

YOLOV8-BASED ADVANCE ASSISTANCE FOR AUTONOMOUS VEHICLES

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering degree in Computer Science and Engineering

By
L SAI KUMAR (Reg. No - 40111097)
YALLA SAI KRISHNA VAMSI (Reg. No - 40111096)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

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

CATEGORY -1 UNIVERSITY BY UGC

**Accredited with Grade "A++" by NAAC | 12B Status by UGC | Approved by AICTE
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI - 600119**

APRIL - 2024



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Category - I University by UGC

Accredited "A++" by NAAC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the Bonafide work of **L SAI KUMAR (40111097)** and **YALLA SAI KRISHNA VAMSI (40111096)** who carried out the project entitled "**YOLOV8-BASED ADVANCE ASSISTANCE FOR AUTONOMOUS VEHICLES**" under my supervision from November 2023 to April 2024.

Internal Guide

Dr. D. POORNIMA M.E., Ph.D.

Head of the Department

HEAD OF THE DEPARTMENT
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Dr. L. LAKSHMANAN, M.E., Ph.D.

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

Jeppiyazh Nagar, Poovanthi, Chennai - 600 119.

Submitted for Viva voce Examination held on 12-04-2024

Internal Examiner

External Examiner

(Dr. P. Vasudevan)

DECLARATION

I, L SAI KUMAR (Reg.No- 40111097) hereby declare that the Project Report entitled "YOLOV8-BASED ADVANCE ASSISTANCE FOR AUTONOMOUS VEHICLES" done by me under the guidance of Dr.D.POORNIMA M.E., Ph.D., is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering**.

DATE: 10-04-2024

PLACE: Chennai

L.Sai Kumar.

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

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

I convey my thanks to **Dr. T. Sasikala M.E., Ph. D, Dean**, School of Computing, **Dr. L. Lakshmanan M.E., Ph.D.**, Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. D. Poornima M.E., Ph.D.**, for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

The artifact search process is based on simple teaching methods and crafts. Combining many low-resolution images with high-definition capture and separation, the integrated building can easily balance their performance. To solve the problems of traditional architecture, due to the rapid development of deep learning, more powerful tools that can learn to speak, advanced and dynamic features absorption are provided. This publication introduces a new method to identify people, vehicles and traffic signals using only publicly available data. Searching for buildings in the dataset is a difficult task as it contains distant images such as those in bright sunlight and the key features of the dataset are not suitable for training. Therefore, to improve its accuracy, we recommend using the standard YOLOv8 for training. The model uses various losses and regularization methods. Using YOLOv8 is essential as it can be used on mobile devices and uses less memory for thinking. It also has more support in Unity for a better port for mobile devices.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO
	ACKNOWLEDGEMENTS	iv
	ABSTRACT	v
	LIST OF FIGURES	ix
1	INTRODUCTION	1
2	LITERATURE SURVEY	5
3	PROPOSED METHODOLOGY	10
4.	IMPLEMENTATION	11
	4.1 Dataset Preparation	12
	4.2 Architecture Creation and Training	13
	4.3 Inference Code	14
5.	ALGORITHM DESCRIPTION	16
	5.1 Deep Learning	16
	5.1.1 Functions of DL	16
	5.2 Convolutional Neural Network	17
	5.2.1 Convolutional Layer	18
	5.2.2 Pooling Layer	18
	5.2.3 Fully Connected Layer	19
	5.3 Transfer Learning	20
	5.3.1 YOLO	21
	5.4 YOLOv3	22
	5.4.1 Bounding Box Optimization	24
	5.4.2 Feature Pyramid Network	26
	5.5 YOLOv5	27

	5.6 YOLOv8	29
	5.6.1 Models available in YOLOv8	30
	5.6.2 YOLOv8 Architecture	33
	5.6.4 Anchor Free Detection	34
	5.6.5 New Convolutions	34
	5.6.6 Closing the Mosaic Augmentation	35
6	SYSTEM REQUIREMENTS	36
	6.1 Hardware Requirements	36
	6.2 Software Requirements	36
	6.3 Software Environment	36
7	SYSTEM DESIGN	40
	7.1 Data Flow Diagram	40
	7.2 UML Diagram	41
	7.3 Use Case Diagram	43
	7.4 Class Diagram	44
	7.5 Sequence Diagram	44
	7.6 Activity Diagram	45
	7.7 Input Design	46
	7.8 Output Design	47
8	SYSTEM STUDY	49
	8.1 Feasibility Study	49
	8.1.1 Economical Feasibility	50
	8.1.2 Technical Feasibility	50
	8.1.3 Social Feasibility	51
9	SYSTEM TESTING	52
	9.1 Types of Tests	52

	9.1.1 Unit Testing	52
	9.1.2 Integration Testing	52
	9.1.3 Functional Testing	53
	9.1.4 System Testing	53
	9.1.5 White Box Testing	53
	9.1.6 Black Box Testing	53
	9.2 Unit Testing	54
	9.3 Integration Testing	55
	9.4 Acceptance Testing	55
10	CONCLUSION	56
	REFERENCE	57
	APPENDIX	61
	RESEARCH PAPER	67

FIG NO.	FIGURE NAME	Page No.
1.1	Object Detection	2
1.2	COCO JSON Format	3
3.1	System Architecture	10
4.1	COCO JSON Format	12
4.2	YOLO Format	12
4.3	Training Yolov8 Model	13
4.4	Training Yolov8 Model	13
4.5	Training Yolov8 Model	14
4.6	Detection of Cars and Traffic lights using Yolov8	14
4.7	Detection of Cars and Traffic lights using Yolov8	15
5.1	Types of DL	16
5.2	Uses of DL Algorithm	17
5.3	CNN Architecture	18
5.4	Transfer Learning	20
5.5	YOLO Algorithm in Object Detection	21
5.6	YOLOv3 Architecture for Multiple Object Detection	23
5.7	Backbone Darknet53 Network Model	24
5.8	Anchor Box Regression	25
5.9	Feature Pyramid Network	26
5.10	The Network Architecture of YOLOv5	27
5.11	Model Available in YOLOv8	30
5.12	Object Detection Using YOLOv8	31
5.13	Object Detection Using YOLOv8	31
5.14	Object Detection Using YOLOv8	32
5.15	Object Detection Using YOLOv8	32
5.16	YOLOv8 Architecture	33
7.1	Use Case Diagram	43

7.2	Class Diagram	44
7.3	Sequence Diagram	45
7.4	Activity Diagram	46
8.1	Types of Feasibility Study	49
8.2	Economic Feasibility	50
8.3	Technical Feasibility	51
A1.1	Converting COCO Json Format into Yolo Format	61
A1.2	Converting COCO Json Format into Yolo Format	62
A1.3	Converting COCO Json Format into Yolo Format	62
A2.1	Training Yolov8 Model	63
A2.2	Training Yolov8 Model	63
A2.3	Training Yolov8 Model	64
A2.4	Training Yolov8 Model	64
A2.5	Training Yolov8 Model	65
A3.1	Object Detection Using Yolov8	66

CHAPTER 1

INTRODUCTION

People can quickly identify and find objects in pictures. Human vision is fast and precise and can perform complex tasks such as recognizing multiple objects and seeing problems without much effort. With the availability of large amounts of data, faster GPUs, and better algorithms, we can now quickly train computers to identify and classify many objects in images with high accuracy. Using this identification and authentication, object detection can be used to count objects in a place as well as locate and track in real time when properly recorded. The object detection problem in computer vision is finding and recognizing objects in images and videos. It is important in many applications, including robotics, self-driving cars and surveillance. The object detection box is a key element that finds the edges of objects marked as squares or rectangles or other sharp rectangles. They carry tags that identify the target object, whether it's a human, a car or a dog. As long as the model knows which objects it is tagging,

bounding boxes can overlap to show multiple objects in a single image.

Single shot detectors and two stage detectors are the two main groups of object detection algorithms. Object detection can be done in several ways:

- Feature Based Object Detection
- Viola Jones Object Detection
- SVM Classifications with HOG Features

Early detection techniques used image classification techniques to find objects while looking at different parts of the image at different scales. These techniques are ineffective and slow.



Fig:1.1. Object Detection

It distinguishes deep learning from other methods by using a convolutional neural network (CNN). Neural networks follow the complex neuronal structure of the human brain. The hidden layer, release layer, and access layer do most of this. These neural networks can learn in a supervised, semi-supervised or unsupervised way, depending on the amount of annotation (if any) on the data presented. Given the ability of CNNs to learn with less manual engineering, deep neural networks for object detection provide the fastest and most accurate results ever for single detection and many things.

If there is no specific design for the job, it will not be possible to have information about the product. These detection models are then trained on hundreds of thousands of pieces of data to achieve better accuracy. Easily accessible data such as COCO (Contextual Content) is used to train and improve models, giving you a good start in evaluating your explanations.

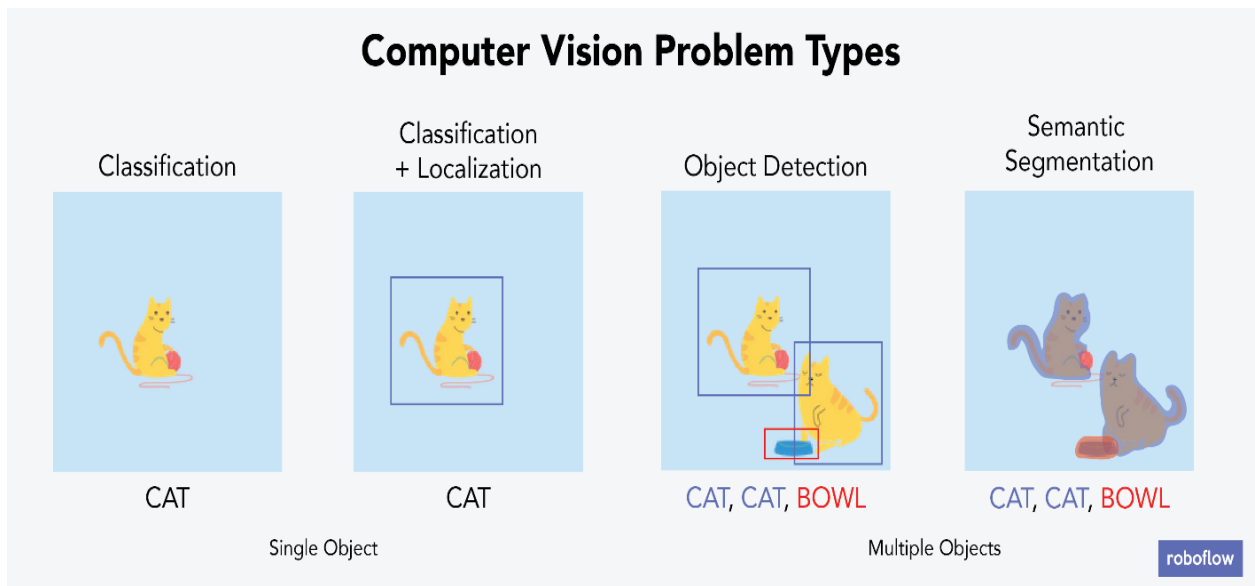


Fig:1.2. COCO JSON Format

R-CNN (Region Based Convolutional Neural Networks), proposed in 2014, is the first family of efficient algorithms. It only removes 2,000 regions (called regional recommendations) from the image compared to pre-developed regions and therefore outperforms its predecessors. After the output image is selected, 2,000 regional comments are extracted according to the R-CNN flowchart. Features will then be extracted from each region separately, and each region will be assigned to one of the designated groups. The main disadvantage of R-CNN is that the process still takes time, even after 2,000 regional recommendations have been issued. This is what makes the new and improved Fast R-CNN possible.

Fast R-CNN reduces processing time by feeding the image into a pre-trained CNN to generate convolutional feature maps, rather than dividing the image into 2,000 regional concepts. Alternatively, it could be as simple as identifying regional recommendations from a particular map and sending them to the ROI pooling layer, which de-features certain regions. After that, the whole linking process outputs the previous process and then splits the output of the model into two parts: one for the category predicted by the SoftMax layer, and soon. for bounding box predictions from the output.

Later, a third and more advanced model called Faster R-CNN was proposed. With some important modifications, the design is comparable to Fast R-CNN. Selection Detection is based on the integration of relevant regions not used by Faster R-CNN.

It will be replaced by the Regional Suggestion Network, which can quickly identify regional proposals. This paper presents the YOLO model for product discovery.

YOLO uses a completely different concept. It only looks at all images once and scans the mesh once for objects. Hence the name time flies.

CHAPTER 2

LITERATURE SURVEY

Z. Q. Zhao, P. Zheng, S. T. Xu, and X. Wu [1] mentioned in their article “**Object Detection with Deep Learning**” how to find the relationship with video analysis and visual perception, object detection has been the subject of much research lately. The artifact search process is based on simple teaching methods and crafts. Combining many low-resolution images with high-definition capture and separation, the integrated building can easily balance their performance. To solve the problems of traditional architecture, due to the rapid development of deep learning, more powerful tools that can learn to speak, advanced and dynamic features absorption are provided. Network architecture, training methods, optimization functions, etc., strange patterns. In this work, we explore deep learning as a search engine. Before going into the analysis, a brief history of deep learning and its descriptive tool, Convolutional Neural Network (CNN), is given. Next, we focus on general discovery tools with some changes and helpful tips to further enhance discovery. We also provide a brief overview of some special functions such as object detection, face detection and pedestrian detection, as different features have special detection functions. In addition, clinical trials are provided to compare different methods and draw some conclusions. Various interesting and functional suggestions are presented to provide suggestions for future work on object recognition and related neural network-based learning.

G. Chandan, A. Jain, and H. Jain [2] who published the paper “**Real Time Object Detection and Tracking Using Deep Learning and OpenCV**” In the past few years, deep learning has had a huge impact on the world's progress towards artificial intelligence. Region-Based Convolutional Neural Networks (RCNN), Faster-RCNN, Single Shot Blocking (SSD), and You Only Look At

One (YOLO) are some well-known search engines. When speed is more important than accuracy, YOLO works better than others, Faster-RCNN and SSD have more accuracy. It combines deep learning, SSD and Mobile Networks to provide effective detection and monitoring. This approach can capture quality without sacrificing performance.

M. Yazdi and T. Bouwmans [3] published about their work “**New trends on moving object detection in video images captured by a moving camera**” in which they said: This work presents a state-of-the-art technique for tracking moving objects in videos captured by motion cameras. While many studies and excellent studies have examined object detection and background removal techniques for fixed cameras, there are currently no studies investigating what technology is used for mobile cameras. Most techniques in this field can be divided into four groups: object tracking, low-order and low-order matrix factorization, orbital classification, and modeling-based background subtraction. We explain each category in detail and offer key ways to suggest improvements to the overall technology concept. In addition to performance tests and some benchmark data that can be used to evaluate the performance of various moving object detection algorithms, we also present challenges and remaining questions in this area.

C.-Y. Wang, A. Bochkovskiy, and H.-Y.-M. Liao [4] published their work “**Scaled-YOLOv4: Scaling cross stage partial network**” in which they mentioned about YOLOv4. They found that the CSP-based YOLOv4 object recognition neural network can scale both small and large networks up and down while maintaining speed and accuracy. We present a similar strategy that changes the structure of the network in addition to differences in depth, width, and resolution. On the MS COCO dataset at around 16 FPS on Tesla V100, the YOLOv4 size model achieves cutting-edge results: 55.5% AP (73.4% AP50), while the YOLOv4 size increases to 56 as test time increases. 0% AP (73.3

AP50). To the best of our knowledge, the published study currently contains the most accurate COCO dataset. On the RTX 2080Ti, the YOLOv4-small sample reaches 22.0% AP (42.0% AP50) at a frame rate of 443 FPS, while YOLOv4- small reaches 1774 FPS using TensorRT, batch size = 4 and FP16 precision.

Y. Tian, P. Luo, X. Wang, and X. Tang [5] did their work on “**Deep learning strong parts for pedestrian detection**” they have discussed about Deep Parts and ConvNet. Recent progress has been made in pedestrian recognition by adapting features learned from a neural network (ConvNet) to pedestrians. Usually this is trained with more general tools like ConvNet, ImageNet. While these features can be adapted to gradation, visibility, and lighting, they don't work well for walking photos with complex occlusions. One of the most important issues in pedestrian consciousness is dealing with occlusion. We recommend using Deep Parts, which has a wide range of tools, instead of previous deep models that only learned one detector to recognize pedestrians. Deep Parts has many advantages. First, weak data or only inclusive and no annotation data can be used to train Deep Parts. Second, Deep Parts can maintain a good IoU strategy that is no different from reality. Third, every Deep Parts device is a powerful electronic device that can detect problems by looking for small clues. Extensive testing on the Caltech dataset shows that Deep Parts works well; It achieves a new error condition of 11:89% while achieving the next best method at 10%.

C. Kwan and B. Budavari. [6] did their work on “**Enhancing Small Moving Target Detection Performance in Low-Quality and Long-Range Infrared Videos Using Optical Flow Techniques**” in which they discussed. In far- infrared images, background clutter, air turbulence, and small targets make it difficult to see small moving objects. In this paper, we summarize our research on effective methods to improve small object recognition performance in long- range, negative infrared recording of moving objects. We pay special attention to the unsupervised, flexible, flexible and efficient process to enable data to be moved from the optical device to improve discovery. Three popular optical

systems are studied. To fix the product, it was found that for the order for the order for the order for the order for the order for the order, the optical system difference must be combined with the difference coupling components and coupling components. The performance of our proposed scheme was demonstrated in a series of tests using medium wave infrared (MWIR) video from the Defense Systems Information Analysis Center (DSIAC) collection.

C. Kwan and J. Larkin [7] did their work on “**Detection of small moving objects in long range infrared videos from a change detection perspective**” in which they said Small moving objects can make it difficult to shoot long infrared (IR) video due to background noise, air turbulence, and small targets. This work presents two neglected, flexible and flexible ways to locate small moving objects. Different poles can be used to analyze motion in displacement (CD) algorithms that support the main point. Our framework includes post- processing techniques such as image noise removal and enlargement, as well as conversion, small object detection, and various post-processing techniques. Extensive testing using long-wavelength medium-wave infrared (MWIR) photography of a target 3500 meters from the camera shows that a method that uses local density gradients (LIGs) only once in the workflow outperforms others at 3500 meters of video. Slightly worse at 4000m and 5000m videos. We also explore the use of synthetics in object search and find support in films of 4000 and 5000 m lengths. Finally, a comparison of the two established methods shows that the performance of our plans is comparable.

C. Kwan and B. Budavari. [8] in which they said “**A high-performance approach to detecting small targets in long-range low-quality infrared videos**” Detection of objects in long range infrared (IR) video is difficult due to the small size of the objects. In this article, we propose an effective method for finding small objects in long-range infrared films. Our method includes

integration of cross-links, component (CC) analysis module, local evidence use and gradient (LIG) as a tool, small, high-definition video enhancement module and small equipment. The effectiveness of the proposed method has been verified in extensive tests using data measurements of medium wave infrared (MWIR) films in the range of 3500 m to 5000 m.

M. S. Uddin, R. Hoque, K. A. Islam, C. Kwan, D. Gribben, and J. Li. [9] published **“Converting optical videos to infrared videos using attention GAN and its impact on target detection and classification performance”** mentioned Additional training data is needed to develop performance models that can be used with deep learning-based algorithms for object recognition and classification in infrared video. However, optical video recording is more common in surveillance applications than infrared recording. If optical video can be converted to infrared video, the shortage of infrared video datasets will be reduced. In this work, we introduce a new method for conversion from optical video to infrared video based on deep learning. The main idea is to maintain the integrity of the target areas using an Attention GAN. The method does not require similar images. Both objective and subjective assessments were used to demonstrate the effectiveness of the proposed GAN. Most importantly, using real infrared video, the results of monitoring GANs have been shown to improve object detection and classification.

CHAPTER 3

PROPOSED METHODOLOGY

The result of object detection in computer vision includes the recognition of many things in pictures or videos. Various methods are used to detect objects, including Single Sided Multibox Detector (SSD), Retina-Net, and Fast R-CNN. These methods address insufficient data and modeling problems in object search, but fail to find objects in a single operation. This article introduces the YOLOv8 deep learning model for object detection. YOLO is a method that uses neural networks to provide real-time object detection.

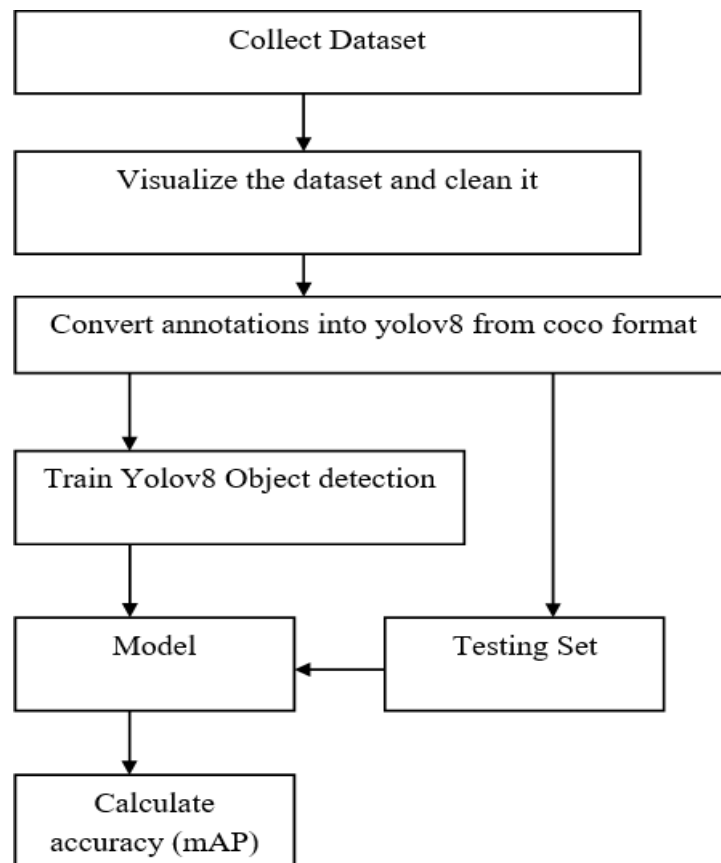


Fig:3.1. System Architecture

CHAPTER 4

IMPLEMENTATION

MODULE DESCRIPTION:

- **Dataset Preparation**
- **Architecture Creation and Training**
- **Inference Code**

4.1. DATASET PREPARATION

- Data including pedestrians, vehicles and traffic signals must be verified before being placed.
- Annotation format should be changed to yolo format, then the groundwork should be drawn that it is correct.
- The first two dots (x, y) of the bounding box are the upper-left function of the box, but the annotation file also contains information about the image, such as the image ID, category, and bbox.
- The corresponding file in the folder must have the same file name as the image file in the image folder.

For example:

images

→0001.jpg

labels

→ 0001.txt

- Each text file must fulfill all the properties of the YOLO format text file which are the following:
 1. Each line starts with a class ID, followed by the box's attributes (x, width, and height).
 2. Normalize bounding box properties (0-1).

3. The center of the box is (x, y).



Fig:4.1. COCO JSON Thermal Format

- In the output method, a new dataset (converted dataset) will be generated with images and tags in YOLO format.

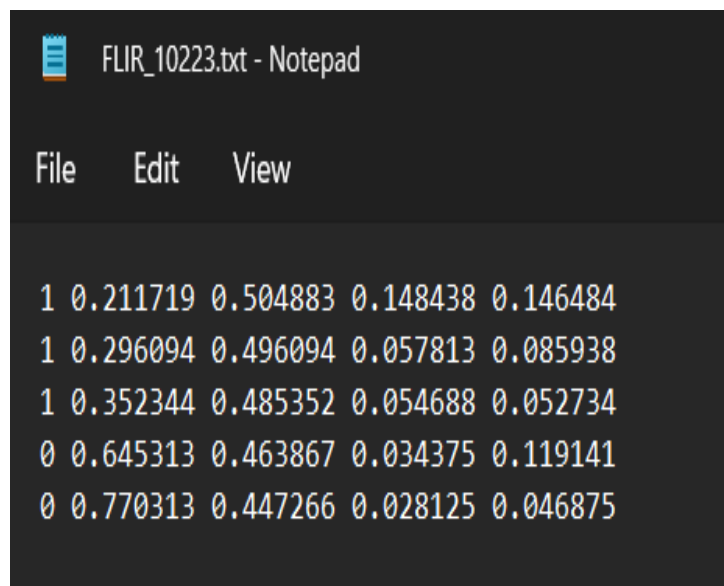


Fig:4.2. YOLO Format

4.2. ARCHITECTURE CREATION AND TRAINING

- For this application we will use the YOLOv8 model, so we will begin to setup the PyTorch environment and create all the necessary elements and initial weights for the design.
- When the schema is ready, the model will start training with the prepared data. Therefore, if the pattern improves on the map and the price drops, stop after watching the price drop carefully to determine how the pattern learned.

```
+ Code + Text

from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

!pip install ultralytics

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting ultralytics
  Downloading ultralytics-8.0.54-py3-none-any.whl (496 kB)
    496.9/496.9 KB 9.8 MB/s eta 0:00:00
Collecting sentry-sdk
  Downloading sentry_sdk-1.17.0-py2.py3-none-any.whl (189 kB)
    189.1/189.1 KB 24.5 MB/s eta 0:00:00
Requirement already satisfied: requests>=2.23.0 in /usr/local/lib/python3.9/dist-packages (from ultralytics) (2.27.1)
Requirement already satisfied: Pillow>=7.1.2 in /usr/local/lib/python3.9/dist-packages (from ultralytics) (8.4.0)
Requirement already satisfied: matplotlib>=3.2.2 in /usr/local/lib/python3.9/dist-packages (from ultralytics) (3.7.1)
Requirement already satisfied: torchvision>=0.8.1 in /usr/local/lib/python3.9/dist-packages (from ultralytics) (0.14.1+cu116)
Requirement already satisfied: opencv-python>=4.6.0 in /usr/local/lib/python3.9/dist-packages (from ultralytics) (4.6.0.66)
Requirement already satisfied: tqdm>=4.64.0 in /usr/local/lib/python3.9/dist-packages (from ultralytics) (4.65.0)
Requirement already satisfied: psutil in /usr/local/lib/python3.9/dist-packages (from ultralytics) (5.9.4)
Collecting thop>=0.1.1
  Downloading thop-0.1.1.post2209072238-py3-none-any.whl (15 kB)
Requirement already satisfied: pandas>=1.1.4 in /usr/local/lib/python3.9/dist-packages (from ultralytics) (1.4.4)
Requirement already satisfied: numpy>=1.21.6 in /usr/local/lib/python3.9/dist-packages (from ultralytics) (1.22.4)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.9/dist-packages (from ultralytics) (1.10.1)
Requirement already satisfied: PyYAML>=5.3.1 in /usr/local/lib/python3.9/dist-packages (from ultralytics) (6.0)
Requirement already satisfied: torch>=1.7.0 in /usr/local/lib/python3.9/dist-packages (from ultralytics) (1.13.1+cu116)
Requirement already satisfied: seaborn>=0.11.0 in /usr/local/lib/python3.9/dist-packages (from ultralytics) (0.12.2)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.2.2->ultralytics) (3.0.9)
```

Fig:4.3. Training YOLOv8 Model

```
+ Code + Text

import ultralytics
ultralytics.checks()

Ultralytics YOLOv8.0.54 Python-3.9.16 torch-1.13.1+cu116 CUDA:0 (Tesla T4, 15102MiB)
Setup complete (2 CPUs, 12.7 GB RAM, 25.5/78.2 GB disk)

[ ] !rsync -a /content/drive/MyDrive/v8_format .

[ ] !yolo task=detect mode=train model=yolov8s.pt data=data.yaml epochs=50 imgsz=640 batch=8 project=/content/drive/MyDrive/output name=automotive_true

Ultralytics YOLOv8.0.53 Python-3.9.16 torch-1.13.1+cu116 CUDA:0 (Tesla T4, 15102MiB)
yolo/engine/trainer: task=detect, mode=train, model=yolov8s.pt, data=data.yaml, epochs=50, patience=50, batch=8, imgsz=640, save=True, save_period=1, cache=False, device=None, workdir=
Overriding model.yaml nc=80 with nc=3

   from  n  params module                  arguments
   ----  -  -
0       1  928    ultralytics.nn.modules.Conv      [3, 32, 3, 2]
1       1 18560    ultralytics.nn.modules.Conv      [32, 64, 3, 2]
2       1 29056    ultralytics.nn.modules.C2f      [64, 64, 1, True]
3       1 73984    ultralytics.nn.modules.Conv      [64, 128, 3, 2]
4       1 197632   ultralytics.nn.modules.C2f      [128, 128, 2, True]
5       1 295424   ultralytics.nn.modules.Conv      [128, 256, 3, 2]
6       1 788480    ultralytics.nn.modules.C2f      [256, 256, 2, True]
7       1 1180672   ultralytics.nn.modules.Conv      [256, 512, 3, 2]
8       1 1838080   ultralytics.nn.modules.C2f      [512, 512, 1, True]
9       1 656896    ultralytics.nn.modules.SPPF      [512, 512, 5]
10      1 0        torch.nn.modules.upsampling.Upsample  [None, 2, 'nearest']
11     [-1, 6] 1        ultralytics.nn.modules.Concat      [1]
12      1 591360    ultralytics.nn.modules.C2f      [768, 256, 1]
13      1 0        torch.nn.modules.upsampling.Upsample  [None, 2, 'nearest']
14     [-1, 4] 1        ultralytics.nn.modules.Concat      [1]
```

Fig:4.4. Training YOLOv8 Model

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	
1/50	2.126	1.971	1.495	1.413	85	640: 100% 366/366 [02:33:00:00, 2.39it/s]	
Class		Images	Instances	Box(P	R	mAP50	mAP50-95): 100% 5/5 [00:02:00:00, 2.14it/s]
all		80	467	0.568	0.247	0.271	0.106
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	
2/50	2.546	1.908	1.324	1.387	71	640: 100% 366/366 [02:26:00:00, 2.50it/s]	
Class		Images	Instances	Box(P	R	mAP50	mAP50-95): 100% 5/5 [00:01:00:00, 3.31it/s]
all		80	467	0.579	0.245	0.287	0.142
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	
3/50	2.546	1.854	1.25	1.35	79	640: 100% 366/366 [02:24:00:00, 2.53it/s]	
Class		Images	Instances	Box(P	R	mAP50	mAP50-95): 100% 5/5 [00:01:00:00, 3.35it/s]
all		80	467	0.895	0.234	0.304	0.145
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	
4/50	2.546	1.822	1.211	1.331	29	640: 100% 366/366 [02:23:00:00, 2.56it/s]	
Class		Images	Instances	Box(P	R	mAP50	mAP50-95): 100% 5/5 [00:01:00:00, 2.86it/s]
all		80	467	0.846	0.276	0.284	0.131
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	
5/50	2.546	1.761	1.138	1.299	104	640: 100% 366/366 [02:23:00:00, 2.54it/s]	
Class		Images	Instances	Box(P	R	mAP50	mAP50-95): 100% 5/5 [00:02:00:00, 2.44it/s]
all		80	467	0.891	0.277	0.312	0.156
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	
6/50	2.546	1.709	1.058	1.27	31	640: 100% 366/366 [02:25:00:00, 2.51it/s]	
Class		Images	Instances	Box(P	R	mAP50	mAP50-95): 100% 5/5 [00:01:00:00, 3.25it/s]
all		80	467	0.358	0.277	0.315	0.158
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	
7/50	2.546	1.696	1.05	1.271	82	640: 100% 366/366 [02:26:00:00, 2.50it/s]	

Fig:4.5. Training YOLOv8 Model

4.3. INFERENCE CODE

- When the model is trained, a decision is made that loads the model and learns the weight (on top of that) and uses them to apply search objects to the image.
- The training model is more about the application interface in this model.

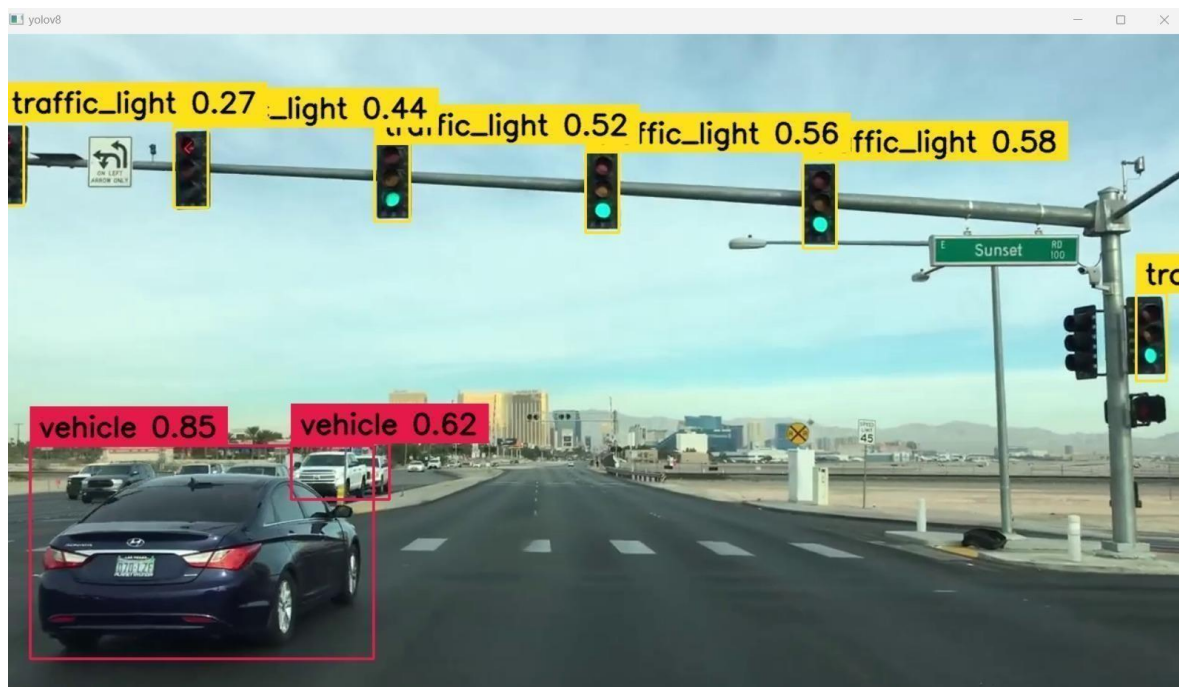


Fig:4.6. Detection of Cars and Traffic lights using YOLOv8



Fig:4.7. Detection of Vehicles and Pedestrian using YOLOv8

CHAPTER 5

ALGORITHM DESCRIPTION

5.1. DEEP LEARNING (DL)

The concept of deep learning has attracted the attention of the scientific and business circles since it was first introduced in 2006. Since its inception, the ILSVRC competition has attracted many artificial intelligence (AI) companies around the world. Support vector machine and machine learning are starting to compete with each other. But winning the first deep learning competition in 2012 led to the rapid development of topics and the introduction of existing deep learning models.

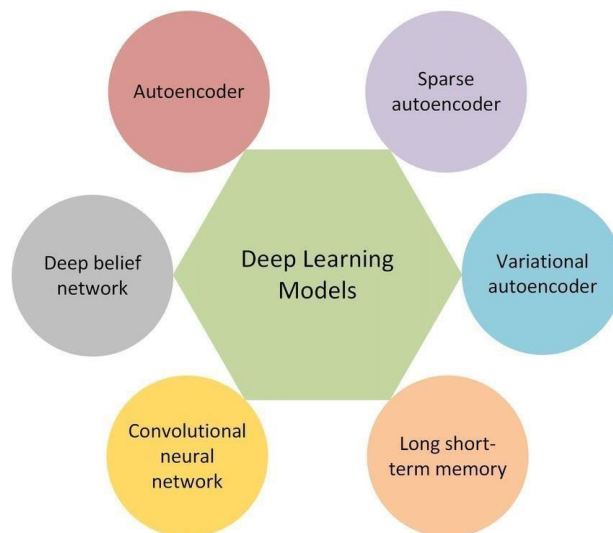


Fig:5.1. Types of DL

5.1.1 FUNCTIONS OF DL

Deep learning techniques extract high-level features from raw data, while traditional machine learning techniques are used to select important features from the data and feed those features into ML-specific algorithms. Low, medium, and high levels are rotated to find or categorize information. Therefore, an important part of deep learning is to use non-linear functions of raw data as input to generate abstract results. Widely used deep

learning techniques require large datasets and advanced processing tools. In recent years, powerful GPUs have been used in deep learning algorithms to increase deployment and reduce learning time.

Deep learning methods (as shown) such as CNN, RNN (Recurrent Neural Network), DBN, DSN (Deep Stack Network) and LSTM (Long Short-Term Memory) are used to decode big data in computer vision., natural language processing sets questions and automatically. sound distribution. The deep learning algorithm used is shown in the figure below. CNNs are modeled after neural networks like ANNs. Neurons in both algorithms have deviations and weights. Although both CNNs and ANNs have layers, the topology of the two networks is very different. In ANN algorithms, the standard set and all connected sets are fully connected. On the other hand, CNN has a set of 3D neurons spanning width, height, and depth.

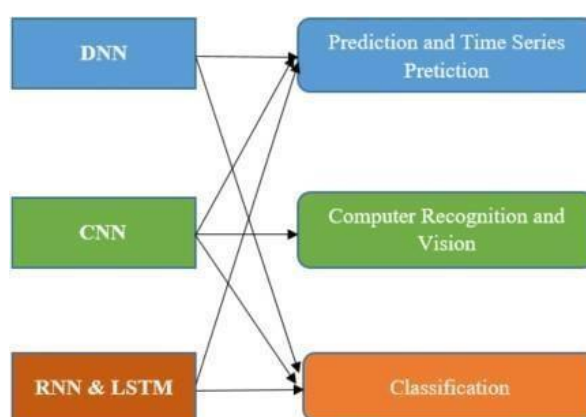


Fig:5.2. Uses of DL Algorithm

5.2. CONVOLUTIONAL NEURAL NETWORKS (CNN)

CNNs are the best-known deep learning algorithms and are very good at detecting patterns in images. CNNs consist of neurons with programmable biases and weights, just like neural networks in the human brain. Multiple inputs are sent to each neuron. Then the weight of the material is calculated. All the weight is fed into the activation function, which then produces the output. What distinguishes

CNN from other neural networks is that it uses convolutional techniques. Convolutional layers, pooling layers, and all layers are the three main layers that make up a CNN.

5.2.1 CONVOLUTIONAL LAYER

The brain of a convolutional neural network is a convolutional process consisting of many convolutional nuclei. Different convolution kernels can be used to obtain different properties. Unlike neural networks, where each neuron must connect to every other neuron in the inclusive previous layer, convolutional neural networks learn only from the local understanding of the previous layer. Because of the relationship between the number of convolution cores and the number of output feature maps, multicore convolution can learn many features. The convolution core speed can be increased to help get more features and improve the presentation of the model to some extent. The convolution layer is a "convolution" process. In a linear process called convolution, the input is multiplied by a set of weights. The core or filter consists of a weight layer. The filter cannot process the amount of input data. The filter is divided into a portion of the input whose size is equal to the size of the filter and which makes up the point material. Point elements are then added to get a value.

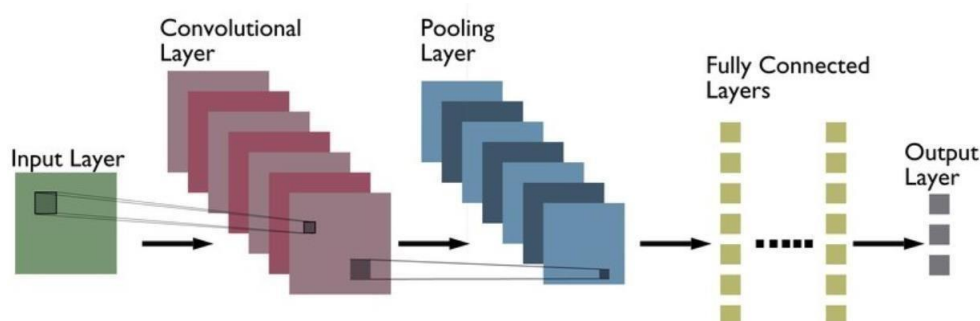


Fig:5.3. CNN Architecture

5.2.2 POOLING LAYER

To simplify network computation and reduce complexity, the pooling techniques often used by the convolutional technique. Therefore, the sensitivity of each depth is reduced by subsampling, which reduces overfitting during network

training. The spatial dimension of the inlet is reduced using pooling, while the depth dimension remains unaffected. The two most common combinations are medium and maximum. The consequences of pooling layer width and height are considered in equations (1), (2), and (3). The width, height, and depth of the entrance are represented in the two equations by W_n , H_n , and D_n , respectively. L represents step (change in kernel of input image) and K represents kernel size.

$$W_n = \frac{W_o - K}{L} + 1 \quad (1)$$

$$H_n = \frac{H_o - K}{L} + 1 \quad (2)$$

$$D_n = D_o \quad (3)$$

5.2.3. FULLY CONNECTED LAYER

The top layer of the CNN architecture is a fully connected layer. Every neuron in a layer in a fully connected layer is connected to every neuron in the layer below it, just like normal ANN. Full tiers show all dataset scores for each class after training. The Rectified Linear Unit (ReLU) enhances CNNs by applying an average activation function (such as a sigmoid) to the output from the activation process. Equation (4) explains the ReLU function.

$$Relu(v) = \max(0, v) \quad (4)$$

The SoftMax classifier is often used in the output process to solve multivariate classification problems because of its small computation and fast learning time. Extraction and classification are the two main steps that CNNs perform during training. Convolutions are applied to the input data using filters or kernels during feature extraction. Then create a custom report. CNNs determine the probability that an image will correspond to a class or tag during the classification process. CNNs are particularly useful for classifying and recognizing images because they can learn features without the need for manual extraction. Transfer learning also allows CNNs to be retrained and used for different purposes. It has been shown that education reform can improve distribution.

5.3. TRANSFER LEARNING

A machine learning method called transfer learning uses previous models as the basis for new models. Simply put, the learning model of the first task concerned is then applied for optimization and rapid progress is achieved in modeling the second task. By applying transfer learning to a new task, better performance can be achieved compared to training with only a small amount of information. Because learning transitions are so common, it's not easy to train a model from scratch to perform graphics or word processing related tasks. The structure of the educational transition is shown below.

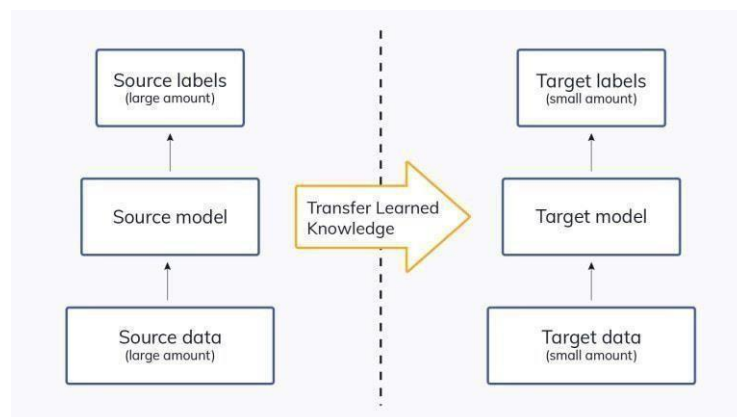


Fig:5.4. Transfer Learning

Various types of transfer learning algorithms are,

- **Alex Net** (2012) — In 2012 Alex Krizhevsky (and others) created Alex Net, an extension and depth to LeNet that had previously won the challenging ImageNet Large Scale Visual Recognition Challenge (ILSVRC). This is a huge improvement over the past, and CNNs are now very popular thanks to this research. ^
- **ZF Net** (2013) — Convolutional community, created via Rob Fergus and Matthew Zeiler, received the 2013 ILSVRC competition. The ZFNet— Zeiler and Fergus internet—became given as its call. It executed better than AlexNet while the architectural hyperparameters were altered.

- **Google-Net** (2014)- Convolutional Networks from Google's Szegedy and others won the 2014 ILSVRC. The development of the Starter Module reduces network headroom (4M compared to Alex Net's 60M), which is a significant plus.
- **VGG-Net** (2014) — At ILSVRC 2014, the VGGNet network took second place. Its main contribution is to demonstrate the importance of mesh depth (number of layers) in terms of overall performance.
- **ResNets** (2015) — The rest of the network is built by Kaiming. Shared the 2015 ILSVRC title with another competitor. The most complex neural network model today is ResNet, which is frequently used in practice (as of May 2016).
- **Dense Net** (August 2016)- Each layer in the densely connected convolutional network, recently published by Gao Huang (and others), is directly connected to all other layers through a feed-forward process.

5.3.1 YOLO

One of the most famous models to master computer vision is called You Look Only Once (YOLO). Object detection equipment is quickly recognized as a complete image. The YOLO algorithm tries to predict the bounding box indicating the class of an object and its position in the input image.

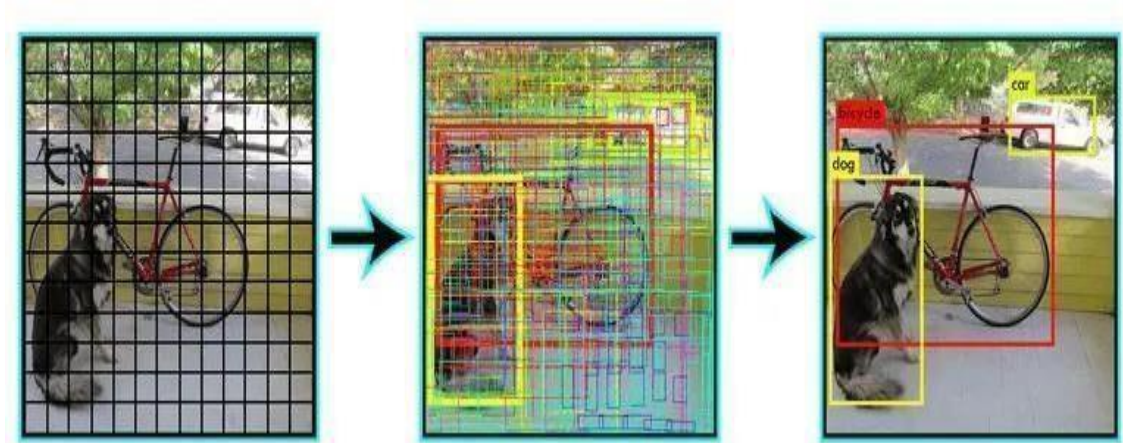


Fig:5.5. YOLO algorithm in object detection

There have been many iterations since Joseph Redmon first introduced YOLO in 2015; the latest was developed by the AI platform Ultralytics, which also developed YOLO v3 and YOLO v5.

5.4. YOLOv3

The 11 folds used by the YOLO operation, meaning "you only look once", ensure that the predicted map is the same size as the previous map. A convolutional neural network (CNN) called YOLO can be implemented quickly. CNNs are classifier-based systems that can identify patterns in an input image by analyzing it based on sequential data. YOLO has the advantage of being faster than other networks while maintaining accuracy. The prediction model is affected by all elements of the image because it allows the model to deal with the entire image during testing. Convolutional neural network methods such as YOLO "grade" regions because they are similar to previous classes. Those with high scores are reported to be the best test for the nearest class.

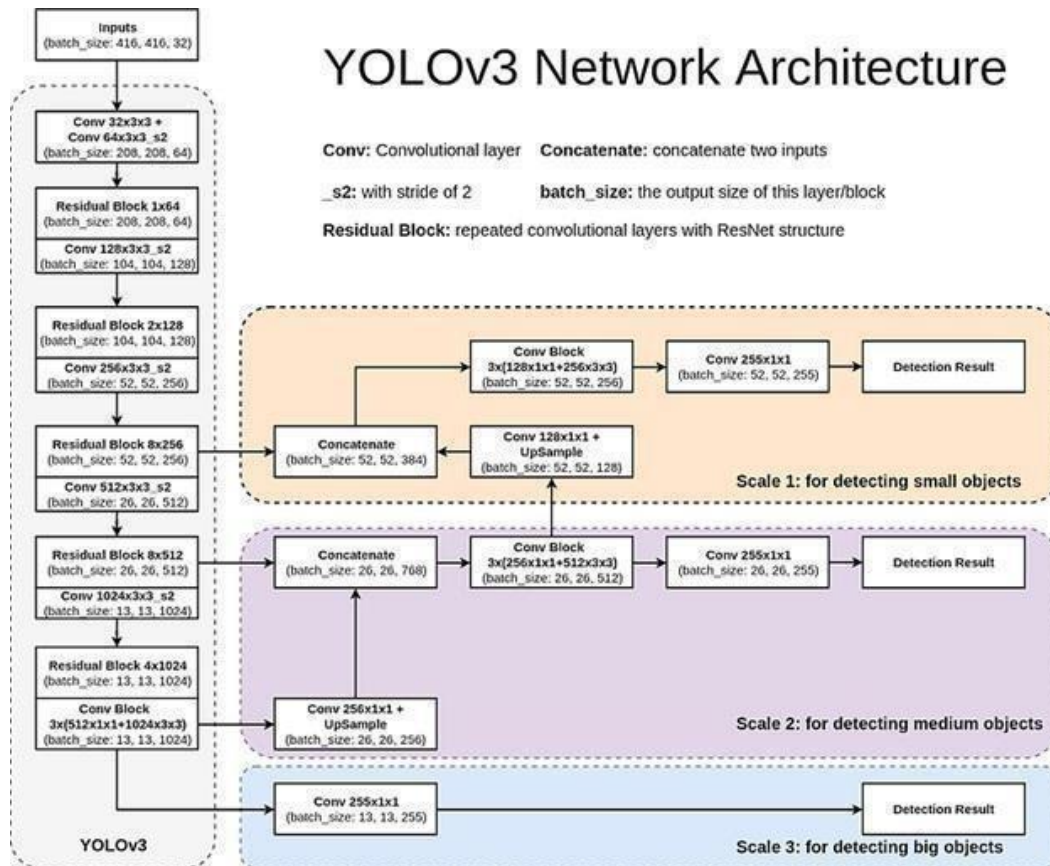


Fig:5.6. YOLOv3 architecture for multiple object detection.

The SoftMax function is used in YOLO versions 1 and 2 to convert scores into probabilities. This strategy only works when opposites can be combined. YOLOv3 uses multiple registers. The YOLOv3 architecture is shown in Figure

1. Unsupervised logistics classifiers are used to evaluate the input probability for a label. The loss is calculated using the binary cross entropy of each tag. The absence of a SoftMax function reduces complexity. YOLOv3 uses Darknet-53. YOLO creators Joseph Redmon and Ali Farhadi also created the Darknet-53 infrastructure. In Darknet53, batch normalization and leaky RELU are applied after each iteration. Because Darknet-53 uses 53 convolutional layers instead of the previous 19 (ResNet-101 or ResNet-152) layers, they are stronger and much more competitive backbones than Darknet-19.

	Type	Filters	Size	Output
	Convolutional	32	3×3	256×256
	Convolutional	64	$3 \times 3 / 2$	128×128
1x	Convolutional	32	1×1	
	Convolutional	64	3×3	
	Residual			128×128
	Convolutional	128	$3 \times 3 / 2$	64×64
2x	Convolutional	64	1×1	
	Convolutional	128	3×3	
	Residual			64×64
	Convolutional	256	$3 \times 3 / 2$	32×32
8x	Convolutional	128	1×1	
	Convolutional	256	3×3	
	Residual			32×32
	Convolutional	512	$3 \times 3 / 2$	16×16
8x	Convolutional	256	1×1	
	Convolutional	512	3×3	
	Residual			16×16
	Convolutional	1024	$3 \times 3 / 2$	8×8
4x	Convolutional	512	1×1	
	Convolutional	1024	3×3	
	Residual			8×8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Fig:5.7. Backbone darknet53 network model.

5.4.1. BOUNDING BOX OPTIMIZATION

Logistic regression for YOLO v3 predicts product scores. The ground truth box is used to identify each object, and the object is considered to have a score of 1 if the junction box most closely matches the ground truth box. Because the junction box overlaps more than the starting point, there is no point for the junction box value. If no junction box is selected and placed in that box, only trust is excluded. The anchor box returns, improving the original box on the

ground as shown in the image below. The available coordinates are defined as follows:

$$bb_x = \partial(p_x) + d_x \quad (5)$$

$$bb_y = \partial(p_y) + d_y \quad (6)$$

$$bb_w = t_w f^{pw} \quad (7)$$

$$bb_h = t_h f^{ph} \quad (8)$$

Where, p_x, p_y, p_w, p_h are the prediction generated by YOLO. d_x, d_y are located at the anchor's grid cell's upper left corner. t_w, t_h are the anchor's width and height. The estimated bounding box are bb_x, bb_y, bb_w, bb_h . $\partial(p_0)$ is denote as bounding box confidence score.

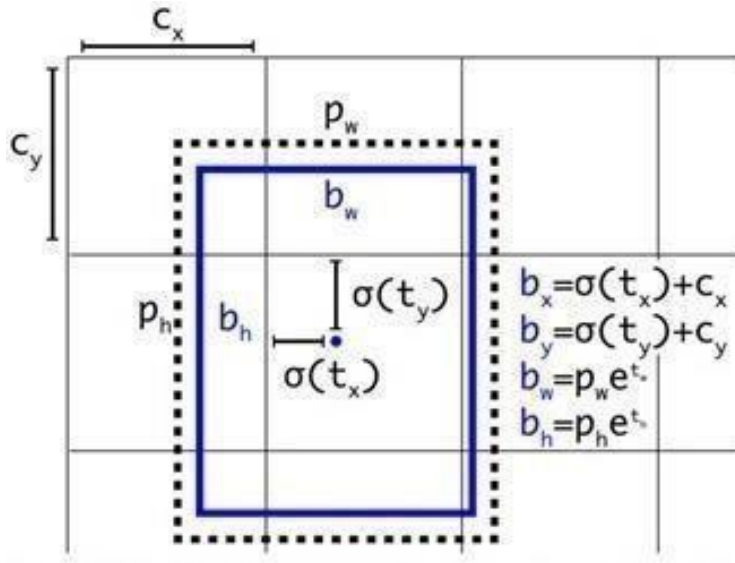


Fig:5.8. Anchor box regression.

5.4.2 FEATURE PYRAMID NETWORK (FPN)

YOLOv3 provides three predictions for each point in the image. We have an estimate $S \times S \times [3 \times (4 + 1 + 80)]$ because the bounding box, negative score, and class score all 80 are included in the estimate. Yes. A typical pyramid network is associated with this concept. Three variables are used to make the estimation, as shown in the figure below. The initial estimate is made in the final process report. Then the feature map is sampled by either one. It uses YOLOv3 supplements to integrate maps and features. The convolutional layer implementation does the second guess. Repeating the second guess takes more semantic information.

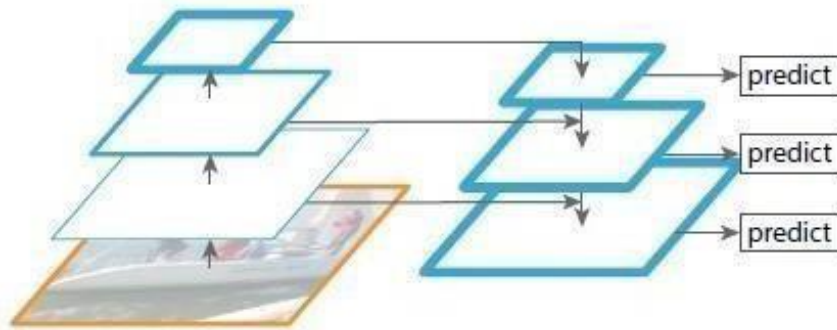


Fig:5.9. Feature pyramid network (FPN).

YOLOv3 is fast and accurate in mean precision (mAP) and intersection over union (IOU). It works faster than other search methods.

5.5. YOLOv5

The COCO dataset is used to train the YOLOv5 series hybrid information models, which also have core functionality of trial time augmentation (TTA), model clustering, hyperparameter enhancement, and deployment for ONNX, CoreML, and TFLite.

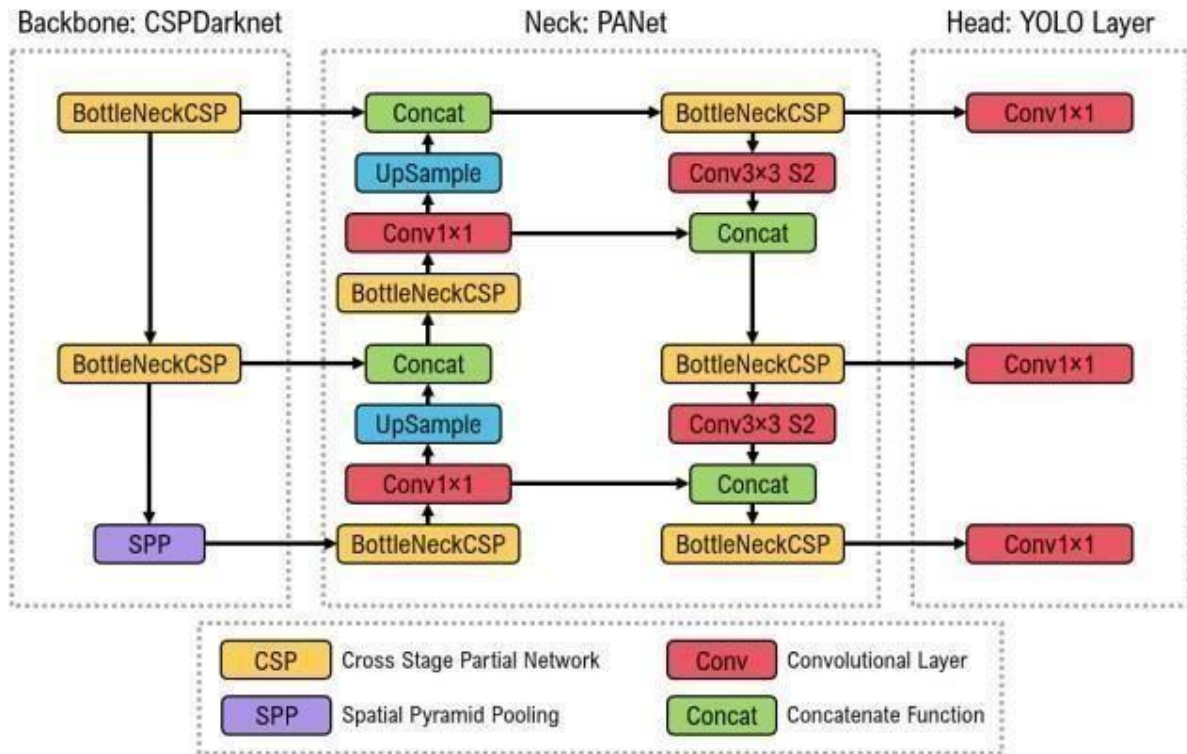


Fig:5.10. The network architecture of YOLOv5.

It consists of three modules: CSPDarknet backbone, PANet backbone, and YOLOv3 (anchor-based) header. Input data also includes junction box computation and tessellation amplification data, which is first fed into CSPDarknet for feature extraction and then fed into PANet to improve data flow. Finally, Head provides benefits such as bounding boxes and class prediction after using Neck's features. Spatial pyramid pool (SPP) and concatenation (Concat) are used here. According to the figure above, the network topology of YOLOv5 is divided into three main groups. The principle is therefore a CNN that collects and

generates image features at various levels of detail. YOLOv5 basically uses a cross-level local area network (CSPNet) to generate image features. With fewer parameters and FLOPS of the model, CSPNet solves the gradient back problem in deeper CNNs by improving inference speed and accuracy while reducing model size. After CSP, the network adds an SPP block to remove the limit on the size of the input image. The SPP block is used to calculate features from the entire image once and then combines features in each region (subimage) to expand the acquisition area and create a long-term representation of detector training. In the model, the neck consists of several layers that collect the information found and send it to the search steps. It uses the Path Aggregation Network (PANet) to improve the data flow process. Bottom-up method augmentation specifically refines feature pyramids with low-level precise localization signals by shortening the information channel of low-level and high-level features. Meanwhile, the adaptive feature pool combines the feature grid and the entire process, transferring useful information from each feature set directly to the next recommendation. Final reconnaissance is usually done by the head of the model. It uses feature junction boxes and bounding box regression produces the final output vector with item scores and class scores. The same anchor is used in YOLOv5 as in YOLOv3. Based on the amount of memory storage (not included), YOLOv5 also includes four models from small to large: YOLOv5s (smallest we used in this post), YOLOv5m (medium), YOLOv5l (large), and YOLOv5x (extra-large, most important). The MS COCO training file was used to train all four models.

5.6 YOLOv8

YOLO v8 is a modern model with new features that increase performance and versatility. Makes requests faster and more accurate

- Object Detection,
- Instance Segmentation, and
- Image Classification.

Here are some key features about the YOLOv8:

- User friendly Python and command line API.
- Faster and more accurate.
- Supports image classification, segmentation, and object discovery.
- Compatible with all previous versions.
- Fresh bones.
- The new title has no anchor.
- Missing new functionality.

Moreover, YOLOv8 is efficient and flexible, supports multiple delivery modes and runs on CPU and GPU. Compared to YOLO v5, YOLO v8 performed better 64% of the time. All YOLO v8 models are trained. Detection and segmentation models are pre-trained on the COCO dataset, while classification models are pre-trained on the ImageNet dataset.

5.6.1 MODELS AVAILABLE IN YOLOv8

For detection, segmentation and classification, there are five models in each category of the YOLOv8 model. The smallest and fastest is the YOLOv8 Nano, while the YOLOv8 Extra Large (YOLOv8x) is the most sensitive but also the slowest.

YOLOv8n	YOLOv8s	YOLOv8m	YOLOv8l	YOLOv8x
---------	---------	---------	---------	---------

Fig 5.11. Model Available in YOLOv8

YOLOv8 comes bundled with the following pre-trained models:

- An object inspection site trained using COCO's exploration data and 640 images.
- Example segmentation checkpoint created using the 640x480 COCO segmentation dataset.
- Pre-trained image classification models with 224pixel image resolution on the ImageNet dataset.



Fig:5.12. Object Detection Using Yolov8

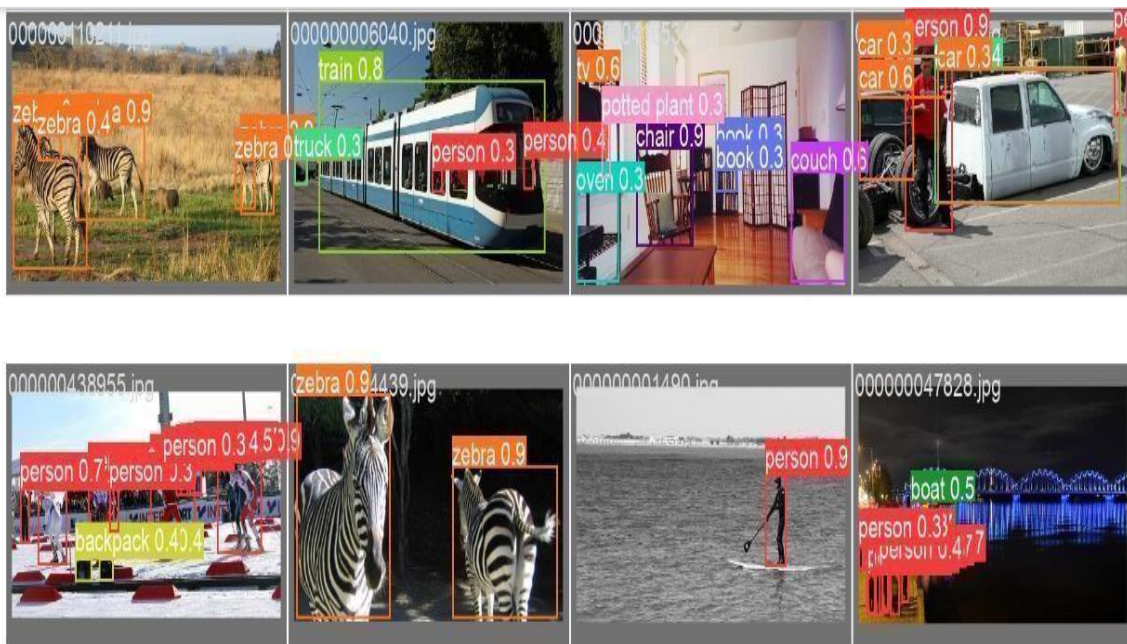


Fig:5.13. Object Detection Using Yolov8

5.6.2 YOLOV8 ARCHITECTURE

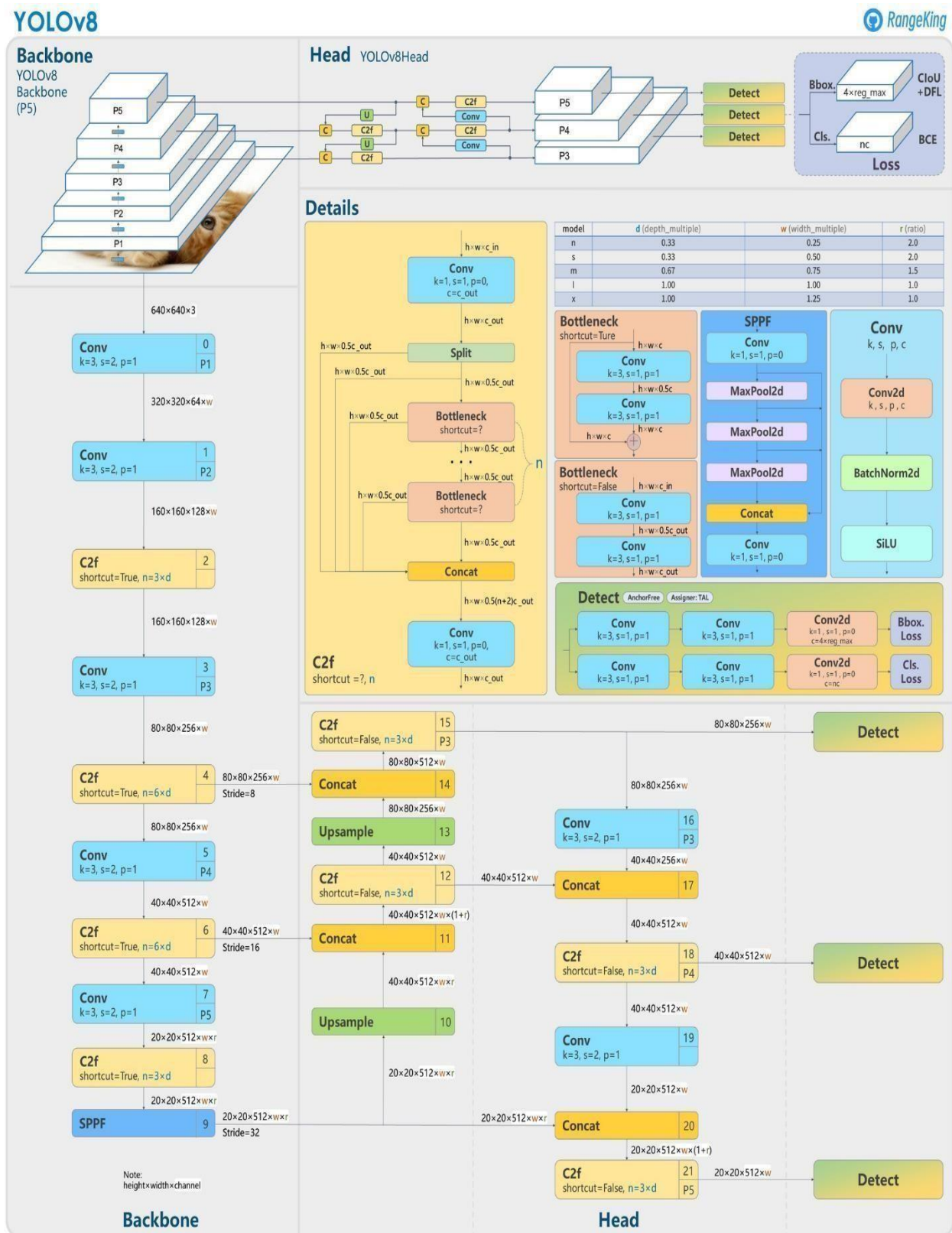


Fig:5.16. YOLOv8 Architecture

5.6.3 ANCHOR FREE DETECTION

- The YOLOv8 model does not have an anchor. In other words, it directly estimates the center of the object instead of estimating the shift of the object relative to the junction box.
- Non-Maximum Suppression (NMS), a complex post-processing step that divides search candidates after computation, can be done faster as there are fewer prediction boxes because there is no object linkage.

5.6.4 NEW CONVOLUTIONS

- The first 6x6 transformation in the hull changes to 3x3, and C2f changes to C3 in the main house. The figure below is a schematic diagram; where "f" represents the feature number, "e" represents the expansion value and CBS is a block containing the following Conv, Batch Norm and SiLU.
- All the results of the flask, ie the two 3x3 convolutions now connected, are combined in C2f. However, only the output of the previous bottle was used in C3.
- The packet size of the first conversion has been changed from 1x1 to 3x3, but the bottleneck is still the same as YOLOv5. From this information, we can deduce that YOLOv8 returns to the ResNet block made in 2015.
- Features attach directly to the neck without forcing them to have the same channel size. By doing this, the number of parameters and the size of the tensor are all reduced. The packet size of the first conversion has been changed from 1x1 to 3x3, but the bottleneck is still the same as YOLOv5. From this information, we can deduce that YOLOv8 returns to the ResNetblock made in 2015.

5.6.5 CLOSING THE MOSAIC AUGMENTATION

- While the architectural model is often the focus of deep research, the educational process in YOLOv5 and YOLOv8 is critical to its outcomes.
- YOLOv8 enhances images when you exercise online. The standard view of the image is slightly different each time.
- The Mosaic enhancement is an improvement. As a result of merging the four images, the model is forced to learn for a lot of things in and around the pixels in a new partially enclosed space.

CHAPTER 6

SYSTEM REQUIREMENTS

6.1. HARDWARE REQUIREMENTS

- System: Pentium i3 Processor.
- Hard Disk: 500 GB.
- Monitor: 15" LED
- Input Devices: Keyboard, Mouse
- Ram: 4 GB

6.2. SOFTWARE REQUIREMENTS

- Operating system: Windows 10.
- Coding Language: Python 3.11.

6.3. SOFTWARE ENVIRONMENT

A common high-level programming language is Python. Code readability is important in its design and uses indentation heavily. Python uses garbage collection and has dynamic typing. It supports a variety of operations and operating procedures, including methods, object orientations, and operation functions (which is unique). It is often referred to as a "battery built-in" language because of its extensive library.

Install Python on your computer system

1. You can use Image AI and TensorFlow, NumPy, OpenCV etc. Install necessary third-party software
2. Save the sample search file (YOLOv8) to your computer. Instructions to follow:
 - 1) Go to the official Python website <https://python.org> and download and install Python version 3.

2) Use pip to install the above dependencies:

i. TensorFlow:

An open-source package called TensorFlow is used for data flow and different programming for various tasks. It is a code library that can be used in machine learning like neural networks. Google uses it for research and development. The Google Brain team developed TensorFlow for use at Google. It was made available under the Apache License 2 on November 9, 2015. The second version of the 0. Google Brain system is called TensorFlow. The first version of TensorFlow was released on February 11, 2017. While TensorFlow can run on multiple CPUs and GPUs (with optional CUDA and SYCL extensions for computational graphics processing), app usage only supports one device. TensorFlow is compatible with many operating systems including 64-bit Linux, macOS, Windows, and mobile operating systems such as Android and iOS. TensorFlow's architecture makes it easy to deploy computations across multiple platforms (CPUs, GPUs and TPUs), from PCs to servers to mobile and edge devices. Stateful dataflow graphs are used to represent calculations in TensorFlow. These neural networks work on different datasets called tensors, from which TensorFlow is named.

ii. NumPy:

The NumPy package for the Python programming language adds support for many different numbers and numbers, as well as many advanced arithmetic operations that can be performed on these arrays. NumPy's predecessor, Numeric, was originally developed by Jim Hugunin with the help of many other developers. Travis Olphant created NumPy in 2005, continuing to combine and replace the Num array functionality with Numeric. Many people have contributed to the open source NumPy program.

iii. SciPy:

SciPy modules cover multiple optimizations, linear equations, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers, and other function engineering. SciPy is part of the NumPy family, which also includes many scientific libraries and tools such as Matplotlib, pandas, and SymPy. SciPy mostly abstracts NumPy array objects. This NumPy stack can be used in parallel with programs such as MATLAB, Octave, and Scilab. SciPy cluster is another name for NumPy cluster.

The SciPy library is currently available under the BSD license and supports and supports open-source community development. Additionally, Num FOCUS is a community-based organization that supports both reversible and viable research.

iv. OpenCV:

A set of programming tools called OpenCV mainly focuses on real-time computation. It was originally developed by Intel and later supported by Willow Garage and Itseez. The cross-platform and open-source BSD license makes the library free to use.

v. Pillow:

Python Imaging Library is a free library for the Python programming language that supports opening, editing, and saving various types of image files. This feature is supported on Linux, Mac OS X and Windows.

vi. Matplotlib:

Python's NumPy numeric extension and Matplotlib plotting tool are written in Python. It provides an object-oriented API for GUI tools such as Tkinter, wxPython, Qt, or GTK+, which can be used to pull images into projects.

vii. H5py:

Python's NumPy numeric extension and Matplotlib plotting tool are written in Python. It provides an object-oriented API for GUI tools like Tkinter, and the h5py program provides high-level and low-level APIs for Python's HDF5 library. High-level components use well-designed Python and NumPy to support access to HDF5 data, files, and groups, while the low-level interface should be a full version of the HDF5 API.

The simplifies the process of reading and writing data from Python by providing priority automatic conversion between Python (NumPy) data types and data structures and their equivalent HDF5. wxPython, Qt or GTK+ can be used to embed projects in projects.

viii. Kera's:

Python-based Kera's is an open-source neural network library. It can run on Microsoft Cognitive Toolkit, Theano, PlaidML or TensorFlow. User friendliness, modularity and scalability are design goals as it aims to facilitate rapid experiments with deep neural networks.

ix. Image AI:

Image AI provides an API that identifies 1000 different objects in an image using pre-trained models built on the ImageNet-1000 dataset. Examples of Squeeze Net, ResNet, InceptionV3, and Dense Net are given.

3) Download the YOLOv8 model file that will be used for object detection.

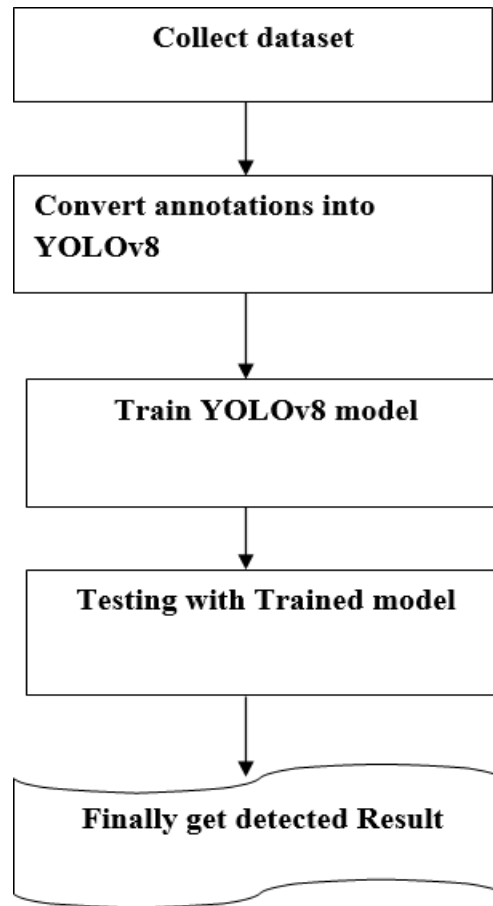
CHAPTER 7

SYSTEM DESIGN

7.1. DATA FLOW DIAGRAM

Bubble diagram is another name of DFD. It is a simple graphical formalism that can be used to describe a system in terms of input data, different operations on the system, and data outputs from these operations.

1. One of the most important modeling tools is the Data Flow Diagram (DFD). The standard component of the system is designed to use it. This content includes the operation of the system, the information it uses, the third parties associated with it and the information transmitted through it.
2. DFD shows how data enters the system and how it affects how various variables change. It is a visualization tool that shows how information flows and changes as it moves from input to output.
3. DFD, also known as bubble diagram. A system using DFD can be described with any level of abstraction. DFD can be split into parallel levels for increased data throughput and processing complexity.



7.2. UML DIAGRAMS

Unified Modeling Language, or UML, it's called. The standard design model in software engineering is UML. The Product Management Team oversees the model and is responsible for its creation. The ultimate goal of UML is to become standard language for modeling computer-oriented objects. Metamodel and notation are two important parts of UML today. In the future, UML will also take method or process. A language for describing, visualizing, designing, and documenting components of software systems, including business models and non-software, is Unified Modeling Language. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

Both the product-oriented software design process and the software design process are based on UML. UML often uses diagrams to express software design.

GOALS:

The Primary goals in the design of the UML are as follows:

1. Provide users with ready-to-use presentation visual language templates so they can create and exchange useful templates.
2. Provide specific tools and equipment to expand the basic idea.
3. It is not tied to a particular development method or programming language.
4. Provide a basis for understanding language patterns.
5. Contribute to the continued commercialization of pointing devices.
6. Encourage the use of high-level development concepts such as objects, patterns, models, and collaboration.

7.3. USE CASE DIAGRAM

Unified Modeling Language (UML) uses the term "use case diagram" to refer to a type of behavior defined and derived from a use case study. Its purpose is to provide a description of the work the system provides about the actors, their purpose (represented as the use of information), and the impact of that use. The use of the diagram shows what work the player is doing as their main goal. The roles played by the participants in the system can be mentioned.

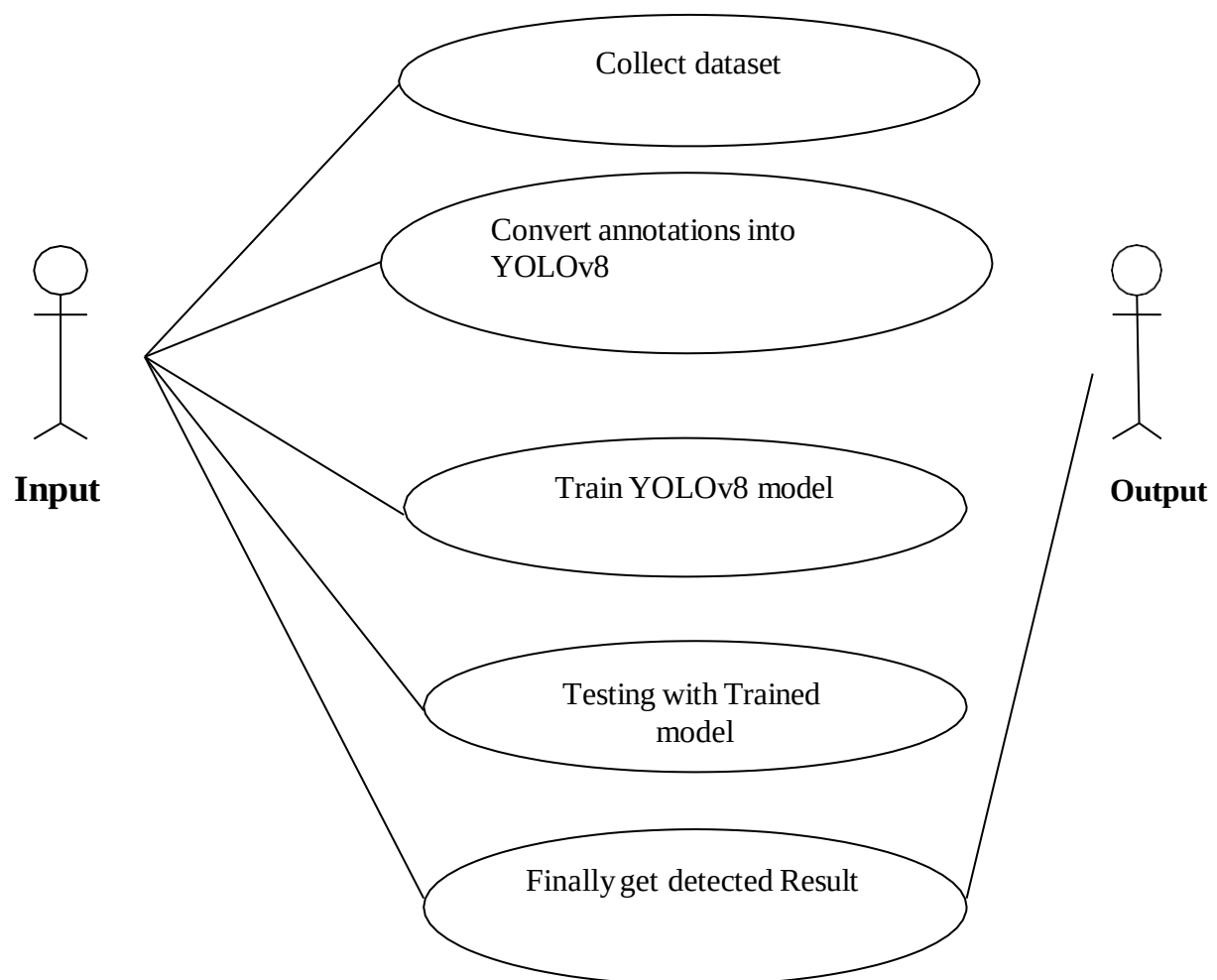


Fig:7.1. Use Case Diagram

7.4. CLASS DIAGRAM

In software engineering, a class diagram in Unified Modeling Language (UML) is a diagram of a static model that describes classes, their attributes, functions (or methods), and the interaction between classes to explain the structure of the system. It shows exactly what the data looks like.

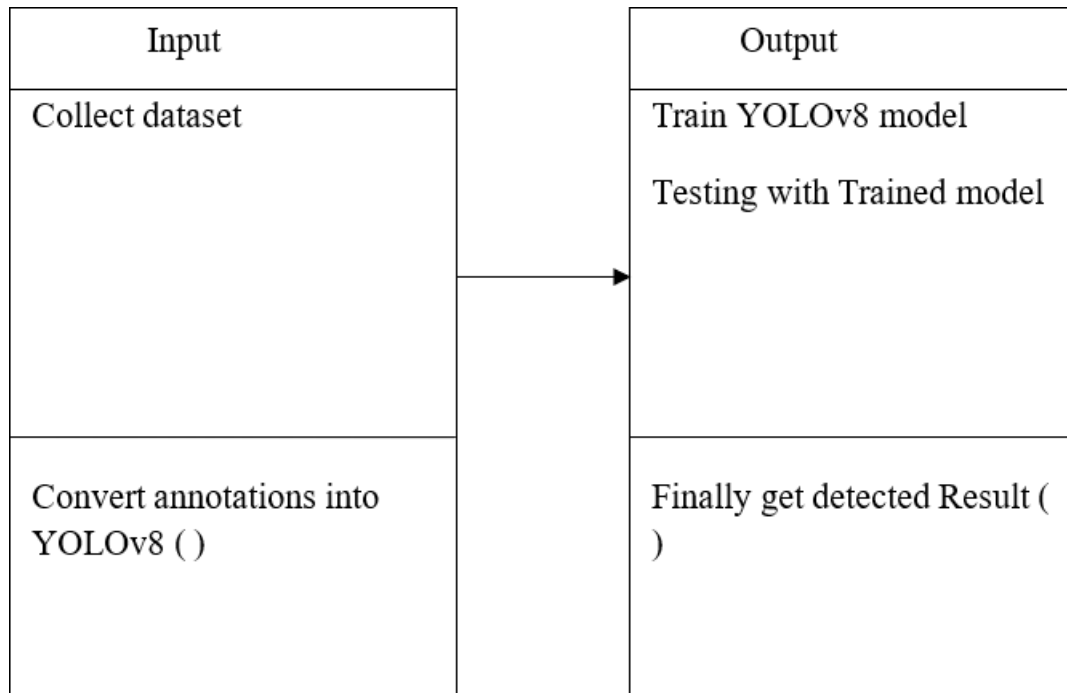


Fig:7.2. Class Diagram

7.5. SEQUENCE DIAGRAM

In Unified Modeling Language (UML), a system diagram is an interaction diagram that shows how and in what order they interact with each other. Sentence structure is structure. Event diagram, case scenario, and sequence diagram are other names for sequence diagram. Software engineers and business professionals use diagrams to understand the details of new systems or to explain existing systems. Charts and event charts are other names for system charts.

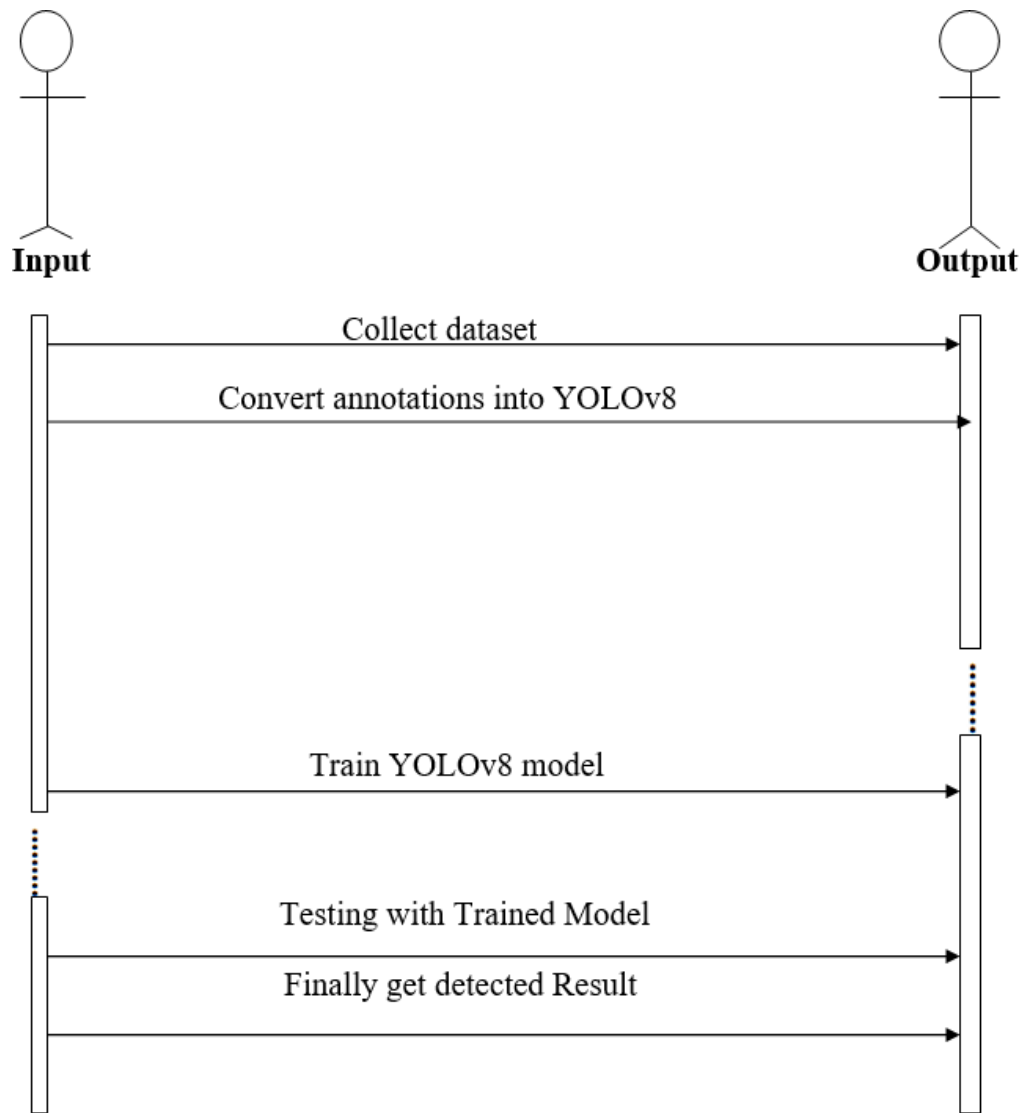


Fig:7.3. Sequence Diagram

7.6. ACTIVITY DIAGRAM

A diagram is a description of the job, including the sequence of tasks and operations, including selection, repetition, and balance. Game maps can be used in Unified Modeling Language to describe step-by-step the operation of physical components and business operations. All flow controls are shown in the diagram.

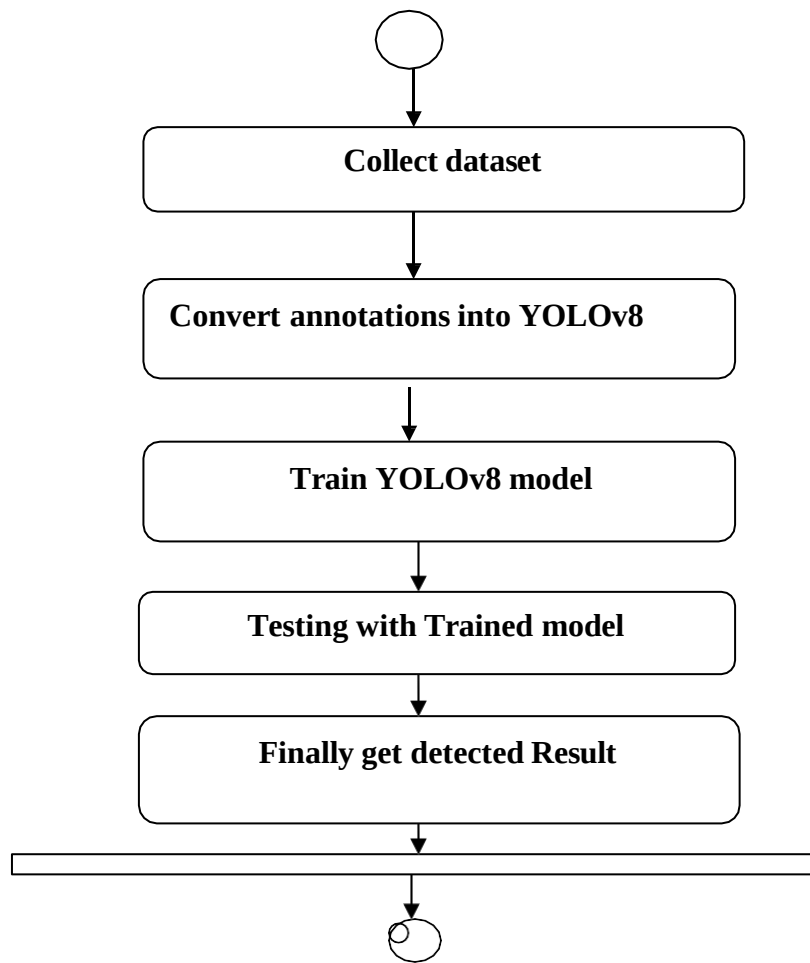


Fig:7.4. Activity Diagram

7.7. INPUT DESIGN

Concept design acts as a conduit between the user and the information system. The process for the preparation of information and should be specifically designed. This step is necessary to convert the data transfer into an operable form. This can be done by watching a computer read information from a written or printed document, or by entering users' information directly into the system. The proposed system is designed with a focus on reducing required equipment, reducing errors, reducing delays, reducing sub-levels and controlling business flexibility. The equipment is designed to provide safety, usability and personal protection. The design strategy takes into account the following factors:

- What information should be provided as input?
- How should data be prepared or encoded?
- Drives the operator input dialog.
- Create procedures for accessing authentication and procedures in the event of an error.

OBJECTIVES

1. The user-oriented description of the input input is translated into the computer -based system through the input creation process. This design is important to avoid errors when entering data and to provide management instructions to retrieve correct data from computer systems.

2. By creating charts that are easy to use and can handle many data entries. According to the design strategy, the input data should be simple and error-free. Due to its design, all applicable data can be processed on the input data. A file viewer is also available.

3. After the data is entered, it checks the accuracy of the data. Information can be accessed using the screen. In this way, the user does not have to search for sending the appropriate message instantly when necessary. Creating easy-to-understand input layouts is the goal of input design.

7.8. OUTPUT DESIGN

Output that satisfies the user's needs and conveys information directly is considered good. Each system has its output so that users and other systems know what results are being received. In the creation of the product, it is determined how the changing information and hard copy will be put on the market according to the urgent needs. It is important, as information directly to users. By creating efficient and smart products, the system's ability to help users make decisions is enhanced.

1. The process of creating a computer-generated product should be organized and well thought out; In order for the body to be used quickly and efficiently, the right products must be created while ensuring that all products released to the market are created. When analyzing computer output, identify the exact output required to meet specifications.
2. Decide how to send the message.
3. Generate reports, documents or other documents containing the generated information.

The output form of an information system should accomplish one or more of the following objectives.

- ❖ Share detailed information about past actions, current events or future predictions.
- ❖ Represents an important time, opportunity, problem or warning.
- ❖ Causes an obstacle.
- ❖ Authentication Event.

CHAPTER 8

SYSTEM STUDY

8.1. FEASIBILITY STUDY

At this stage the feasibility of the project is evaluated and a business proposal is developed with a fairly simple plan and some cost estimates. As part of the systems analysis, the feasibility assessment of the proposal should be made. By doing this, he ensured that there was no burden on the company's planning process. In order to perform the analysis, the basic requirements of the system must be known.

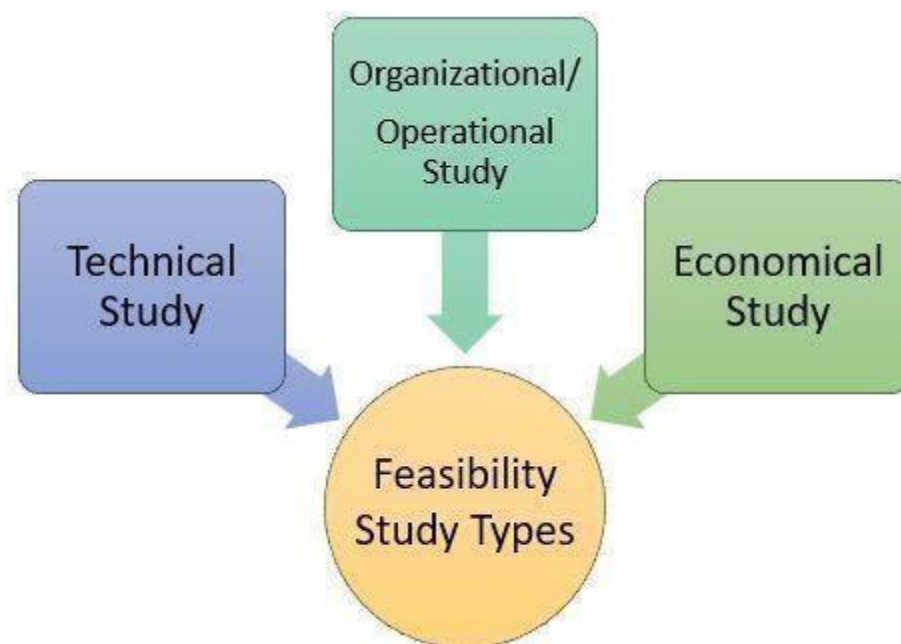


Fig:8.1. Types of Feasibility Study

Three key considerations involved in the feasibility analysis are

- ◆ ECONOMICAL FEASIBILITY
- ◆ TECHNICAL FEASIBILITY
- ◆ SOCIAL FEASIBILITY

8.1.1. ECONOMICAL FEASIBILITY

This study looks at what the company's financial impact will be on the company. Companies have less money for system development. Debts must be supported by evidence. As a result, the design process is within budget as most of the equipment is in the public domain. Only special items need to be purchased.

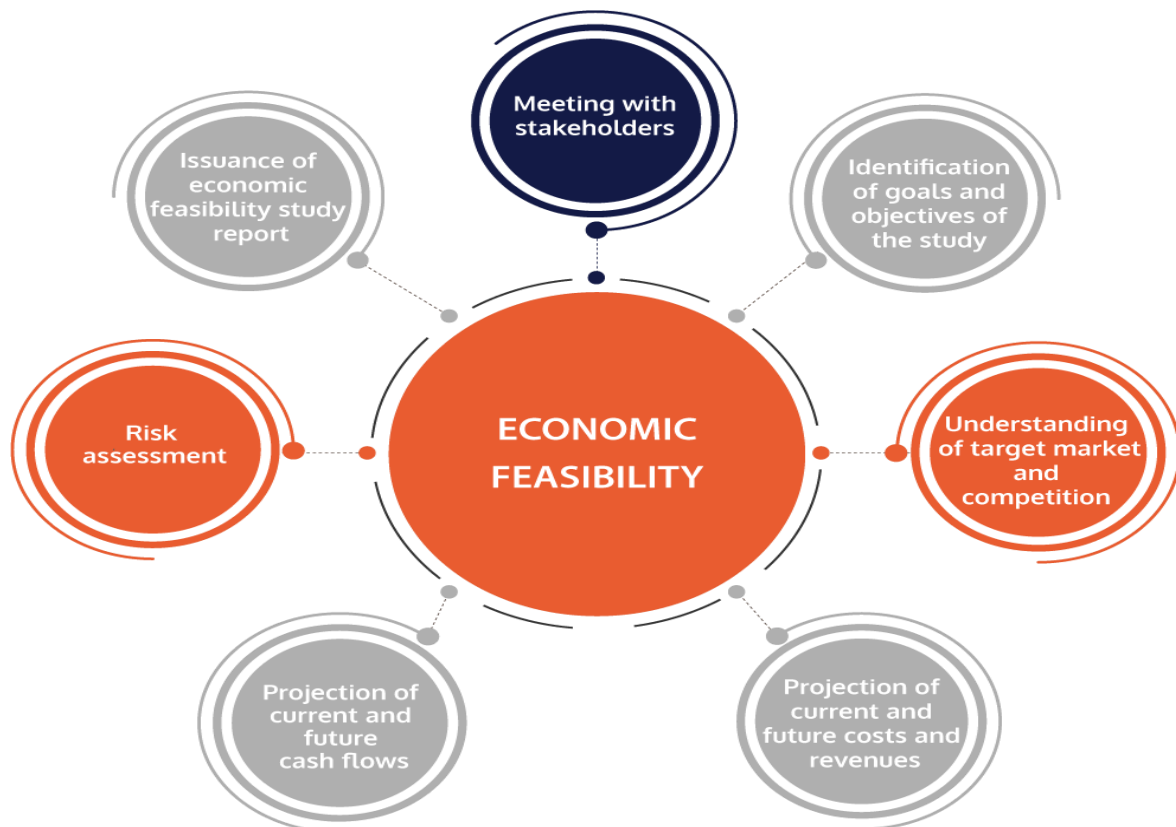


Fig:8.2. Economic Feasibility

8.1.2. TECHNICAL FEASIBILITY

In other words, as part of this study, rules were analyzed to determine the effectiveness of the system. Any system design should not strain available resources. As a result, the market value will be very high. Therefore, customers will face high expectations. Because the system created by the application requires little or no modification, there should be very little demand.

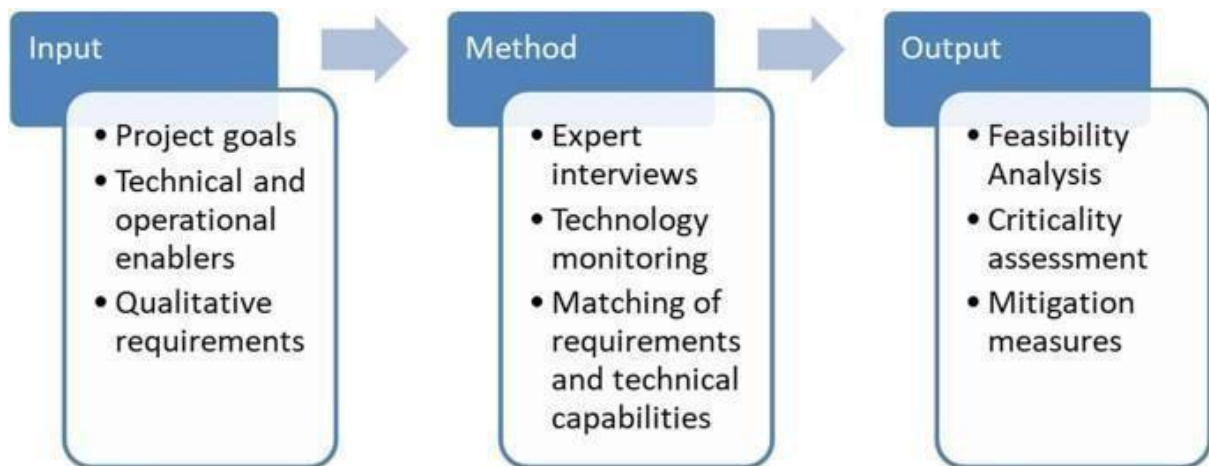


Fig:8.3. Technical Feasibility

8.1.3. SOCIAL FEASIBILITY

The purpose of the research is to examine the user's acceptance of the system. This includes the training process for users to use the system correctly. Users should not be threatened by the system, but should be accepted as necessary. User acceptance is dependent on the application for the user to introduce and get to know the system. As he is the end user of the system, his confidence level should be raised so that he can still take some criticism.

CHAPTER 9

SYSTEM TESTING

Locating mistakes is the aim of trying out. The system of testing includes seeking out any capability flaws or weaknesses in a chunk of work. It gives a way of evaluating the capability of elements, subassemblies, assemblies, and/or a completed excellent. in an effort to make certain that the software device satisfies its necessities and person expectancies and does now not fail in an unacceptable manner, it should be positioned via a manner known as software checking out. there are many distinctive take a look at types. each test kind specializes in a positive checking out requirement.

9.1.1. UNIT TESTING

Unit testing involves creating states that check that internal processes are working properly and program inputs produce usable products. All branch decisions and internal code flow must be defined. It is a test of individual software components of an application. It is done after each unit before integration. This is an evaluation model based on information about its structure and impact. Unit testing performs simple tests at the component level and checks the specific configuration of a system, application or business process. The testing unit verifies that each particular method of the business process works according to the published instructions and has the correct instructions and expected results.

9.1.2. INTEGRATION TESTING

Integration software products are tested collaboratively to see if they work as a single program. Evaluation is event-driven and focuses more on the results of the screens or areas. Although the product combination has been tested successfully, the combination shows that the product combination is correct and consistent. The purpose of integration testing is to identify problems that arise when combining components.

9.1.3. FUNCTIONAL TESTING

Functional testing provides evidence that functional testing is compliant with business needs and requirements, system documentation, and user guides. performance measures focus on:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and planning of job tests focusing on needs, important tasks or specific situations. In addition, the assessment should consider the scope of the data area, the previous and subsequent procedures, and the work process. Additional experiments are discovered before the experimental work is completed and the results of existing experiments are evaluated.

9.1.4. SYSTEM TEST

System testing ensures that the integrated software is fully compliant with specifications. Evaluates settings to provide recognizable and visible results. Configuration-oriented system integration testing is an example of system testing. System evaluation is based on methods and descriptions that focus on the elements before integration and connection.

9.1.5. WHITE BOX TESTING

White box testing is a type of testing where the software tester knows the inner workings, structure and language of the software, or at least what he wants to do. It has a purpose. It is used to measure inaccessible areas at the black box level.

9.1.6. BLACK BOX TESTING

In a "black box" software testing is done without knowledge of the inner workings, architecture or language of the model under test. Like many other tests, black box testing must be based on explicit information such as instructions or

data requirements. It is a type of testing in which the tested software is treated like a black box. I can't "see" the inside. Regardless of the functionality of the software, tests generate input and respond to results.

9.1. UNIT TESTING

While it is not unusual for coding and unit testing to be performed as two separate phases, unit testing is often done as part of the mixed code and unit testing phase of the software lifecycle.

Test strategy and approach

An on-site assessment will be completed and full operational testing scheduled.

Test objectives

- All entries must be correct.
- The specified link must be used to open the page.
- Delayed access to screens, messages or replies.

Features to be tested

- Make sure the data entry is correct.
- Do not enter text.
- All links should direct the user to the correct page.

9.3 INTEGRATION TESTING

Incremental testing of two or more integrated software components on a single platform to reduce the effects of interoperability is called "software integration testing". The purpose of integration is to ensure that software applications or devices, such as those found in the software system, or, at a higher level, enterprise-wide software applications do not work together badly.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

9.4 ACCEPTANCE TESTING

User acceptance the testing phase of any project is important and requires the involvement of end users. It also ensures that the system meets certain requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

CHAPTER 10

CONCLUSION

Object detection is a critical skill for most computer and robot vision systems. While significant progress has been made in recent years with some technologies found in many consumer electronics (such as face detection for autofocus in smartphones) or incorporated into driving aid, we are still far from reaching the human goal. Especially the level of quality in open world education. Deep learning-based object detection has become a research hotspot in recent years because of its learning potential and advantages of dealing with occlusion, scale conversion, and background offset. This s collision and bounded problem-solving multiple problems in the YOLO new model. As we train and develop the YOLOV7 object detection model to detect our objects (car, truck, bus, etc.), the DEEP-SORT algorithm will be responsible for tracking vehicles identified by the framework. The development of traffic management in smart cities and video surveillance for driverless vehicles is the sole goal of the final project. The algorithm has high recognition rate and fast recognition in difficult situations and bad weather conditions. YOLOv5s greatly reduces the storage space of the model while still running very fast. We combined YOLOv7 with deep-SORT as an object search network to create YOLOv7-Deep SORT. Experiments show that this network is more accurate in tracking than YOLOv5-DeepSORT. Experimental results of complex data analyzed at different camera heights, angles and rear view of the vehicle demonstrate the flexibility and accuracy of road preparation. One initiative that will improve the reliability of the system in the future is the distribution of tools. The inclusion of a classification system is very helpful as it further improves detection performance.

REFERENCE

- [1] Z. Q. Zhao, P. Zheng, S. T. Xu, and X. Wu, “Object Detection with Deep Learning: A Review,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212–3232, 2019, Doi: 10.1109/TNNLS.2018.2876865.
- [2] G. Chandan, A. Jain, and H. Jain, “Real Time Object Detection and Tracking Using Deep Learning and OpenCV,” *2018 Int. Conf. Inven. Res. Comput. Appl.*, no. Icirca, pp. 1305–1308, 2018.
- [3] M. Yazdi and T. Bouwmans, “New trends on moving object detection in video images captured by a moving camera: A survey,” *Comput. Sci. Rev.*, vol. 28, pp. 157–177, 2018
- [4] C.-Y. Wang, A. Bochkovskiy, and H.-Y.-M. Liao, “Scaled-YOLOv4: Scaling cross stage partial network,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 13024–13033.
- [5] Y. Tian, P. Luo, X. Wang, and X. Tang, “Deep learning strong parts for pedestrian detection,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1904–1912.
- [6] C. Xie, P. Li, and Y. Sun, “Pedestrian Detection and Location Algorithm Based on Deep Learning,” *Proc. - 2019 Int. Conf. Intell. Transp. Big Data Smart City, ICITBS 2019*, pp. 582–585, 2019, Doi: 10.1109/ICITBS.2019.00145.
- [7] S. Sumit, J. Watada, A. Roy, and D. Rambli, in *object detection deep learning methods*, Y O L O shows supremum to Mask R-CNN. 2019.

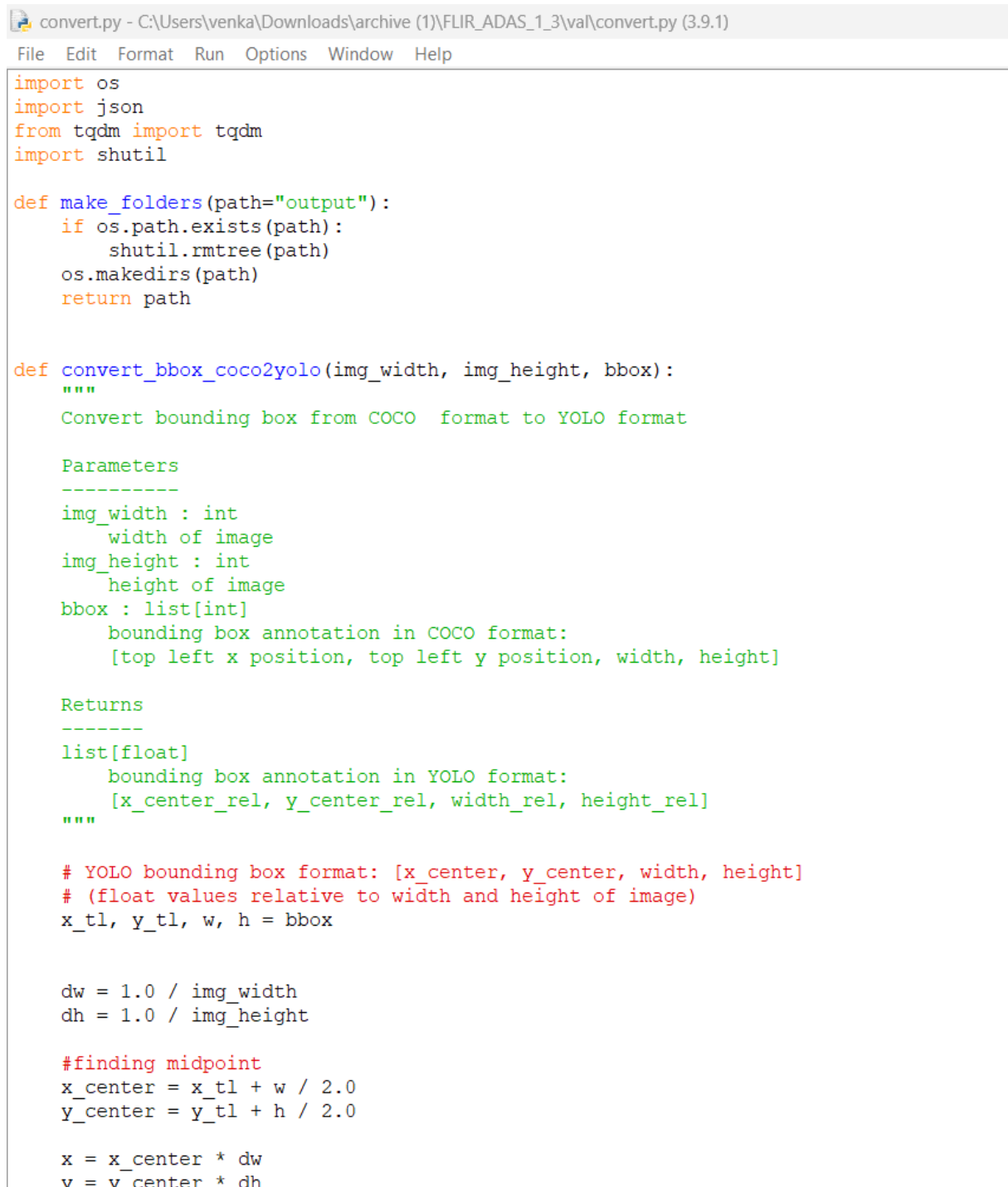
- [8] R. Huang, J. Pedoeem, and C. Chen, “YOLO-LITE: a real-time object detection algorithm optimized for non-GPU computers,” in 2018 IEEE International Conference on Big Data (Big Data), 2018, pp. 2503–2510.
- [9] A. Bochkovski, C. Wang, and H. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection.”
- [10] Y. Jin, “Embedded Real-Time Pedestrian Detection System Using YOLO Optimized by LNN,” no. June, pp. 1– 5, 2020.
- [11] G. Li, Y. Yang, and X. Qu, “Deep learning approaches on pedestrian detection in hazy weather,” *IEEE Trans. Ind. Electron.*, vol. 67, no. 10, pp. 8889 – 8899, 2020, doi: 10.1109/TIE.2019.2945295.
- [12] C. Kwan and B. Budavari, “Enhancing small moving target detection performance in low-quality and long-range infrared videos using optical flow techniques,” *Remote Sens.*, vol. 12, no. 24, p. 4024, Dec. 2020.
- [13] C. Kwan and J. Larkin, “Detection of small moving objects in long range infrared videos from a change detection perspective,” *Photonics*, vol. 8, no. 9, p. 394, Sep. 2021.
- [14] C. Kwan and B. Budavari, “A high-performance approach to detecting small targets in long-range low-quality infrared videos,” *Signal, Image Video Process.*, vol. 16, no. 1, pp. 93–101, Feb. 2022.
- [15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Proc. NIPS*, 2012, pp. 1097–1105.

- [16] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial pyramid pooling in deep convolutional networks for visual recognition,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 9, pp. 1904–1916, Sep. 2015.
- [17] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.
- [18] W. Liu et al., “SSD: Single shot multibox detector,” in *Computer Vision—ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham, Switzerland: Springer, 2016, pp. 21–37.
- [19] J. Redmon and A. Farhadi, “YOLO9000: Better, faster, stronger,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 6517–6525.
- [20] K. Ehsani, H. Bagherinezhad, J. Redmon, R. Mottaghi, and A. Farhadi, “Who let the dogs out? Modeling dog behavior from visual data,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4051–4060.
- [21] M. S. Uddin, R. Hoque, K. A. Islam, C. Kwan, D. Gribben, and J. Li, “Converting optical videos to infrared videos using attention GAN and its impact on target detection and classification performance,” *Remote Sens.*, vol. 13, no. 16, p. 3257, Aug. 2021.
- [22] C. Kwan, B. Chou, J. Yang, and T. Tran, “Deep learning-based target tracking and classification for infrared videos using compressive measurements,” *J. Signal Inf. Process.*, vol. 10, no. 4, pp. 167–199, 2019.

- [23] M. Beauvais and S. Lakshmanan, “CLARK: A heterogeneous sensor fusion method for finding lanes and obstacles,” *Image Vis. Comput.*, vol. 18, no. 5, pp. 397–413, Apr. 2000.
- [24] B. Steux, C. Laurgeau, L. Salesse, and D. Wautier, “Fade: A vehicle detection and tracking system featuring monocular color vision and radar data fusion,” in *Proc. Intell. Vehicle Symp.*, Jun. 2002, pp. 632–639.
- [25] L. Bombini, P. Cerri, P. Medici, and G. Aless, “Radar-vision fusion for vehicle detection,” in *Proc. Int. Workshop Intell. Transp.*, vol. 3, 2006, pp. 65–70.
- [26] J.-G. Wang, S. J. Chen, L.-B. Zhou, K.-W. Wan, and W.-Y. Yau, “Vehicle detection and width estimation in rain by fusing radar and vision,” in *Proc. 15th Int. Conf. Control, Autom., Robot. Vis. (ICARCV)*, Nov. 2018, pp. 1063–1068.
- [27] V. John and S. Mita, “RVNet: Deep sensor fusion of monocular camera and radar for image-based obstacle detection in challenging environments,” in *Image and Video Technology*, C. Lee, Z. Su, and A. Sugimoto, Eds. Cham, Switzerland: Springer, 2019, pp. 351–364.

APPENDIX

Converting COCO Json Format Into Yolo Format:



```
convert.py - C:\Users\venka\Downloads\archive (1)\FLIR_ADAS_1_3\val\convert.py (3.9.1)
File Edit Format Run Options Window Help

import os
import json
from tqdm import tqdm
import shutil

def make_folders(path="output"):
    if os.path.exists(path):
        shutil.rmtree(path)
    os.makedirs(path)
    return path

def convert_bbox_coco2yolo(img_width, img_height, bbox):
    """
    Convert bounding box from COCO format to YOLO format

    Parameters
    -----
    img_width : int
        width of image
    img_height : int
        height of image
    bbox : list[int]
        bounding box annotation in COCO format:
        [top left x position, top left y position, width, height]

    Returns
    -----
    list[float]
        bounding box annotation in YOLO format:
        [x_center_rel, y_center_rel, width_rel, height_rel]
    """

    # YOLO bounding box format: [x_center, y_center, width, height]
    # (float values relative to width and height of image)
    x_tl, y_tl, w, h = bbox

    dw = 1.0 / img_width
    dh = 1.0 / img_height

    #finding midpoint
    x_center = x_tl + w / 2.0
    y_center = y_tl + h / 2.0

    x = x_center * dw
    y = y_center * dh
```

Fig A1.1 Converting COCO Json Format Into Yolo Format

```
convert.py - C:\Users\venka\Downloads\archive (1)\FLIR_ADAS_1_3\val\convert.py (3.9.1)
File Edit Format Run Options Window Help

x = x_center * dw
y = y_center * dh
w = w * dw
h = h * dh

return [x, y, w, h]

def convert_coco_json_to_yolo_txt(output_path, json_file):
    dat = 0
    dat_1 = 0
    path = make_folders(output_path)

    with open(json_file) as f:
        json_data = json.load(f)

    # labels files holds names of all classes (one class per line)
    label_file = os.path.join(output_path, "object.labels")
    with open(label_file, "w") as f:
        for category in tqdm(json_data["categories"], desc="Categories"):
            category_name = category["name"]
            f.write(f"{category_name}\n")

    for image in tqdm(json_data["images"], desc="Annotation txt for each iamge"):
        img_id = image["id"]
        img_name = image["file_name"]
        img_width = image["width"]
        img_height = image["height"]

        anno_in_image = [anno for anno in json_data["annotations"] if anno["image_id"] == img_id]
        #anno_txt = os.path.join(output_path, img_name.split(".")[0] + ".txt")
        img_temp = img_name.split("/")
        anno_txt = str(output_path) + "/" + str(img_temp[-1].split(".")[0]) + ".txt"
        with open(anno_txt, "w") as f:
            for anno in anno_in_image:
                category = anno["category_id"]
                if (anno["category_id"] == 1):
                    bbox_COCO = anno["bbox"]
                    x, y, w, h = convert_bbox_coco2yolo(img_width, img_height, bbox_COCO)
                    f.write(f"{0} {x:.6f} {y:.6f} {w:.6f} {h:.6f}\n")
                    #dat +=1
                    #print (img_name)
                if (anno["category_id"] == 3):
                    #dat_1 +=1
                    bbox_COCO = anno["bbox"]
                    x, y, w, h = convert_bbox_coco2yolo(img_width, img_height, bbox_COCO)
                    f.write(f"{1} {x:.6f} {y:.6f} {w:.6f} {h:.6f}\n")
```

Fig A1.2 Converting COCO Json Format Into Yolo Format

```
        x, y, w, h = convert_bbox_coco2yolo(img_width, img_height, bbox_COCO)
        f.write(f"{1} {x:.6f} {y:.6f} {w:.6f} {h:.6f}\n")
    if (anno["category_id"] == 10):
        #dat_1 +=1
        bbox_COCO = anno["bbox"]
        x, y, w, h = convert_bbox_coco2yolo(img_width, img_height, bbox_COCO)
        f.write(f"{2} {x:.6f} {y:.6f} {w:.6f} {h:.6f}\n")

    print("Converting COCO Json to YOLO txt finished!")

convert_coco_json_to_yolo_txt("output", "thermal_annotations.json")
```

Fig A1.3 Converting COCO Json Format Into Yolo Format

Training Yolov8 Model:

```
Yolov8.ipynb
File Edit View Insert Runtime Tools Help Last edited on March 23
+ Code + Text
[ ] from google.colab import drive
drive.mount('/content/drive')

[ ] !pip install ultralytics

[ ] import ultralytics
ultralytics.checks()

Ultralytics YOLOv8.0.54 Python-3.9.16 torch-1.13.1+cu116 CUDA:0 (Tesla T4, 15102MiB)
Setup complete (2 CPUs, 12.7 GB RAM, 25.5/78.2 GB disk)

[ ] !rsync -a /content/drive/MyDrive/v8 format .

[ ] !yolo task=detect mode=train model=yolov8s.pt data=data.yaml epochs=50 imgsz=640 batch=8 project=/content/drive/MyDrive/output name=automotive_true

Ultralytics YOLOv8.0.54 Python-3.9.16 torch-1.13.1+cu116 CUDA:0 (Tesla T4, 15102MiB)
yolo/engine/trainer: task=detect, mode=train, model=yolov8s.pt, data=data.yaml, epochs=50, patience=50, batch=8, imgsz=640, save=True, save_period=1, cache=False, device=None, workers=0
Overriding model.yaml nc=80 with nc=3

      from n  params module                        arguments
0         1    928  ultralytics.nn.modules.Conv [3, 32, 3, 2]
1         1  18560  ultralytics.nn.modules.Conv [32, 64, 3, 2]
2         1   29056 ultralytics.nn.modules.C2f [64, 64, 1, True]
3         1   73984 ultralytics.nn.modules.Conv [64, 128, 3, 2]
4         2  197632 ultralytics.nn.modules.C2f [128, 128, 2, True]
5         1  295424 ultralytics.nn.modules.Conv [128, 256, 3, 2]
6         2  788480 ultralytics.nn.modules.C2f [256, 256, 2, True]
```

Fig A2.1 Training Yolov8 Model

```
+ Code + Text
!yolo task=detect mode=train model=yolov8s.pt data=data.yaml epochs=50 imgsz=640 batch=8 project=/content/drive/MyDrive/output name=automotive_true

Epoch GPU mem box_loss cls_loss dfl_loss Instances Size
1/50 2.126 1.971 1.495 1.413 85 640: 100% 366/366 [02:33<00:00, 2.39it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:02<00:00, 2.14it/s]
all 80 467 0.568 0.247 0.271 0.106

Epoch GPU mem box_loss cls_loss dfl_loss Instances Size
2/50 2.546 1.988 1.324 1.387 71 640: 100% 366/366 [02:26<00:00, 2.50it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01<00:00, 3.31it/s]
all 80 467 0.379 0.245 0.287 0.142

Epoch GPU mem box_loss cls_loss dfl_loss Instances Size
3/50 2.546 1.854 1.25 1.35 79 640: 100% 366/366 [02:24<00:00, 2.53it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01<00:00, 3.35it/s]
all 80 467 0.895 0.234 0.304 0.145

Epoch GPU mem box_loss cls_loss dfl_loss Instances Size
4/50 2.546 1.822 1.211 1.331 29 640: 100% 366/366 [02:23<00:00, 2.56it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01<00:00, 2.86it/s]
all 80 467 0.846 0.276 0.284 0.131

Epoch GPU mem box_loss cls_loss dfl_loss Instances Size
5/50 2.546 1.761 1.138 1.299 104 640: 100% 366/366 [02:23<00:00, 2.54it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:02<00:00, 2.44it/s]
all 80 467 0.891 0.277 0.312 0.156

Epoch GPU mem box_loss cls_loss dfl_loss Instances Size
6/50 2.546 1.709 1.058 1.27 31 640: 100% 366/366 [02:25<00:00, 2.51it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01<00:00, 3.25it/s]
all 80 467 0.358 0.277 0.315 0.158
```

Fig A2.2 Training Yolov8 Model

```

+ Code + Text
Connect
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01:00:00, 3.24it/s]
all 80 467 0.41 0.298 0.361 0.183

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
48/50 2.54G 1.196 0.5955 1.048 42 640: 100% 366/366 [01:28:00:00, 4.14it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01:00:00, 3.33it/s]
all 80 467 0.423 0.368 0.364 0.187

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
49/50 2.54G 1.192 0.5903 1.051 71 640: 100% 366/366 [01:28:00:00, 4.12it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01:00:00, 3.35it/s]
all 80 467 0.496 0.363 0.371 0.192

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
50/50 2.54G 1.171 0.5802 1.042 33 640: 100% 366/366 [01:27:00:00, 4.16it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:06:00:00, 1.33s/it]
all 80 467 0.475 0.369 0.375 0.194

50 epochs completed in 1.951 hours.
Optimizer stripped from /content/drive/MyDrive/output/automotive_true/weights/last.pt, 22.5MB
Optimizer stripped from /content/drive/MyDrive/output/automotive_true/weights/best.pt, 22.5MB

Validating /content/drive/MyDrive/output/automotive_true/weights/best.pt...
Ultralytics YOLOv8.0.53 Python-3.9.16 torch-1.13.1+cu116 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 11126745 parameters, 0 gradients, 28.4 GFLOPs
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:07:00:00, 1.46s/it]
all 80 467 0.535 0.409 0.394 0.198
vehicle 80 447 0.715 0.617 0.667 0.36
pedestrian 80 20 0.355 0.2 0.122 0.0364
Speed: 1.8ms preprocess, 12.3ms inference, 0.0ms loss, 8.8ms postprocess per image
Results saved to /content/drive/MyDrive/output/automotive_true

```

Fig A2.3 Training YOLOv8 Model

```

+ Code + Text
Connect
[ ] lyolo task-detect mode=train model=yolov8s.pt data=data.yaml epochs=50 imgsz=640 batch=8 project=/content/drive/MyDrive/output name=automotive optimizer='AdamW' close_mosaic=10 cos_lr

Downloading https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8s.pt to yolov8s.pt...
100% 21.5M/21.5M [00:00:00:00, 52.0MB/s]
Ultralytics YOLOv8.0.53 Python-3.9.16 torch-1.13.1+cu116 CUDA:0 (Tesla T4, 15102MiB)
yolo/engine/trainer: task=detect, mode=train, model=yolov8s.pt, data=data.yaml, epochs=50, patience=50, batch=8, imgsz=640, save=True, save_period=1, cache=False, device=None, work
Downloading https://ultralytics.com/assets/Arial.ttf to /root/.config/Ultralytics/Arial.ttf...
100% 755k/755k [00:00:00:00, 48.8MB/s]
Overriding model.yaml nc=80 with nc=3

      from  n  params module arguments
      0      -1  1    928 ultralytics.nn.modules.Conv [3, 32, 3, 2]
      1      -1  1  18560 ultralytics.nn.modules.Conv [32, 64, 3, 2]
      2      -1  1  29056 ultralytics.nn.modules.C2f [64, 64, 1, True]
      3      -1  1  73984 ultralytics.nn.modules.Conv [64, 128, 3, 2]
      4      -1  2  197632 ultralytics.nn.modules.C2f [128, 128, 2, True]
      5      -1  1  295424 ultralytics.nn.modules.Conv [128, 256, 3, 2]
      6      -1  2  788480 ultralytics.nn.modules.C2f [256, 256, 2, True]
      7      -1  1  1180672 ultralytics.nn.modules.Conv [256, 512, 3, 2]
      8      -1  1  1838880 ultralytics.nn.modules.C2f [512, 512, 1, True]
      9      -1  1  656896 ultralytics.nn.modules.SPPF [512, 512, 5]
     10      -1  1      0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
     11      [-1, 6] 1      0 ultralytics.nn.modules.Concat [1]
     12      -1  1  591360 ultralytics.nn.modules.C2f [768, 256, 1]
     13      -1  1      0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
     14      [-1, 4] 1      0 ultralytics.nn.modules.Concat [1]
     15      -1  1  148224 ultralytics.nn.modules.C2f [384, 128, 1]
     16      -1  1  147712 ultralytics.nn.modules.Conv [128, 128, 3, 2]
     17      [-1, 12] 1      0 ultralytics.nn.modules.Concat [1]
     18      -1  1  493056 ultralytics.nn.modules.C2f [384, 256, 1]
     19      -1  1  590336 ultralytics.nn.modules.Conv [256, 256, 3, 2]

```

Fig A2.4 Training YOLOv8 Model

Yolov8.ipynb

File Edit View Insert Runtime Tools Help Last edited on March 23

+ Code + Text

```
[ ]
```

	all	80	407	0.771	0.354	0.406	0.196
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	
49/50	2.52G	0.995	0.4869	0.9385	71	640: 100% 366/366 [01:25<00:00, 4.27it/s]	
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100% 5/5 [00:01<00:00, 3.36it/s]
	all	80	467	0.786	0.344	0.407	0.197

	Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
	50/50	2.52G	0.9737	0.4783	0.9307	33	640: 100% 366/366 [01:25<00:00, 4.28it/s]
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100% 5/5 [00:06<00:00, 1.23s/it]
	all	80	467	0.729	0.332	0.404	0.195

50 epochs completed in 1.868 hours.
Optimizer stripped from /content/drive/MyDrive/output/automotive4/weights/last.pt, 22.5MB
Optimizer stripped from /content/drive/MyDrive/output/automotive4/weights/best.pt, 22.5MB

Validating /content/drive/MyDrive/output/automotive4/weights/best.pt...
Ultralytics YOLOv8.0.53 Python-3.9.16 torch-1.13.1+cu116 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 11126745 parameters, 0 gradients, 28.4 GFLOPs

Class	Images	Instances	Box(P	R	mAP50	mAP50-95)
all	80	467	0.771	0.369	0.429	0.207
vehicle	80	447	0.807	0.637	0.704	0.374
pedestrian	80	20	0.734	0.1	0.153	0.0408

Speed: 4.2ms preprocess, 13.3ms inference, 0.0ms loss, 13.4ms postprocess per image
Results saved to /content/drive/MyDrive/output/automotive4

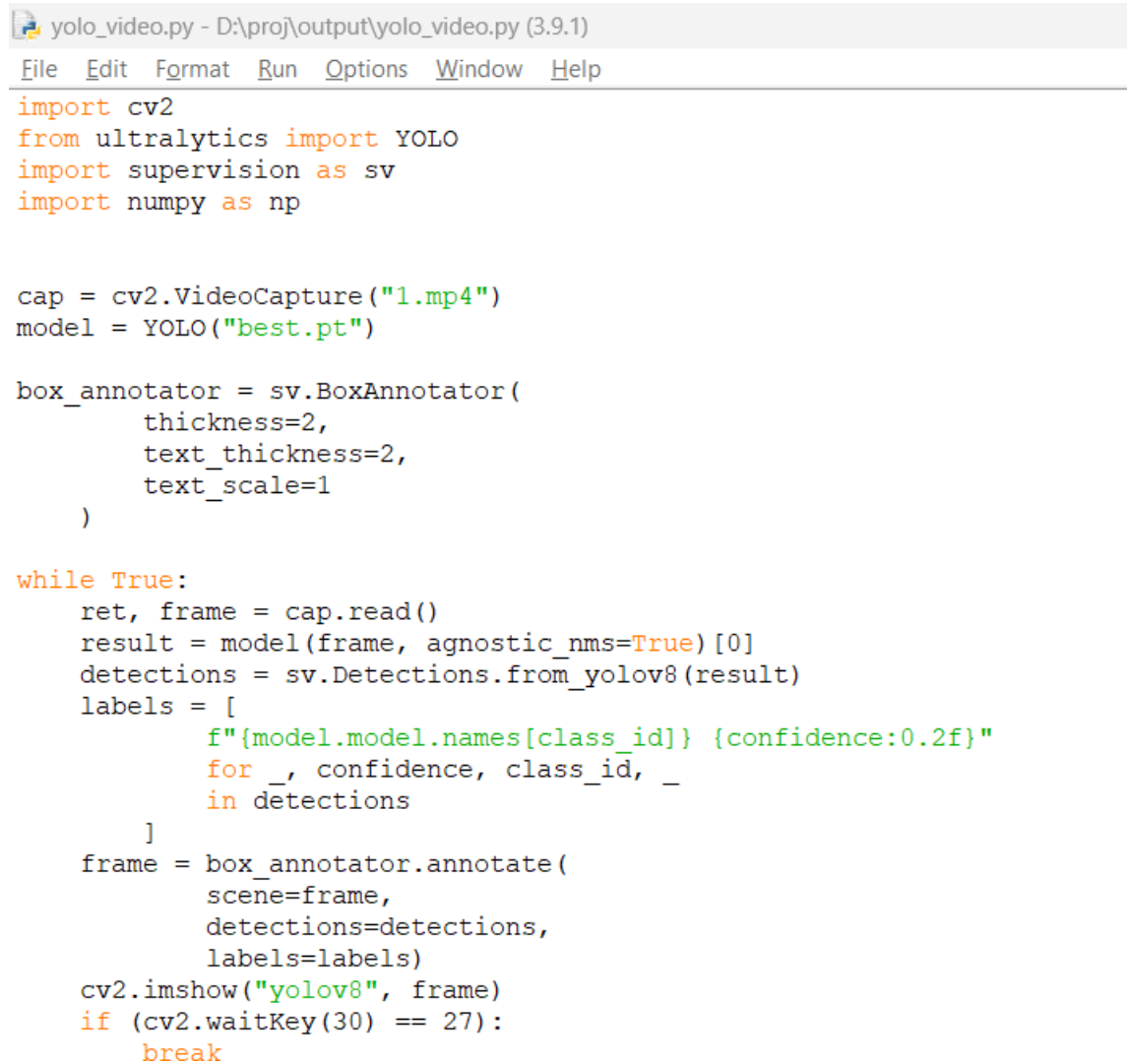
```
[ ] |yolo task=detect mode=predict model=/content/drive/MyDrive/output/automotive4/weights/best.pt conf=0.55 source=/content/drive/MyDrive/dataset/1478019952686311006_jpg.rf.54e2d12dbabc4
```

Ultralytics YOLOv8.0.54 Python-3.9.16 torch-1.13.1+cu116 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 11126745 parameters, 0 gradients, 28.4 GFLOPs

image 1/1 /content/drive/MyDrive/dataset/1478019952686311006_img.rf.54e2d12dbabc46ba3c78995b6eaf3fee_img-640x640 2 vehicles 1 pedestrian 17.2ms

Fig A2.5 Training YOLOv8 Model

Object Detection Using YOLOv8:



```
yolo_video.py - D:\proj\output\yolo_video.py (3.9.1)
File Edit Format Run Options Window Help

import cv2
from ultralytics import YOLO
import supervision as sv
import numpy as np

cap = cv2.VideoCapture("1.mp4")
model = YOLO("best.pt")

box_annotator = sv.BoxAnnotator(
    thickness=2,
    text_thickness=2,
    text_scale=1
)

while True:
    ret, frame = cap.read()
    result = model(frame, agnostic_nms=True)[0]
    detections = sv.Detections.from_yolov8(result)
    labels = [
        f"{model.model.names[class_id]} {confidence:0.2f}"
        for _, confidence, class_id, _
        in detections
    ]
    frame = box_annotator.annotate(
        scene=frame,
        detections=detections,
        labels=labels)
    cv2.imshow("yolov8", frame)
    if (cv2.waitKey(30) == 27):
        break
```

Fig A3.1 Object Detection Using YOLOv8

RESEARCH PAPER

YOLOv8-based Advance Assistance for Autonomous Vehicles

Dr. D. Poornima, M.E., (Ph.D.)
Computer Science & Engineering
Sathyabama Institute of Science & Technology
Sathyabama University, Chennai, India.
poornima.d.cse@sathyabama.ac.in

L Sai Kumar
Computer Science & Engineering
Sathyabama Institute of Science & Technology
Sathyabama University, Chennai, India.
lsaikumarreddy48@gmail.com

Yalla Sai Krishna Vamsi
Computer Science & Engineering
Sathyabama Institute of Science & Technology
Sathyabama University, Chennai, India.
yallasaiakrishnavamsi@gmail.com

ABSTRACT

The characteristics of the system and the specific model form the basis of the product discovery process. Its performance can be easily balanced by creating a combination of many low-resolution images and high-resolution elements for capture and separation. Due to the rapid development of machine learning, more powerful tools that can take semantic, advanced and deeper features have emerged to solve the problems of legacy architectures. This research offers a new way to identify people, cars and traffic lights using only publicly available data. It is difficult to integrate search into the database because it contains long-dimensional images (such as those in sunlight) and the features of the dataset are less reliable for training. For training purposes, we introduce a customized YOLOv8 model to improve its accuracy. The model uses many constant and lossy ideas. Since YOLOv8 can be used on mobile devices and requires less RAM to think about, it makes sense to use it. Additionally, Unity provides additional support to simplify the migration process.

Index terms: object detection, instance segmentation, image classification.

1.INTRODUCTION

The image is easily visible and depends on the person. Humans have fast, accurate vision and the ability to easily perform complex tasks such as problem solving and multitasking. Big data, faster GPUs, and advanced algorithms allow us to quickly

train computers to recognize and classify various attributes in images with high accuracy. An important element for defining the boundaries of objects defined as squares, rectangles, or other different types of rectangles is the object recognition box. Early detection techniques used image classification to determine the location of objects while reexamining the differences between images at different scales. This is a slow and ineffective strategy.

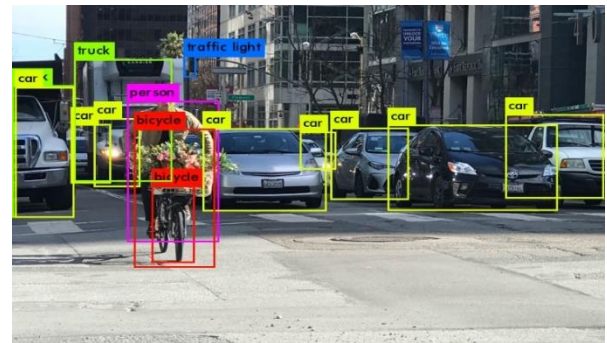


Fig 1: Object Detection

R-CNN (region-based convolutional neural network) was proposed in 2014 and is the best method of the first category. The disadvantage of R-valued CNN is that the process is difficult even if 2000 regional recommendations are removed. Later, a third, more advanced model, Faster R-CNN, was proposed. The architecture is similar to Fast R-CNN with some important changes. This article introduces the YOLO method for target detection. YOLO adopts an entirely different strategy. It only makes one network

connection and one full image view. Hence the nickname. time flies.

II.BACKGROUND STUDY

C. Kwan and B. Budavari.,(2021) Author of the article "Using Optical Flow to Improve the Accuracy of Small Moving Object Detection in Low Long-Distance Infrared Lenses." Long infrared films face problems in detecting small moving objects due to air turbulence, background clutter, and small targets. This article describes practical research methods for improving small object detection performance in long-range, low-resolution infrared movies containing moving objects. Our particular interest is in the use of mobile data with optical devices to improve product detection performance through surveillance, alteration, modification and processing. enjoy your work. Various tests on long medium-wave infrared (MWIR) videos collected by the Defense Systems Data Analysis Center (DSIAC) demonstrate the effectiveness of our strategy.

[2] C. Kwan and J. Larkin.,(2022) Far infrared (IR) film may have trouble detecting small moving objects due to air turbulence, background noise, and small targets. This work provides both adaptive, non-adaptive and non-observer approaches to objects moving in small space. The main idea behind the displacement (CD) algorithm is to use the difference between variables to detect movement. Extensive testing with true mid-infrared (MWIR) lenses at a distance of more than 3500 meters from the camera's lens showed that a single-use operation is possible better than other methods of using light. Both work on 3500m film, but they don't do well on 4000m and 5000m film. Finally, an analysis comparing our proposed method with two well-known methods shows that its performance is comparable.

[3] C. Kwan and B. Budavari.,(2020) The small size of objects makes them difficult to identify in far infrared (IR) movies. In this paper, we propose a robust method for locating small objects in low-mass long-range infrared images. The three methods include solving the animation problem, a small object, solid and gradient evidence (LIG) based on the small object, coupled object (CC) analysis module, and connectivity model for connectivity from multiple images. The effectiveness of the scheme was confirmed by extensive testing with medium-wave infrared (MWIR) videos of intersections in the reference material range of 3,500 m and 5,000

[4] C.-Y. Wang, A. Bochkovskiy, and H.-Y.-M. Liao.,(2020) We found that the YOLOv4 object detection neural network based on CSP technology can scale to large and small networks while maintaining speed and accuracy. Using the MS COCO dataset, the YOLOv4-wide model achieved 55.5% AP (73.4% AP50) at approximately 16 IPS on Tesla V100, while with increasing testing time, YOLOv4-wide model achieved 56.0% AP (73%). ,4) 3AP50. Using TensorRT, batch size = 4 and FP16 exposure, the smaller YOLOv4 model reaches 1774 FPS, while the RTX 2080Ti model achieves 22.0% AP (42.0 AP50%) and 443 FPS.

[5] M. S. Uddin, R. Hoque, K. A. Islam, C. Kwan, D. Gribben, and J. Li., (2022) More information is needed to develop a performance model that can be used with deep learning techniques for object recognition and classification in infrared film. Once it is possible to convert video from optical to infrared, the lack of infrared film data will be reduced. In this paper, we propose a deep learning algorithm to convert optical video to infrared. This method does not

require similar images. The performance of the proposed GAN is demonstrated by evaluating the purpose and content. Moreover, the focus effect has been shown to improve the object recognition and classification abilities of GANs using real infrared video.

III.METHODOLOGY

A. Architecture of yolov8

Because it has not yet been published, we cannot evaluate future methods to directly produce YOLOv8 by ablation studies. However, we looked at simple repositories and template files to start documenting the new YOLOv8 features. After a brief summary of the reformulation model, we present a description of the model itself. The image below is a full view of the network architecture and was created by GitHub user Range King.

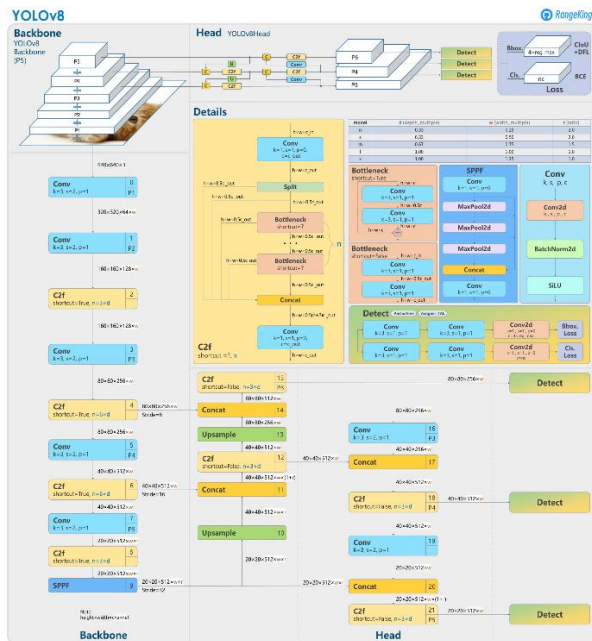


Fig 2: Architecture of yolov8

B. Deepsort

The SORT method uses the Hungarian

method to evaluate the relationship and uses the Kalman filter technique to process the data. The method performs well when frame rates are high. SORT, on the other hand, ignores the properties of the objects it finds and instead converts its accuracy into the uncertainty of the state estimate. To improve tracking performance, SORT also removes non-matching elements from consecutive rows. However, since this causes identity change problems, the product identity may change from time to time. Once an item is assigned a unique ID, it is tracked using Deep SORT, a computerized tracking system. Deep SORT extends this (Easy Online Sorting Time) technology. To reduce individual variables and improve detection, Deep SORT incorporates deep learning into the SORT algorithm.

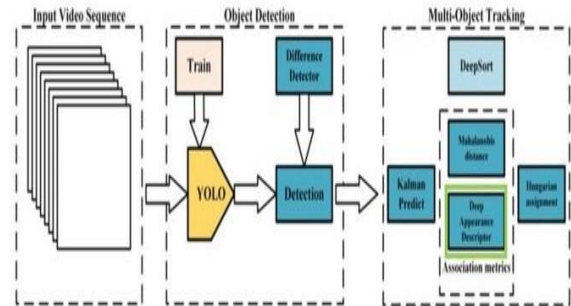


Fig 3: DEEPSORT

Kalman filter is required for depth measurement. Eight variables lead to three states: $(u, v, a, h, u', v', a \text{ and } h')$; (u, v) is the center of the bounding box; (a) comparable; (h) is the height of the image. Another difference is the difference between differences. Kalman filter helps deal with noise by using a good fit of the boundary layer by estimating the previous state. A combination of the Hungarian algorithm and the Kalman filter is used for tracking. When a Kalman filter is used on the image space, a measure is used that determines the overlap between the boxes, in this case the

Hungarian method is used to provide the connection between the data. As the final matching level, Deep SORT adopts Io U matching [16], which can reduce significant changes caused by obvious changes or partial occlusions. In addition, Deep SORT calculates similarity using the Re ID model to extract unique features from the output of network detection objects.

IV.RESULTS AND DISCUSSIONS

1)Dataset Preparation

- Content detection control is learned using COCO data detection; image resolution is 640x480.
- Split checkpoint example study using the COCO dataset 640x480 split dataset.
- ImageNet data contains 224 pixels per inch images and is initially used to train image classification models.



Fig:4.1. COCO JSON Thermal Format

- A new file (converted dataset) will be created in the specified output path, containing text and images in YOLO format.

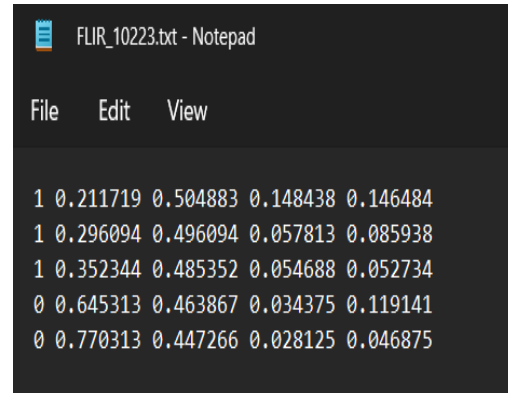


Fig:4.2. YOLO Format

2) Architecture creation and training

- Since we will be using the YOLOv8 model for this application, we will install the PyTorch environment, create everything necessary and start the weight of the design.
- Once the architecture is completed, the model will immediately start training using the prepared data. So, when you look at the churn rate to determine how well the model is learning, if the mAP model is improving and the churn rate is low, stop learning.

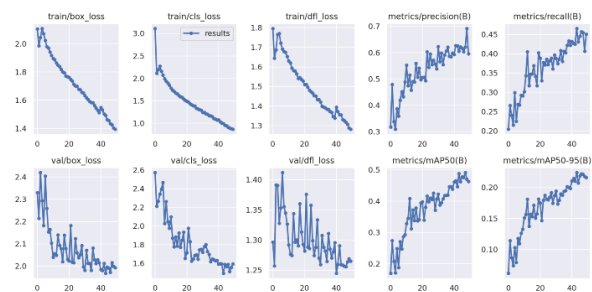


Fig: 4.2. Training YOLOv8

3) Inference Code

- The inference process loads the architecture and weights found during model training to identify objects in images.
- The main purpose of this module is to base it on a practice-oriented training mode

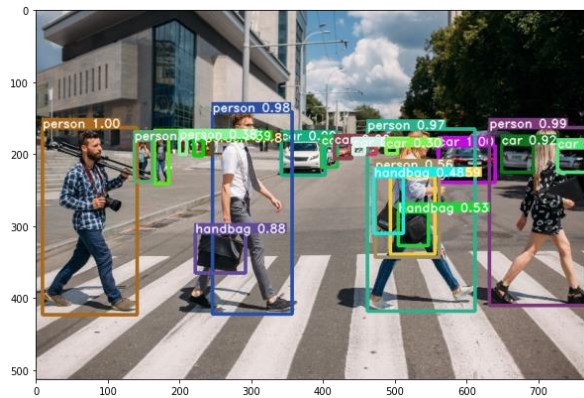


Fig: 4.3. Object Detection with YOLOv8

V.YOLO-PROPOSED ALGORITHM

One of the most famous models of computer vision is the "look-at-once" (YOLO) model. The search engine is known for its fast and accurate image processing. The goal of the YOLO algorithm is to predict the class of an object and the bounding box that indicates its position in the input image.

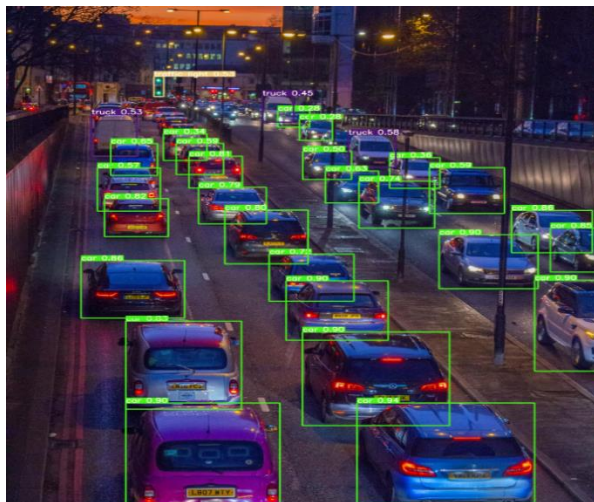


Fig:5.1. YOLO algorithm in object detection

YOLOv8

For many object detection and picture segmentation applications, the YOLOv8 system is the ideal approach since it is quick,

precise, and simple to use. It can be trained on large datasets and run on multiple hardware platforms, including CPUs and GPUs. It's made to be a framework that works with any of YOLO's efforts, so switching between them and tracking their progress is simple. Therefore, YOLOv8 is the best choice for customers who want to benefit from the latest YOLO technology while using the existing YOLO model. In addition to its simplicity, YOLOv8 has many new features that make it ideal for many tasks related to object detection and image segmentation. These include a new unempyment, a new spine and a head without an anchor. In addition to being very efficient, YOLOv8 is also compatible with various CPU and GPU hardware platforms. YOLOv8 is a very powerful and versatile object detection and image segmentation tool that combines the best of previous YOLO versions in a new SOTA approach.

VI. CHALLENGES

Second, separate the red and green lights of the movie from the traffic signs. The second thing that affects tracking accuracy is the location where the video will be recorded. Fog, dust, or dim light can reduce accuracy and cause more errors and artifacts. Sensory exposure is amplified by Gaussian noise introduced by electronic devices such as sensors. Finally, there is a need for real-time analysis and critical detection of vehicle acceleration or deceleration. For hardware acceleration, this requires running the program on the FPGA.

VII. CONCLUSION

YOLOv8 is the newest model in the YOLO series and raises the bar in product discovery. For developers, the YOLOv8 software is made as user-friendly as feasible. The newly released

Ultralytics YOLOv8 pip or bundle makes working with models in code easier than ever. Learning is more straightforward with the new shell interface.

VIII. REFERENCE

- [1] M. Jones and P. Viola, “Robust real-time object detection,” *Int. J. Comput. Vis.*, vol. 57, no. 2, p. 87, 2001.
- [2] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, vol. 1, Jun. 2005, pp. 886–893.
- [3] C. Kwan and B. Budavari, “Enhancing small moving target detection performance in low-quality and long-range infrared videos using optical flow techniques,” *Remote Sens.*, vol. 12, no. 24, p. 4024, Dec. 2020.
- [4] C. Kwan and J. Larkin, “Detection of small moving objects in long range infrared videos from a change detection perspective,” *Photonics*, vol. 8, no. 9, p. 394, Sep. 2021.
- [5] C. Kwan and B. Budavari, “A high-performance approach to detecting small targets in long-range low-quality infrared videos,” *Signal, Image Video Process.*, vol. 16, no. 1, pp. 93–101, Feb. 2022.
- [6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Proc. NIPS*, 2012, pp. 1097–1105.
- [7] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 580–587.
- [8] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial pyramid pooling in deep convolutional networks for visual recognition,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 9, pp. 1904–1916, Sep. 2015.
- [9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.
- [10] W. Liu et al., “SSD: Single shot multibox detector,” in *Computer Vision—ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham, Switzerland: Springer, 2016, pp. 21–37.