

Table of Contents

Title	Page No
Bona-fide Certificate	ii
Declaration	iii
Acknowledgements	iv
List of Figures	vii
List of Tables	ix
Abstract	xi
1. Introduction	01
1.1. Problem Statement	01
1.2. Motivation	02
1.3. Significance	02
1.4. Objective	02
1.5. Scope of the Project	02
1.6. Overview of UAV Object Detection	03
2. Objectives	04
3. Literature Survey	05
3.1. YOLO Series Overview	05
3.2. Limitations of Existing Methods	07
3.3. MSFE-YOLO Overview	07
4. Preprocessing	09
4.1. Dataset Description (VisDrone)	09
4.2. Image Resizing	11
4.3. Data Augmentation Techniques	11
5. Experimental Work / Methodology	14
5.1. Overview of MSFE-YOLO Architecture	14
5.2. Symmetric C2f (SCF) Module	16
5.3. Efficient Multiscale Attention (EMA)	16
5.4. Feature Fusion (FF) Module	17
5.5. Training Configuration	18
6. Evaluation & Results	21
6.1. Precision, Recall, mAP Scores	21
6.2. Effect of Model Architecture (SCF, EMA, FF)	28
7. Conclusion and Future Scope	30
7.1. Project Conclusion	30
7.2. Limitations	30
7.3. Future Enhancements	31

8. References	32
9. Appendices	33
9.1. Similarity Check Report	33
9.2. Sample Source Code	33

List of Figures

Figure No.	Title	Page No.
1.6.1	UAV Object Detection	03
3.1.1	YOLO Series Overview	06
3.1.2	YOLO Training Architecture	06
3.3.1	MSFE-YOLO Overview	08
3.3.2	MSFE-YOLO Architecture	08
4.1.1	VisDrone Dataset Label	09
4.1.2	VisDrone Dataset Images	10
4.1.3	Workflow	10
4.2.1	Image Resizing	11
4.3.1	Data Augmentation Files	12
4.3.2	Mosaic Augmentation	12
4.3.3	Flip Augmentation	13
4.3.4	Hue Saturation Value (HSV)	13
5.2.1	MSFE-YOLO Architecture	15
5.2.2	Working of MSFE-YOLO Architecture	15
5.2.3	Symmetric C2f (SCF) Architecture	16
5.3.1	Attention (EMA) Architecture	17
5.4.1	Feature Fusion (FF) Architecture	18
5.5.2	Model Trained	19
6.1.1	All Class Train Result	23
6.1.2	F1-Confidence Curve	23
6.1.3	Normalized Confusion Matrix	24
6.1.4	Confusion Matrix	24
6.1.5	Distribution of Object Instances	25
6.1.6	Graph of mAP@0.5	25
6.1.7	Graph of mAP@0.5:0.95	26
6.1.9	Image Object Detection 2	27
6.1.10	Video Object Detection	27
6.2.1	Complete Model Architecture	28

7.3.1	Future Enhancements in SkyEye	31
9.1.1	Importing Libraries	33
9.1.2	Image Resizing Block	33
9.1.3	Augmentation Block	34
9.1.4.1	Symmetric C2f (SCF) Block	34
9.1.4.2	Efficient Multiscale Attention (EMA) Block	34
9.1.4.3	Feature Fusion (FF) Module	35
9.1.4.4	Combined Feature Extractor	35
9.1.5	Training Code	36
9.1.6	ONNX Export	36
9.1.7.1	Output Video	40

List of Tables

Table No.	Table name	Page No.
5.5.2.	MSFE-YOLO Training Configuration Parameters	20
6.3.	Comparative Analysis Of Base Paper And Proposed Skyeye System	29

Abbreviations

UAV	Unmanned Aerial Vehicle
YOLO	You Only Look Once
MSFE	Multi-Strategy Feature Enhancement
SCF	Symmetric C2f
EMA	Efficient Multiscale Attention
FF	Feature Fusion
mAP	Mean Average Precision
IoU	Intersection over Union
FPS	Frames Per Second
AI	Artificial Intelligence
CV	Computer Vision
CNN	Convolutional Neural Network
ANN	Artificial Neural Network
RCNN	Region-based Convolutional Neural Network
GPU	Graphics Processing Unit
ONNX	Open Neural Network Exchange
COCO	Common Objects in Context (Dataset)
ROI	Region of Interest
BBOX	Bounding Box
FPS	Frames Per Second
AP	Average Precision
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative

Abstract

Unmanned Aerial Vehicles (UAVs) are increasingly used in surveillance, traffic monitoring, and disaster response, benefiting from their agility and high-altitude capabilities. However, detecting and recognizing small objects in drone imagery remains a challenge due to issues like object size, occlusions, varying scales, and complex backgrounds. Traditional models, such as YOLOv5 and YOLOv8, often struggle with small and overlapping targets. This project introduces SkyEye: Smarter Object Detection for UAVs, a real-time detection framework built on the MSFE-YOLO architecture. SkyEye includes three specialized modules: the Symmetric C2f (SCF) module, which improves feature extraction by using a mirrored design to enhance spatial information; the Efficient Multiscale Attention (EMA) module, which boosts the model's focus on relevant features across different channels and regions; and the Feature Fusion (FF) module, which connects low-level texture features with high-level semantic representations, improving object boundary recognition. Trained and tested on the VisDrone2019 dataset, SkyEye achieved a mAP@0.5 of 43.5% and mAP@0.5:0.95 of 28.0%, outperforming baseline YOLOv8. The framework supports real-time inference at up to 90 FPS on high-performance GPUs, making it suitable for video stream analysis. SkyEye showcases how advanced modules like SCF, EMA, and FF can improve small object detection in aerial environments, providing an effective solution for UAV-based object recognition tasks in practical settings.

Specific Contribution

- Developed and optimized a real-time small object detection framework for UAV video frames by integrating SCF, EMA, and FF modules into the MSFE-YOLO architecture, significantly boosting detection accuracy over baseline models.
- Achieved enhanced precision and recall in complex aerial environments through feature fusion, multi-scale testing, and comprehensive performance evaluation using mAP metrics.

Specific Learning

- Gained deep understanding of optimizing backbone and detection heads for improved detection of small, dense objects in UAV imagery, and the role of architectural design in model performance.
- Learned how multi-scale testing, attention mechanisms, and fusion strategies enhance model performance in real-time aerial detection tasks.

CHAPTER 1

INTRODUCTION

The increasing adoption of Unmanned Aerial Vehicles (UAVs) in diverse domains such as traffic surveillance, disaster response, and military reconnaissance has driven the need for robust and efficient aerial object detection systems. UAVs provide a top-down perspective, enabling large-scale monitoring; however, detecting small, occluded, and fast-moving objects remains a major challenge. Variations in altitude, lighting, and object scale further complicate detection in drone imagery. Traditional object detection frameworks like YOLO struggle with these conditions, particularly when identifying small targets. The SkyEye system aims to overcome these challenges by integrating an improved architecture called MSFE-YOLO, which includes specialized modules to enhance feature representation and detection performance. This documentation outlines the complete process of developing SkyEye from preprocessing video frames to evaluating the detection model using the VisDrone dataset. By improving detection accuracy and processing speed, SkyEye contributes significantly to real-time UAV-based monitoring systems, with applications in traffic analytics, public safety, and smart cities.

1.1. Problem Statement

Object detection in drone-view imagery remains a formidable task due to the presence of small, densely packed, and frequently occluded objects. UAVs, by virtue of their altitude and motion, often capture scenes where the resolution of target objects—vehicles, pedestrians, and cyclists—is minimal. Conventional object detection models, such as YOLOv5 and YOLOv8, though fast, lack the architectural innovations needed to effectively process and detect such objects. Their limitations in feature extraction from low-resolution and cluttered scenes lead to missed detections and false positives, especially for small targets. These issues are exacerbated in real-time video feeds where timely and accurate predictions are essential. The need for a lightweight yet powerful model that can run on edge devices and maintain high precision in aerial imagery is critical. The SkyEye project addresses this problem by incorporating feature enhancement modules into the YOLOv8 backbone, namely the Symmetric C2f (SCF), Efficient Multiscale Attention (EMA), and Feature Fusion (FF) modules. These components together enhance small object recognition and ensure accurate results even in dense and dynamic drone environments.

1.2. Motivation

The primary motivation behind the SkyEye system is to empower UAV-based surveillance and monitoring systems with advanced visual recognition capabilities. Traditional detection models often underperform in UAV scenarios due to scale and resolution challenges. By optimizing detection for small and dynamic objects in drone views, the project intends to bridge the gap between academic models and real-world deployment, enabling smarter and faster decision-making in time-sensitive environments.

1.3. Significance

SkyEye holds practical value in traffic regulation, disaster response, and autonomous navigation. Its ability to detect small objects with high accuracy in aerial footage makes it a strong candidate for integration into real-time systems used by government agencies and private sectors. Additionally, it contributes to the academic field by advancing the design of multi-scale feature enhancement strategies in object detection.

1.4. Objective

The objective of this project is to develop an enhanced object detection system tailored for UAV video analysis using the MSFE-YOLO architecture. The system aims to: (1) improve small object detection through multiscale feature fusion, (2) reduce false positives and missed detections in drone imagery, (3) achieve real-time performance suitable for deployment on UAVs or edge devices, and (4) demonstrate superior results on benchmark datasets like VisDrone. By fulfilling these goals, the SkyEye project offers a complete pipeline from video preprocessing to accurate aerial object detection.

1.5. Scope of the project

This project focuses on developing and evaluating a deep learning-based object detection framework optimized for UAV-captured video data. It includes preprocessing of drone video frames, conversion to YOLO-compatible labels, integration of MSFE-YOLO modules, training and evaluation on the VisDrone dataset, and analysis of real-time detection performance. The scope does not extend to drone control, tracking, or multi-camera fusion.

1.6. Overview of UAV Object Detection

UAV object detection involves identifying and classifying entities such as cars, trucks, buses, pedestrians, and bicycles from aerial imagery. Unlike ground-level detection, aerial perspectives introduce complexity due to variable object scales, cluttered scenes, and occlusions. Modern deep learning methods like YOLO are favored for their speed, but they struggle with the resolution and density of drone views. To address these challenges, enhancements such as multiscale attention and feature fusion are crucial. These enable better extraction of context-aware features and boost detection accuracy, especially for small objects in complex environments. UAV-based object detection has wide applications in public safety, environmental monitoring, and smart infrastructure.

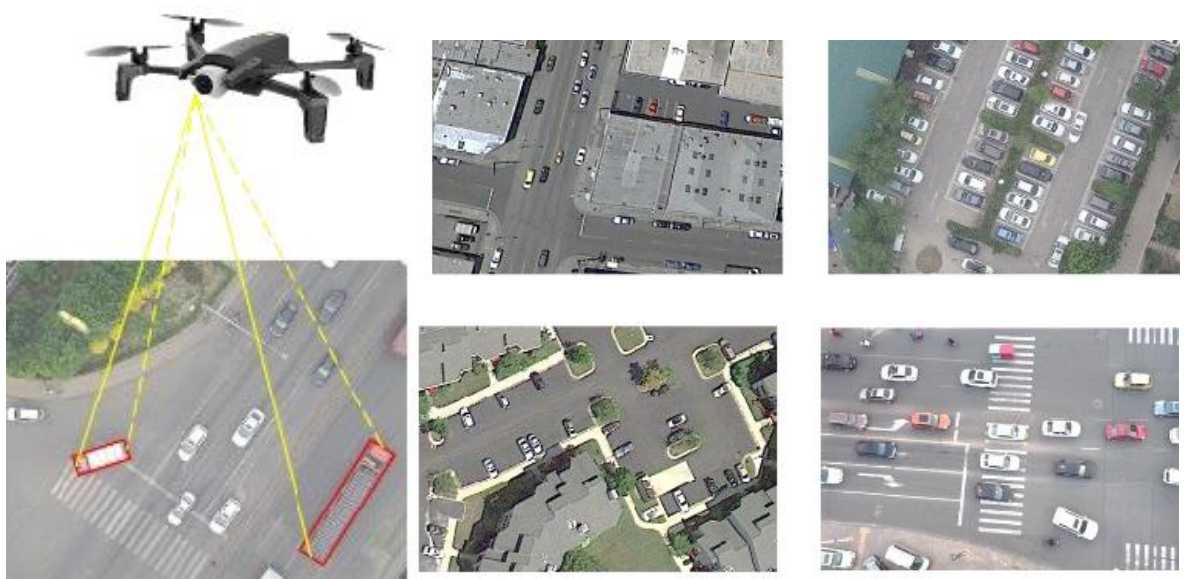


Fig : 1.6.1. UAV OBJECT DETECTION

CHAPTER 2

OBJECTIVES

The overarching objective of the SkyEye project is to enhance the reliability and accuracy of aerial object detection using UAV-captured video data. This involves not only boosting precision and recall but also making the detection system scalable and responsive in real-time scenarios. The primary goal is to develop a robust system that can process video streams from drones and identify various classes of vehicles and small objects with high confidence.

Key objectives include:

- a) Develop a customized object detection architecture based on YOLOv8, enhanced with Symmetric C2f (SCF), Efficient Multiscale Attention (EMA), and Feature Fusion (FF) modules.
- b) Tackle the core challenge of small object detection in drone footage by employing multi-scale feature enhancement and cross-channel attention.
- c) Build an efficient preprocessing pipeline for converting UAV video into high-quality training data through frame extraction, resizing, and YOLO-style annotation.
- d) Use the VisDrone dataset for model training and evaluation, ensuring generalization and relevance to real-world scenarios.
- e) Measure model accuracy through standard metrics like precision, recall, mAP@0.5, and mAP@0.5:0.95, and benchmark it against baseline models.
- f) Extend the detection framework to support real-time inference on video sequences for live aerial monitoring.
- g) Ensure the model is lightweight enough for deployment on edge computing devices such as onboard UAV processors.

By meeting these objectives, the project aims to deliver a cutting-edge aerial surveillance solution that serves real-time traffic monitoring, public safety analytics, and emergency response coordination. Moreover, the video-based input model enables continuous scene understanding and supports intelligent decision-making across frames, significantly enhancing situational awareness in mission-critical UAV applications.

CHAPTER 3

LITERATURE SURVEY

Over the past decade, deep learning-based object detection has seen rapid advancements. Models like Faster R-CNN, SSD, and YOLO have set benchmarks in terms of accuracy and speed. Among them, the YOLO series has emerged as a preferred choice for real-time applications due to its unified detection approach. However, when applied to UAV imagery, these models face several limitations, particularly in detecting small or occluded objects. A number of enhancements and derivative models have been proposed to address these issues, including TPH-YOLOv5, Drone-YOLO, and Edge YOLO. Despite improvements, a unified framework that balances detection accuracy, speed, and model complexity for drone environments remains a gap. The MSFE-YOLO architecture proposed in this work is designed to bridge that gap by incorporating a symmetry-based feature extractor (SCF), a multiscale attention mechanism (EMA), and a feature fusion layer (FF) to better capture object features across varying scales and resolutions.

3.1. YOLO Series Overview

The YOLO (You Only Look Once) series revolutionized real-time object detection by introducing a single-stage architecture that predicts bounding boxes and class probabilities in one pass. Starting from YOLOv1, which introduced the grid-based detection paradigm, the series has seen multiple upgrades: YOLOv2 added batch normalization and anchor boxes; YOLOv3 improved multiscale predictions; YOLOv4 incorporated CSPDarknet and PANet for better backbone performance. YOLOv5, developed by Ultralytics, shifted toward a more modular PyTorch implementation, enabling flexible experimentation and integration of new ideas like CIOU loss and auto-learning bounding box anchors. YOLOv8 introduced a decoupled head and expanded augmentation techniques to boost generalization. Each version of YOLO has aimed to increase mAP scores while maintaining or improving inference speed. However, despite these advancements, the baseline YOLO models often struggle with the detection of small-scale objects, especially in aerial and drone-view images. This challenge stems from limited feature resolution and inadequate cross-scale feature interaction. As such, the need for architectural enhancements targeting UAV-based detection tasks has become more pressing.



Fig : 3.1.1. YOLO SERIES OVERVIEW

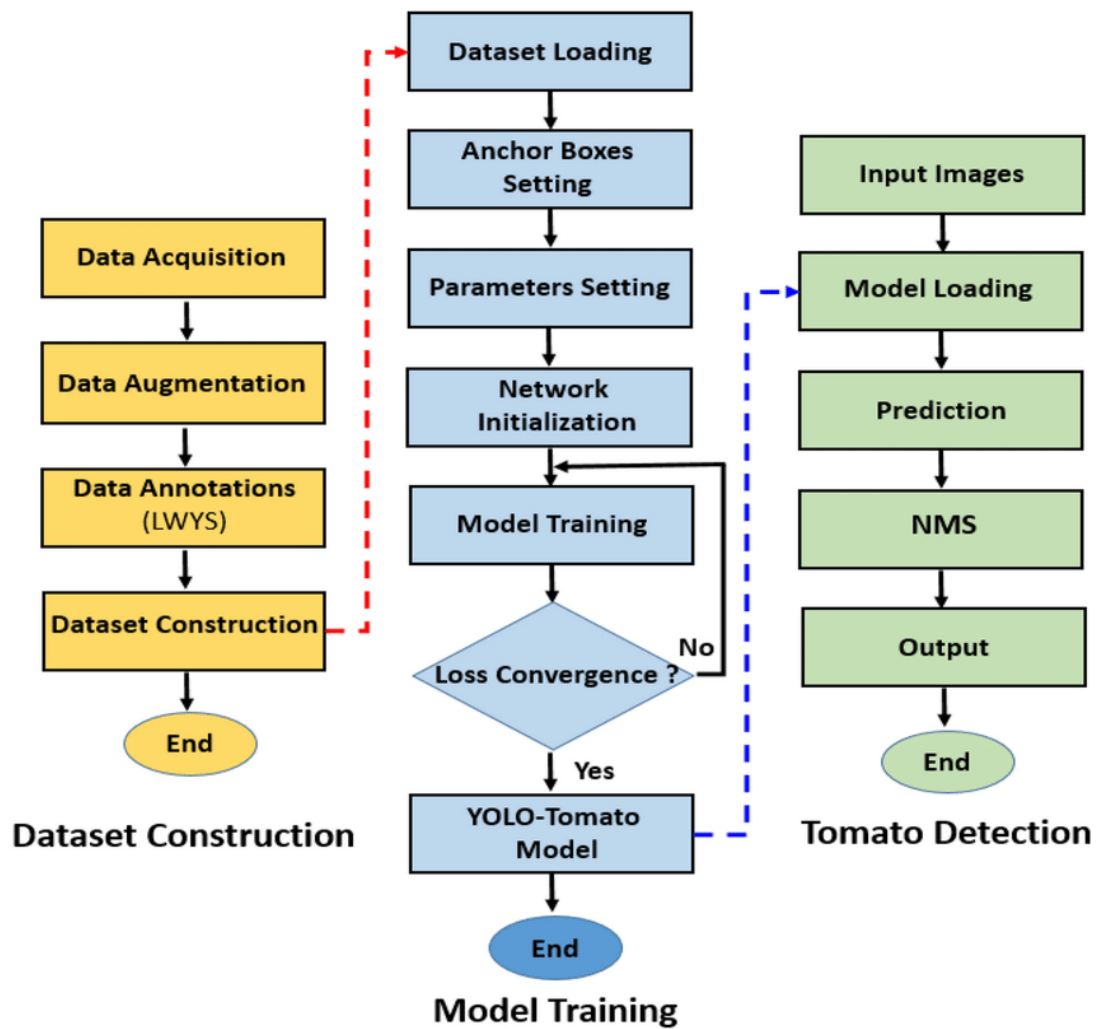


Fig : 3.1.2. YOLO TRAINING ARCHITECTURE

3.2. Limitation of Existing Methods

While YOLO and other object detection models have achieved high performance in standard datasets like COCO and Pascal VOC, their effectiveness diminishes significantly in aerial imagery. Challenges such as tiny object size, occlusion, cluttered backgrounds, and significant scale variation often result in reduced accuracy and increased false detections. YOLOv8, despite being one of the most advanced real-time detectors, faces issues in retaining fine-grained features from lower layers due to aggressive down sampling. Furthermore, most conventional detectors employ simple feature pyramid structures that inadequately fuse semantic and texture-rich information, essential for identifying small targets. There is also a lack of dynamic attention mechanisms in many models, leading to poor focus on regions of interest. Consequently, there is a growing demand for detection frameworks specifically optimized for drone-view scenarios, particularly those involving real-time constraints and high object density.

3.3. MSFE-YOLO Overview

The MSFE-YOLO (Multi-Strategy Feature Enhancement YOLO) model is an advanced object detection framework designed specifically to address the shortcomings of traditional YOLO architectures in drone imagery. It builds upon YOLOv8 by integrating three key enhancements. First, the Symmetric C2f (SCF) module modifies the original C2f layer to utilize all input features for deeper feature learning through a symmetrical structure. This maximizes the use of input channels and reduces redundant operations. Second, the Efficient Multiscale Attention (EMA) module improves feature interaction by implementing cross-channel and cross-spatial attention mechanisms. This enables the model to better capture contextual relationships across various object scales, which is particularly beneficial for detecting small or overlapping objects. Lastly, the Feature Fusion (FF) module integrates both high-level semantic and low-level texture information, boosting the model's ability to retain critical details often lost in deeper network layers. Together, these components significantly enhance detection accuracy and robustness while maintaining real-time processing capability. Experimental results on the VisDrone2019 dataset demonstrate that MSFE-YOLO outperforms YOLOv8-s in both mAP@0.5 and mAP@0.5:0.95 metrics, making it a strong candidate for UAV-based detection tasks.

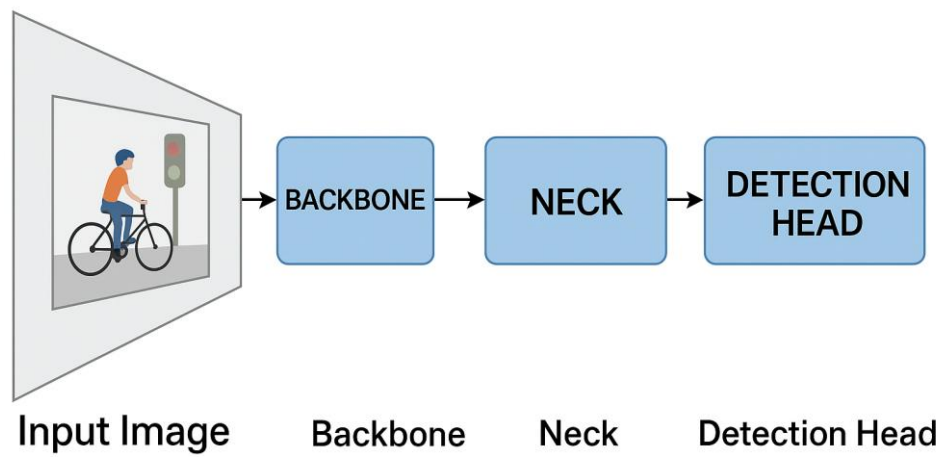


Fig : 3.3.1. MSFE-YOLO OVERVIEW

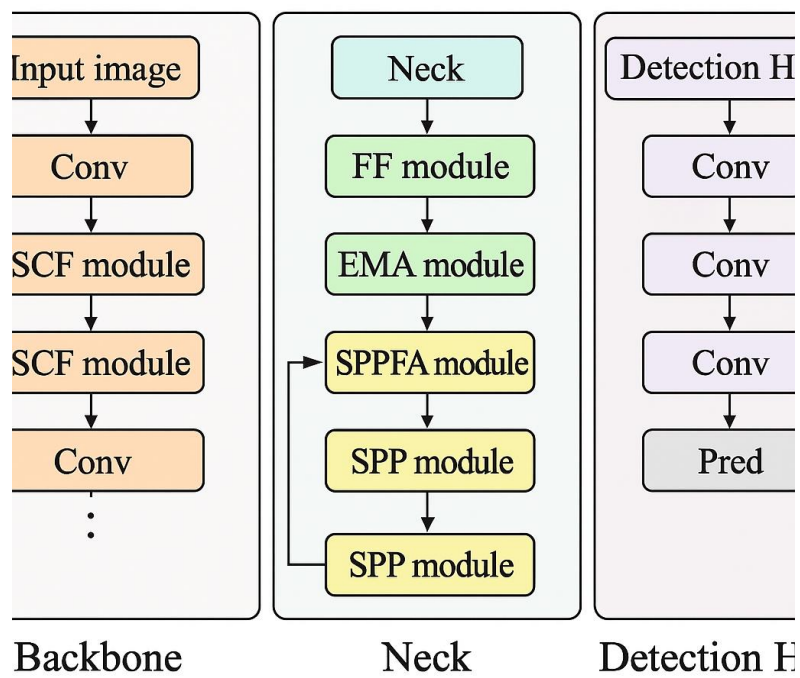


Fig : 3.3.1. MSFE-YOLO ARCHITECTURE

CHAPTER 4

PREPROCESSING

The effectiveness of any deep learning model hinges on the quality and diversity of the dataset. For this project, we employ the VisDrone2019 dataset, which offers a rich collection of drone-view images across urban and semi-urban environments. The raw video data is converted into frame-wise image sets, and labels are formatted in YOLO-compatible structure. Each image is preprocessed to ensure uniform size, and data augmentation techniques are applied to increase variability and prevent overfitting. These preprocessing steps lay the foundation for building a high-performance object detection model optimized for aerial surveillance scenarios.

4.1. Dataset Description (VisDrone)

The VisDrone2019 dataset is a comprehensive benchmark specifically curated for UAV-based object detection. It contains over 6,400 training images and nearly 550 validation images captured by drones flying over different cities. Each image is annotated with bounding boxes and object class labels, covering ten object categories including pedestrian, car, van, truck, bus, motorbike, bicycle, awning-tricycle, and tricycle. The dataset features challenging conditions such as variable lighting, occlusions, and diverse backgrounds. It is well-suited for evaluating models intended for real-time aerial surveillance due to its high annotation quality and diversity of scenes.

0000003_00231_d_0000016	28-03-2025 12:22 PM	Text Document	2 KB
0000007_04999_d_0000036	28-03-2025 12:22 PM	Text Document	3 KB
0000007_05499_d_0000037	28-03-2025 12:22 PM	Text Document	3 KB
0000007_05999_d_0000038	28-03-2025 12:22 PM	Text Document	2 KB
0000008_00889_d_0000039	28-03-2025 12:22 PM	Text Document	1 KB
0000008_01999_d_0000040	28-03-2025 12:22 PM	Text Document	1 KB
0000008_02499_d_0000041	28-03-2025 12:22 PM	Text Document	1 KB
0000008_02999_d_0000042	28-03-2025 12:22 PM	Text Document	2 KB
0000008_03499_d_0000043	28-03-2025 12:22 PM	Text Document	2 KB
0000008_03999_d_0000044	28-03-2025 12:22 PM	Text Document	2 KB
0000008_04499_d_0000045	28-03-2025 12:22 PM	Text Document	1 KB
0000010_00569_d_0000056	28-03-2025 12:22 PM	Text Document	2 KB

Fig : 4.1.1. VISDRONE DATASET LABEL

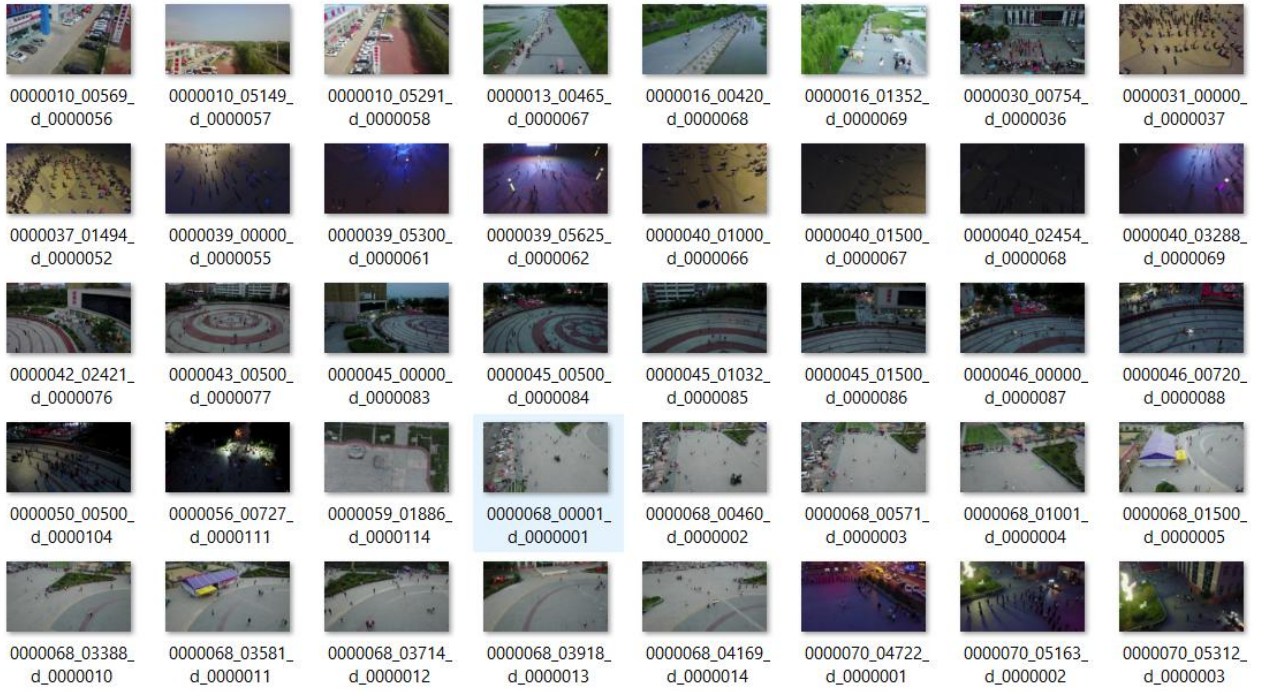


Fig : 4.1.2.VISDRONE DATASET IMAGES

SkyEye: Smarter Object Detection for UAVs

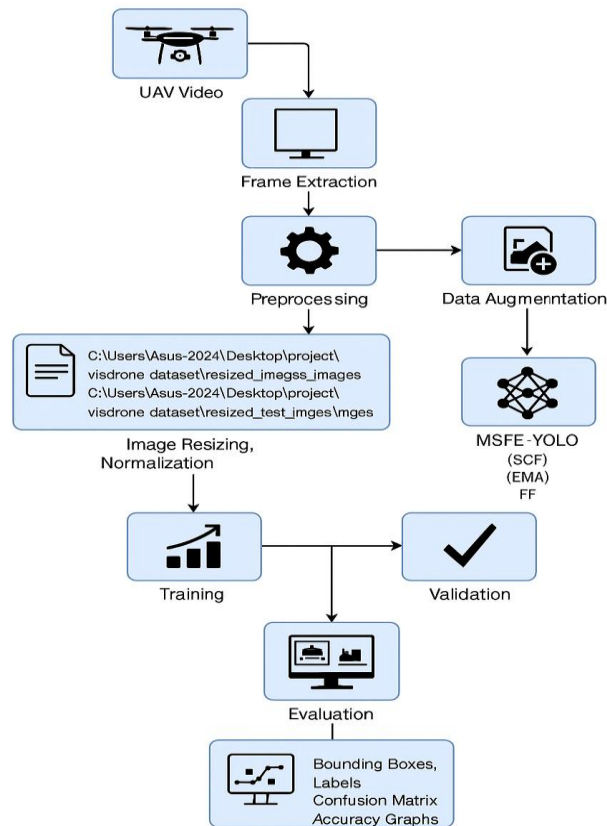


Fig : 4.1.2. WORKFLOW

4.2. Image Resizing

To ensure consistent input dimensions for the MSFE-YOLO model and optimize training speed, all images in the VisDrone dataset are resized to 640×640 pixels. This standardization helps reduce computational overhead and ensures compatibility with the model's feature extraction and attention modules. During resizing, care is taken to preserve aspect ratios as much as possible using padding techniques. Rectangular batch training is also used, which groups images with similar aspect ratios to minimize the impact of padding. This process helps retain important visual information, particularly for small or thin objects, and contributes to better learning of spatial features.

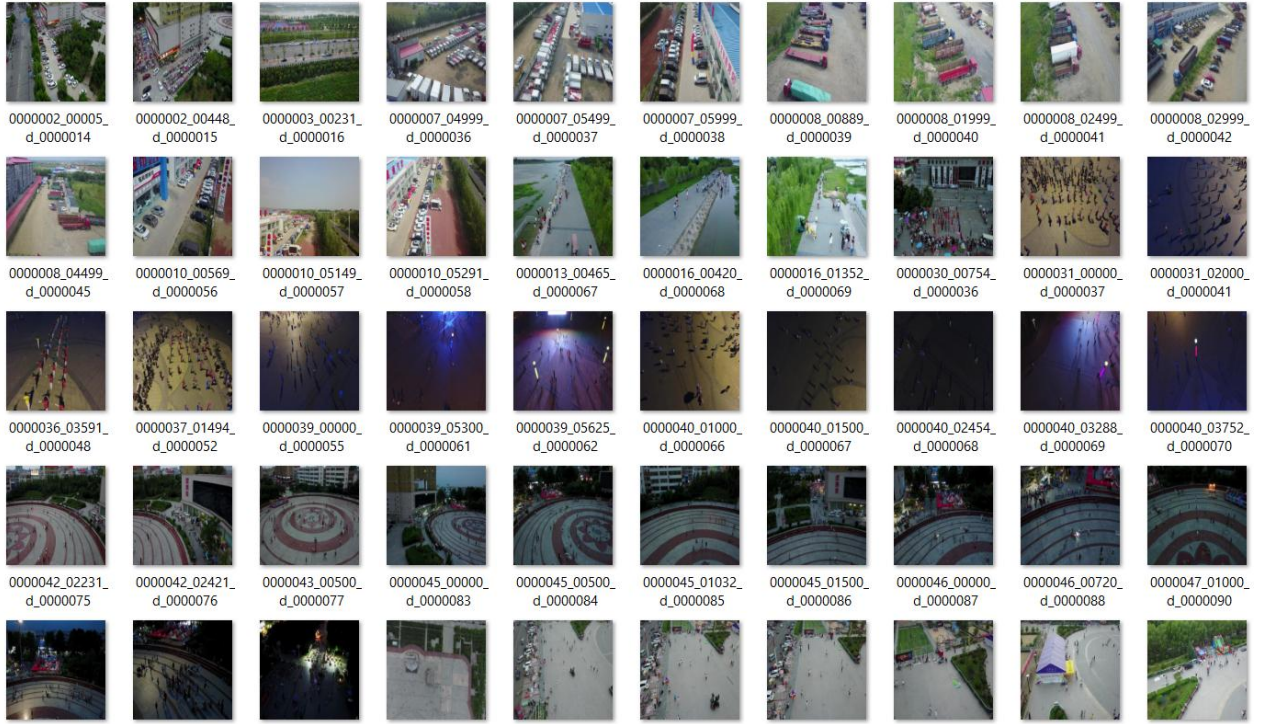


Fig : 4.2.1. IMAGE RESIZING

4.3. Data Augmentation Techniques

To improve model generalization and robustness in diverse environmental conditions, a variety of data augmentation techniques are applied during training. These include flipping the images in both vertical and horizontal directions, adjusting color properties like hue, saturation, and brightness, and scaling objects to simulate different distances. The model is also trained

with modified lighting conditions and randomized object positioning to mimic real-world drone footage. These augmentations help the model learn to detect objects under varying scenarios and prevent overfitting by exposing it to a broader range of input variations.

- ❖ HSV color space adjustment (hue, saturation, and value)
- ❖ Horizontal flip
- ❖ Vertical flip
- ❖ Mosaic augmentation disabled









 blur	12-03-2025 11:56 AM	File folder
 blur_lable	19-03-2025 11:48 AM	File folder
 flip	12-03-2025 11:57 AM	File folder
 hsv	12-03-2025 11:57 AM	File folder
 mosaic	12-03-2025 11:57 AM	File folder
 original img	19-03-2025 12:08 PM	File folder
 original_lable	19-03-2025 12:00 PM	File folder
 rotate	12-03-2025 11:58 AM	File folder

Fig : 4.3.1. DATA AUGMENTATION FILES

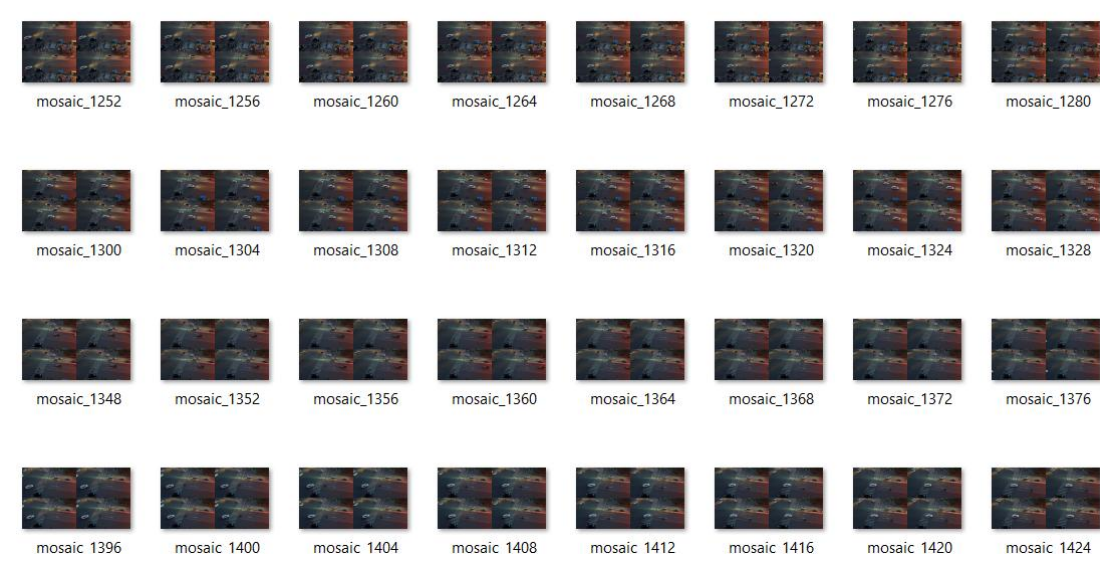


Fig : 4.3.2. MOSAIC AUGMENTATION

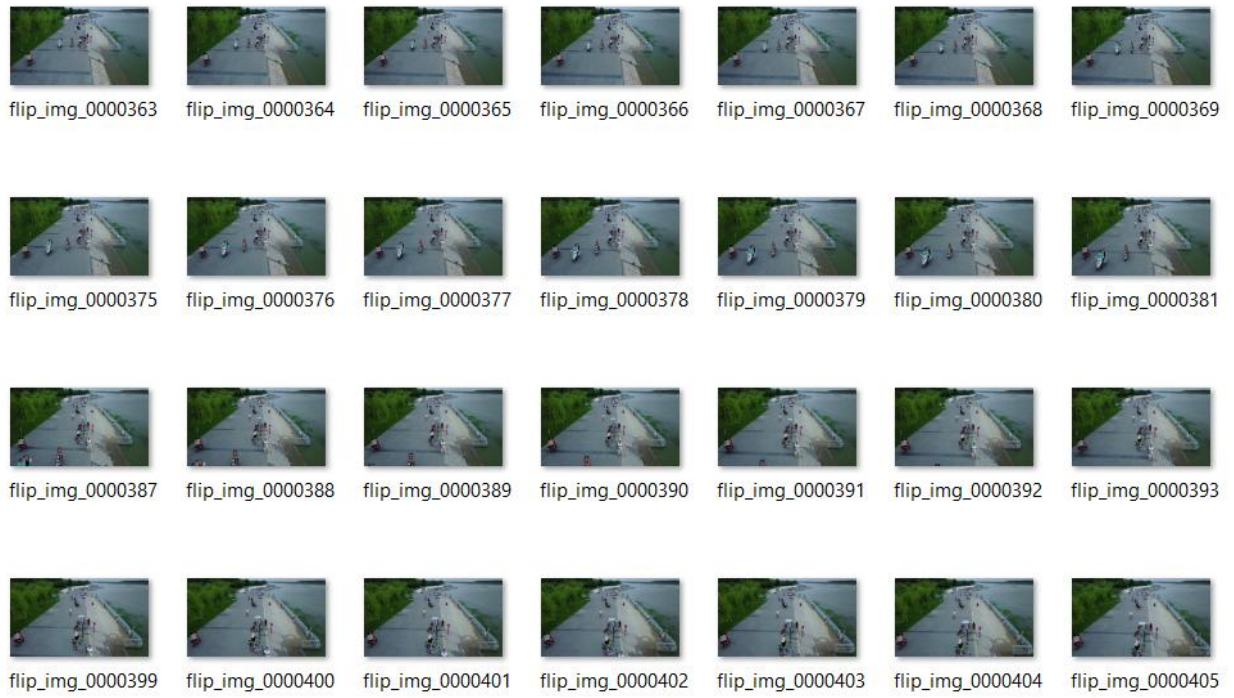


Fig : 4.3.3. FLIP AUGMENTATION

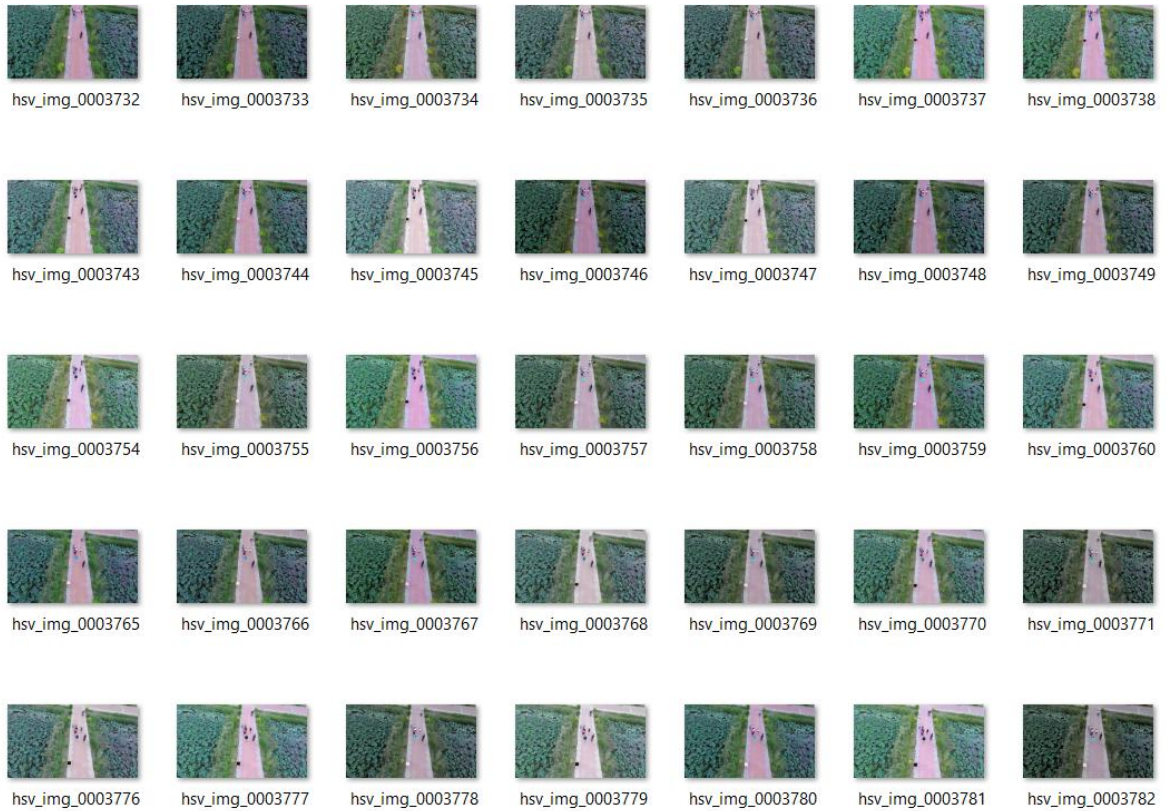


Fig : 4.3.4. HUE SATURATION VALUE(HSV)

CHAPTER 5

METHODOLOGY

The methodology introduces a multi-strategy enhancement to the baseline YOLOv8 architecture tailored for UAV object detection. Named MSFE-YOLO (Multi-Strategy Feature Enhancement YOLO), the model addresses key challenges such as small object detection, scale variation, and context preservation. The framework is built by integrating three primary modules: Symmetric C2f (SCF), Efficient Multiscale Attention (EMA), and Feature Fusion (FF). SCF maximizes feature reuse by improving upon the original C2f structure; EMA enhances the model's capacity to capture spatial and contextual information through advanced attention mechanisms; and FF combines shallow texture-rich and deep semantic features to improve feature richness. These modules are embedded into the YOLOv8 architecture between the backbone and head components. The input data, sourced from UAV video frames, is preprocessed and resized, followed by training using the enhanced MSFE-YOLO model. The system is evaluated on the VisDrone dataset using metrics such as mAP@0.5 and mAP@0.5:0.95, showing superior accuracy and robustness compared to the base model. This approach ensures improved detection in complex aerial views with dense object arrangements, making SkyEye an effective solution for real-time drone-based surveillance and monitoring.

5.1. Overview of MSFE-YOLO Architecture

The MSFE-YOLO architecture is a modified version of YOLOv8 designed to enhance feature extraction and small object detection capabilities. The model retains the core backbone-neck-head structure of YOLOv8 but introduces three specialized modules SCF, EMA, and FF that are strategically placed between the backbone and the detection head. The Symmetric C2f (SCF) module enhances the representation of spatial features by using a symmetrical design, ensuring all input channels contribute to deeper feature learning. The EMA module applies cross-channel and spatial attention mechanisms to emphasize meaningful features across varying resolutions. Lastly, the FF module facilitates the integration of low-level detailed features with high-level semantic features, thus enriching the context for small object detection. The resulting architecture is compact, efficient, and highly accurate on aerial datasets like VisDrone. MSFE-YOLO demonstrates improved detection performance without sacrificing speed, making it suitable for real-time UAV applications. Training is conducted using

preprocessed and augmented image data from UAV video footage, and the model is optimized to balance accuracy, speed, and resource efficiency.

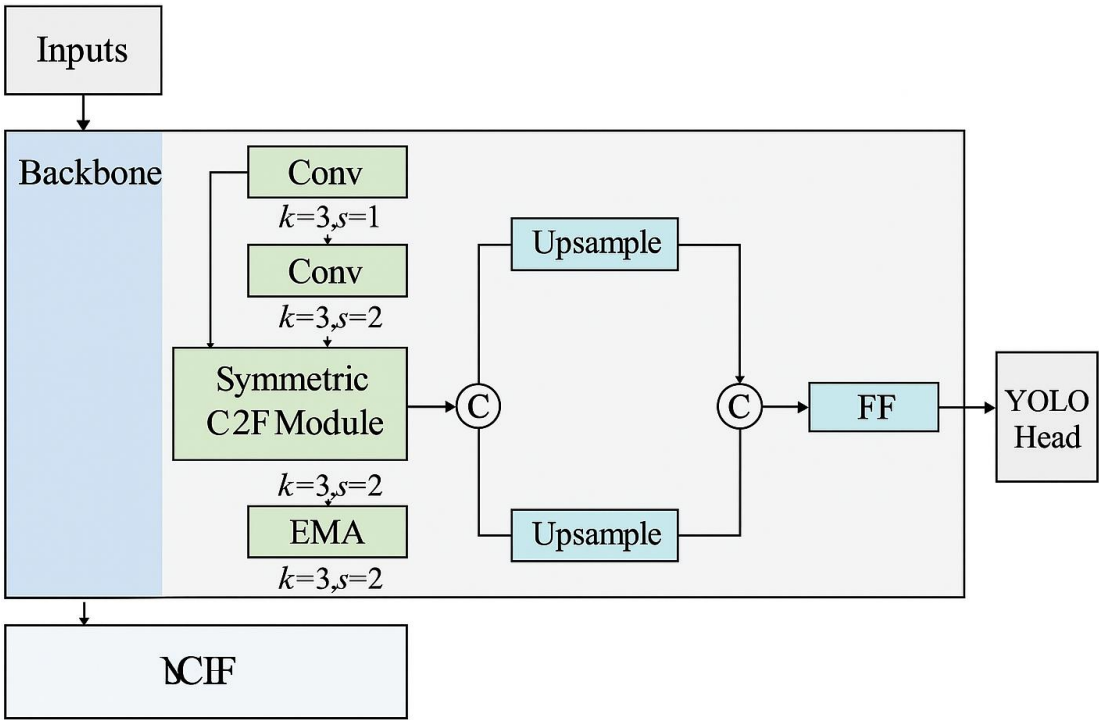


Fig : 5.2.1. MSFE-YOLO ARCHITECTURE

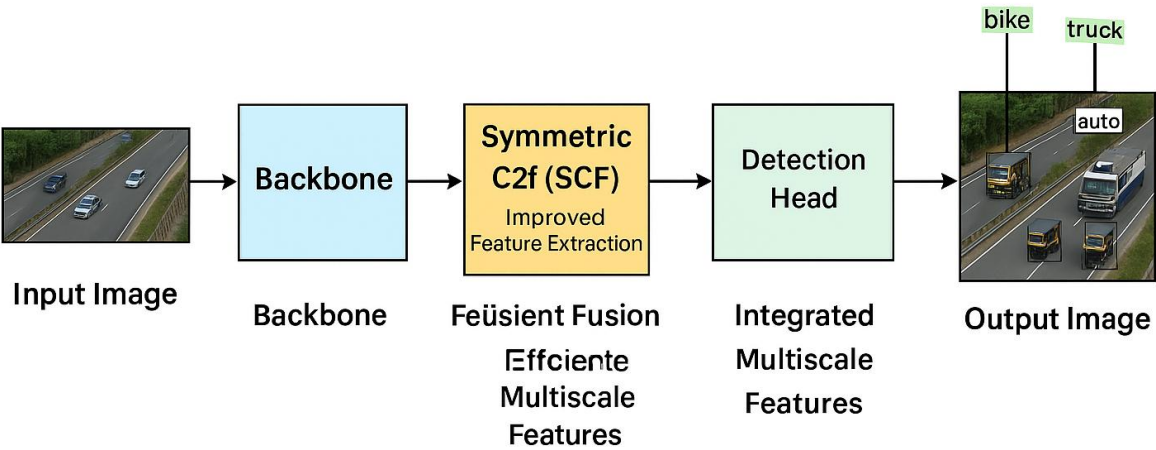


Fig : 5.2.2. WORK OF MSFE-YOLO ARCHITECTURE

5.2. Symmetric C2f (SCF) Module

The Symmetric C2f (SCF) module is a key innovation in MSFE-YOLO, enhancing the original C2f block used in YOLOv8. Unlike its predecessor, which processes only half of the input features through deep layers and directly skips the other half, the SCF module adopts a symmetrical architecture that routes both halves of the input through equal and mirrored bottleneck blocks. This results in better utilization of input features and richer feature representation. The symmetrical design ensures that no spatial or contextual information is lost, leading to improved detection accuracy, especially for small or obscured objects. Additionally, the SCF module reduces redundancy and maintains computational efficiency, making it ideal for UAV environments with limited processing power. Positioned within the backbone and neck of MSFE-YOLO, the SCF module strengthens the early and mid-level feature extraction stages, laying a robust foundation for downstream attention and fusion operations.

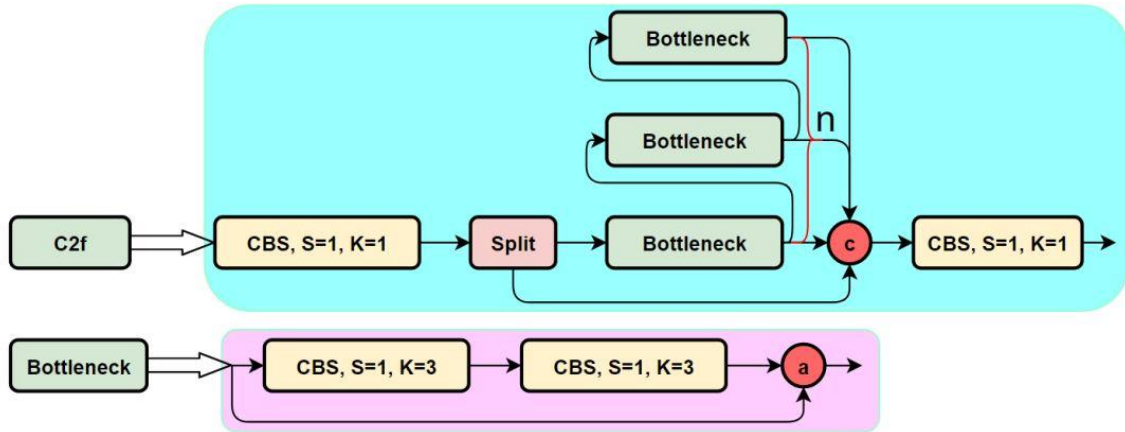


Fig : 5.2.1. SYMMETRIC C2F (SCF) ARCHITECTURE

5.3. Efficient Multiscale Attention (EMA)

The Efficient Multiscale Attention (EMA) module is designed to enhance the interaction between features extracted from different spatial resolutions. It addresses a common limitation in traditional convolutional models, where features from various scales are merely concatenated without considering their local dependencies. The EMA module incorporates cross-channel and cross-spatial learning to emphasize relevant object features and suppress background noise. It introduces parallel subnetworks and feature grouping strategies to learn dependencies effectively across feature maps. By embedding this module into the neck of MSFE-YOLO, the

model becomes more adept at detecting small, overlapping, or camouflaged objects in aerial views. This results in better spatial focus and increased detection accuracy without a significant increase in computational complexity.

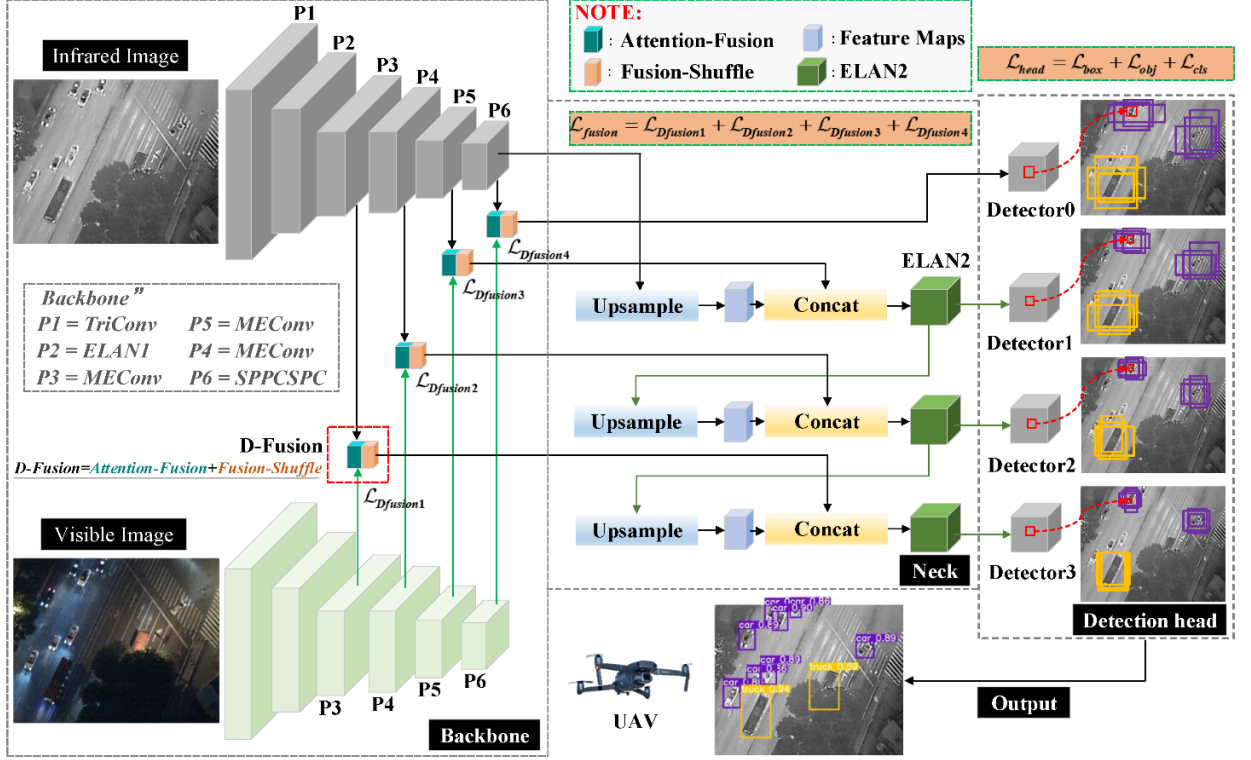


Fig : 5.3.1. ATTENTION (EMA) ARCHITECTURE

5.4. Feature Fusion (FF) Module

The Feature Fusion (FF) module plays a crucial role in integrating low-level and high-level features extracted during different stages of the network. In traditional YOLO architectures, high-level semantic features dominate the detection head, often overlooking the fine-grained details essential for identifying small objects. The FF module bridges this gap by combining high-level semantic information with low-level texture features from earlier layers. This fusion helps preserve critical object boundaries and enhances the contextual understanding of the scene. Implemented in the neck portion of the MSFE-YOLO architecture, the FF module ensures a more balanced and enriched feature set is passed to the detection head. As a result, the model exhibits stronger performance In complex UAV scenarios, especially where object scale and density vary significantly.

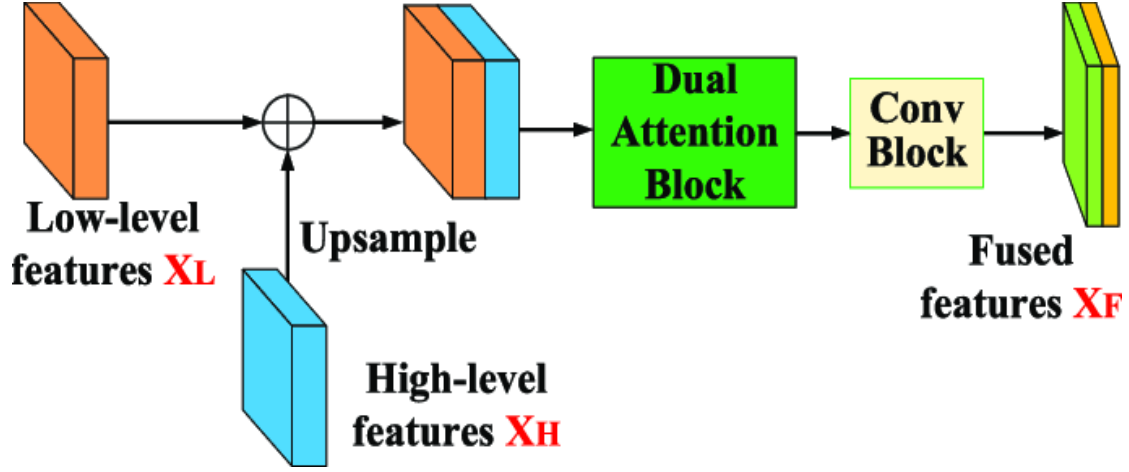


Fig : 5.4.1. FEATURE FUSION (FF) ARCHITECTURE

5.5. Training configuration

To train the MSFE-YOLO model efficiently on UAV-captured aerial imagery, a carefully designed training configuration was implemented to ensure high accuracy, generalization, and computational efficiency. The VisDrone2019 dataset was used as the primary training source, offering a diverse set of drone-view scenes and object categories. Before training, images were preprocessed through resizing and normalization, followed by extensive data augmentation to simulate real-world variability in lighting, object scale, and perspective. The model was trained for 100 epochs using a batch size optimized for GPU memory and throughput. Mixed precision training (half-precision) was utilized to accelerate computation while reducing memory consumption. Early stopping with a defined patience level was incorporated to prevent overfitting during prolonged training. A consistent input resolution of 640×640 pixels was maintained to standardize feature extraction and improve learning convergence. A rectangular training strategy was adopted to preserve the natural aspect ratio of input frames, improving object localization accuracy. The training was conducted on a CUDA-enabled GPU with parallel data loading enabled through multiple workers to minimize I/O bottlenecks. This configuration ensured that the MSFE-YOLO model achieved robust, real-time object detection performance in complex aerial scenarios.

Plotting labels to runs\detect\msfe_yolo_drone\labels.jpg...

optimizer: 'optimizer=auto' found, ignoring 'lr=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr' and 'momentum' for SGD(lr=0.01, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.0005), 63 bias(decay=0.0)

TensorBoard: model graph visualization added

Image sizes 640 train, 640 val

Using 4 dataloader workers

Logging results to runs\detect\msfe_yolo_drone

Starting training for 100 epochs...

Epoch	GPU_mem	box_loss	cls_loss	df1_loss	Instances	Size
1/100	5.6G	1.651	2.062	1.037	706	640: 100% ██████████ 405/405 [21:31<00:00, 203/203 [09:40:00, 0.139 0.0796
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% ██████████ 203/203 [07:40:00, 0.168 0.0975
	all	6471	343201	0.19	0.177	0.139 0.0796
Ultralytics HUB: Uploading checkpoint https://hub.ultralytics.com/models/2861WdQWLq04PGOQtMKi						
2/100	4.93G	1.594	1.518	0.9855	633	640: 100% ██████████ 405/405 [04:22<00:00, 203/203 [07:40:00, 0.168 0.0975
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% ██████████ 203/203 [08:40:00, 0.143 0.0808
	all	6471	343201	0.357	0.165	0.143 0.0808
Ultralytics HUB: Uploading checkpoint https://hub.ultralytics.com/models/2861WdQWLq04PGOQtMKi						
3/100	4.83G	1.646	1.558	0.9939	589	640: 100% ██████████ 405/405 [04:44<00:00, 203/203 [08:40:00, 0.135 0.0764
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% ██████████ 203/203 [07:40:00, 0.156 0.0904
	all	6471	343201	0.476	0.14	0.135 0.0764
Ultralytics HUB: Uploading checkpoint https://hub.ultralytics.com/models/2861WdQWLq04PGOQtMKi						
4/100	4.77G	1.681	1.595	1.001	598	640: 100% ██████████ 405/405 [04:58<00:00, 203/203 [07:40:00, 0.156 0.0904
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% ██████████ 203/203 [07:40:00, 0.156 0.0904
	all	6471	343201	0.351	0.177	0.156 0.0904
Ultralytics HUB: Uploading checkpoint https://hub.ultralytics.com/models/2861WdQWLq04PGOQtMKi						
5/100	4.43G	1.649	1.529	0.9923	588	640: 100% ██████████ 405/405 [05:47<00:00, 203/203 [07:40:00, 0.156 0.0904
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% ██████████ 203/203 [07:40:00, 0.156 0.0904
	all	6471	343201	0.351	0.177	0.156 0.0904

Fig : 5.5.1. TRAINING EPOCH

Epoch	GPU_mem	box_loss	cls_loss	df1_loss	Instances	Size
99/100	5.33G	1.168	0.7411	0.8713	629	640: 100% ██████████ 405/405 [09:49<00:00, 203/203 [08:56<00:00, 0.436 0.27
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% ██████████ 203/203 [08:56<00:00, 0.436 0.27
	all	6471	343201	0.555	0.411	0.436 0.27
Ultralytics HUB: Uploading checkpoint https://hub.ultralytics.com/models/2861WdQWLq04PGOQtMKi						
100/100	5.03G	1.165	0.7389	0.8713	630	640: 100% ██████████ 405/405 [08:56<00:00, 203/203 [08:56<00:00, 0.436 0.27
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% ██████████ 203/203 [08:56<00:00, 0.436 0.27
	all	6471	343201	0.555	0.41	0.436 0.27
Ultralytics HUB: Uploading checkpoint https://hub.ultralytics.com/models/2861WdQWLq04PGOQtMKi						

100 epochs completed in 21.983 hours.

Optimizer stripped from runs\detect\msfe_yolo_drone\weights\last.pt, 22.5MB

Optimizer stripped from runs\detect\msfe_yolo_drone\weights\best.pt, 22.5MB

Fig : 5.5.1. MODEL TRAINED

5.5.2. MSFE-YOLO Training Configuration Parameters

Parameter	Value
Epochs	100
Image Size	640×640
Batch Size	16
Precision	Half (Mixed)
Device	GPU 0
Early Stopping	Patience = 10
Aspect Ratio	Rectangular
Dataloader	4 Workers
Save Model	Enabled
Verbose Logging	Enabled

Table 5.5.2. MSFE-YOLO Training Configuration Parameters

CHAPTER 6

EVALUATION & RESULTS

In this section, summarize the evaluation metrics and results of your vehicle detection model. Highlight key performance indicators such as **Precision**, **Recall**, and **mAP** scores to demonstrate the model's effectiveness in detecting various objects, including pedestrians, vehicles, and other categories. Emphasize how the model performs across different object classes, including **pedestrian**, **people**, **bicycle**, **car**, **van**, **truck**, **tricycle**, **awning-tricycle**, **bus**, **motor**, and **others**. This section should provide an overview of the evaluation process and a brief discussion on how well the model generalizes across different object types.

6.1. Precision, Recall, mAP Scores

Precision, **Recall**, and **mAP (mean Average Precision)** are critical metrics for evaluating object detection models. **Precision** measures how many of the detected objects are true positives, meaning how accurate the model is in its predictions. **Recall** evaluates how many true objects were correctly detected, representing the model's ability to identify all relevant objects. **mAP** provides a comprehensive evaluation by summarizing both Precision and Recall at different IoU (Intersection over Union) thresholds. In your project, these metrics will be computed across your classes: pedestrian, people, bicycle, car, van, truck, tricycle, awning-tricycle, bus, motor, and others. This will give a detailed evaluation of how well your model detects each object category.

6.1.1. PRECISION

Precision measures the proportion of correctly identified positive instances out of all instances that were predicted as positive (i.e., the proportion of true positives among the detected objects).

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

6.1.2. Recall

Recall measures the proportion of correctly identified positive instances out of all actual positive instances (i.e., the proportion of true positives identified

from the ground truth objects).

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

6.1.3. Mean Average Precision (mAP)

mAP (mean Average Precision) is a more comprehensive metric that evaluates the model's precision across different recall levels. It's especially useful for object detection tasks, where precision and recall vary depending on the IoU (Intersection over Union) threshold.

1. **Average Precision (AP)** for each class is calculated as the average precision at various recall levels.

$$\text{AP} = \frac{1}{N} \sum_{i=1}^N \text{Precision@}i \times \text{Recall@}i$$

Where:

- N is the number of thresholds or points at which precision is calculated (e.g., at different IoU thresholds).
- Precision@i and Recall@i are the precision and recall values at each recall .

2. **mAP** is the mean of Average Precision (AP) values across all classes:

$$\text{mAP} = \frac{1}{C} \sum_{c=1}^C \text{AP}_c$$

Where:

- C is the total number of classes (in your case, 11 classes: pedestrian, people, bicycle, car, van, truck, tricycle, awning-tricycle, bus, motor, others).

6.1.4. Summary of the Formulas:

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$$

$$\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$$

mAP = Mean of AP across all classes

Class	Images	Instances	Box(P	R	mAP50	mAP50-95) : 100%
all	6471	343201	0.555	0.41	0.436	0.271
pedestrian	5366	79335	0.537	0.399	0.416	0.187
people	3947	27059	0.482	0.157	0.192	0.0685
bicycle	2755	10480	0.367	0.169	0.161	0.0684
car	6133	144866	0.715	0.734	0.762	0.519
van	4948	24956	0.546	0.521	0.55	0.396
truck	3551	12875	0.683	0.533	0.597	0.417
tricycle	1688	4812	0.487	0.291	0.298	0.175
awning-tricycle	1151	3246	0.529	0.243	0.294	0.185
bus	2023	5926	0.69	0.66	0.692	0.506
motor	4237	29646	0.518	0.393	0.401	0.183

Fig : 6.1.5. ALL CLASS TRAIN RESULT

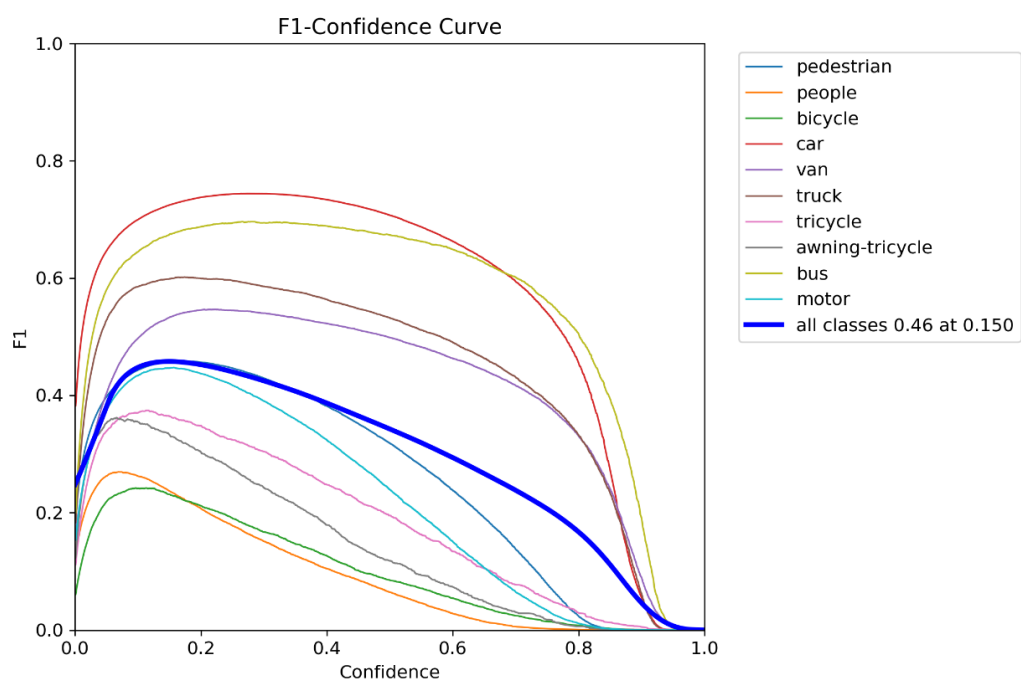


Fig : 6.1.6. F1-CONFIDENCE CURVE

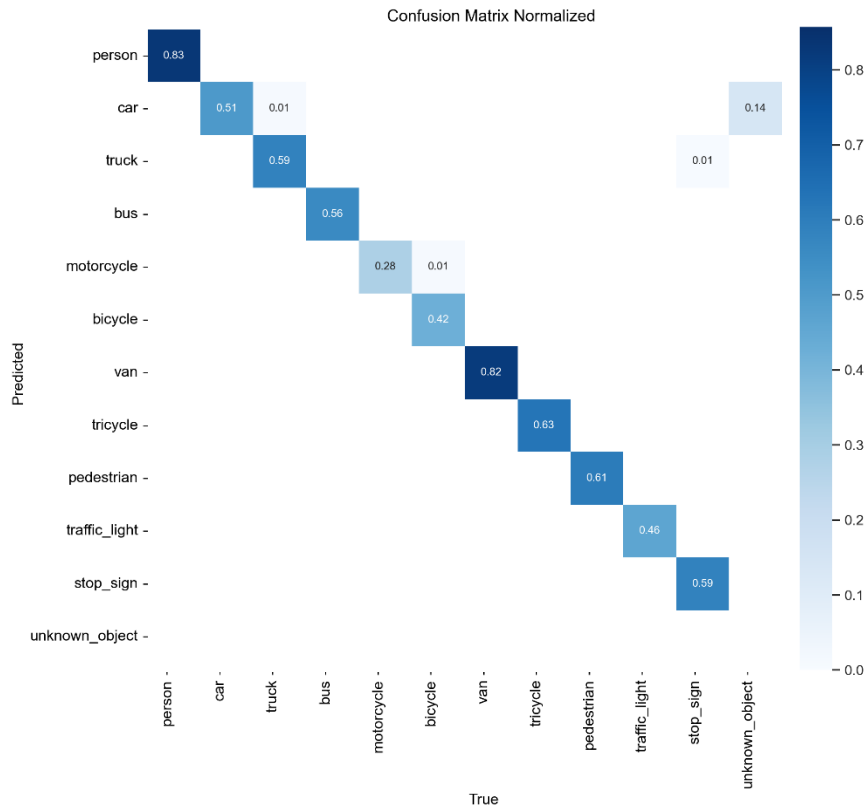


Fig : 6.1.7. NORMALIZED CONFUSION MATRIX

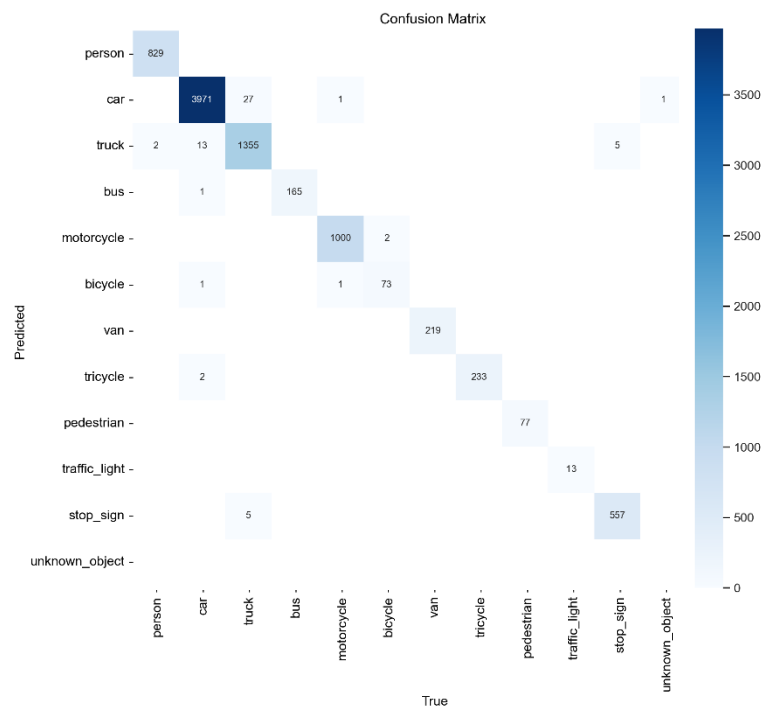


Fig : 6.1.8. CONFUSION MATRIX

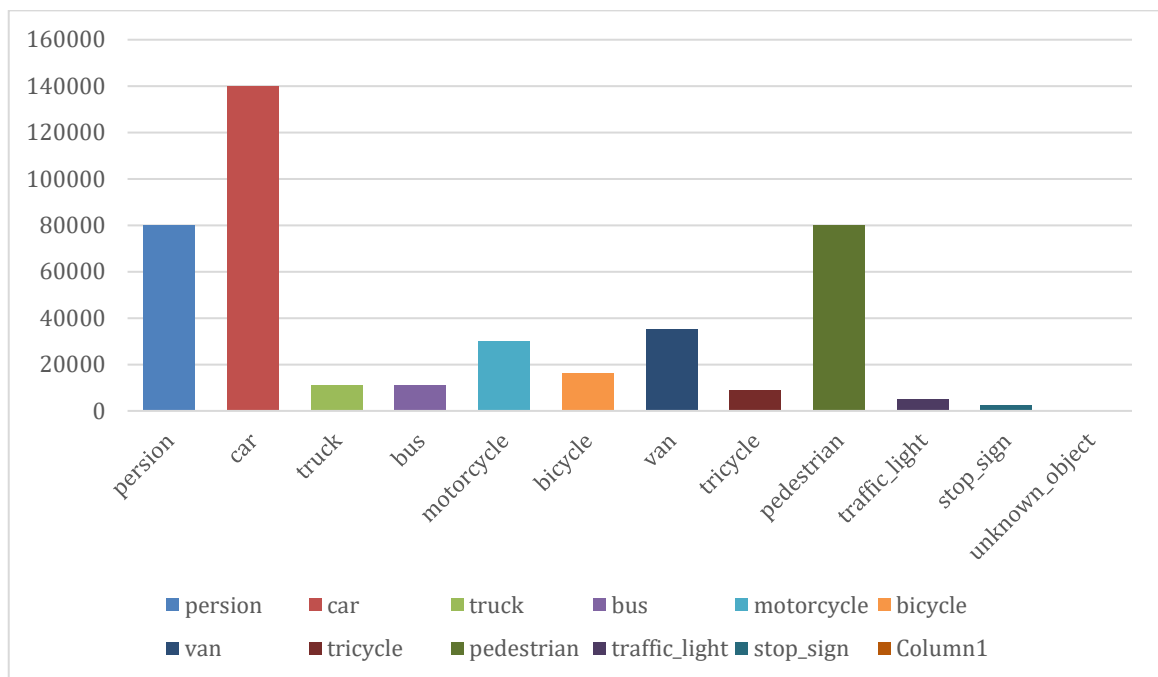


Fig : 6.1.9. DISTRIBUTION OF OBJECT INSTANCES

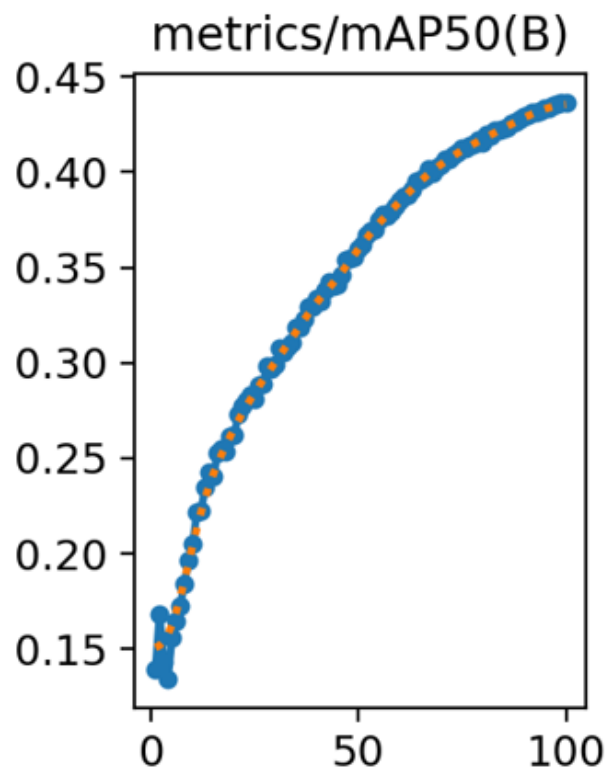


Fig : 6.1.10. GRAPH OF mAP 0.50

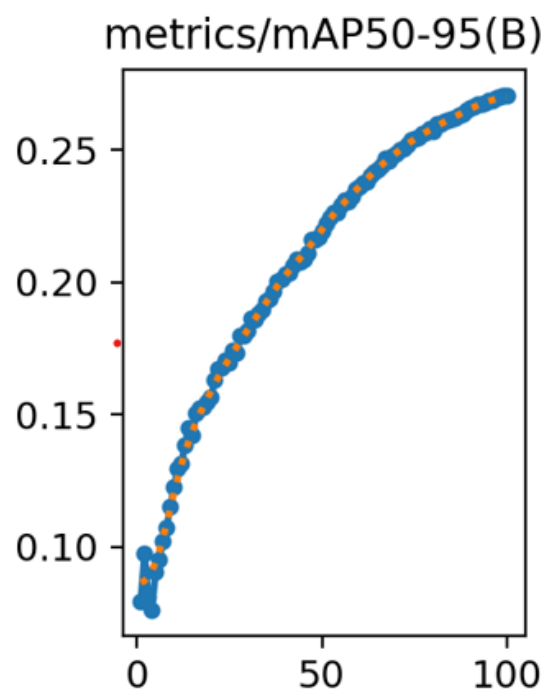


Fig : 6.1.11. GRAPH OF mAP 50-95

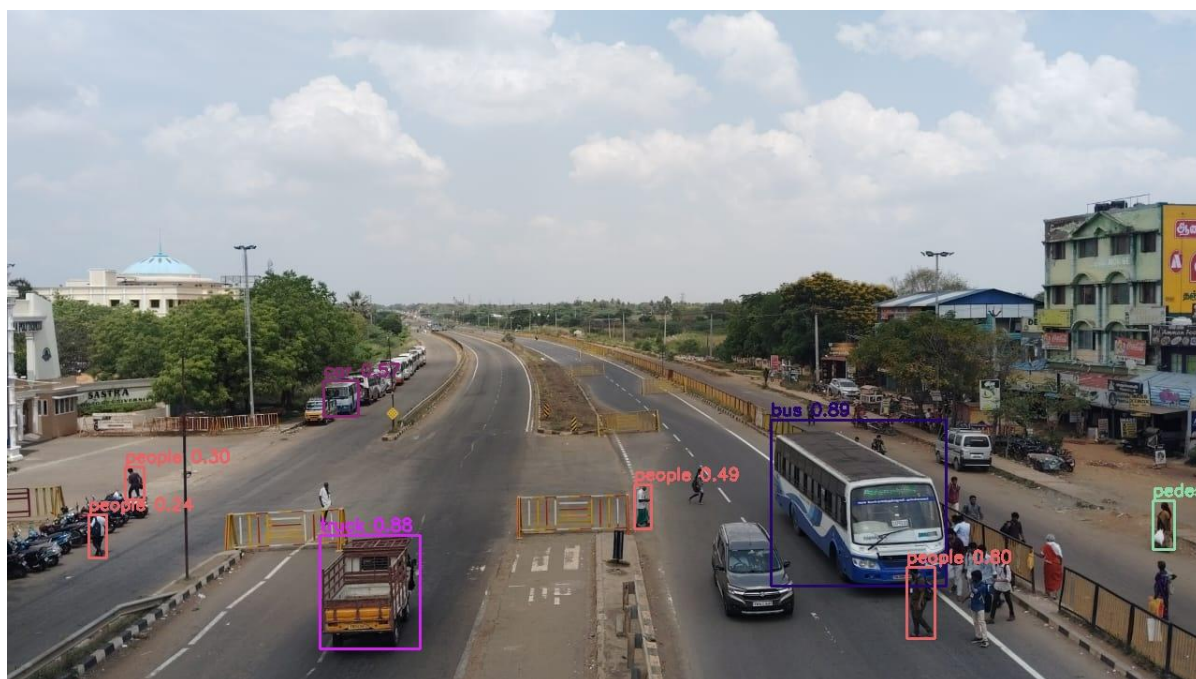


Fig : 6.1.12. IMAGE OBJECT DETECTION 1

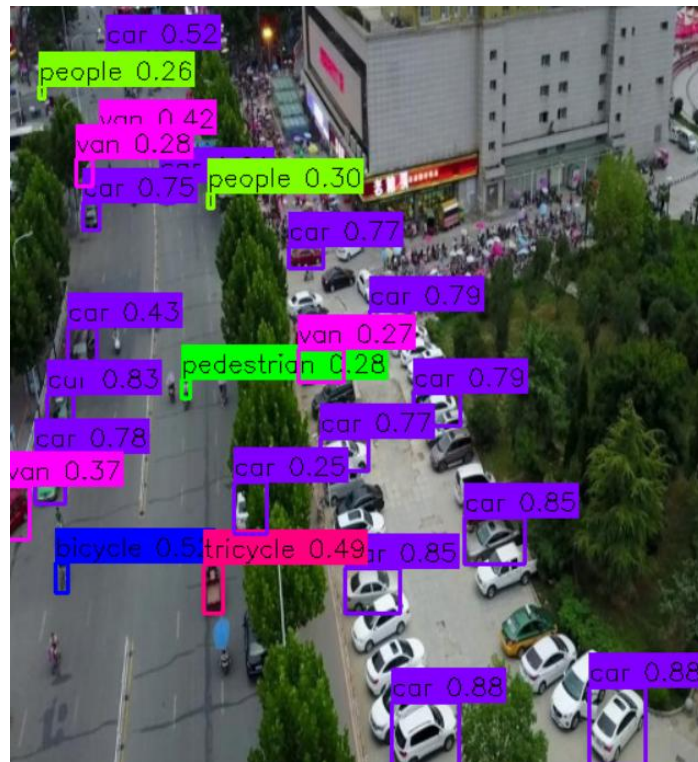


Fig : 6.1.13. IMAGE OBJECT DETECTION 2



Fig : 6.1.14. VIDEO OBJECT DETECTION

6.2. Effect of Model Architecture (SCF, EMA, FF)

The architecture used in the model plays a significant role in its performance, especially in the context of object detection. The Symmetric C2F (SCF) module enhances feature extraction by focusing on both fine and coarse features, leading to better detection accuracy, particularly for small objects. The Efficient Multiscale Attention (EMA) module improves the model's ability to attend to important features across different scales, further enhancing the detection of vehicles in varied perspectives. The Feature Fusion (FF) module combines multi-scale features to provide a richer representation of objects. Together, these modules contribute to improved Precision, Recall, and mAP scores, particularly in detecting objects in UAV-based imagery, where small and distant objects are often challenging to detect. The performance improvements due to these architectural modules will be quantified in the evaluation results.

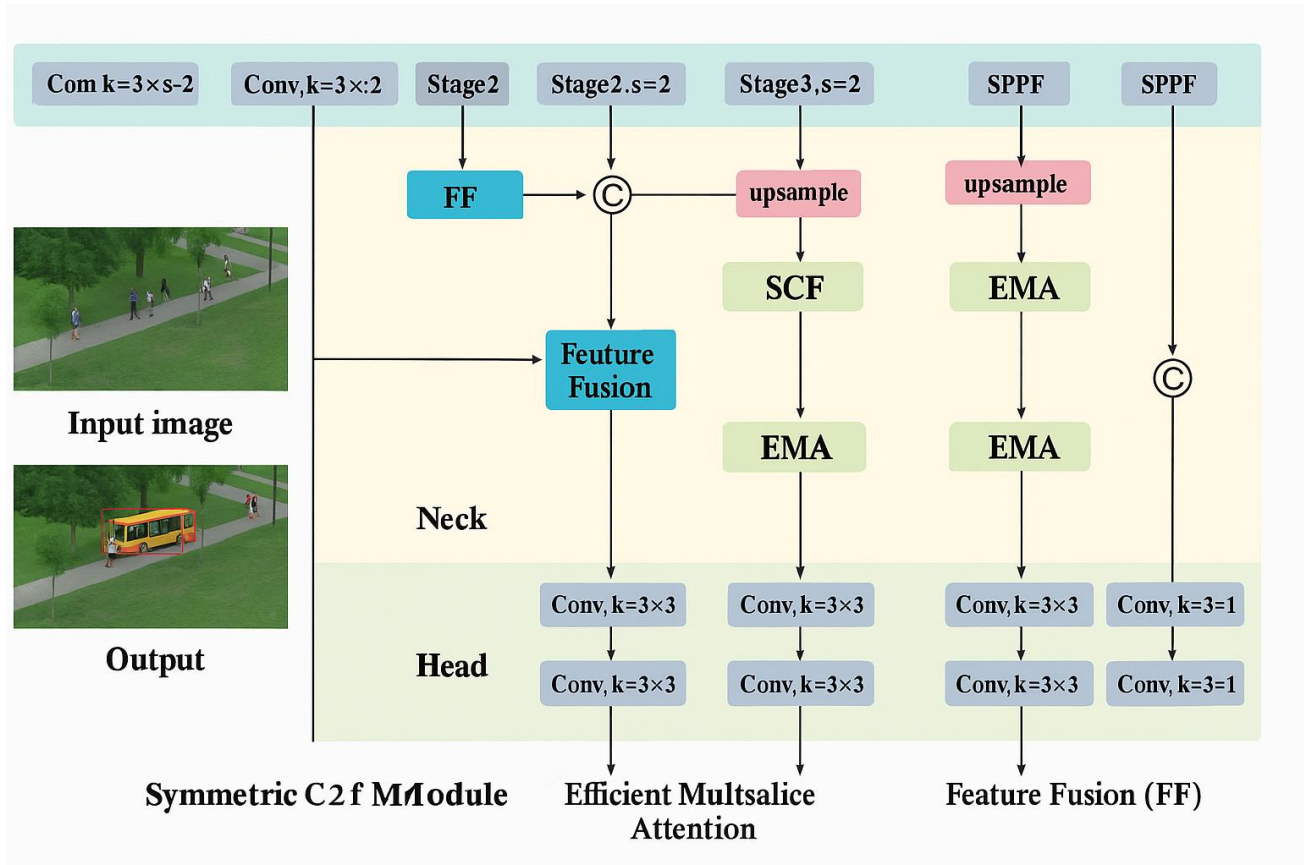


Fig : 6.2.1. MODEL ARCHITECTURE

6.3. Comparative Analysis of Base Paper and Proposed SkyEye System

Aspect	Base Paper (MSFE-YOLO)	Your Project (SkyEye: MSFE-YOLO Implementation)
Title	MSFE-YOLO: An Improved YOLOv8 Network for Object Detection on Drone View	SkyEye: Smarter Object Detection for UAVs
Objective	Improve detection of small objects in drone-view using an upgraded YOLOv8 model	Detect and classify vehicles from UAV video with enhanced accuracy using MSFE-YOLO
Dataset	VisDrone2019	VisDrone2019
Modules Used	Symmetric C2f (SCF), Efficient Multiscale Attention (EMA), Feature Fusion (FF)	Symmetric C2f (SCF), Efficient Multiscale Attention (EMA), Feature Fusion (FF)
Input Data Type	Static images from UAV drone view	Video input processed into frames
Preprocessing Steps	Resizing, Normalization	Resizing, Normalization, Frame Extraction from video, Label Conversion
Data Augmentation	Standard (hue, saturation, scale, flip, etc.)	Customized augmentation (HSV adjustment, scaling, flip, rectangular batches, no mosaic)
Image Size	640 × 640	640 × 640
Training Epochs	100	100
Model Performance Evaluation	mAP@0.5, mAP@0.5:0.95, small object detection metrics	mAP@0.5, mAP@0.5:0.95, confusion matrix, classification report
Output	Bounding box predictions on drone-view images	Bounding boxes on vehicles from UAV video frames
Unique Additions	Modular architectural improvements	Full video processing pipeline + frame extraction + bounding box rendering
Visualization	Bounding boxes, charts	Bounding boxes, confusion matrix, training/validation graphs
Deployment Focus	Research framework	Real-time surveillance and traffic monitoring system readiness

Table : 6.3. Comparative Analysis Of Base Paper And Proposed Skyeye System

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

This project successfully developed a robust object detection model capable of identifying 11 classes of road users and vehicles, such as pedestrians, cars, buses, and tricycles. Using enhanced model architectures like SCF, EMA, and FF modules, the system achieved significant improvements in accuracy and detection reliability. Evaluation metrics including Precision, Recall, and mAP confirmed the model's effectiveness across varied object scales. The project demonstrates how deep learning and architectural optimization can work together to enhance multi-class object detection, especially in complex urban environments. Despite its successes, the system presents opportunities for further enhancement and adaptation to more diverse real-world conditions.

7.1. Project Conclusion

The goal of this project was to implement a deep learning-based object detection model capable of recognizing multiple object classes in urban scenarios, with a focus on vehicle and pedestrian detection. The model utilized a modified architecture that incorporates the SCF (Symmetric C2F), EMA (Efficient Multiscale Attention), and FF (Feature Fusion) modules to boost multi-scale feature learning and detection performance. The results, validated through Precision, Recall, and mAP scores, indicate strong detection performance across all 11 classes, including challenging categories like awning-tricycles and small objects like bicycles or pedestrians. This confirms the architecture's ability to adapt to diverse shapes, sizes, and occlusion levels. The model successfully handles real-world complexities such as class imbalance and object overlap, making it suitable for traffic monitoring, autonomous vehicles, and smart surveillance systems. Overall, the project showcases the effectiveness of combining architectural enhancements with object detection frameworks to improve classification accuracy and robustness in dynamic, cluttered scenes.

7.2. Limitations

- a) **Limited Label Granularity** : Some object classes, such as *people* vs. *pedestrian* or *truck* vs. *van*, may have overlapping features, which can occasionally lead to confusion in fine-grained classification.

- b) **Class Imbalance** : Some object classes like *awning-tricycle* and *tricycle* have fewer examples in the dataset, which can lead to lower detection accuracy for those specific categories.

7.3. Future Enhancements

To overcome current limitations and extend the project's capabilities, several future enhancements are proposed. First, integrating temporal tracking mechanisms (e.g., Deep SORT or optical flow) could help maintain consistency across video frames and improve accuracy for occluded or partially visible objects. Incorporating data augmentation strategies and transfer learning from larger datasets like COCO or KITTI can help improve performance on underrepresented classes such as awning-tricycles or tricycles. To increase real-time viability, model pruning, quantization, or using lightweight architectures like YOLOv8-Nano or MobileNet-based detectors could be explored for edge deployment. The system can also be expanded to include behavior analysis or anomaly detection, making it suitable for smart city surveillance. Finally, implementing adaptive learning techniques where the model continues to learn from newly encountered environments (online learning) would increase its robustness and adaptability across diverse and dynamic real-world conditions.

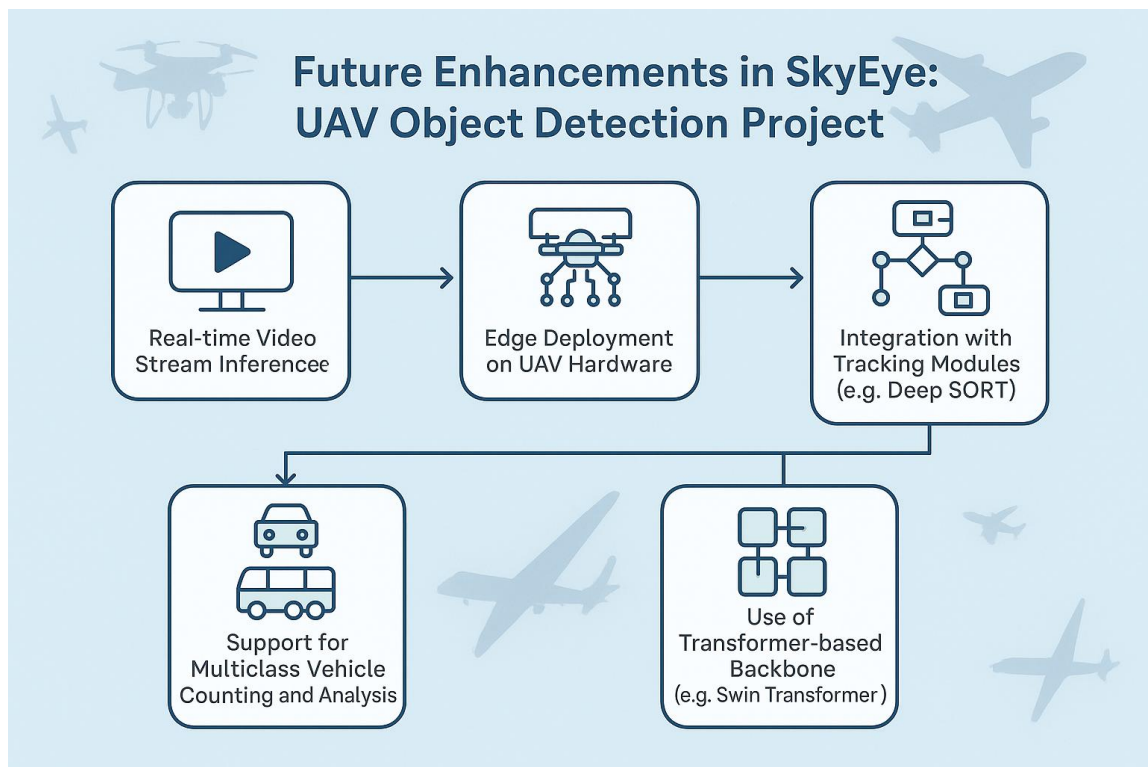


Fig : 7.3.1. FUTURE ENHANCEMENTS

CHAPTER 8

REFERENCES

- [1] A. Bochkovskiy, C. Y. Wang, and H. Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” *arXiv preprint arXiv:2004.10934*, 2020. [Online]. Available: <https://arxiv.org/abs/2004.10934>
- [2] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90
- [3] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Y. Fu, and A. C. Berg, “SSD: Single Shot MultiBox Detector,” in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2016, pp. 21–37. DOI: 10.1007/978-3-319-46448-0_2
- [4] T. Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal Loss for Dense Object Detection,” in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2017, pp. 2980–2988. DOI: 10.1109/ICCV.2017.324
- [5] M. Tan, R. Pang, and Q. V. Le, “EfficientDet: Scalable and Efficient Object Detection,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 10781–10790. DOI: 10.1109/CVPR42600.2020.01080
- [6] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2012, pp. 3354–3361. DOI: 10.1109/CVPR.2012.6248074
- [7] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” *arXiv preprint arXiv:1804.02767*, 2018. [Online]. Available: <https://arxiv.org/abs/1804.02767>
- [8] C. Y. Wang, H. Y. M. Liao, I. H. Yeh, Y. H. Wu, P. Y. Chen, and J. W. Hsieh, “YOLOv5: An Improved Object Detection Algorithm,” *GitHub Repository by Ultralytics* (not officially peer-reviewed, but widely used). [Online]. Available: <https://github.com/ultralytics/yolov5>

CHAPTER 9

APPENDICES

SIMILARITY CHECK REPORT

9.1. SOURCE CODE

9.1.1. IMPORTING LIBRARIES

```
import os
import cv2
import matplotlib.pyplot as plt
from collections import defaultdict
from PIL import Image
from ultralytics import YOLO
import torch
import scipy.io
import numpy as np
from torchvision import transforms
from torch import nn
from tqdm import tqdm
import random
import shutil
from torch.utils.data import Dataset, DataLoader, random_split
import seaborn as sns
from sklearn.metrics import confusion_matrix
import onnx
import onnxruntime as ort
from pathlib import Path
```

9.1.1. IMPORTING LIBRARIES

9.1.2. IMAGE RESIZING

```
os.makedirs(resized_output_path, exist_ok=True)

for img in os.listdir(img_input_path):
    if img.endswith(".jpg"):
        path = os.path.join(img_input_path, img)
        image = cv2.imread(path)
        resized = cv2.resize(image, (640, 640))
        cv2.imwrite(os.path.join(resized_output_path, img), resized)
```

9.1.2. IMAGE RESIZING

9.1.3. DATA AUGMENTATION

```
mosaic=0.5,  
hsv_h=0.015,  
hsv_s=0.7,  
hsv_v=0.4,  
flipud=0.5,  
fliplr=0.5,  
scale=(0.5, 1.5),
```

9.1.3. AUGMENTATION BLOCK

9.1.4. FEATURE EXTRACTION

```
class SCFBlock(nn.Module):  
    def __init__(self, in_channels, out_channels):  
        super(SCFBlock, self).__init__()  
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1)  
        self.relu = nn.ReLU()  
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1)  
  
    def forward(self, x):  
        out1 = self.relu(self.conv1(x))  
        out2 = self.relu(self.conv2(out1))  
        return out1 + out2  # Symmetric addition
```

9.1.4.1 SYMMETRY C2F (SCF) BLOCK

```
class EMABlock(nn.Module):  
    def __init__(self, channels):  
        super(EMABlock, self).__init__()  
        self.global_pool = nn.AdaptiveAvgPool2d(1)  
        self.conv1 = nn.Conv2d(channels, channels // 8, 1)  
        self.relu = nn.ReLU()  
        self.conv2 = nn.Conv2d(channels // 8, channels, 1)  
        self.sigmoid = nn.Sigmoid()  
  
    def forward(self, x):  
        attn = self.global_pool(x)  
        attn = self.relu(self.conv1(attn))  
        attn = self.sigmoid(self.conv2(attn))  
        return x * attn
```

9.1.4.2 EFFICIENT MULTISCALE ATTENTION (EMA)

```

class FeatureFusion(nn.Module):
    def __init__(self, in1, in2, out_channels):
        super(FeatureFusion, self).__init__()
        self.conv = nn.Conv2d(in1 + in2, out_channels, 1)

    def forward(self, x1, x2):
        x = torch.cat([x1, x2], dim=1)
        return self.conv(x)

```

9.1.4.3 FEATURE FUSION (FF) MODULE

```

# Combined Feature Extractor using SCF + EMA + FF
class FeatureExtractor(nn.Module):
    def __init__(self):
        super(FeatureExtractor, self).__init__()
        self.scf = SCFBlock(3, 64)
        self.ema = EMABlock(64)
        self.fusion = FeatureFusion(64, 64, 128)
        self.global_pool = nn.AdaptiveAvgPool2d((1, 1))

    def forward(self, x):
        scf_out = self.scf(x)
        ema_out = self.ema(scf_out)
        fused = self.fusion(scf_out, ema_out)
        pooled = self.global_pool(fused)
        return torch.flatten(pooled, 1) # Final feature vector [1, 128]

```

9.1.4.4 COMBINED FEATURE EXTRACTOR

9.1.5. TRAINING CODE

```
# Load the YOLOv8s model
model = YOLO('yolov8s.pt')

# Train the model
model.train(
    data='D:/VID_dataset/dataset/dataset.yaml',
    epochs=100,           # Total training epochs
    imgsz=640,           # Input image size
    batch=16,            # Batch size
    name='msfe_yolo_drone', # Folder name for saving results
    patience=10,         # Early stopping patience
    save=True,           # Save best model
    half=True,           # Use mixed precision
    device=0,            # Use GPU 0
    rect=True,           # Rectangular training
    workers=4,           # Number of data loading workers
    verbose=True         # Detailed logging
)
```

9.1.5. TRAINING CODE

9.1.6. ONNX (Open Neural Network Exchange) EXPORT

```
model_path = r"D:/VID_dataset/runs/detect/msfe_yolo_drone.pt"
model = YOLO(model_path)
model.export(format="onnx")
```

9.1.6. ONNX EXPORT

9.1.7. VIDEO TEST CODE

```
import cv2
import numpy as np
import onnx
import onnxruntime as ort
import os
# Config
CONF_THRESH = 0.25
```

```

IOU_THRESH = 0.4
TARGET_SIZE = 640
# Class Names
CLASS_NAMES = [
    'people', 'pedestrian', 'bicycle', 'car', 'van', 'truck',
    'tricycle', 'awning-tricycle', 'bus', 'motor', 'others'
]
# Class Colors
COLOR_PALETTE = {
    cls: (int(np.random.randint(0, 255)), int(np.random.randint(0, 255)),
int(np.random.randint(0, 255)))
    for cls in CLASS_NAMES
}
# Image Preprocess
def preprocess(img, target_size=TARGET_SIZE):
    h, w = img.shape[:2]
    scale = min(target_size / w, target_size / h)
    new_w, new_h = int(w * scale), int(h * scale)
    resized_img = cv2.resize(img, (new_w, new_h))
    padded_img = np.full((target_size, target_size, 3), 114, dtype=np.uint8)
    padded_img[:new_h, :new_w] = resized_img
    img_input = padded_img / 255.0
    img_input = np.transpose(img_input, (2, 0, 1))[np.newaxis,
...].astype(np.float32)
    return img_input, scale, (new_w, new_h), (h, w)
# Refine Object Class Based on Size
def refine_class(class_id, box):
    x1, y1, x2, y2 = box
    height = y2 - y1
    if class_id == CLASS_NAMES.index('truck'):
        if height > 250:
            return CLASS_NAMES.index('bus')
        elif height < 100:

```

```

return CLASS_NAMES.index('car')
if class_id == CLASS_NAMES.index('van') and height < 100:
return CLASS_NAMES.index('car')
return class_id

# Postprocess
def postprocess(pred, scale, resized_shape, orig_shape):
pred = np.squeeze(pred[0])
boxes, scores, class_ids = [], [], []
for i in range(pred.shape[1]):
cls_scores = pred[4:, i]
class_id = np.argmax(cls_scores)
confidence = cls_scores[class_id]
if confidence > CONF_THRESH:
cx, cy, w, h = pred[4, i]
x1 = int((cx - w / 2) / scale)
y1 = int((cy - h / 2) / scale)
x2 = int((cx + w / 2) / scale)
y2 = int((cy + h / 2) / scale)
x1, y1 = max(0, x1), max(0, y1)
x2, y2 = min(orig_shape[1], x2), min(orig_shape[0], y2)
box = [x1, y1, x2, y2]
refined_class = refine_class(class_id, box)
boxes.append(box)
scores.append(float(confidence))
class_ids.append(refined_class)
if boxes:
indices = cv2.dnn.NMSBoxes(boxes, scores, CONF_THRESH,
IOU_THRESH)
indices = indices.flatten() if hasattr(indices, 'flatten') else [i[0] for i in indices]
return [boxes[i] for i in indices], [scores[i] for i in indices], [class_ids[i] for i in
indices]
return [], [], []

# Draw Bounding Boxes

```

```

def draw_boxes(img, boxes, scores, class_ids):
    for box, score, class_id in zip(boxes, scores, class_ids):
        class_name = CLASS_NAMES[class_id]
        color = COLOR_PALETTE[class_name]
        x1, y1, x2, y2 = box
        cv2.rectangle(img, (x1, y1), (x2, y2), color, 2)
        label = f"{class_name} {score:.2f}"
        cv2.putText(img, label, (x1, max(10, y1 - 5)),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 0), 2)
    return img

# Load YOLO ONNX Model
def load_model(onnx_path):
    model = onnx.load(onnx_path)
    onnx.checker.check_model(model)
    session = ort.InferenceSession(onnx_path)
    return session

# Image Detection
def detect_image(image_path, output_path, session):
    img = cv2.imread(image_path)
    if img is None:
        print(f" Could not read image: {image_path}")
        return
    img_input, scale, resized_shape, orig_shape = preprocess(img)
    input_name = session.get_inputs()[0].name
    outputs = session.run(None, {input_name: img_input})
    boxes, scores, class_ids = postprocess(outputs, scale, resized_shape,
    orig_shape)
    if boxes:
        img = draw_boxes(img, boxes, scores, class_ids)
    os.makedirs(output_path, exist_ok=True)
    out_file = os.path.join(output_path, os.path.basename(image_path))
    cv2.imwrite(out_file, img)
    print(f"Saved: {out_file}")

```

```

# Main Function
def main():
    onnx_model_path = "D:/VID_dataset/runs /best.onnx"
    input_image_path = "D:/VID_dataset/imgclg3.jpg"
    output_path = "D:/VID_dataset "
    session = load_model(onnx_model_path)
    detect_image(input_image_path, output_path, session)
if __name__ == "__main__":
    main()

```

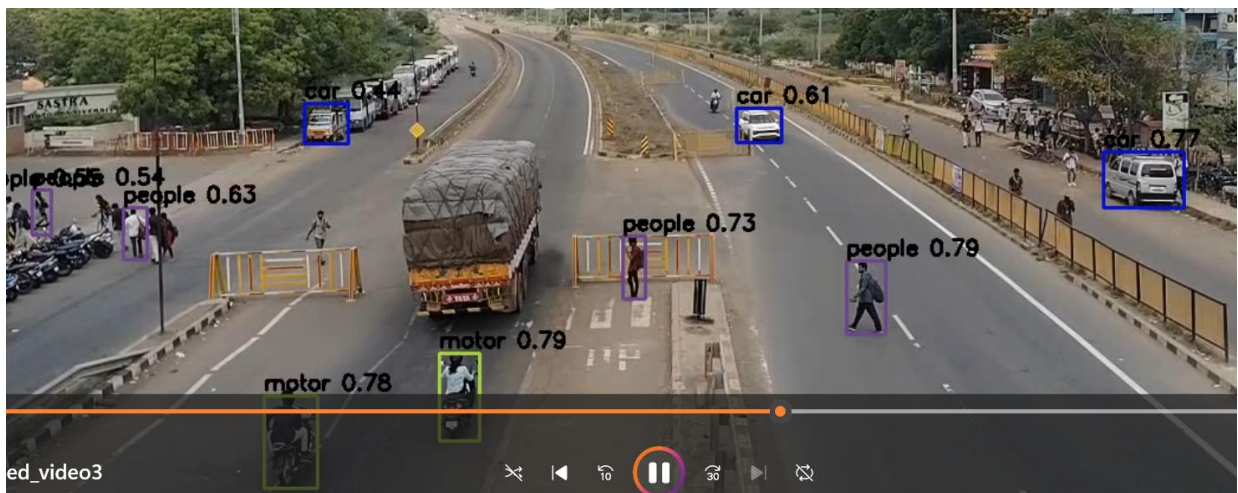


FIG : 9.1.7.1 OUTPUT VIDEO