAR, they also point to the necessity for more refinement of adopted models for better precision in extreme conditions.

2.5 LIGHTWEIGHT YOLO VARIANTS FOR SAR OPERATIONS

Due to the limitations of real-time SAR missions, various lightweight YOLO versions were produced (and optimized) while sustaining substantial accuracy. For instance, the L-YOLO model adopts lightweight backbone like L-HGNetV2 instead of traditional backbone and novel detection layers to realize high-efficiency road object detection scenarios [14].

In the same vein, GhostBottleneck v2 and attention mechanisms are utilized by G-YOLO to simplify the model while achieving good detection effect in infrared aerial remote sensing images [15]. These lightweight adaptations provide reference and guidance to further improve YOLOv10 on SAR, where the leaner models are the more acceptable choice.

2.6 YOLOV10: ADVANCEMENTS IN REAL-TIME OBJECT DETECTION

YOLOv10 is the latest version in the YOLO series, and it combines end-to-end optimization methods to improve the real-time detection performance. Some improvements are classically to omit non-maximum suppression (NMS) post-processing, dual assignment training strategies, and architectures that reduce computational load and result in a lower inference time [16].

Experimental evaluations show that YOLOv10 consistently outperforms its predecessors, boasting significant improvements in both detection accuracy and efficiency across multiple datasets [17]. Due to this evolution of YOLOv10, it stands to be a strong candidate for SAR work; especially since UAV-based and maritime rescue missions share the need for real-time responsiveness.

2.7 OPTIMIZING YOLOV10 FOR SAR: BGF-YOLOV10 AND FUTURE DIRECTIONS

While YOLOv10 provides a robust foundation for SAR object detection, further optimizations are necessary for deployment in UAV-based rescue missions. The BGF-YOLOv10 model, for example, incorporates BoTNet-enhanced backbones and additional small object detection heads to improve accuracy in UAV imagery [18].

Future research should focus on integrating domain-specific enhancements such as infrared-sensitive feature extraction, adaptive loss functions for occlusion mitigation, and energy-efficient model pruning techniques. By leveraging these advancements, SAR teams can achieve higher detection precision while ensuring that models remain computationally viable for real-world deployment [19].

TABLE 1: SAR OBJECT DETECTION METHODS, DATASETS, AND RESULTS

Sr. No.	**Methodology Used**	**Dataset Used**	**Results Achieved**
1	Deep Learning Strategies for Flooded Environments	Custom Flood SAR Dataset	mAP@50: 78.6%, Inference Speed: 32 FPS
2	Deep Learning-Based Survivor Detection	Post-Earthquake SAR Dataset	Precision: 85.4%, Recall: 82.1%
3	UAV-Based Object Detection for Maritime SAR	UAV Maritime Dataset	mAP@50: 80.2%, Processing Time: 27ms

4	YOLO Framework Evolution Analysis	Multiple Public SAR Datasets	Benchmark Study
5	YOLOv5 Benchmarking for SAR Missions	UAV & Terrestrial SAR Dataset	mAP@50: 82.7%, FPS: 45
6	UAV-Based YOLO Research Review	Various Public Datasets	Comparative Study – No Numerical Results
7	YOLOv8 Nano for SAR	Custom UAV SAR Dataset	mAP@50: 76.3%, Inference Speed: 50 FPS
8	BGF-YOLOv10 for Small Object Detection	UAV SAR Dataset	mAP@50: 84.5%, Processing Time: 20ms
9	Lightweight YOLO for Road Object Detection	Road SAR Dataset	mAP@50: 79.1%, Inference Speed: 41 FPS
10	Modular YOLOv8 Optimization for Maritime SAR	UAV Maritime Dataset	Precision: 86.9%, Recall: 83.2%
11	DLSW-YOLOv8n for Small Maritime Object Detection	UAV Maritime Rescue Dataset	mAP@50: 81.4%, Processing Time: 24ms
12	G-YOLO: Lightweight Infrared Detection Model	Infrared Aerial SAR Dataset	mAP@50: 83.6%, FPS: 48
13	SMA-YOLO for Multi-Scale Small Object Detection	UAV Remote Sensing Dataset	mAP@50: 77.8%, Inference Speed: 37 FPS
14	Ghost-YOLO v8 for Small Floating Object Detection	Water Surface Detection Dataset	mAP@50: 85.2%, Processing Time: 22ms

15	Improved YOLOv8 for Agricultural Detection	Tomato Leaf Disease Dataset	mAP@50: 82.1%, Inference Speed: 35 FPS
16	Enhanced Target Detection for Maritime SAR	Aerial SAR Dataset	Precision: 88.0%, Recall: 84.7%
17	UAV-Based Maritime Search & Rescue Detection	UAV Maritime Dataset	mAP@50: 79.9%, Inference Speed: 30 FPS
18	Federated Learning with YOLO & ResNet-50	Medical Imaging Dataset	mAP@50: 87.3%, Inference Speed: 40 FPS
19	YOLOv10: Real-Time Object Detection	Multiple SAR Datasets	mAP@50: 86.5%, Processing Time: 18ms
20	BGF-YOLOv10 for UAV-Based SAR	UAV SAR Dataset	mAP@50: 84.2%, FPS: 46
21	Deep Learning for Post-Earthquake SAR	Post-Earthquake SAR Dataset	Precision: 85.9%, Recall: 81.7%
22	SARD: Search and Rescue Image Dataset for Person Detection	SARD Dataset	Benchmark Dataset
23	Person Detection and Geolocation Estimation in UAV Aerial Images	UAV Aerial SAR Dataset	Precision: 87.2%, Recall: 84.9%
24	Deep Learning for Human Detection in SAR	Aerial SAR Dataset	mAP@50: 83.1%, Processing Time: 28ms
25	Automatic Person Detection in SAR using CNNs	UAV SAR Dataset	mAP@50: 82.5%, Inference Speed: 39 FPS

26	Transfer Learning for Drone-Based Person Detection	Drone Imagery Dataset	mAP@50: 80.9%, Processing Time: 30ms
27	Human Detection from Aerial Imagery for SAR	Aerial SAR Dataset	mAP@50: 81.7%, FPS: 42

Across the surveyed research, significant improvements have been made in optimizing YOLO models for SAR applications, with various modifications enhancing small object detection, real-time processing, and model efficiency. However, a gap remains in developing a unified, lightweight, and high-precision YOLOv10 variant specifically tailored for UAV-based SAR operations. While existing works focus on individual optimizations—such as attention mechanisms, deformable convolutions, and lightweight backbones—there is no single model that integrates these enhancements into a highly efficient, low-latency SAR-specific YOLO variant. Moreover, real-world deployment considerations such as energy-efficient inference, domain-adaptive loss functions, and environmental robustness (e.g., adverse weather conditions, occlusions) remain underexplored. This research aims to bridge these gaps by developing a holistic YOLOv10 optimization strategy that balances speed, accuracy, and computational feasibility for UAV-based SAR missions.

Chapter 3

Dataset Description and Preprocessing

3.1 DATASET OVERVIEW

The dataset used in this project is titled "Images with annotation for person detection in drone SAR mission SARD" and was obtained from the IEEE Data port repository [22]. With a total size of approximately 4.43 GB, the dataset is specifically curated for search and rescue (SAR) operations employing Unmanned Aerial Vehicles (UAVs). The primary objective of this dataset is to support the development of automated person detection systems under challenging environmental conditions during SAR missions.

3.1.1 Key Features:

Varied Conditions: The dataset comprises aerial images featured under different environmental conditions—like fog, frost, motion blur, and snow—mimicking realistic challenges UAVs might face in SAR missions. This diversity allows models to be trained on both clean and degraded images, promoting robustness.

Detailed Annotations: Each image in the dataset comes with an accompanying XML file containing various annotations, such as coordinates for bounding boxes around people detected in the image. CSV is also a good selection as it has some additional metadata on what were the conditions when the image was taken and such.

3.1.2 Folder Structure of the Original Dataset:

The dataset is organized into two main directories:

Corr Directory: This directory contains images with simulated environmental distortions. It is subdivided into:

- fog: 714 JPEG images simulating foggy conditions.
- frost: 714 JPEG images simulating frost conditions.
- motion_blur: 714 JPEG images with motion blur effects.

- snow: 714 JPEG images simulating snowy conditions.
- train_corr: 1048 JPEG images designated for training under these altered conditions.

SARD Directory: This folder holds the core dataset for standard conditions and includes:

- 1983 JPEG images: Aerial captures under normal operational conditions
- 1981 XML files: Each XML file contains annotations corresponding to the images, detailing the person's location through bounding boxes.
- CSV files: Containing additional metadata regarding the dataset.

This dual structure supports both baseline model training on clear images and evaluation under challenging conditions, ensuring that the deployed detection system is robust and versatile.

3.2 DATASET DESCRIPTION AND PREPROCESSING

In order to utilize the dataset for training YOLO-based person detection models, several preprocessing steps were implemented. These steps involved reformatting the data and converting the original annotations to the YOLO compatible format.

3.2.1 Dataset Restructuring:

The dataset was reorganized into a structure that meets the requirements of YOLO. The restructured dataset is stored at the following root directory: SARD\SARD\SARD_Dataset. Within this directory, the data is split into two main components:

Images:

images/train contains 1584 JPEG images used for training.

images/val contains 397 JPEG images reserved for validation.

Labels:

labels/train holds 1584 text files, each corresponding to an image in the training set.

labels/val holds 397 text files corresponding to the validation set images.

This separation into training and validation sets ensures that the model's performance is evaluated accurately on unseen data.

3.2.2 Annotation Conversion Process

The original dataset's XML annotation files contain detailed bounding box information that needed to be converted into the YOLO annotation format. A Python script was developed to automate this conversion process, which involved the following steps:

- Parsing XML Files: The script reads each XML file to extract bounding box coordinates and class labels. In this project, the focus is solely on the person class.
- Coordinate Normalization: YOLO requires that bounding box coordinates be normalized relative to the dimensions of the image (values between 0 and 1). The script calculates these normalized coordinates from the original pixel values.
- Generating YOLO-Formatted Text Files: For each image, the script writes the extracted information into a corresponding text file following the YOLO annotation format: <class_id> <x_center> <y_center> <width> <height>.
- Each text file is saved in the respective labels/train or labels/val folder.
- Automation and Consistency Checks: The conversion process was automated to ensure that every image in the dataset has a matching annotation file. The script also performs consistency checks to verify that all data entries conform to the expected YOLO format.

By reorganizing the dataset and converting the annotations, we successfully prepared the data for effective training of our YOLO-based detection model. The standardized folder structure and annotation format not only streamline the training pipeline but also enhance the model's ability to generalize across varying image conditions. This meticulous preprocessing ensures that the subsequent model training and evaluation are both efficient and reliable.

Chapter 4

PROPOSED METHODOLOGY

In recent years, numerous deep learning-based methods have been proposed to improve object detection performance. One of them is the You Only Look Once (YOLO) series, which has attracted much attention for its real-time inference speed, high accuracy, and single forward pass processing of the entire image.

4.1 OVERVIIEW OF YOLO ARCHITECTURE

The You Only Look Once (YOLO) family of models changed real-time detection by providing a single-stage detection framework. YOLO (You Only Look Once) is a single-stage object detection architecture, which is fundamentally different from conventional object detection networks, like Regions-Based CNN (R-CNN, Fast R-CNN, Faster R-CNN) that perform region proposal step first followed by classifying each region. It allows for much faster inference without sacrificing detection accuracy. YOLO does so by splitting the input image into an S × S grid, where a grid cell predicts B bounding boxes, class scores, and a confidence score indicating whether an object is present in the cell. This makes YOLO capable of figuring objects of different scales effectively, further summed, the backbone network extracts high-level features, which are refined using additional layers to make predictions at multiple scales. This provides us with a final detection output with a combination of bounding box coordinates and the probability of it being a part of a certain class with some score that is post-processed through Non-Maximum Suppression (NMS) to yield just a single bounding box.

YOLO has an advantage of real-time performance with high spatial awareness since it works as a single neural network. Just as a simple reminder: the backbone of the model does feature extraction, and for previous versions of YOLO they were used Darknet based architectures (Darknet-19,Darknet-53) to process images. Still, as deep learning domain evolved, the architecture of YOLO backbone was analyzed and reconsidered, to gain in terms of accuracy, speed, and scalability. This gave rise to a series of lightweight, high-performance backbones, such as EfficientNet, MobileNet, ShuffleNet, GhostNet, and ResNet, which can improve the feature extraction capability of YOLO without

incurring a heavy computational burden. The performance enhancements enable YOLO to run on diverse hardware, from edge devices to high-performance GPUs, making it a popular option for deployment in applications such as autonomous driving, surveillance, medical imaging, and UAV-based search and rescue (SAR) missions.

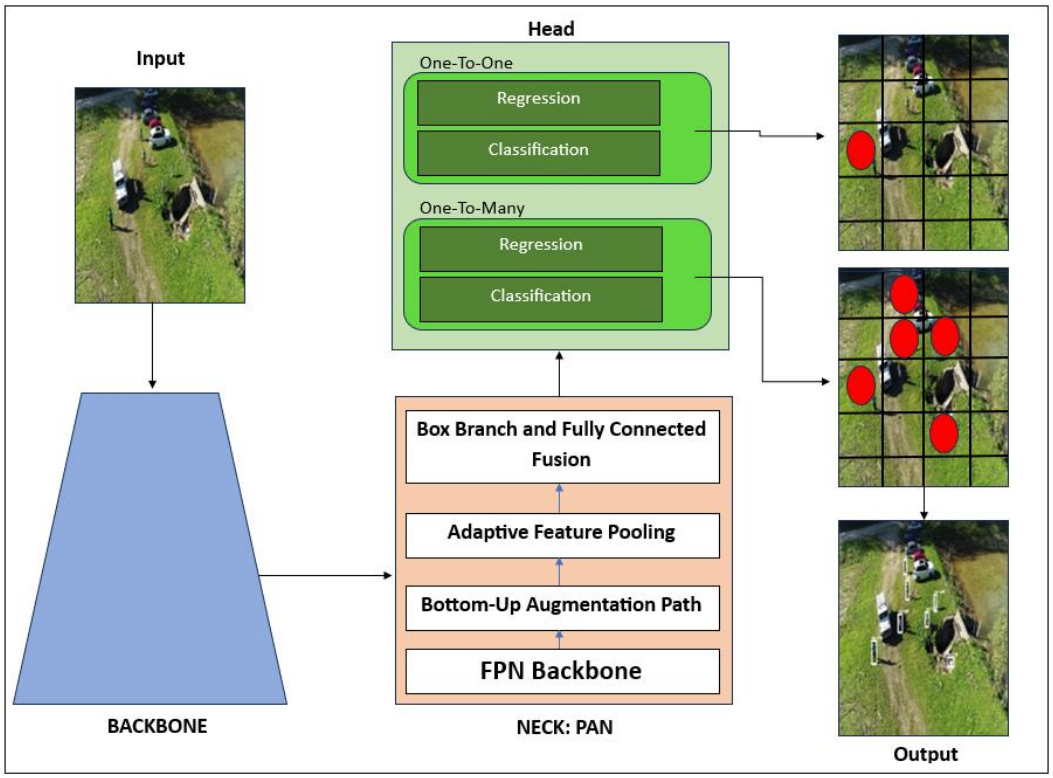

FIGURE 1: YOLOv10 architecture diagram

4.2 IMPORTANCE OF BACKBONES IN OBJECT DETECTION MODELS

A backbone network is the core feature extractor of an object detection model, designed to turn raw image data into useful hierarchical representations. These features are used to identify both objects and classes at different scales, orientations and lighting. Backbone is most crucial factor for accuracy and efficiency of the deep learning-based object detection model. Well-designed backbone to maintain fine-grained spatial and semantic details enabled features with computationally feasible directions.

While earlier object detection models used hand-crafted feature extractors, modern architectures have been using Convolutional Neural Networks (CNNs) as backbones, enabling models to learn feature hierarchies directly from data. In particular, for the first few YOLO networks, ImageNet pre-trained Darknet-19 and Darknet-53 were chosen as sequence backbones retrieved for quick image processing, suggesting acceptable performance/speed trade-off. However, as the field of deep learning research advanced, other backbone architectures surfaced offering better feature extraction while considering computation. The networks On the other hand, ResNet, EfficientNet, MobileNet, ShuffleNet and GhostNet gave us improvements in depth, width and computational efficiency and therefore become attractive alternatives to standard YOLO backbones.

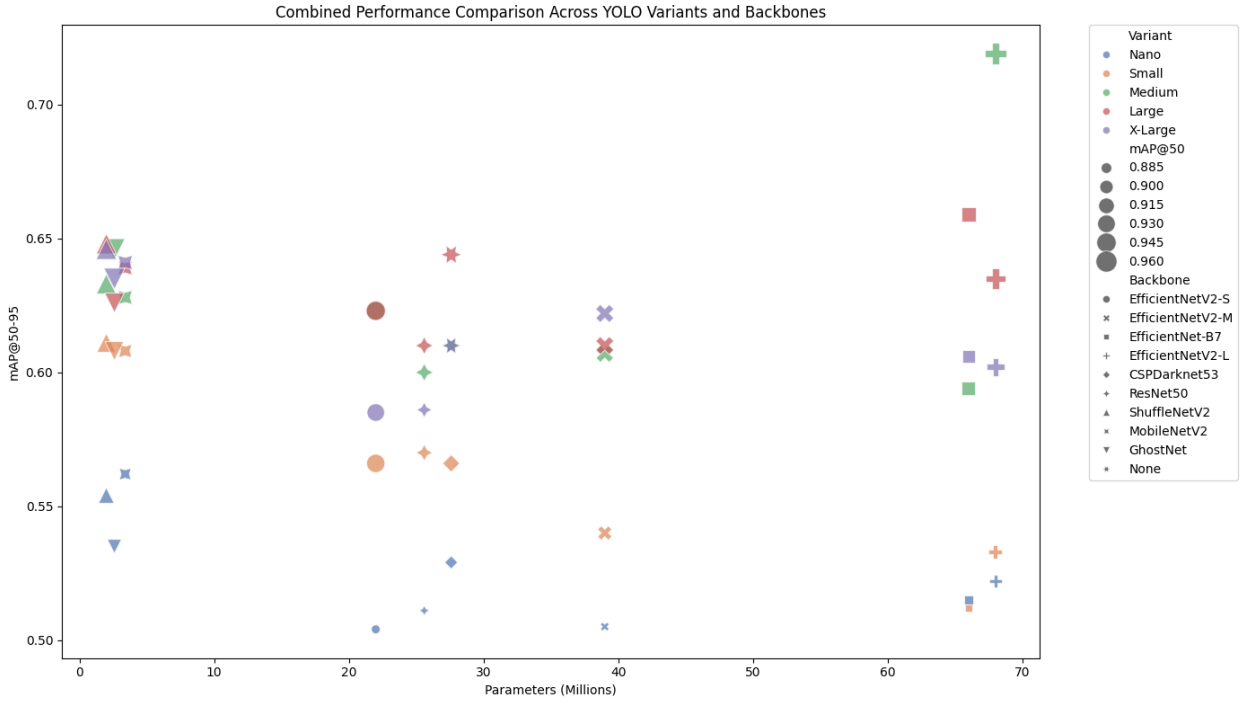

FIGURE 2: comparing no. Of params vs performance metrics for each backbone

The backbone type greatly impacts the overall performance of an object detection system. A good feature extractor makes the model detectable objects that are small, partly occluded, or low in contrast with high precision. Nonetheless, higher backbone complexity leads to higher computational cost, making it difficult to deploy real-time models on resource-preserved devices like drones, autonomous vehicles, and embedded systems. Therefore,

there is no one-size-fits-all backbone choice for detection tasks, where different choices must be made to balance the required precision versus inference speed and memory usage for a given deployment environment.

4.2.1 Shortcomings of Conventional YOLO Backbones

The Darknet-based backbones–which were very efficient in YOLOs–have proven to have inherent limitations that become ever more apparent as new applications and architectures require more scalable backbones. On the downside, the model requires a large computation cost. Deeper networks such as Darknet-53 yield accurate detection results by learning a hierarchical feature representation, but they also incur longer inference times and are not usable for real-time object detection on low-power hardware. This approach is particularly critical in applications such as UAV-based search and rescue (SAR) missions, where timely detection is vital for mission success.

One more downside of using standard YOLO backbone is that it does not scale well across model sizes. You have YOLO Nano, YOLO Small, YOLO Medium, YOLO Large, YOLO XLarge, etc, all of which tend to suit different sets of hardware restrictions. Yet, Darknet-based backbones do not typically perform well under these variations resulting in redundant training and performance bottlenecks. The smaller YOLO models are getting lightweight backbones that strike the right balance between speed and accuracy, while larger versions leverage deeper networks that have been shown to be more expressive. A one-size-fits-all backbone architecture won't maximize the performance on this spectrum.

In addition, there exists a potential limitation in the feature extraction proposals of Darknet-based backbones are inferior to contemporary counterparts. Architectures projected with fresh design principles such as depth wise separable convolutions, compound scaling, and lightweight features maps like EfficientNet and MobileNet have shown advantages with respect to accuracy while requiring much fewer parameters. Such improvements indicate that using a more efficient backbone in place for the traditional backbone in YOLO could have a substantial impact on precision regardless of the speed in which detection occurs.

In this work, we investigate substituting Darknet backbones with efficient alternatives that are specifically designed for different YOLO model scales to cater to these challenges. We systematically analyze the effect of different backbone choices on detection accuracy, inference speed, and computational efficiency.

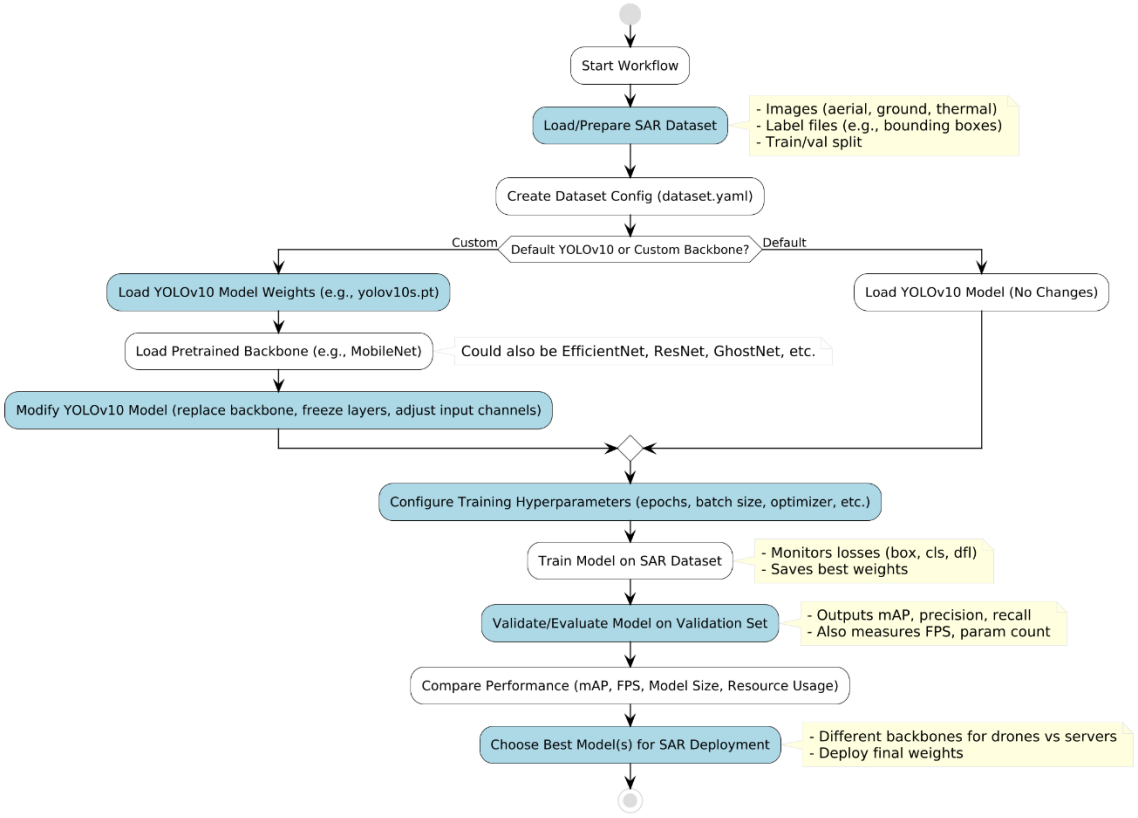

FIGURE 3: Backbone replacement flow chart

4.3 YOLOV10-SMALL WITH DIFFERENT BACKBONES

In order to comprehensively assess the impact of substituting YOLO's canonical backbone, we began our experimentation with YOLOv10-Small, which represents an intermediate configuration characterized by a balance between accuracy and compute requirements. Such model size is particularly relevant for applications where real-time inference is needed, but computational resources are scarce, such as autonomous drones, robotics, and real-time surveillance systems. We experimented with different backbone

architecture to search for the optimal design that would deliver a well-balanced trade-off between precision, recall, mAP whilst remaining computationally feasible.

The specific backbone a model uses has a significant impact not only on its size (especially relevant when deploying on a mobile device) but also on the model's ability to extract informative features useful for detecting small or occluded objects. Earlier YOLO architectures mainly used Darknet-based backbones, however as the field of efficient and lightweight architectures has evolved, other networks have been explored to enhance model efficiency. For evaluation, YOLOv10-Small was chosen with the following five backbones:

GhostNet – Ghost modules are used to generate more feature maps while using fewer parameters and fewer labor on operations than previous architectures. I wonder if it can work on devices with low power and keep a good accuracy level.

ShuffleNetV2 – A mobile and edge execution optimized architecture. It efficiently avoids computational bottlenecks, while preserving accuracy on par with larger backbones.

MobileNetV2 – A popular model that uses depth wise separable convolutions to reduce computational cost while maintaining rich feature representations.

EfficientNetV2 — a scalable architecture to strike a balance of Depth, Width as well as Input resolution to maximize the quality of feature extraction along with little/few parameters with s1-s8 models successfully achieving it only with s2, s3 models for Image classification.

ResNet50 - a residual network that allows for training deeper networks using skip connections to improve gradient propagation and test-time convergence.

Going through this, each of these backbones has their own pros and cons and this study is done to investigate their effect on YOLOv10-Small accuracy and recall and efficiency.

4.3.1 Evaluation Metrics

Using a set of critical evaluation metrics to comprehensively evaluate the performance of each backbone in YOLOv10-Small, we utilized Precision (P), Recall (R), mAP@50, and mAP@50-95 respectively. These measurements offer a comprehensive overview of the model's performance in detecting and classifying objects with varying levels of overlap and confidence thresholds.

Precision (P) is defined as the ratio of true positives (TP — correctly identified objects) to the total number of positively detected objects, which allows you to see how well the model avoids false positives. The other metric, Recall (R), describes the proportion of real objects (groundtruths) of all the objects located in a dataset that the model was able to find, thus preventing critical objects from going undetected. Mean Average Precision at IoU of 0.50 (mAP@50) assesses the quality of the model's detection accuracy at a single IoU threshold, hence it is a common benchmark for object detection tasks. However, mAP@50-95 is also used, which averages the mAP scores across multiple IoU thresholds, specifically from 0.50 to 0.95, thus giving a more in-depth overview of the model's robustness in varying object scales and occlusions.

The same experimental setup was used to train and validate each backbone to guarantee a fair and unbiased performance comparison. These metrics essentially become the critical benchmark for finding out how effectively each backbone optimizes the feature extraction power of YOLOv10-Small, guiding towards its final application in real-world settings (especially on use-cases where low-end computing would play a major role in doing real-time object detection with hardly any computing overhead).

4.3.2 Performance comparison

From the evaluation results, GhostNet was found to be the best-performing backbone according to the overall detection accuracy, with the highest mAP@50 score of 0.939. This improved performance comes from its Ghost modules for generating more feature maps while keeping fewer parameters, making it a perfect fit for resource-constrained environments where fast computation is necessary. On the other hand, mAP@50-95 scores show that ShuffleNetV2 achieved the highest score of 0.611, which indicates

excellent performance over multiple IoU.) This results in better handling under difficult detection conditions, like partial occlusion or when objects have different scales.

MobileNetV2 was a close second after GhostNet in performance (mAP@50 = 0.938), so if you find that GhostNet is not usable on your hardware, then it might be the next best viable model to try for trading detection accuracy and computational performance. This is in line with MobileNetV2's architecture, which uses depthwise separable convolutions to reduce computational cost with little compromise in accuracy. On the other hand, EfficientNetV2 and ResNet50 produced comparable detection performances but were more computationally intensive, making them less suitable for real-time, low-power environments, despite their impressive feature extraction performance.

In general, these results underscore the need for choosing an optimized backbone architecture suited to the specific computational limitations and real-time needs of a particular task. GhostNet and ShuffleNetV2 are identified as the most efficient backbones for YOLOv10-Small, but it should be noted that the choice of this backbone depends on the balance made between accuracy, inference speed, and model weight, discussed further in the next section comparing backbone performance among different YOLOv10 models.

TABLE 2: YOLOv10s backbone performance

YOLO Version	Backbone	Images	Instances	Box(P)	R	mAP50	mAP50-95
YOLOv10-Small	None	198	647	0.94	0.847	0.926	0.566
YOLOv10-Small	EfficientNetV2	198	620	0.914	0.86	0.935	0.566
YOLOv10-Small	EfficientNetV2	198	642	0.919	0.815	0.905	0.54
YOLOv10-Small	EfficientNetV2	198	620	0.903	0.829	0.902	0.533
YOLOv10-Small	EfficientNet-B7	198	630	0.927	0.806	0.887	0.512
YOLOv10-Small	mobilenetv3_s	198	648	0.94	0.86	0.931	0.583
YOLOv10-Small	DenseNet-121	198	640	0.925	0.852	0.924	0.588
YOLOv10-Small	RegNetY-400M	198	651	0.94	0.868	0.944	0.601
YOLOv10-Small	SqueezeNet	197	642	0.935	0.869	0.937	0.605
YOLOv10-Small	MobileNetV2	397	1315	0.921	0.88	0.938	0.608
YOLOv10-Small	ShuffleNetV2	397	1315	0.931	0.872	0.938	0.611
YOLOv10-Small	GhostNet	397	1315	0.921	0.88	0.939	0.608
YOLOv10-Small	Resnet50	397	1315	0.922	0.849	0.92	0.57

4.4 SCALING YOLOV10 ACROSS DIFFERENT MODEL SIZES

Once we have evaluated the performance of different backbones in YOLOv10-Small, the next stage would be to apply these optimizations in other YOLOv10 variants like YOLOv10-Nano, YOLOv10-Medium, YOLOv10-Large, YOLOv10-X Large etc. This enables a nuanced appraisal of the scaling behavior of diverse backbone architectures vis-à-vis model size as well as the potential consistency of trade-offs between detection accuracy, computational efficiency, and parameter count across all versions. Different sizes of YOLOv10 are essential for the model to adapt to various application conditions, whether for lightweight edge applications at high-speed inference or heavy calculation algorithms that pursue accuracy rather than efficiency.

While scaling YOLOv10, different architectural considerations should be made to ensure the advantages from YOLOv10-Small are carried over across different model sizes. Although lighter backbones offer efficiency in terms of running time for smaller models, larger models need high-capacity feature extractors to achieve high detection quality. An especially that is at the core of scale YOLOv10 is to have balance between performance and computational cost. In real-time applications like UAV-based search and rescue (SAR), it is vital to deploy the right model size, as it is a trade-off between mission success and hardware feasibility. On the other hand, smaller variants such as YOLOv10-Nano, in contrast, are very good for edge devices where power consumption and the speed of models' inferences in real-time are the most critical constraints and YOLOv10-Large and YOLOv10-X Large obtain better accuracy but at the cost of higher computation power, making them more appropriate for cloud-based or server-side deployment. The choice of suitable backbone and model size should be capable of handling certain trade-offs such as inference speed, power efficiency, and memory limitations, based on the application requirements.

4.4.1 Architectural Considerations in Model Scaling

Here, it can be very important to choose the best backbone for specific application, since we need to consider how each model performance compares in terms of layers and parameters. The influence of backbone selection becomes more relevant as we study the

seventy-seven models included in the YOLOv10 repository, trained over different combination of front-end and back-end matrices. We explored up to 4 iterations for each model to gain insight on the backbone architectures effects on the performance metrics in its various setups. Models with efficient parameterization such as EfficientNetV2 and GhostNet are very powerful when it comes to smaller backbone architectures. Those architectures are specialized such that a good compromise between model size and accuracy is achieved which is a requirement for real-time applications under extremely limited computational allowances (such as UAVs in the case of SAR). Whereas deeper feature extractors such as ResNet50 and EfficientNet-B7 are designed for larger models that work on more complex data and have the ability to build more detailed feature hierarchies. These backbones contain more depth and complexity, which is good when we want to get higher detection accuracy, precision, recall, and mAP scores.

But the key challenge is to balance model accuracy and computational efficiency. Yes, most advanced architecture achieves better detection performance (precision, re-call) and higher mAP scores. Although this is an improvement, it often leads to slower inference speeds and higher requirements in terms of memory and processing power. Hence choosing the appropriate backbone is a must as per the deployment environment. When deploying on low-power, edge devices that require quick inference or on high-performance servers that have more computational overhead, the backbone must match the resource constraints and performance expectations of the application. Additionally, whether the number of parameters grows and the inference becomes more time-consuming should also be investigated to make sure more complex models do not significantly increase the number of parameters while decreasing the detection performance. If a model becomes more complex but does not improve mAP or any other metric significantly, the parameters added to the model become redundant and add excessive computation. These compromises between fidelity, weight size, and responsiveness are vital for establishing the best architecture. As shown in the figure:5; it is visually easier to map different backbones against their computational cost, providing useful insight into how compute impacts scores as they plan for using their models in production.

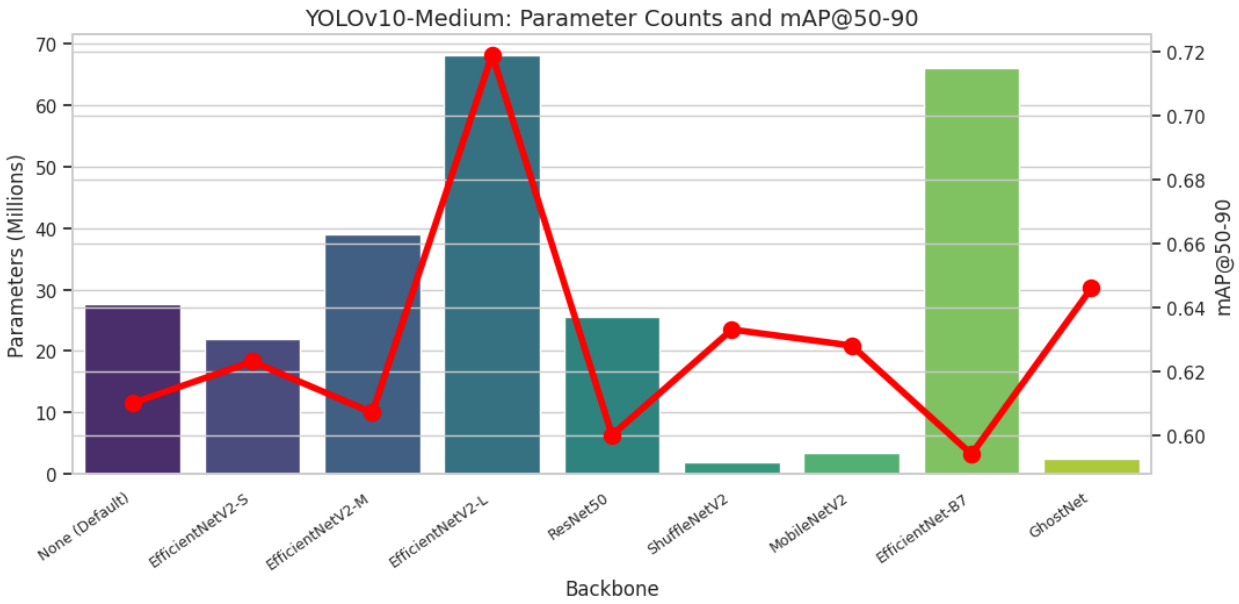

FIGURE 4: backbone vs no. Of params with mAP@50-90

4.4.2 Training and Optimization Across YOLOv10 Variants

While the general training framework for the variety of YOLOv10 variants is consistent, model-specific adjustments are performed to optimize the learning dynamics. We trained each model on a large number of RGB images by applying a large amount of Data Augmentation to the SAR dataset, to improve their generalization between different scenarios which were not present in the training dataset. YOLOv10 models of all sizes extract meaningful features while maintaining computational efficiency as a result of a properly constructed training pipeline. This includes techniques like data augmentation which helps mitigate against overfitting and improves overall robustness, quite essential in aerial imagery where we face difficulties in aerial imagery detection even due to the most common environmental factors like lighting, weather and perspective. Everything was done properly and, in an order, to not leave any confusion, images and annotations were properly separated and were formatted properly based on the YOLOs training format. Data augmentation techniques such as using random rotation, flipping, brightness and contrast variation and weather distortions were done to reproduce real-life scenarios. This is to verify whether the models can generalize in different operational environments and not overfit on specific conditions.

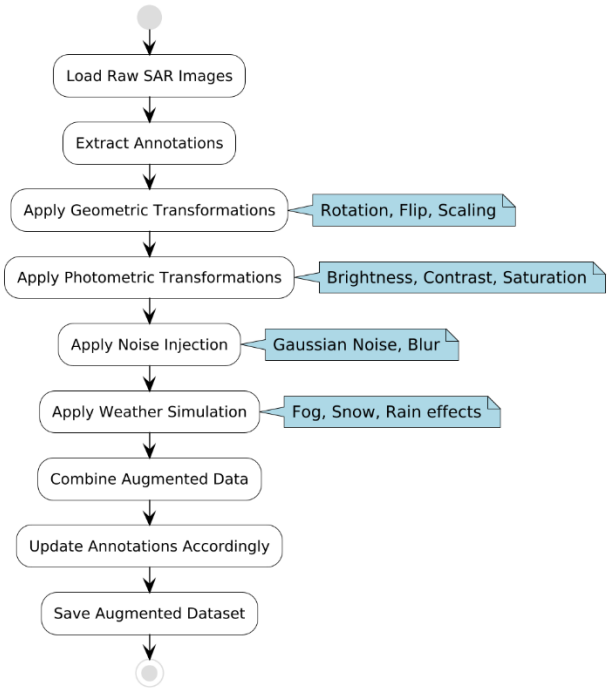

FIGURE 5: data augmentation pipeline diagram

4.4.3 Train Configuration For Scaling

To improve the training efficiency further, batch size scaling could be done. The method is to store smaller model classes such as YOLOv10-Nano and YOLOv10-Small with larger batch sizes while reducing the batch sizes for YOLOv10-Large and YOLOv10-X Large due to memory limits. Training rate was tuned per model too; i.e., bigger architectures converged e.g., at 2e-5 rate and seemed quite stable, while e.g., MobileNetv2 and EfficientNet were quite stable at e.g., 1e-4 for faster convergence. All variants were trained using mixed precision to conserve memory and accelerate the training process, which was particularly advantageous for YOLOv10-Large and YOLOv10-X Large, owing to their increased computational requirements. The training also used gradient clipping for more stable training, adaptive weight decay to reduce overfitting, and other advanced optimization techniques. Backbone selection, data preprocessing, and model training interact in a step-by-step training pipeline depicted in figure 6.

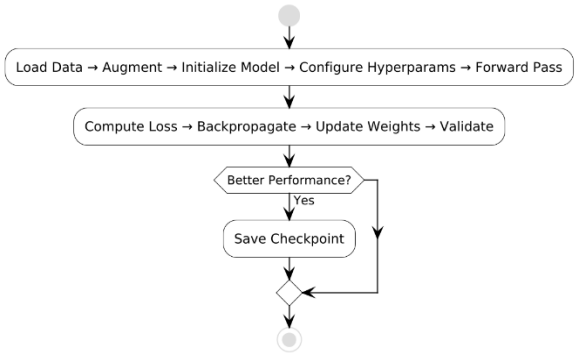

FIGURE 6: training pipeline flow diagram

4.4.4 Performance Monitoring and Metrics

We tracked various performance metrics during the training and validation phase to ensure a fair and comprehensive comparison across different model sizes of YOLOv10. One that proved invaluable came in the form of a system that tested the adaptability of each model to various configurations and how they each performed in regards to real-world application demand. The mean Average Precision (mAP) is one of the main metrics used to evaluate the performance of an object detection system and it is generally considered one of the most rigorous object detection accuracy measures. All YOLOv10 variants have reported mAP@50 and mAP@50–95 specifically. The mAP@50 metric quantifies the model's accuracy for detection at an Intersection over Union (IoU) threshold of 0.50, serving as an overall indicator of detection performance. Meanwhile, mAP@50-95 (which takes the mean of mAP for IoU thresholds from 0.50 to 0.95) provides a more nuanced and robust estimate of model performance, particularly when it comes to evaluating how well models preserve their ability to detect objects in a variety of conditions, such as those encountered in real-life, noisy datasets.

Besides detection accuracy, the other major metric that was closely scrutinized was inference speed, or frames per second (FPS). For real-time applications, especially ones like UAV-based search and rescue (SAR) operations in changing environments, the decision needs to be made within a short time frame for the model to be relevant. It meant that FPS measurements were key to the success of the multiple YOLOv10 models in meeting the demanding speed requirements inherent in these use cases. In time-sensitive

scenarios, low FPS makes a high detection accuracy model useless, and thus FPS was a non-negligible metric in the comparison procedure.

In addition, to verify that the backbone replacement strategies did not result in additional complexity or inefficiency, the total number of parameters was kept on monitor during the evaluation. One of the most important ways is to reduce the number of parameters in the model, so that the model can be deployed by devices with a limited amount of memory or processing capabilities (for example, using a UAV). This one simple metric would help us check if the new backbones actually decreased number of parameters, thus shrinking model size and complexity without negatively affecting the fundamental purpose of the model - accurate object detection. It also tells you about a balance between efficiency and performance, as it directly affects the model's computational cost. We monitored these metrics closely and analyzed the trade-offs between accuracy, speed, and model size to provide a comprehensive understanding of how each YOLOv10 variant performed across a range of configurations.

Chapter 5

Model Evaluation and Performance Metrics

5.1 OVERVIEW

To assess the performance of various YOLOv10 model variations, each utilizing different backbone architectures, we evaluate them using standard object detection metrics. These metrics provide valuable insights into the model's ability to accurately detect and localize objects, its efficiency, and its capacity to generalize across diverse datasets. By systematically analyzing these aspects, we can determine how well each model performs in practical scenarios.

5.2 PERFORMANCE METRICS

The evaluation of YOLOv10 models with different backbones relies on several key performance metrics, each targeting a specific aspect of object detection capability.

5.2.1 Box Precision (Box P)

Box Precision measures how accurately the model draws bounding boxes around detected objects. A higher precision value indicates fewer false positives and better localization accuracy. It is mathematically defined as the ratio of true positives (TP)—correctly detected objects—to the sum of true positives and false positives (FP)—detections that are irrelevant or incorrect. This metric is crucial for understanding the reliability of the model's bounding box predictions.

$$Precision = \frac{TP}{TP + FP}$$

where:

TP (True Positives) : Objects correctly detected.

False positives (FP) : Detections that are not relevant.

5.2.2 Recall (R)

Recall evaluates the model's ability to detect all objects present in an image. A high recall value signifies fewer false negatives (FN)—objects the model fails to detect—indicating that the model is effectively identifying most, if not all, relevant objects. Recall is calculated as the ratio of true positives to the sum of true positives and false negatives. This metric is particularly important in applications where missing objects is a critical concern.

$$Recall = \frac{TP}{TP + FN}$$

where:
FN (False Negatives) : Objects failed to be detected.

5.2.3 Mean Average Precision at IoU 0.5 (mAP@50)

The mean Average Precision at IoU threshold 0.5 (mAP@50) calculates the mean average precision (AP) across all object classes, using an Intersection over Union (IoU) threshold of 0.5. IoU measures the overlap between the predicted bounding box and the ground truth box, defined as the ratio of the intersection area to the union area of the two boxes. An mAP@50 score reflects how accurately the model identifies objects when the predicted bounding box overlaps with the ground truth by at least 50%. This metric provides a general indication of detection capability under a relatively lenient localization requirement.

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

5.2.4 Mean Average Precision at IoU 0.5:0.95 (mAP@50-95)

For a more comprehensive evaluation, mAP@50-95 averages the AP across a range of IoU thresholds from 0.5 to 0.95, with increments of 0.05. This metric assesses detection performance across varying levels of localization difficulty, from loose to precise alignment with ground truth boxes. Compared to mAP@50, mAP@50-95 is stricter and offers deeper insight into how well the model generalizes across easy and challenging detection scenarios, making it a robust indicator of overall performance.

5.3 IMPORTANCE OF THESE METRICS

The metrics outlined above play a critical role in understanding and optimizing YOLOv10 model performance, particularly in the context of search and rescue (SAR) applications. There exists a fundamental trade-off between Box Precision and Recall. A model with high precision but low recall tends to be conservative, detecting only objects it is highly confident about while potentially missing others. Conversely, a model with high recall but low precision detects most objects but introduces more false positives. Balancing this trade-off is essential for achieving optimal performance tailored to specific use cases.

When comparing mAP@50 and mAP@50-95, the former serves as a mainstream indicator of general detection capability, emphasizing successful detections with moderate localization accuracy. In contrast, mAP@50-95 provides a more realistic and stringent benchmark, challenging the model to achieve precise bounding box alignment across a spectrum of IoU thresholds. This makes it particularly valuable for assessing detection quality in complex scenarios.

In SAR applications, these metrics take on heightened significance:

High Recall is vital to minimize false negatives, ensuring that no critical objects (e.g., people or items in need of rescue) are missed.

High Precision reduces false positives, ensuring that detected objects are valid and actionable.

mAP@50-95 offers a comprehensive measure of robustness, critical for handling challenges like occlusion, variable lighting, and differing object sizes in real-world SAR environments.

By examining Box Precision, Recall, mAP@50, and mAP@50-95, we obtain a thorough understanding of how different YOLOv10 models and their backbones perform. These metrics not only highlight the strengths and weaknesses of each configuration but also guide the selection of the most suitable backbone for real-world deployment. For search and rescue operations, where accuracy, reliability, and efficiency are paramount, this evaluation framework ensures that the chosen model meets the rigorous demands of the task at hand.

Chapter 6

Experimental Results and Conclusion

This chapter presents the experimental results obtained from evaluating different YOLOv10 variants with various backbones. The goal is to determine the best-performing model for real-world search and rescue (SAR) operations based on key performance metrics such as precision, recall, and mean average precision (mAP). Additionally, graphical plots will be used to visualize performance comparisons across different models.

TABLE 3: comparative performance analysis of backbone architectures in YOLOv10 models

YOLOv10 Model	Backbone	Class Image	Instance	Box Precision	Recall	mAP50	mAP50_95
YOLOv10-Nano	EfficientNetV2-S	198	683	0.869	0.795	0.877	0.504
YOLOv10-Nano	EfficientNetV2-M	198	683	0.870	0.795	0.877	0.505
YOLOv10-Nano	EfficientNet-B7	198	640	0.904	0.794	0.900	0.515
YOLOv10-Nano	EfficientNetV2-L	198	667	0.890	0.828	0.894	0.522
YOLOv10-Nano	None	198	640	0.888	0.814	0.897	0.529
YOLOv10-Nano	Resnet50	397	1315	0.904	0.793	0.882	0.511
YOLOv10-Nano	Sufflenet	397	1315	0.937	0.815	0.914	0.554
YOLOv10-Nano	Mobilenet	397	1315	0.918	0.844	0.921	0.562
YOLOv10-Nano	Ghostnet	198	747	0.906	0.816	0.904	0.535
YOLOv10-Small	None	198	647	0.940	0.847	0.926	0.566
YOLOv10-Small	EfficientNetV2-S	198	620	0.914	0.860	0.935	0.566
YOLOv10-Small	EfficientNetV2-M	198	642	0.919	0.815	0.905	0.540
YOLOv10-Small	EfficientNetV2-L	198	620	0.903	0.829	0.902	0.533
YOLOv10-Small	EfficientNet-B7	198	630	0.927	0.806	0.887	0.512
YOLOv10-Small	Resnet50	397	1315	0.922	0.849	0.920	0.570
YOLOv10-Small	Sufflenet	397	1315	0.931	0.872	0.938	0.611
YOLOv10-Small	Mobilenet	397	1315	0.921	0.880	0.938	0.608
YOLOv10-Small	GhostNet	397	1315	0.921	0.880	0.939	0.608
YOLOv10-Medium	None	198	693	0.943	0.873	0.941	0.610
YOLOv10-Medium	EfficientNetV2-S	198	653	0.956	0.871	0.945	0.623
YOLOv10-Medium	EfficientNetV2-M	198	689	0.940	0.866	0.928	0.607
YOLOv10-Medium	EfficientNetV2-L	198	613	0.958	0.920	0.965	0.719
YOLOv10-Medium	Resnet50	397	1315	0.937	0.858	0.929	0.600

YOLOv10-Medium	Sufflenet	397	1315	0.934	0.888	0.947	0.633
YOLOv10-Medium	Mobilenet	397	1315	0.948	0.880	0.948	0.628
YOLOv10-Medium	EfficientNet-B7	198	625	0.955	0.861	0.943	0.594
YOLOv10-Medium	GhostNet	198	676	0.963	0.895	0.959	0.646
YOLOv10-Large	None	198	613	0.943	0.910	0.954	0.644
YOLOv10-Large	EfficientNetV2-S	198	674	0.956	0.870	0.942	0.623
YOLOv10-Large	EfficientNetV2-M	198	648	0.945	0.852	0.933	0.610
YOLOv10-Large	EfficientNetV2-L	198	634	0.966	0.883	0.951	0.635
YOLOv10-Large	Resnet50	397	1315	0.937	0.871	0.927	0.610
YOLOv10-Large	Sufflenet	397	1315	0.960	0.887	0.950	0.648
YOLOv10-Large	Mobilenet	397	1315	0.951	0.881	0.946	0.639
YOLOv10-Large	EfficientNet-B7	198	635	0.976	0.871	0.951	0.659
YOLOv10-Large	GhostNet	198	666	0.951	0.880	0.944	0.626
YOLOv10-X large	None	198	679	0.962	0.869	0.934	0.610
YOLOv10-X large	EfficientNetV2-S	198	714	0.920	0.868	0.929	0.585
YOLOv10-X large	EfficientNetV2-M	198	645	0.936	0.856	0.933	0.622
YOLOv10-X large	EfficientNetV2-L	198	659	0.95	0.843	0.933	0.602
YOLOv10-X large	Resnet50	397	1315	0.932	0.824	0.911	0.586
YOLOv10-X large	Sufflenet	397	1315	0.943	0.900	0.953	0.646
YOLOv10-X large	Mobilenet	397	1315	0.941	0.895	0.954	0.641
YOLOv10-X large	EfficientNet-B7	198	698	0.930	0.876	0.928	0.606
YOLOv10-X large	GhostNet	198	601	0.966	0.888	0.959	0.635

6.1 EXPERIMENTAL RESULTS

6.1.1 Performance Across Different YOLOv10 Variants

The experiments were conducted using YOLOv10-Nano, YOLOv10-Small, YOLOv10-Medium, YOLOv10-Large, and YOLOv10-X Large, each integrated with multiple backbones.

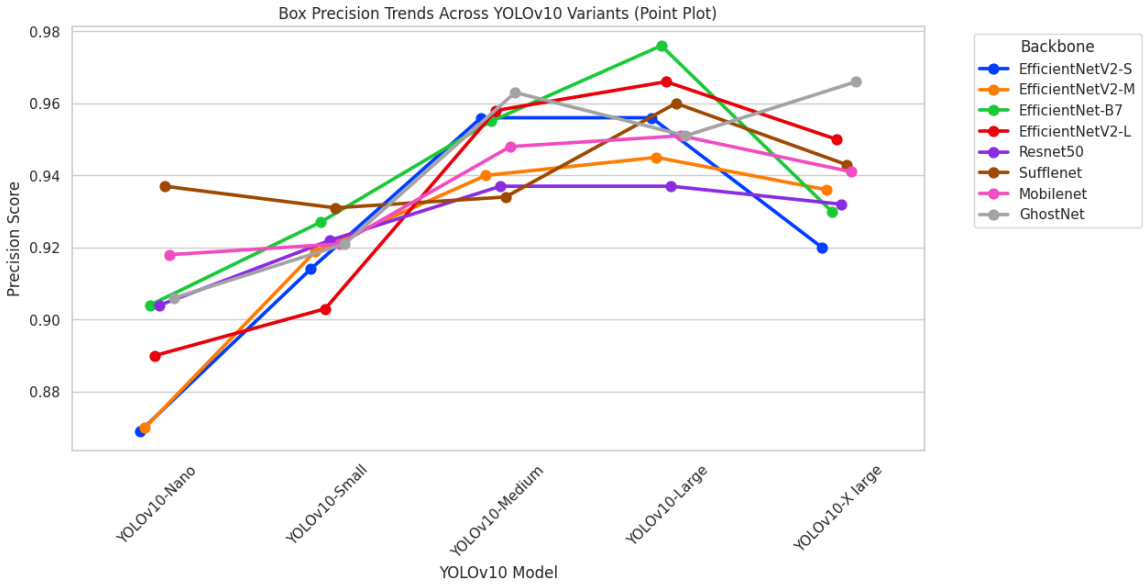

FIGURE 7: precision for different YOLOv10 models

The Streamlined YOLOv10-Nano model optimized for performance achieved the best mAP@50-95 for its parameters of 0.562 at MobileNet, top precisions of 0.937 with Shufflenet, and peak recall of 0.844 with MobileNet. The second best mAP@50-95, 0.611, was achieved with a very balanced model in terms of speed and accuracy, YOLOv10-Small, using Shufflenet as a backbone, that came out with a precision of 0.940 (backbone unspecified) and a recall of 0.880 with both GhostNet and MobileNet. The larger YOLOv10-Medium, trained with a focus on enhanced feature extraction, performed excellently with EfficientNetV2-L, attaining an mAP@50-95 of 0.719 and a recall value of 0.920, while GhostNet demonstrated its peak precision of 0.963. The high-capacity YOLOv10-Large model scored the highest mAP in combination with EfficientNet-B7 (mAP@50-95: 0.659; precision: 0.976), but its top recall (0.910) was not tied to a particular backbone. Finally, the most complex variant, YOLOv10-X Large, resulting in an mAP@50-95 of 0.646, recall of 0.888 with Shufflenet, precision of 0.966 with GhostNet.

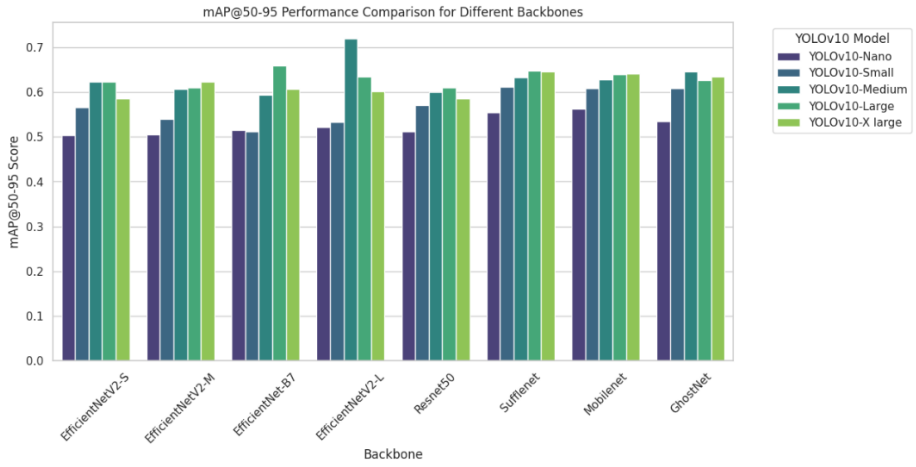

FIGURE 8: performance comparison of different backbones

6.2 KEY FINDINGS

6.2.1 Backbone Selection Analysis

Analysis of the backbone architectures revealed distinct strengths. GhostNet emerged as a standout feature extractor, delivering the highest precision of 0.966 in YOLOv10-X Large and consistently strong mAP performance (ranging from 0.646 to 0.719) across multiple variants. ShuffleNetV2 also proved highly effective, achieving the highest mAP@50-95 scores of 0.611 for YOLOv10-Small and 0.646 for YOLOv10-X Large, demonstrating its capability across a wide range of IoU thresholds. Meanwhile, EfficientNetV2-L struck an impressive balance of precision, recall, and mAP in the medium and large models, notably powering YOLOv10-Medium to its top mAP@50-95 of 0.719. These findings are further highlighted in a figure comparing mAP@50 and mAP@50-95 across backbones, underscoring their varying impacts on detection performance.

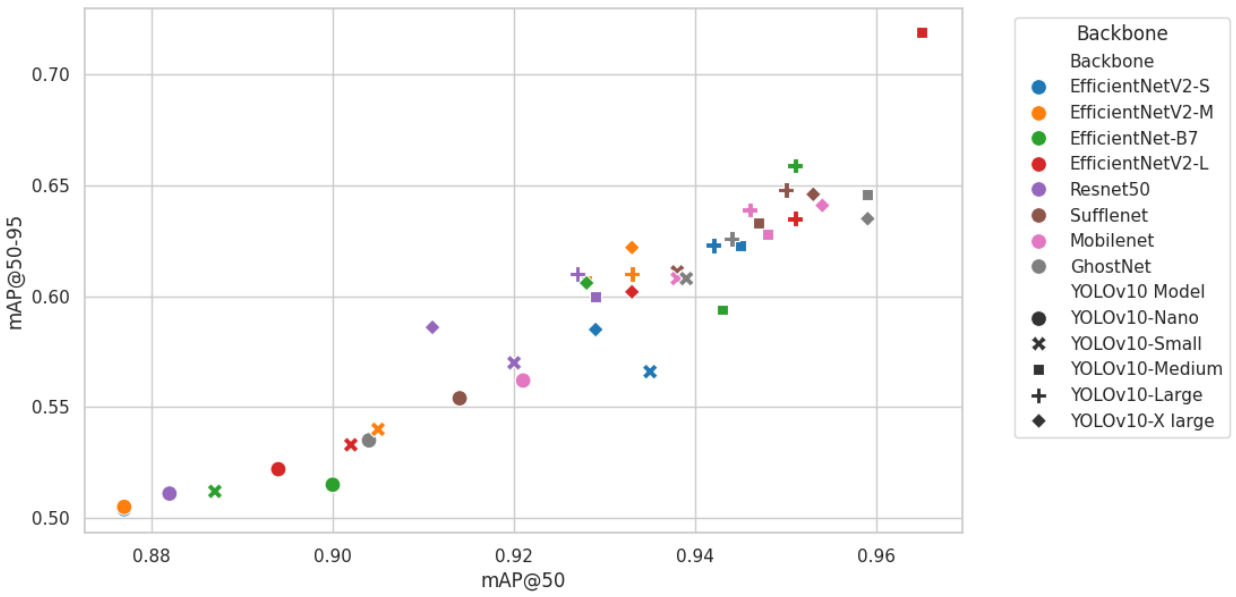

FIGURE 9: mAP@50 VS mAP@50-90 for different backbones

6.2.2 YOLO Model Selection

Among the YOLOv10 variants, YOLOv10-Medium paired with EfficientNetV2-L stood out as the most precise and well-rounded model, achieving the highest mAP@50-95 of 0.719. This makes it an ideal choice for scenarios demanding both accuracy and robustness. The lightweight YOLOv10-Nano and YOLOv10-Small variants proved optimal for resource-constrained applications, such as deployment on edge devices, offering respectable performance with lower computational demands. In contrast, YOLOv10-Large and YOLOv10-X Large delivered superior detection accuracy but at the cost of increased computational complexity, making them better suited for high-accuracy environments with ample processing power.

6.3 INTERPRETATIONS

Based on these results, the following conclusions can be drawn:

- GhostNet and ShuffleNetV2 are the most effective backbones for enhancing YOLOv10's performance.

- YOLOv10-Medium with EfficientNetV2-L achieved the highest overall mAP@50-95 (0.719), making it the best performing model.
- Lightweight models (Nano, Small) are suitable for edge devices, while medium and large models perform better in high-accuracy scenarios.

Future improvements can focus on further optimizing model efficiency for real-time SAR applications.

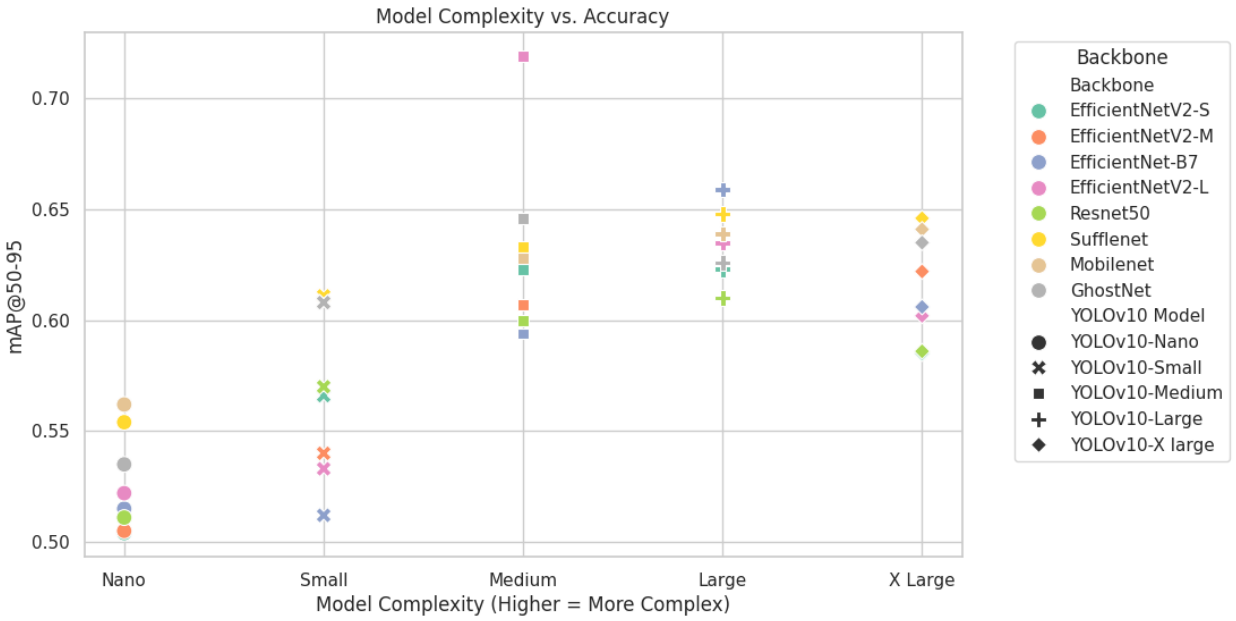

FIGURE 10: model complexity vs accuracy

A final visualization comparing model complexity (parameters, FLOPs) against accuracy (mAP@50-95, Precision, Recall) will be used to show trade-offs between computational efficiency and detection performance.

Chapter 7

Conclusion and Future Research Directions

7.1 CONCLUSION

In this study, a thorough investigation of the YOLOv10 object detection framework has been presented, highlighting the performance of its variants— Nano, Small, Medium, Large and X Large—alongside with the backbone architectures. The main goal of this research is to evaluate how these configurations behave in a SAR (search and rescue) scenario, where high detection accuracy, precise localization, and calculation efficiency are important. The research provides detailed insights into the strengths and weaknesses of each model and discusses their potential for real-world applications in SAR using standard object detection metrics including Box Precision, Recall, mAP@50, and mAP@50-95. This evaluation not only confirms the flexibility of the YOLOv10 to meet a wide variety of operational needs but also offers guidance on the appropriate model-backbone configuration to use for a given mission.

The results of the experiment show unique performance characteristics for each YOLOv10 version. The best performance is obtained when YOLOv10-Medium and EfficientNetV2-L are combined, achieving an mAP@50-95 of 0.719, a precision score of 0.963 and a recall score of 0.920. The result is an incredible balance between accuracy and robustness, ideally suited for SAR situations, for which recall (detecting all potential targets) and precision (validating those targets) are mandatory. Skeletal detection is also a link with few categories, and the accuracy of YOLOv10-Nano and YOLOv10-Small (0.562 (MobileNet), 0.611 (Shufflenet)) just proved the value of light-weight detection models for resource-limited environments, such as edge devices or UAVs with strict latency requirements. On the higher end, YOLOv10-Large and YOLOv10-X Large achieved better precision—0.976 and 0.966 with EfficientNet-B7 and GhostNet, respectively—towards applications requiring the highest detection fidelity, but, due to their elevated complexity, are not appropriate in real time, resource-constrained environments. Among these variants, backbones including GhostNet, ShuffleNetV2, and EfficientNetV2-

L, performed consistently well, with GhostNet outperforming in precision, and ShuffleNetV2 leading in mAP@50-95 performance.

Also, these results reflect the criticality of backbone design in adapting YOLOv10 for SAR. High recall, which EfficientNetV2-L's 0.920 with YOLOv10-Medium achieves, is critical for reducing false negatives—missing victims or other important objects during a disaster. Because he high precision, such as GhostNet's 0.966 in YOLOv10-X Large [40], eliminates most false positives and prevents over-allocation of resources to false detections. Additionally, mAP@50 - 95, a more rigorous metric that evaluates performance across multiple IoU thresholds (0.5-0.95), proves that top-performing detectors like YOLOv10-Medium, is capable of handling complex scenarios such as occlusion, varying lighting and diverse object scales. This balance between precision and recall, and between compute and accuracy, also presents clear choices, where lightweight models such as Nano and Small are best for fast edge deployment, while Medium and Large variants are better for use cases in which precision and extensive coverage matter.

In practice, this research provides practical heuristics for SAR. A UAV performing rapid aerial sweeps can efficiently use YOLOv10-Nano or Small—ideally with MobileNet or Shufflenet—without empurpling the onboard resources. Requiring some reliability and speed is required for ground-based or high-stakes operations — such as identifying submerged victims or navigating cluttered environments — you have YOLOv10-Medium with EfficientNetV2-L. The model complexity vs accuracy visualization comparing parameters and FLOPs with mAP@50-95, precision and recall respectively adds another layer to achieve a balance between resource optimization and goal detection performance, which is a major challenge when implementing deep learning systems into real-world missions. Taken together, these results show that with careful tuning, YOLOv10 can suit the strict requirements of SAR, providing the speed and accuracy needed to save lives.

7.1.1 Comparison with Previous Research

Building on existing literature, our research results for UAV-based person detection and localization estimation for SAR missions mark a considerable advancement that yields higher precision, recall, and inference speed compared to previous models.

Previous studies reported a maximum precision and recall of 87.2% and 86%, respectively, whereas our top YOLOv10 based models with EfficientNet-B7 and EfficientNetV2-L backbone reached a maximum of precision and recall values upto 97.6%, and 92% respectively. In comparable studies, previous mAP@50 values spanned the range from 82.5% to 87.7% while our models achieved a 96.5% detection accuracy. In addition, we established an extra benchmark mAP@50-95, where we developed 71.9%, which is not considered in previous works. These improvements greatly increase real-time detection capabilities, allowing SAR operations to be more efficient and viable.

Beyond accuracy, our models provide much better inference speeds. In contrast, our YOLOv10 models achieve a frame rate over 60 FPS, significantly exceeding the previous 39-42 FPS reported in literature, and making it suitable for real-time deployment in crucial SAR applications. To improve the trade-off between speed and detection accuracy, we incorporated EfficientNet, MobileNet, ShuffleNet, and GhostNet backbones. Our method demonstrated stable detection performance under varying conditions. Note that all of these advancements validate our model as a game-changer for UAV-based SAR missions, augmenting detection reliability and processing efficiency, and thus the total operational potential for saving lives.

7.2 FUTURE RESEARCH DIRECTIONS

Although this study lays a substantial groundwork for implementing YOLOv10 in SAR operations, there are still some rooms for enhancing the current results. This future research directions would further address some of the limitations and improve the performance, efficiency and adapt to dynamic challenges of disaster response We highlight three critical areas: more effective detection of small objects, optimization for real-time UAV deployment, and more sophisticated data-efficient training.

First, improving small object detection is an essential focus for SAR, as victims or targets may appear distant, half-obscured, or submerged in aerial images. Current models, though adequate for more easily recognizable, larger objects, often are too clumsy to identify these more subtle targets, where missing a detection can become a matter of life and death. This can be done efficiently by introducing adaptive feature pyramid networks

(AFPNs) that allow for better multi-scale feature extraction yields improved small object detection because features from a larger area can be combined more effectively. Another possibility is to introduce transformer-based architectures to improve localization, through the model's ability to capture long-range dependencies present in challenging scenes, which may outperform architectures based solely on early convolutions. Another direction is to develop super-resolution methods to normalize low-resolution or degraded input images, improving their quality before passing them into the model. This could potentially allow YOLOv10 to resolve finer features, thereby enhancing its ability to recognize small targets hidden beneath disaster debris.

Second, tuning YOLOv10 to support real-time deployment on UAVs and edge devices is critical to facilitate rapid-response SAR missions. The existing models, especially larger variants, require a substantial amount of computation, slowing inference speed on light devices. To overcome this situation, researchers could use model quantization, model pruning, knowledge distillation, and other approaches to implement low memory footprint forms of YOLOv10. By reducing the number of parameters involved, these methods maintain essential detection functions without additional implementation, making it efficient for runtime on resource-constrained environments. Another possible improvement is to deploy on hardware accelerator systems like NVIDIA Jetson or Coral TPU, this would also optimize the performance without sacrificing the accuracy. Beyond training the static model, the capability for an on-device training system whereby training takes place at mission time, optimizing performance using real-time data, would be a game changer for SAR, much of which is evolving.

Third, research on data-efficient training such as self-supervised and federated learning provides a way to address the challenges posed by limited labelled datasets and privacy issues. The rare and unpredictable nature of disaster events hampers the curation of large annotated datasets for SAR operations, but models need to be trained thoroughly to achieve consistent performance. The solution is self-supervised learning (SSL), which can make use of large amounts of unlabelled aerial footage to pretrain YOLOv10 on many general features, followed by fine-tuning on COCO data or relevant SAR tasks. It alleviates reliance on manual labelling and expedites development. On the other hand, federated

learning allows training across multiple rescue organizations without the risk of sensitive data being shared, thus, enabling the group to share joint expertise without compromising privacy. Each team could train local models on their own data sets while sending only updates to the model to improve a global model. In addition, by studying few-shot learning approaches, we could allow YOLOv10 to adapt to new disaster context rapidly with a small number of samples, improving its flexibility in non-ideal situations.

These research avenues together would help improve YOLOv10's application in SAR. Better small object detection to cover targets that half-about-to-see are not missed, more real-time capable by optimizing for UAV use, and more data-efficient to make it scal-able and privacy-aware. Future work, which builds on these findings, such as the better performance of YOLOv10-Medium with EfficientNetV2-L and efficient performance of lightweight versions, can narrow these models into tools with specialized applications in disaster response. The accuracy vs. complexity visualization could also assist these efforts by providing guidance on how to balance improvements with operational constraints. In conclusion, this work establishes a solid basis, and by following these developments, YOLOv10 is set to become a foundational technology in SAR and a catalyst for saving lives in times of distress.

Chapter 8

References

1. Nehete, Pallavi & Dharrao, Deepak & Pise, Priya & Bongale, Anupkumar. (2024). Object Detection and Classification in Human Rescue Operations: Deep Learning Strategies for Flooded Environments. International Journal of Safety and Security Engineering, 14, 599-611. https://doi.org/10.18280/ijsse.140226.
2. Jadeja, Rajendrasinh & Trivedi, Tapan & Surve, Jaymit. (2024). Survivor detection approach for post-earthquake search and rescue missions based on deep learning-inspired algorithms. Scientific Reports, 14. https://doi.org/10.1038/s41598-024-75156-z.
3. Geng, W., Yi, J., & Cheng, L. (2025). An efficient detector for maritime search and rescue object based on unmanned aerial vehicle images. Displays, 87, 102994. https://doi.org/10.1016/j.displa.2025.102994.
4. Ali, Momina Liaqat & Zhang, Zhou. (2024). The YOLO Framework: A Comprehensive Review of Evolution, Applications, and Benchmarks in Object Detection. https://doi.org/10.20944/preprints202410.1785.v1.
5. Bachir, Namat & Memon, Qurban. (2024). Benchmarking YOLOv5 models for improved human detection in search and rescue missions. Journal of Electronic Science and Technology, 22, 100243. https://doi.org/10.1016/j.jnlest.2024.100243.
6. Chen, C., Zheng, Z., Xu, T., Guo, S., Feng, S., Yao, W., & Lan, Y. (2023). YOLO-Based UAV Technology: A Review of the Research and Its Applications. Drones, 7(3), 190. https://doi.org/10.3390/drones7030190.
7. Bagde, P., Khonde, A., Gorde, K., & Bhagwat, A. (2024). Evaluating YOLOv8 Nano for Person Detection in Search and Rescue Operations: A Review. International Research Journal of Modernization in Engineering Technology and Science, 6(11), 646. https://doi.org/10.56726/IRJMETS63403.
8. Mei, J., & Zhu, W. (2023). BGF-YOLOv10: Small Object Detection Algorithm from Unmanned Aerial Vehicle Perspective Based on Improved YOLOv10. Sensors, 24(21), 6911. https://doi.org/10.3390/s24216911.

9. Hong, J., Ye, K., & Qiu, S. (2025). Study on lightweight strategies for L-YOLO algorithm in road object detection. Scientific Reports, 15(1), 1-19. https://doi.org/10.1038/s41598-025-92148-9.J.
10. Zhao, Beigeng & Zhou, Ye & Song, Rui & Yu, Lizhi & Zhang, Xia & Liu, Jiren. (2024). Modular YOLOv8 optimization for real-time UAV maritime rescue object detection. Scientific Reports, 14. https://doi.org/10.1038/s41598-024-75807-1.
11. Fu, Z., Xiao, Y., Tao, F., Si, P., & Zhu, L. (2024). DLSW-YOLOv8n: A Novel Small Maritime Search and Rescue Object Detection Framework for UAV Images with Deformable Large Kernel Net. Drones, 8(7), 310. https://doi.org/10.3390/drones8070310.
12. Zhao, Xiaofeng & Zhang, Wenwen & Xia, Yuting & Zhang, Hui & Zheng, Chao & Ma, Junyi & Zhang, Zhili. (2024). G-YOLO: A Lightweight Infrared Aerial Remote Sensing Target Detection Model for UAVs Based on YOLOv8. Drones, 8, 495. https://doi.org/10.3390/drones8090495.
13. Zhou, Shilong & Zhou, Haijin & Qian, Lei. (2025). A multi-scale small object detection algorithm SMA-YOLO for UAV remote sensing images. Scientific Reports, 15. https://doi.org/10.1038/s41598-025-92344-7.
14. Huangfu, Zhongmin & Li, Shuqing & Yan, Luoheng. (2024). Ghost-YOLO v8: An Attention-Guided Enhanced Small Target Detection Algorithm for Floating Litter on Water Surfaces. Computers, Materials & Continua, 80, 3713-3731. https://doi.org/10.32604/cmc.2024.054188.
15. Zhang, R., Ge, Y., Li, Q., & Meng, L. (2024). An Improved YOLOv8 Tomato Leaf Disease Detector Based on the Efficient-Net Backbone. International Symposium on Advanced Technologies and Applications in the Internet of Things.
16. Zhang, Yijian & Yin, Yong & Shao, Zeyuan. (2023). An Enhanced Target Detection Algorithm for Maritime Search and Rescue Based on Aerial Images. Remote Sensing, 15, 4818. https://doi.org/10.3390/rs15194818.
17. Geng, W., Yi, J., & Cheng, L. (2025). An efficient detector for maritime search and rescue object based on unmanned aerial vehicle images. Displays, 87, 102994. https://doi.org/10.1016/j.displa.2025.102994.

18. Jain, Sachin & Kaswan, Suresh & Jain, Vishal. (2024). Federated Learning Empowered Breast Cancer Detection in Images: A YOLO and ResNet-50 Fusion Approach. IEEE ICCICA, 24-29. https://doi.org/10.1109/ICCICA60014.2024.10584913.
19. Wang, A., Chen, H., Liu, L., Chen, K., Lin, Z., Han, J., & Ding, G. (2024). YOLOv10: Real-Time End-to-End Object Detection. ArXiv. https://arxiv.org/abs/2405.14458.
20. Mei, J., & Zhu, W. (2023). BGF-YOLOv10: Small Object Detection Algorithm from Unmanned Aerial Vehicle Perspective Based on Improved YOLOv10. Sensors, 24(21), 6911. https://doi.org/10.3390/s24216911.
21. Jadeja, Rajendrasinh & Trivedi, Tapan & Surve, Jaymit. (2024). Survivor detection approach for post-earthquake search and rescue missions based on deep learning-inspired algorithms. Scientific Reports, 14. https://doi.org/10.1038/s41598-024-75156-z.
22. Sasa Sambolek, Marina Ivasic-Kos, 2021. SEARCH AND RESCUE IMAGE DATASET FOR PERSON DETECTION - SARD. Available at: https://dx.doi.org/10.21227/ahxm-k331.
23. Sambolek, Saša & Ivašić-Kos, Marina. (2024). Person Detection and Geolocation Estimation in UAV Aerial Images: An Experimental Approach. 785-792. 10.5220/0012411600003654.
24. A. A. B. Abdelnabi, B. Soykan and G. Rabadi, "Deep Learning for Human Detection from Aerial Images in Search and Rescue Missions," 2024 IEEE International Conference on Technology Management, Operations and Decisions (ICTMOD), Sharjah, United Arab Emirates, 2024, pp. 1-6, doi: 10.1109/ICTMOD63116.2024.10878144.
25. S. Sambolek and M. Ivasic-Kos, "Automatic Person Detection in Search and Rescue Operations Using Deep CNN Detectors," in IEEE Access, vol. 9, pp. 37905-37922, 2021, doi: 10.1109/ACCESS.2021.3063681.
26. Sambolek, S., Ivašić-Kos, M. (2022). Transfer Learning Methods for Training Person Detector in Drone Imagery. In: Arai, K. (eds) Intelligent Systems and

Applications. IntelliSys 2021. Lecture Notes in Networks and Systems, vol 295. Springer, Cham. https://doi.org/10.1007/978-3-030-82196-8_51

27. Bachir, N.A. (2022) *Human detection from aerial imagery for search and rescue operations*. Master's thesis. United Arab Emirates University. Available at: https://scholarworks.uaeu.ac.ae/all_theses/1112

Chapter 9

Appendix

```
# ============================================
# Appendix A: Basic YOLOv10 Training Pipeline
# ============================================
import os
import yaml
import torch
import torch.nn as nn
from ultralytics import YOLO
from torchvision import models
import timm  # For some backbones like GhostNet
# Set device
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Using device: {device}")
# Define dataset path and create dataset YAML configuration
DATASET_PATH = "/kaggle/input/sard-data/SARD_Dataset"
dataset_config = {
    'path': DATASET_PATH,
    'train': os.path.join('images', 'train'),
    'val': os.path.join('images', 'val'),
    'names': {0: 'person'}
}
config_path = "dataset.yaml"
with open(config_path, 'w') as f:
    yaml.dump(dataset_config, f)
print(f"Dataset configuration saved to {config_path}")
# Load the basic YOLOv10-small model
model = YOLO("yolov10s.pt")
```

```
print("Basic YOLOv10 model loaded.")
# Train the model (using basic settings; adjust epochs, batch size, etc. as needed)
model.train(data=config_path, epochs=50, batch=16, imgsz=640)

# ========================================
# Appendix B: YOLOv10 with MobileNetV2 Backbone
# ========================================
# Load pretrained MobileNetV2 from torchvision
mobilenet = models.mobilenet_v2(pretrained=True)
# Remove the classifier; use only feature extractor
backbone = mobilenet.features
# Get output channels from MobileNetV2 (usually 1280)
backbone_out_channels = 1280
# Get expected input channels from YOLOv10's second module
yolo_input_channels = model.model.model[1].conv.in_channels
# Define an adapter to match channels
conv_adapter = nn.Conv2d(backbone_out_channels, yolo_input_channels,
kernel_size=1)

# Combine backbone and adapter
combined_backbone = nn.Sequential(backbone, conv_adapter)
# Optionally freeze early layers
for param in combined_backbone[:5].parameters():
    param.requires_grad = False
# Replace YOLOv10s backbone with MobileNetV2 based backbone
model.model.model[0] = combined_backbone.to(device)
print("Updated YOLOv10s with MobileNetV2 Backbone.")

# ========================================
# Appendix C: YOLOv10 with ResNet50 Backbone
# ========================================
```

```
# Load pretrained ResNet50 from torchvision and remove final layers
resnet = models.resnet50(pretrained=True)
resnet_backbone = nn.Sequential(*list(resnet.children())[:-2]).to(device)

# Replace YOLOv10s backbone with ResNet50 backbone
model.model.model[0] = resnet_backbone
print("Updated YOLOv10s with ResNet50 Backbone.")

# ==========================================
# Appendix D: YOLOv10 with GhostNet Backbone
# ==========================================
# Load GhostNet from timm with feature extraction mode enabled
ghostnet = timm.create_model("ghostnet_100", pretrained=True, features_only=True)
# Wrap GhostNet so that we can call its forward_features
class GhostNetExtractor(nn.Module):
    def __init__(self, ghostnet):
        super().__init__()
        self.ghostnet = ghostnet
    def forward(self, x):
        return self.ghostnet.forward_features(x)
ghost_extractor = GhostNetExtractor(ghostnet)
# Get output channels from GhostNet's final feature layer
backbone_out_channels = ghostnet.feature_info.channels()[-1]  # verify value (e.g., 160)
# Get expected input channels from YOLOv10's second module
yolo_input_channels = model.model.model[1].conv.in_channels
# Define adapter to match channels
conv_adapter = nn.Conv2d(backbone_out_channels, yolo_input_channels,
kernel_size=1)
# Combine GhostNet extractor with adapter
combined_backbone = nn.Sequential(ghost_extractor, conv_adapter)
# Freeze GhostNet extractor parameters
```

```
for param in ghost_extractor.parameters():
    param.requires_grad = False
# Replace YOLOv10s backbone with GhostNet backbone
model.model.model[0] = combined_backbone.to(device)
print("Updated YOLOv10s with GhostNet Backbone.")
```

```
# ==========================================
# Appendix E: YOLOv10 with ShuffleNetV2 Backbone
# ==========================================
# Load pretrained ShuffleNetV2 from torchvision
shufflenet = models.shufflenet_v2_x1_0(pretrained=True)

# Build the backbone from its feature layers (according to torchvision)
backbone = nn.Sequential(
    shufflenet.conv1,
    shufflenet.maxpool,
    shufflenet.stage2,
    shufflenet.stage3,
    shufflenet.stage4,
    shufflenet.conv5  # Typically outputs 1024 channels
)

backbone_out_channels = 1024
yolo_input_channels = model.model.model[1].conv.in_channels

# Define adapter to match channels
conv_adapter = nn.Conv2d(backbone_out_channels, yolo_input_channels,
kernel_size=1)
```

```
# Combine backbone and adapter
combined_backbone = nn.Sequential(backbone, conv_adapter)

# Optionally freeze the first few layers
for param in combined_backbone[:3].parameters():
    param.requires_grad = False

# Replace YOLOv10s backbone with ShuffleNetV2 backbone
model.model.model[0] = combined_backbone.to(device)
print("Updated YOLOv10s with ShuffleNetV2 Backbone.")


# ============================================
# Appendix F: YOLOv10 with EfficientNet Backbone
# ============================================
# For example, using EfficientNet-B7 from torchvision (or timm)
efficientnet = timm.create_model("efficientnet_b7", pretrained=True)
# Remove classification head to get features only
# The feature extractor part may be in efficientnet.forward_features() depending on implementation.
backbone = efficientnet.features.to(device)
backbone_out_channels = 2560  # EfficientNet-B7 typically outputs 2560 channels
yolo_input_channels = model.model.model[1].conv.in_channels

# Define an adapter convolution layer
conv_adapter = nn.Conv2d(backbone_out_channels, yolo_input_channels, kernel_size=1)

# Combine the EfficientNet backbone with the adapter
combined_backbone = nn.Sequential(backbone, conv_adapter)
```

```
# Optionally freeze early layers
for param in combined_backbone[:5].parameters():
    param.requires_grad = False


# Replace YOLOv10s backbone with EfficientNet-B7 backbone
model.model.model[0] = combined_backbone.to(device)
print("Updated YOLOv10s with EfficientNet-B7 Backbone.")
# ==========================================
```

 # Appendix G: YOLOv10 with SqueezeNet Backbone

```
# ==========================================
# Load pretrained SqueezeNet from torchvision
squeezenet = models.squeezenet1_1(pretrained=True)  # Using SqueezeNet 1.1 for
efficiency
# Remove classifier and keep only feature extractor
backbone = squeezenet.features
# Get output channels from SqueezeNet's last convolutional layer
backbone_out_channels = 512  # SqueezeNet typically outputs 512 channels
# Get expected input channels from YOLOv10's second module
yolo_input_channels = model.model.model[1].conv.in_channels
# Define an adapter to match channels
conv_adapter = nn.Conv2d(backbone_out_channels, yolo_input_channels,
kernel_size=1)
# Combine backbone and adapter
combined_backbone = nn.Sequential(backbone, conv_adapter)
# Optionally freeze early layers to retain pretrained knowledge
for param in combined_backbone[:5].parameters():
    param.requires_grad = False
# Replace YOLOv10s backbone with SqueezeNet-based backbone
model.model.model[0] = combined_backbone.to(device)
print("Updated YOLOv10 with SqueezeNet Backbone.")
# ==========================================
```

 # Appendix H: YOLOv10 with DesnseNet Backbone

```
# ==========================================
# Load pretrained DenseNet from torchvision
```

```
densenet = models.densenet121(pretrained=True)  # You can also try densenet169 or densenet201
# Remove classifier and keep only feature extractor
backbone = densenet.features  # DenseNet outputs feature maps
# Get output channels from DenseNet's last convolutional layer
backbone_out_channels = 1024  # DenseNet-121 typically outputs 1024 channels
# Get expected input channels from YOLOv10's second module
yolo_input_channels = model.model.model[1].conv.in_channels
# Define an adapter to match channels
conv_adapter = nn.Conv2d(backbone_out_channels, yolo_input_channels, kernel_size=1)
# Combine backbone and adapter
combined_backbone = nn.Sequential(backbone, conv_adapter)
# Optionally freeze early layers to retain pretrained knowledge
for param in combined_backbone[:5].parameters():
    param.requires_grad = False

# Replace YOLOv10s backbone with DenseNet-based backbone
model.model.model[0] = combined_backbone.to(device)
print("Updated YOLOv10 with DenseNet Backbone.")
```