International Research Journal of Education and Technology

Peer Reviewed Journal, **ISSN 2581-7795**

Improved YOLO Architecture for Real-Time Jellyfish Detection

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE &

ENGINEERING (DATA SCIENCE)

Submitted by

JAISWAL KASHISH 21WJ1A6718

PADIGELA AKSHAYA 21WJ1A6744

SHAIK SHOYAB 21WJ1A6754

Under the Guidance of

Dr.Ch. Subbalakshmi

HOD - CSE(DATA SCIENCE)

Department of CSE(DATA SCIENCE)

School of Engineering and Technology

Guru Nanak Institutions Technical Campus

Ibrahimpatnam, Hyderabad, R.R. District – 501506

June, 2025

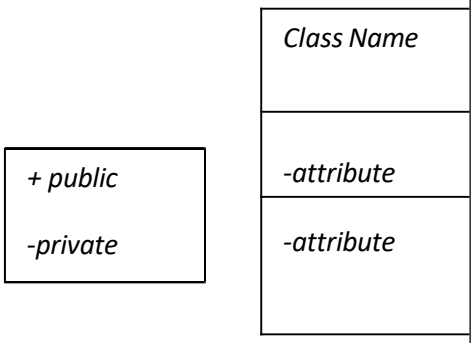
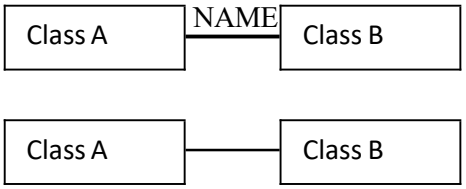
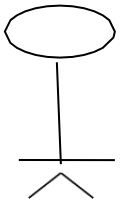
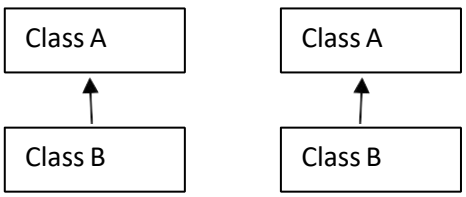
The objective of this project is to develop an efficient and accurate system for detecting jellyfish in underwater environments using deep learning techniques. Massive jellyfish outbreaks pose serious risks to human safety and marine ecosystems, highlighting the urgent need for reliable detection methods. To address this, the project leverages optical imagery and a convolutional neural network (CNN)-based object detection approach.

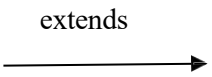
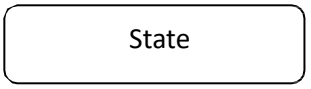
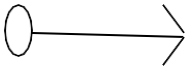
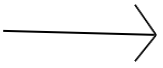
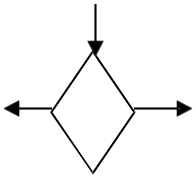
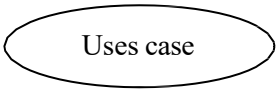
Due to the scarcity of labeled jellyfish datasets, a novel dataset was created using a model-assisted labeling strategy, significantly minimizing the need for manual annotation. Based on this dataset, we propose an enhanced YOLOv11 model that incorporates the Global Attention Mechanism (GAM) and CoordConv modules to improve feature extraction and spatial awareness.

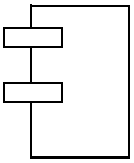
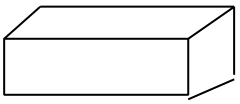
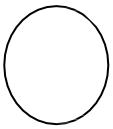
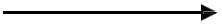
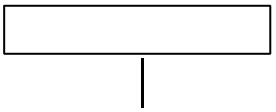
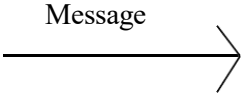
Experimental evaluations demonstrate that the proposed model outperforms several state-of-the-art detection frameworks in terms of accuracy and robustness. The results indicate the system's potential for real-time jellyfish detection, contributing to improved marine safety and ecosystem monitoring.

LIST OF FIGURES

LIST OF SYMBOLS

S.NO	NOTATION NAME	NOTATION	DESCRIPTION
1.	Class		Represents a collection of similar entities grouped together.
2.	Association		Associations represents static relationships between classes. Roles represents the way the two classes see each other.
3.	Actor		It aggregates several classes into a single class.
4.	Aggregation		Interaction between the system and external environment

5.	Relation (uses)	uses	Used for additional process communication.
6.	Relation (extends)		Extends relationship is used when one use case is similar to another use case but does a bit more.
7.	Communication		Communication between various use cases.
8.	State		State of the processes.
9.	Initial State		Initial state of the object
10.	Final state		Final state of the object
11.	Control flow		Represents various control flow between the states.
12.	Decision box		Represents decision making process from a constraint
13.	Use case		Interaction between the System and external Environment.

14.	Component		Represents physical Modules which are a collection of components.
15.	Node		Represents physical Modules which are a collection of components.
16.	Data Process/State		A circle in DFD represents a state or process which has been triggered due to some event or action.
17.	External entity		Represents external entities such as keyboard, sensors, etc.
18.	Transition		Represents communication that occurs between processes.
19.	Object Lifeline		Represents the vertical dimensions that the object communications.
20.	Message		Represents the message exchanged.

CHAPTER-1

INTRODUCTION

1.1 INTRODUCTION

Jellyfish outbreaks have emerged as a significant concern globally, affecting marine ecosystems and endangering human lives. These outbreaks disrupt ecological balance, impact fisheries, and pose challenges to coastal industries, including tourism and power generation. As a result, accurate and efficient jellyfish detection has become a crucial area of research. Current detection methods are broadly categorized into optical imaging and sonar imaging approaches, each with its own advantages and limitations. Optical imagery, in particular, offers a detailed and intuitive visual representation, making it highly suitable for real-time jellyfish detection. However, the application of optical methods in conjunction with advanced deep-learning techniques remains underexplored.

Manual annotation of datasets is a labour-intensive and time-consuming process, particularly for large datasets. To overcome this challenge, we adopted a model-assisted labelling method, which significantly reduces the burden of manual labelling while ensuring high-quality annotations. Using this dataset, we developed and enhanced a YOLOv11-based detection model. The proposed improvements involve integrating advanced architectural components, such as the Global Attention Mechanism (GAM) and CoordConv layers, to optimize the detection capabilities of the YOLOv11- model.

The enhanced YOLOv11 model is designed to balance real-time performance with robust detection accuracy, addressing the trade-offs often observed in conventional object detection models. Extensive experiments were conducted to evaluate its performance against several state-of-the-art object detection frameworks, including Faster R-CNN and YOLOv11. The results highlight the effectiveness of our proposed approach in accurately detecting jellyfish in complex underwater environments, offering a promising solution for mitigating the impact of jellyfish outbreaks.

1.2 SCOPE OF THE PROJECT

The scope of this project revolves around utilizing YOLOv11, a cutting-edge object detection framework, to develop an advanced jellyfish detection system using optical imagery. This project aims to address the critical challenges posed by jellyfish outbreaks, which significantly impact marine ecosystems, fisheries, and human activities. The focus includes creating a diverse and annotated dataset of underwater jellyfish images to train and evaluate the model. YOLOv11's state-of-the-art capabilities, such as Transformer-based backbones and advanced feature fusion mechanisms, will be fine-tuned for underwater environments to ensure robust performance in complex scenarios involving low visibility, overlapping objects, and diverse jellyfish morphologies. The system is designed to achieve real-time detection with a balance between computational efficiency and accuracy, making it suitable for deployment on platforms such as edge devices, unmanned underwater vehicles, and cloud-based systems. Additionally, the model's performance will be benchmarked against other object detection frameworks to demonstrate its effectiveness. This scalable and deployable system aims to support ecosystem management, providing valuable insights into jellyfish populations and assisting industries in mitigating the adverse effects of outbreaks, ultimately contributing to advancements in marine conservation and monitoring.

In sectors like autonomous vehicles, healthcare imaging, and robotics, YOLOv11 provides unparalleled efficiency in detecting intricate details and handling challenging environments. It can be integrated into retail for inventory tracking, surveillance systems for real-time threat detection, and industrial automation for quality control. With a reduction in latency and enhanced real-time performance, YOLOv11 offers a flexible and robust foundation for addressing specific challenges across diverse domains, ensuring adaptability to future advancements in computer vision technologies

1.3 OBJECTIVE

The primary objective of this project is to design and implement a cutting-edge jellyfish detection system using YOLOv11, a state-of-the-art deep learning model for real-time object detection. The goal is to address critical challenges posed by massive jellyfish outbreaks, which impact marine ecosystems, disrupt fisheries, and pose risks to human activities. By leveraging YOLOv11's advanced features—such as improved feature extraction, enhanced detail recognition, and faster processing speeds—this project aims to create a detection system that excels in accuracy and efficiency under challenging underwater conditions.

The project emphasizes creating a comprehensive dataset of underwater jellyfish images that reflect diverse species, environments, and morphological traits. These images will serve as a foundation for training YOLOv11, ensuring the model is robust against issues like low visibility, overlapping objects, and complex underwater backgrounds. The integration of YOLOv11's innovative architecture, including its ability to handle multiple vision tasks such as object detection and instance segmentation, will significantly enhance the system's ability to detect jellyfish with precision and speed.

Another objective is to develop a solution that is not only highly accurate but also scalable and deployable across various platforms, such as underwater vehicles, edge devices, and cloud-based systems. This will facilitate real-time monitoring and analysis, enabling proactive measures to mitigate the adverse effects of jellyfish outbreaks on marine and coastal industries. Furthermore, the system is designed to be versatile, with the potential to extend its capabilities to other underwater object detection tasks, contributing to broader advancements in environmental monitoring and industrial applications.

CHAPTER 2

LITERATURE SURVEY

2.1 REVIEW OF RELATED RESEARCH PAPERS

[1] **TITLE:** Visual Instruction Tuning

AUTHORS: H. Liu, C. Li, Q. Wu, and Y. J. Lee

YEAR: 2023

DESCRIPTION: This paper introduces a novel approach called Visual Instruction Tuning (VIT), which aims to enhance the capabilities of visual models in understanding and executing complex visual instructions. Traditional visual models are primarily trained on large-scale datasets with minimal task-specific guidance, which can lead to suboptimal performance in specialized tasks. Visual instruction tuning addresses this limitation by incorporating task-specific visual cues during the model's training process. This allows the model to better interpret nuanced visual cues and instructions, improving its accuracy in tasks such as object detection, image captioning, and semantic segmentation. The authors also explore the potential applications of VIT in multi-modal AI systems where understanding and generating tasks from both visual and textual data is crucial. Experimental results show significant performance improvements when using VIT on several benchmark datasets, demonstrating the efficacy of visual instruction as a means to bridge the gap between vision and language-based tasks. This research opens new avenues for developing more flexible and task-aware visual models.

[2] **TITLE:** Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation

AUTHORS: R. Girshick, J. Donahue, T. Darrell, and J. Malik

YEAR: 2014

DESCRIPTION: The paper introduces the concept of rich feature hierarchies for the dual tasks of object detection and semantic segmentation, significantly advancing the state-of-the-art in both areas. Traditional object detection approaches struggled with low detection accuracy and the inability to handle complex background clutter. The authors propose a solution through the integration of multiple levels of convolutional feature hierarchies. By leveraging deep convolutional networks (CNNs), they enhance the ability of the model to recognize objects at various scales, which is crucial for real-world applications where objects may appear in different sizes or orientations. The paper introduces R-CNN, a model that extracts region proposals from an image and classifies each proposal using CNNs, yielding better detection accuracy. Additionally, the authors combine CNN-based features with robust object proposal algorithms like selective search to significantly improve the detection process. The method's success demonstrated the power of deep learning in the field of object detection and laid the foundation for further developments such as Fast R-CNN and Faster R-CNN, which optimize and accelerate the detection process.

[3] **TITLE:** Fast R-CNN

AUTHORS: R. Girshick

YEAR: 2015

DESCRIPTION: Fast R-CNN is an advanced object detection model designed to improve the performance and efficiency of the original R-CNN framework. While R-CNN revolutionized object detection, it had notable limitations, such as its slow processing speed and high computational cost. Fast R-CNN addresses these limitations by introducing a novel Region of Interest (RoI) pooling layer that enables faster and more efficient processing of region proposals. The key idea behind Fast R-CNN is that, instead of performing separate feature extraction for each region proposal, it extracts features from the entire image in one pass and then applies RoI pooling to map the relevant portion of the image to a fixed-size feature map. This method reduces redundant calculations, leading to faster training and inference times. In addition to increasing speed, Fast R-CNN achieves better accuracy by improving the classification and bounding box regression steps. As a result, it became a widely adopted approach in computer vision, significantly enhancing real-time object detection systems. This paper's contributions were instrumental in pushing forward the development of efficient object detection models.

[4] **TITLE:** Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

AUTHORS: S. Ren, K. He, R. Girshick, and J. Sun

YEAR: 2017

DESCRIPTION: Faster R-CNN represents a groundbreaking advancement in real-time object detection by introducing Region Proposal Networks (RPNs), which allow for joint training of the region proposal generation and object detection processes. Unlike previous models that used external algorithms like selective search to generate region proposals, Faster R-CNN generates proposals directly from the feature map of the convolutional layers. This unified approach eliminates the need for costly external region proposal algorithms, thus reducing computational overhead and enabling faster processing times. RPNs use a small sliding window to propose regions of interest, which are then classified and refined by the subsequent layers of the Faster R-CNN pipeline. The paper shows that this method not only speeds up object detection but also maintains or even improves accuracy compared to traditional methods. The authors validate their model on several challenging datasets, demonstrating that Faster R-CNN achieves state-of-the-art performance in real-time object detection. This paper marks a significant shift towards integrated, end-to-end learning systems in computer vision, influencing future research in object detection architectures.

[5] **TITLE:** You Only Look Once: Unified, Real-Time Object Detection

AUTHORS: J. Redmon, S. Divvala, R. Girshick, and A. Farhadi

YEAR: 2016

DESCRIPTION: YOLO (You Only Look Once) is a revolutionary object detection algorithm that departs from traditional multi-stage detection systems by framing object detection as a single regression problem. Unlike earlier models that separately localized objects and classified them, YOLO performs both tasks simultaneously in a single forward pass through the network. This unified approach allows YOLO to process images extremely quickly, making it one of the first real-time object detection models. The authors demonstrate that YOLO achieves impressive speed without sacrificing accuracy, making it suitable for time-critical applications such as autonomous vehicles, real-time surveillance, and robotics. YOLO's architecture is based on a single convolutional network that divides an image into a grid, with each grid cell predicting bounding boxes and class probabilities. By handling multiple objects in an image at once, YOLO performs significantly faster than its predecessors, paving the way for real-time applications in object detection. Despite its speed, YOLO maintains high detection accuracy, especially for large objects, and its ability to generalize across datasets makes it a versatile model for various detection tasks.

[6] **TITLE:** SSD: Single Shot Multibox Detector

AUTHORS: W. Liu et al.

YEAR: 2016

DESCRIPTION: The Single Shot Multibox Detector (SSD) is another key advancement in real-time object detection, designed to offer a good balance between speed and accuracy. SSD achieves this by generating multiple bounding box predictions for each object in an image using feature maps from several layers of a convolutional neural network. The model applies a small convolutional filter over different layers of the image to detect objects at various scales and aspect ratios, which is especially beneficial for detecting small and large objects within the same image. Unlike YOLO, which uses a grid-based approach, SSD predicts bounding boxes for multiple aspect ratios and scales at each location, enabling it to handle a wider range of object sizes. SSD also uses a multibox loss function to jointly optimize both classification and localization, further enhancing its accuracy. This paper demonstrates that SSD not only matches or exceeds the performance of prior models but also processes images significantly faster, making it a valuable tool for real-time object detection applications in industries such as autonomous driving and surveillance.

CHAPTER 3
SYSTEM REQUIREMENTS & SPECIFICATIONS

3.1 EXISTING SYSTEM

In the domain of real-time object detection, the YOLO (You Only Look Once) series has evolved significantly, with each version addressing limitations and introducing enhancements to improve accuracy, speed, and versatility. YOLOv5, a widely adopted model, became notable for its simplicity, modularity, and efficiency in object detection tasks. Despite its popularity, it faced challenges in balancing real-time performance with detection accuracy, especially in complex scenarios such as underwater environments. YOLOv5 provided a baseline for numerous applications but required external optimizations for tasks like instance segmentation.

Building on this foundation, YOLOv6 introduced structural refinements tailored for industrial use, focusing on lightweight design and improved processing speed. It incorporated enhancements like better activation functions and loss calculations, enabling superior performance for high-speed applications. YOLOv7 further advanced the architecture by integrating novel techniques such as extended bag-of-freebies and trainable bag-of-specials, which contributed to achieving state-of-the-art accuracy and efficiency. This model also introduced additional capabilities for instance segmentation and object tracking, making it suitable for more dynamic and complex environments.

YOLOv8 represented another leap forward by focusing on panoptic segmentation, key point estimation, and a more refined feature pyramid network.

3.1.1 EXISTINGSYSTEM DISADVANTAGES

Accuracy-Speed Trade-off: YOLOv5 struggles to balance real-time detection with precision, especially in complex environments.

Limited Task Versatility: YOLOv5 and YOLOv6 lack native support for advanced tasks like instance segmentation.

High Computational Overhead: Training and deployment require significant computational resources, especially in YOLOv6 and YOLOv7.

Sub optimal Performance in Challenging Conditions: Detection falters in environments with poor lighting, occlusion, or noise, affecting reliability.

Training Complexity: YOLOv7 and YOLOv8 feature advanced architectures that complicate the training process.

3.2 PROPOSED SYSTEM

YOLOv11 represents a significant leap forward in the field of real-time object detection, building upon the strengths of its predecessors while addressing many of their limitations. This model is designed with an optimized architecture that balances the need for high accuracy with the constraints of real-time processing, making it suitable for a wide range of applications, including autonomous vehicles, healthcare imaging, security surveillance, and environmental monitoring. By introducing innovative features like improved feature extraction, dynamic head design, and enhanced handling of multiple vision tasks, YOLOv11 significantly enhances the performance and flexibility of the model.

One of the key advantages of YOLOv11 is its ability to process high-resolution images with minimal computational overhead. It incorporates advanced techniques such as the use of more efficient convolutional layers, reducing the computational burden without sacrificing detection accuracy. This makes YOLOv11 ideal for deployment in resource-constrained environments, including edge devices, drones, and mobile platforms. The model's architecture also ensures faster inference times, which is critical for applications requiring real-time feedback and decision-making, such as traffic monitoring, emergency response, and industrial automation.

Another standout feature of YOLOv11 is its superior adaptability to diverse environments and tasks. Whether dealing with occlusions, low visibility, or overlapping objects, YOLOv11 excels in challenging scenarios where other models may struggle. It provides more precise object localization and segmentation, even in crowded or cluttered scenes.

Furthermore, YOLOv11's modular design and easy integration with existing pipelines make it accessible for developers and researchers across various domains. It allows for quick fine-tuning

and customization, ensuring that it can be adapted to specific tasks without requiring extensive retraining or complex adjustments. This ease of use, combined with its cutting-edge performance, positions YOLOv11 as a powerful tool for advancing the field of computer vision.

3.2.1 PROPOSED SYSTEM ADVANTAGES

Decision trees are easy to understand and interpret.

Random forests are less sensitive to noise and outliers in the data.

Both decision trees and random forests offer strong advantages in terms of interpretability, robustness, and handling various types of data.

Random forests can be used for both classification and regression tasks.

3.3 GENERAL

We can see from the results that on each database, the error rates are very low due to the discriminatory power of features and the regression capabilities of classifiers. Comparing the highest accuracies (corresponding to the lowest error rates) to those of previous works, our results are very competitive.

3.4 HARDWARE REQUIREMENTS

The hardware requirements may serve as the basis for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system. They are used by software engineers as the starting point for the system design. It should what the system do and not how it should be implemented.

PROCESSOR	:	DUAL CORE 2 DUOS.
RAM	:	4GB DD RAM
HARD DISK	:	250 GB

3.5 SOFTWARE REQUIREMENTS

The software requirements document is the specification of the system. It should include both a definition and a specification of requirements. It is a set of what the system should do rather than how it should do it. The software requirements provide a basis for creating the software requirements specification. It is useful in estimating cost, planning team activities, performing

tasks and tracking the teams and tracking the team's progress throughout the development activity.

Operating System : Windows 7/8/10

Platform : Spyder3

Programming Language : Python

Front End : Spyder3

3.6 FUNCTIONAL REQUIREMENTS

A functional requirement defines a function of a software-system or its component. A function is described as a set of inputs, the behavior, Firstly, the system is the first that achieves the standard notion of semantic security for data confidentiality in attribute-based deduplication systems by resorting to the hybrid cloud architecture.

3.7 NON-FUNCTIONAL REQUIREMENTS

The major non-functional Requirements of the system are as follows

Usability

The system is designed with completely automated process hence there is no or less user intervention.

Reliability

The system is more reliable because of the qualities that are inherited from the chosen platform python. The code built by using python is more reliable.

Performance

This system is developing in the high level languages and using the advanced back-end technologies it will give response to the end user on client system with in very less time.

Supportability

The system is designed to be the cross platform supportable. The system is supported on a wide range of hardware and any software platform, which is built into the system.

Implementation

The system is implemented in web environment using Jupyter notebook software. The server is used as the intelligence server and windows 10 professional is used as the platform. Interface the user interface is based on Jupyter notebook provides server system.

CHAPTER 4
PROJECT DESCRIPTION

4.1 GENERAL

Jellyfish detection using YOLOv11 marks a cutting-edge approach in real-time marine monitoring, leveraging the advancements in object detection to address the complexities of underwater environments. The unique characteristics of jellyfish, such as their semi-transparent bodies, fluid movements, and varying sizes, make them a challenging target for detection systems. However, YOLOv11's innovative architecture, designed for both speed and accuracy, offers a robust solution for identifying jellyfish even under challenging underwater conditions.

YOLOv11 builds on its predecessors by integrating advanced techniques like more efficient convolutional layers, improved feature extraction, and enhanced multi-task learning capabilities. These enhancements allow the model to better detect jellyfish, which often appear in cluttered, noisy environments or in large, overlapping groups. One of the standout features of YOLOv11 is its ability to achieve accurate detection in real-time without compromising speed, making it ideal for continuous monitoring in dynamic marine settings.

The model excels at distinguishing jellyfish from their background, which is often composed of water, sand, or other marine organisms. YOLOv11's ability to handle transparency and irregular shapes significantly enhances its ability to detect jellyfish even when they blend into their environment. This capability is crucial for applications such as real-time jellyfish bloom detection, where accurate identification is necessary for early warnings to coastal communities or the fishing industry.

4.2 METHODOLOGIES

4.2.1 MODULES NAME

1. CoordConv Modules
2. Global Average Pooling (GAP)
3. Generative Attention Modules (GAM)
4. Loss Functions
5. Data Augmentation Techniques
6. Post-Processing with Non-Maximum Suppression (NMS)

4.2.2 MODULES EXPLANATION

CoordConv Modules: YOLOv11 replaces conventional convolution layers with CoordConv (Coordinate Convolution) layers in its backbone. This modification allows the model to better understand spatial information, which is crucial for detecting transparent or semi-transparent objects like jellyfish in underwater scenes.

Global Average Pooling (GAP): YOLOv11 uses GAP to aggregate features across the entire image. This technique improves the model's robustness in handling different object scales and complex backgrounds, enabling more accurate jellyfish detection across a variety of underwater environments.

Generative Attention Modules (GAM): These modules help the model learn more focused and discriminative features, improving its ability to distinguish jellyfish from surrounding noise or similar marine organisms.

Loss Functions: YOLOv11 introduces new loss functions that improve the localization accuracy of jellyfish detections. These include both bounding box regression loss and classification loss, ensuring the model not only detects the jellyfish but also localizes it precisely.

Data Augmentation Techniques: YOLOv11 applies a range of data augmentation strategies, such as random cropping, scaling, and rotation, to increase the model's robustness to different

jellyfish appearances and environmental conditions. This helps the model generalize better to unseen underwater scenes.

Post-Processing with Non-Maximum Suppression (NMS): To refine the detection results and eliminate redundant bounding boxes, YOLOv11 uses NMS. This technique ensures that only the most confident detections remain, improving the precision of jellyfish identification in complex or crowded scenes.

4.3 TECHNIQUE USED OR ALGORITHM USED

4.3.1 EXISTING TECHNIQUE

The YOLO series of object detection models, from YOLOv5 to YOLOv8, have significantly advanced real-time object detection, each iteration introducing improvements in speed, accuracy, and versatility. YOLOv5, while not officially part of the original YOLO family, is known for its flexibility and ease of use, becoming popular for real-time detection tasks. However, it faces challenges in generalization to diverse datasets and struggles with small object detection. YOLOv6 brought improvements in speed and efficiency, particularly for mobile and embedded systems, but its trade-off for speed is a slight reduction in detection accuracy, especially for small or occluded objects. YOLOv7, with its introduction of Efficient Layer Aggregation Networks (ELAN) and attention mechanisms, enhanced detection capabilities, particularly for small objects, though it requires more computational resources, making it less suitable for resource-constrained environments. YOLOv8 pushed performance further by focusing on energy efficiency and modularity, but its complexity and larger model size can hinder deployment on devices with limited resources, and its inference speed may be slower than earlier versions.

DRAWBACKS

Optimization Trade-offs: While YOLOv6 is faster and more optimized, it sacrifices some level of accuracy, particularly when detecting smaller or highly occluded objects.

Transparency and Complex Shapes: Jellyfish detection, for instance, involves transparent and irregularly shaped objects. All YOLO versions face challenges with transparency and complex, non-rigid objects, as they rely heavily on pixel-based features that can blend with the background.

Inconsistent Performance Across Datasets: While YOLO versions are highly efficient, their performance can degrade when applied to diverse and unbalanced datasets without fine-tuning, especially when detecting rare or atypical object types.

Lack of Robustness to Occlusions: Despite improvements in later versions, occlusions (where jellyfish overlap or are obscured by other objects) remain a limitation in real-world applications.

4.3.2 PROPOSED TECHNIQUE USED OR ALGORITHM USED

The proposed YOLOv11 model represents a significant advancement in detecting jellyfish, particularly in underwater environments where detection challenges include transparency, irregular shapes, and cluttered backgrounds. YOLOv11 builds on the strengths of its predecessors by integrating cutting-edge techniques such as **Enhanced Attention Mechanisms** and **CoordCov Modules** within its backbone architecture, improving the model's ability to detect small and transparent objects like jellyfish. These innovations enable YOLOv11 to capture finer details and spatial relationships, which is crucial for identifying jellyfish in complex underwater scenes. Additionally, YOLOv11 leverages **Global Attention Mechanisms (GAM)**, allowing the model to focus on the most relevant features, significantly enhancing detection accuracy in real-time scenarios. The model also benefits from an **optimized computational pipeline**, making it faster and more efficient in processing underwater imagery with minimal loss in performance. This makes YOLOv11 highly suitable for deployment in both research and practical applications such as marine monitoring and conservation efforts, where accurate and swift detection of jellyfish populations is critical. With its improved detection capabilities and optimized architecture, YOLOv11 outperforms previous versions like YOLOv5, YOLOv6, and YOLOv8, especially in terms of handling challenging underwater conditions and ensuring accurate identification of jellyfish across varying environments.

ADVANTAGES

Enhanced Detection of Small and Transparent Objects: The integration of Enhanced Attention Mechanisms and CoordCov Modules allows YOLOv11 to better capture intricate features, making it more effective in detecting small and transparent jellyfish in cluttered underwater environments.

Global Attention Mechanisms (GAM): This enables YOLOv11 to focus on the most relevant features in the image, reducing noise and ensuring that jellyfish are accurately detected even in complex, feature-rich environments.

Improved Accuracy in Cluttered Backgrounds: YOLOv11's ability to focus on key features enhances its performance in detecting jellyfish among other marine life and debris, making it particularly useful for underwater scenarios with challenging visibility.

Real-Time Detection: YOLOv11 is optimized for computational efficiency, allowing it to run in real-time applications without sacrificing detection quality. This is crucial for time-sensitive marine monitoring and conservation efforts.

Optimized Computational Pipeline: The model's design ensures faster processing of underwater imagery, allowing it to be deployed on devices with limited computational resources without compromising performance.

CHAPTER 5

DESIGN ENGINEERING

5.1 GENERAL

Design Engineering deals with the various UML [Unified Modelling language] diagrams for the implementation of project. Design is a meaningful engineering representation of a thing that is to be built. Software design is a process through which the requirements are translated into representation of the software. Design is the place where quality is rendered in software engineering.

5.2 UML DIAGRAMS

5.2.1 USE CASE DIAGRAM

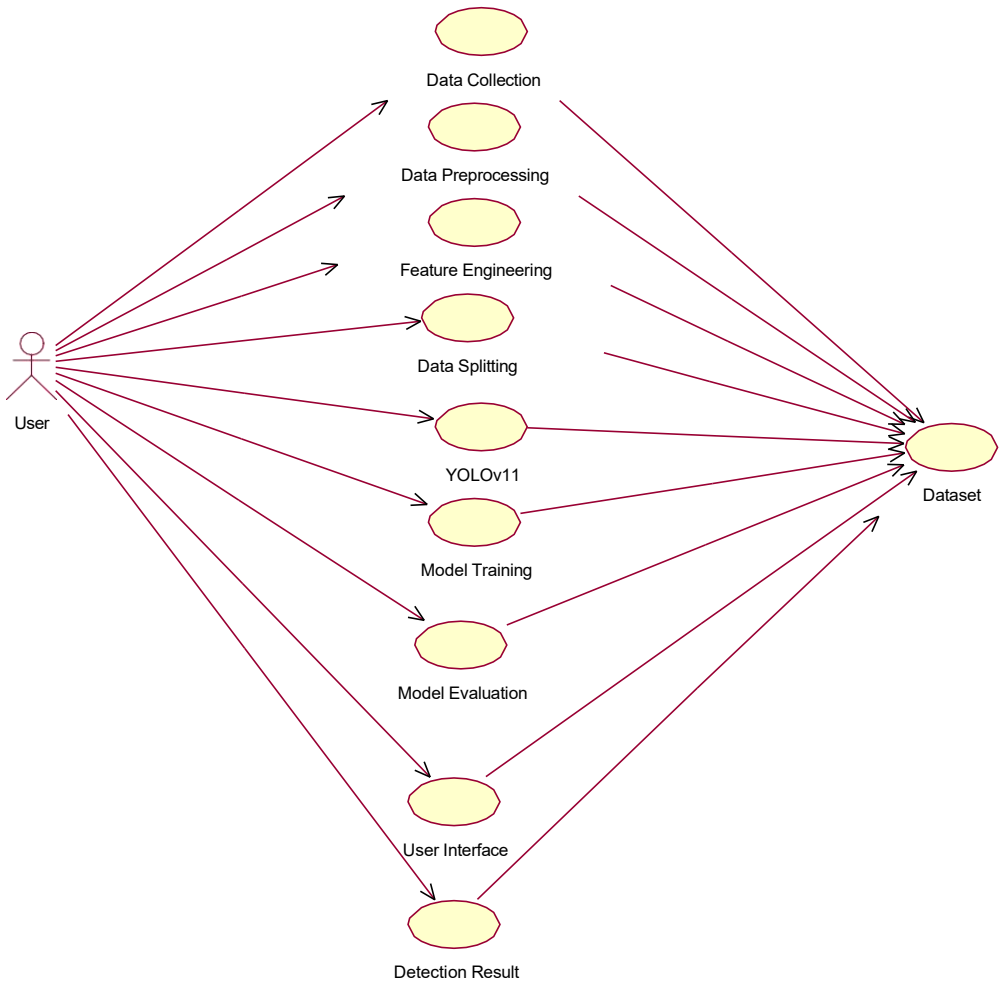


Fig 5.2.1 : Use Case Diagram

EXPLANATION

The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted. The above diagram consists of user as actor. Each will play a certain role to achieve the concept.

5.2.2 CLASS DIAGRAM

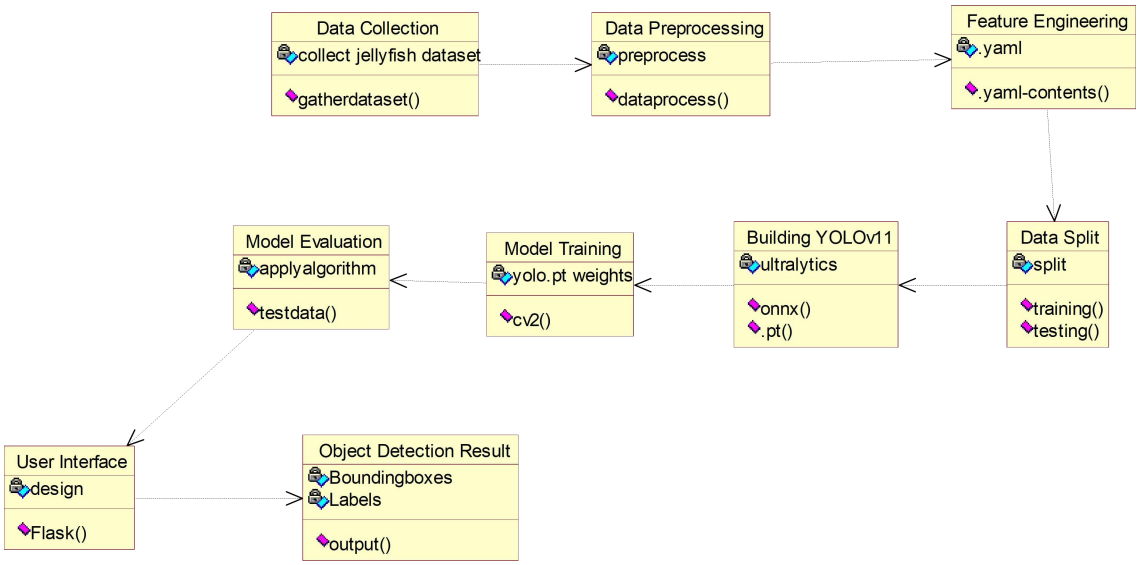


Fig 5.2.2: Class Diagram

EXPLANATION

In this class diagram represents how the classes with attributes and methods are linked together to perform the verification with security. From the above diagram shown the various classes involved in our project.

5.2.3 OBJECT DIAGRAM

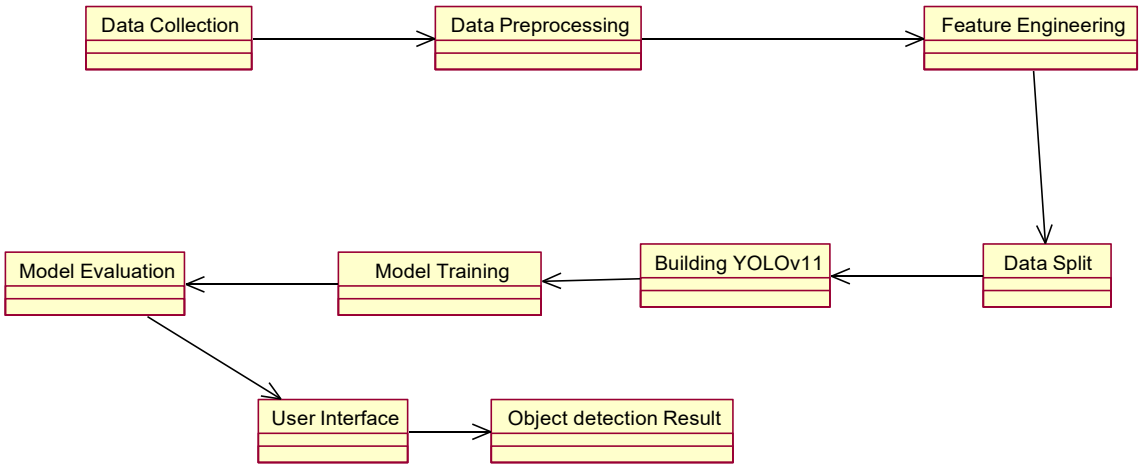


Fig 5.2.3: Object Diagram

EXPLANATION

In the above digram tells about the flow of objects between the classes. It is a diagram that shows a complete or partial view of the structure of a modeled system. In this object diagram represents how the classes with attributes and methods are linked together to perform the verification with security.

5.2.4 STATE DIAGRAM

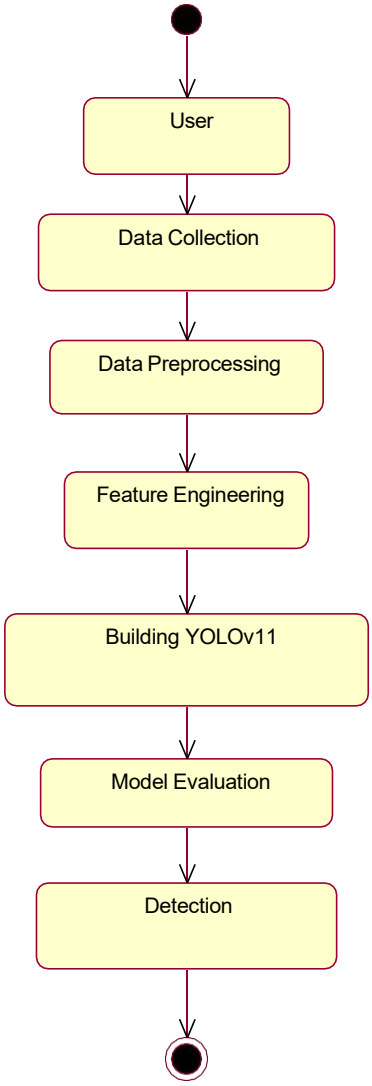


Fig 5.2.4: State Diagram

EXPLANATION

State diagram is a loosely defined diagram to show workflows of stepwise activities and actions, with support for choice, iteration and concurrency. State diagrams require that the system described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction. Many forms of state diagrams exist, which differ slightly and have different semantics.

5.2.5 ACTIVITY DIAGRAM

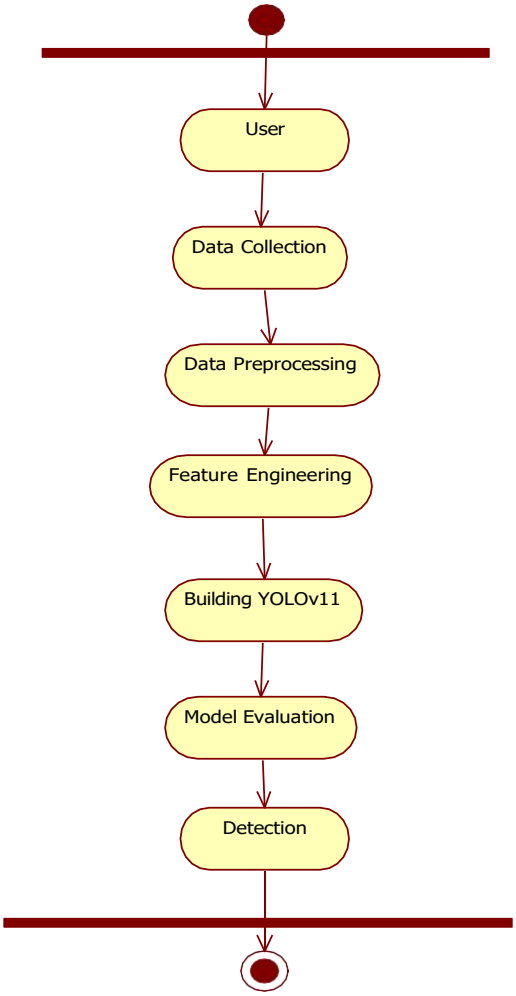


Fig: 5.2.5 : Activity Diagram

EXPLANATION

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modelling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

5.2.6 SEQUENCE DIAGRAM

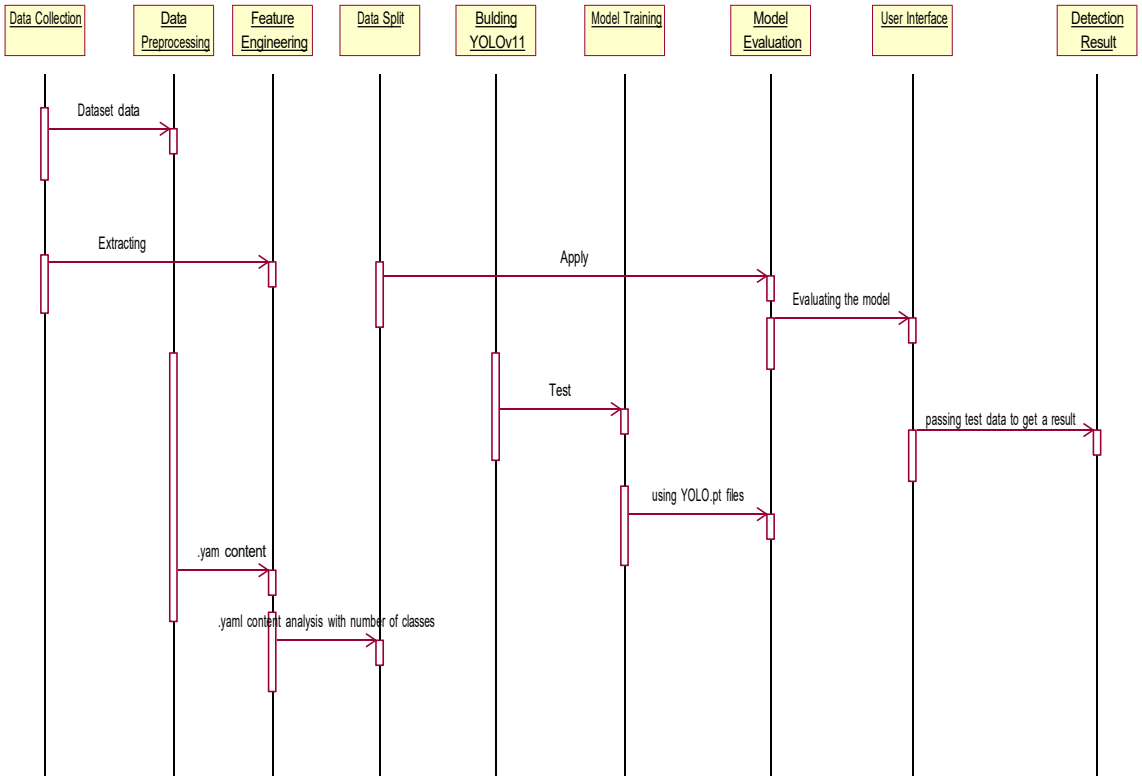


Fig 5.2.6: Sequence Diagram

EXPLANATION

A sequence diagram in Unified Modelling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

5.2.7 COLLABORATION DIAGRAM

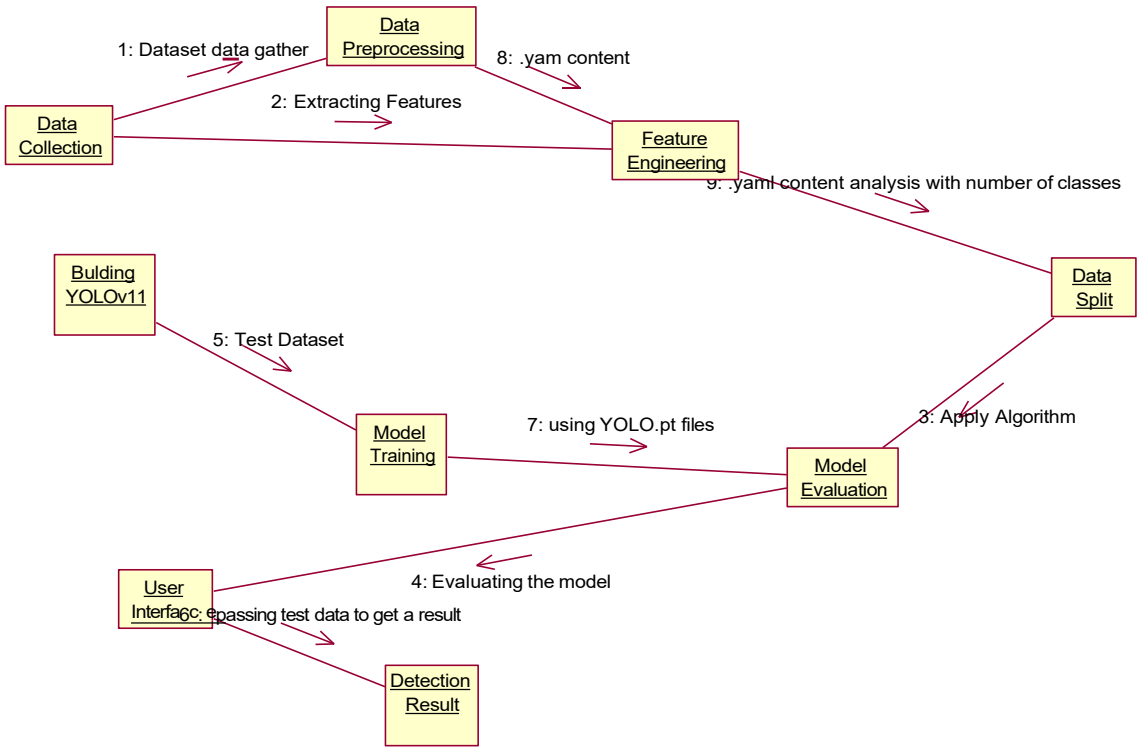


Fig 5.2.7: Collaboration Diagram

EXPLANATION

A collaboration diagram, also called a communication diagram or interaction diagram, is an illustration of the relationships and interactions among software objects in the Unified Modelling Language (UML). The concept is more than a decade old although it has been refined as modelling paradigms have evolved.

5.2.8 COMPONENT DIAGRAM

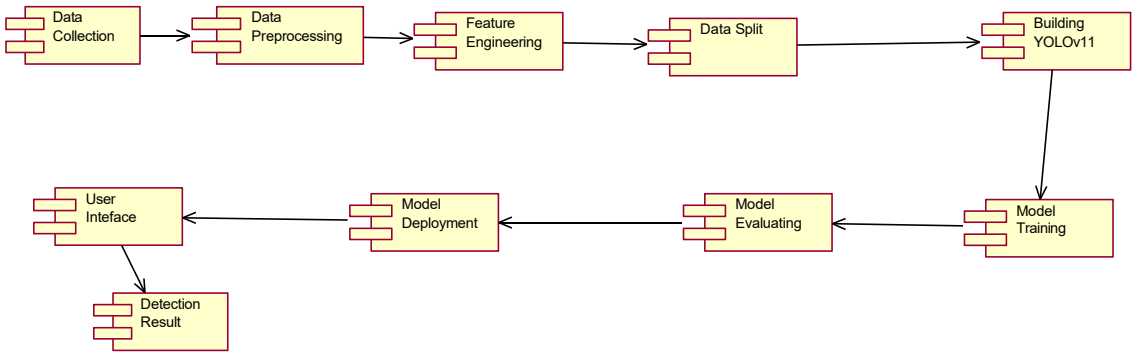


Fig 5.2.8: Component Diagram

EXPLANATION

In the Unified Modelling Language, a component diagram depicts how components are wired together to form larger components and or software systems. They are used to illustrate the structure of arbitrarily complex systems. User gives main query and it converted into sub queries and sends through data dissemination to data aggregators. Results are to be showed to user by data aggregators. All boxes are components and arrow indicates dependencies.

5.2.9 DATA FLOW DIAGRAM

Level 0

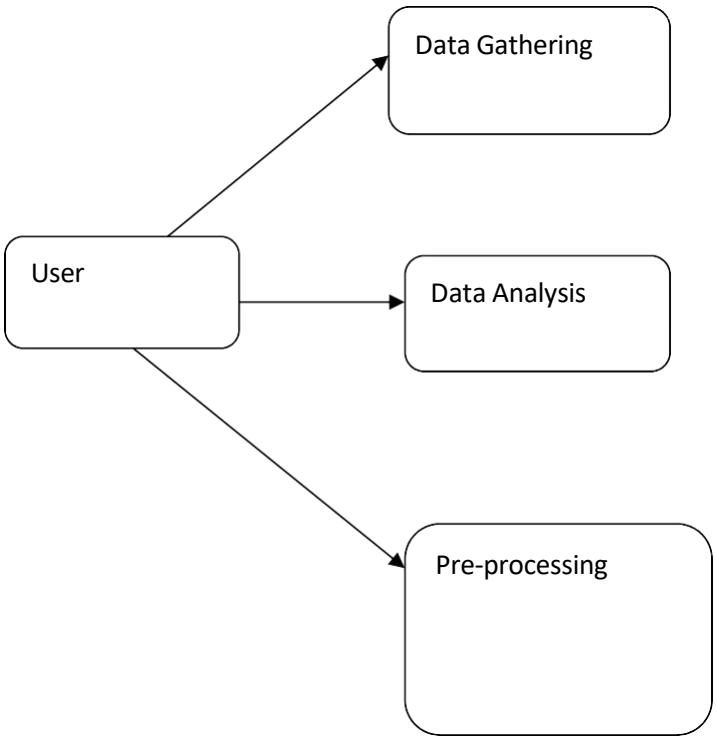


Fig 5.2.9.1: Data Flow Diagrams

Level 1

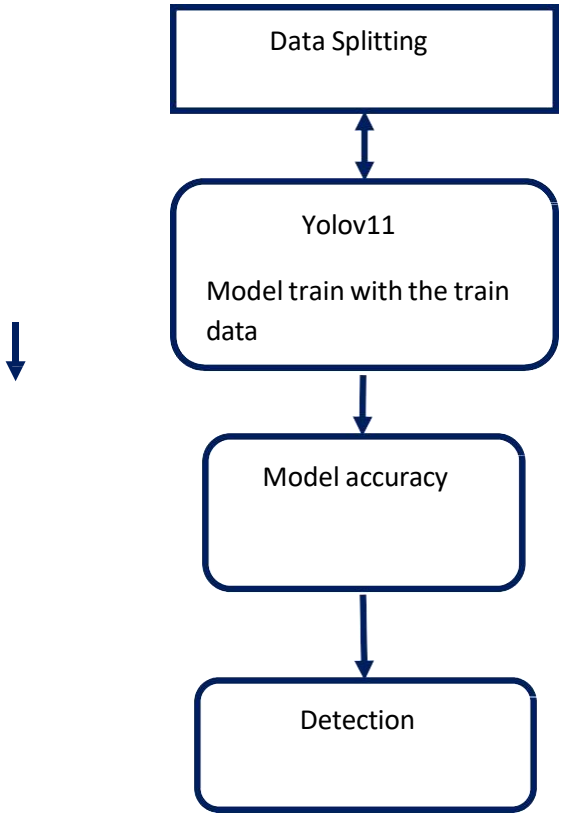


Fig 5.2.9.2: Data Flow Diagrams

EXPLANATION

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modelling its process aspects. Often they are a preliminary step used to create an overview of the system which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

A DFD shows what kinds of data will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of processes, or information about whether processes will operate in sequence or in parallel.

5.2.10 DEPLOYMENT DIAGRAM

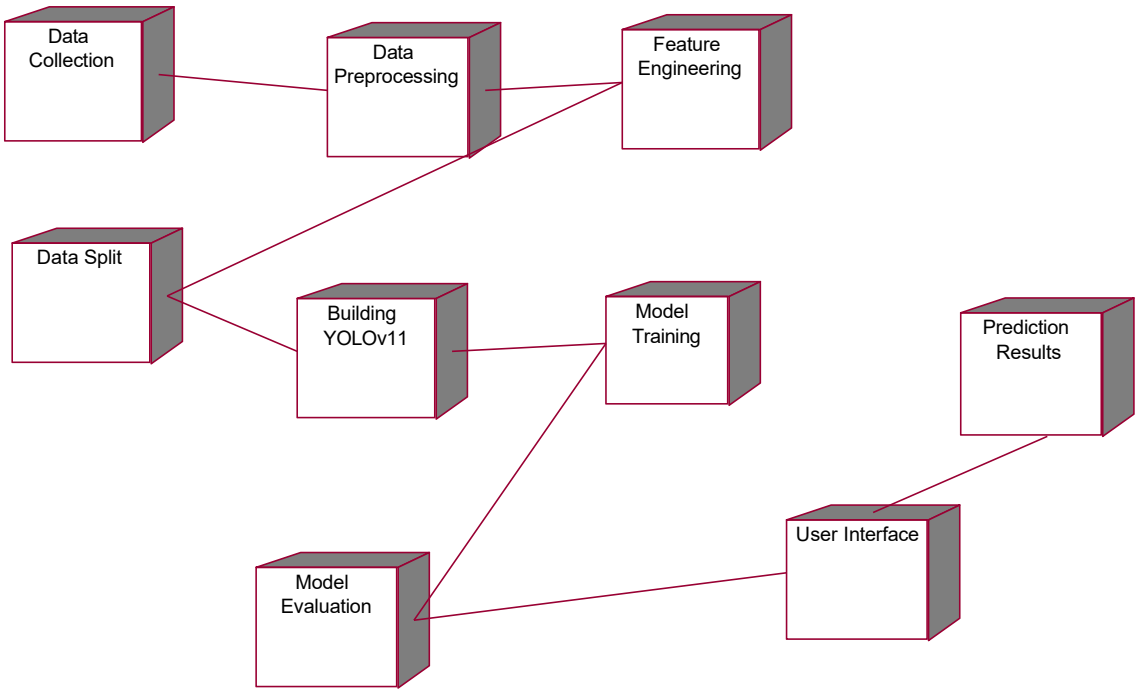


Fig: 5.2.10: Deployment Diagram

EXPLANATION

Deployment Diagram is a type of diagram that specifies the physical hardware on which the software system will execute. It also determines how the software is deployed on the underlying hardware. It maps software pieces of a system to the device that are going to execute it.

5.2.11 SYSTEM ARCHITECTURE

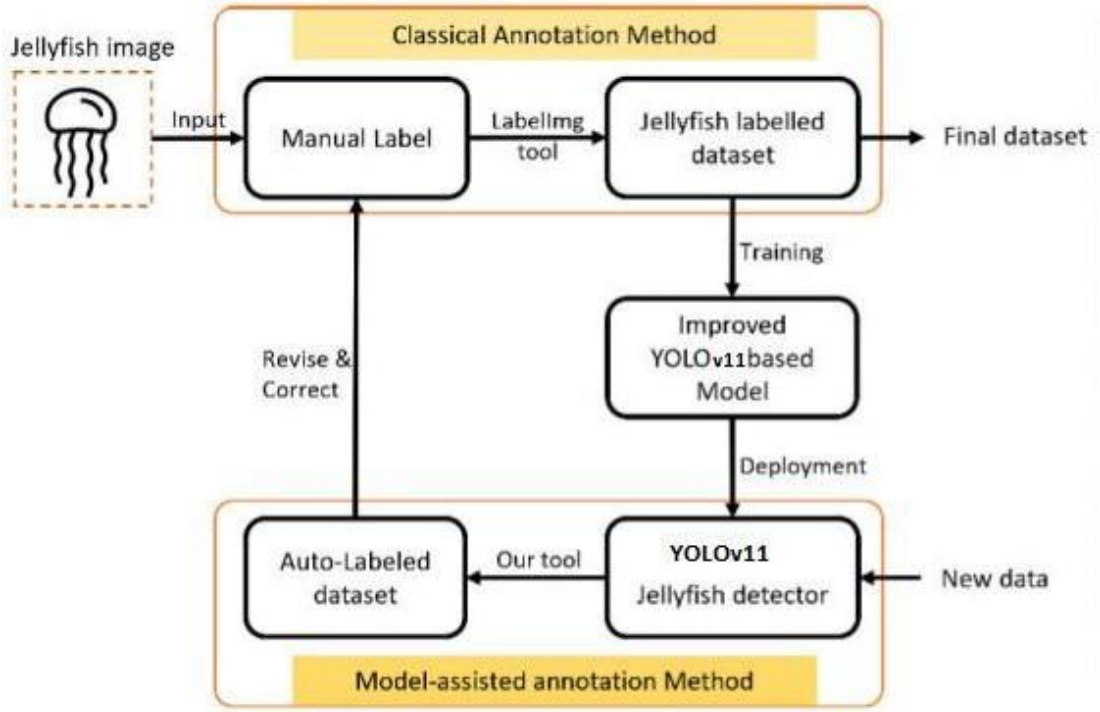


Fig 5.2.11 System Architecture

CHAPTER 6

DEVELOPMENT TOOLS

6.1 GENERAL

Fig 6.1.1: download python

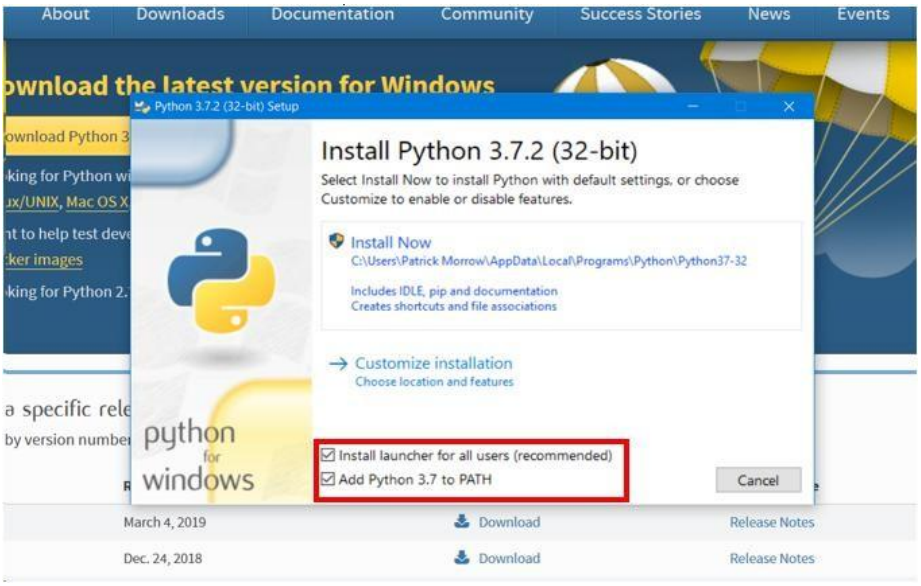


Fig 6.1.2: Installation of python

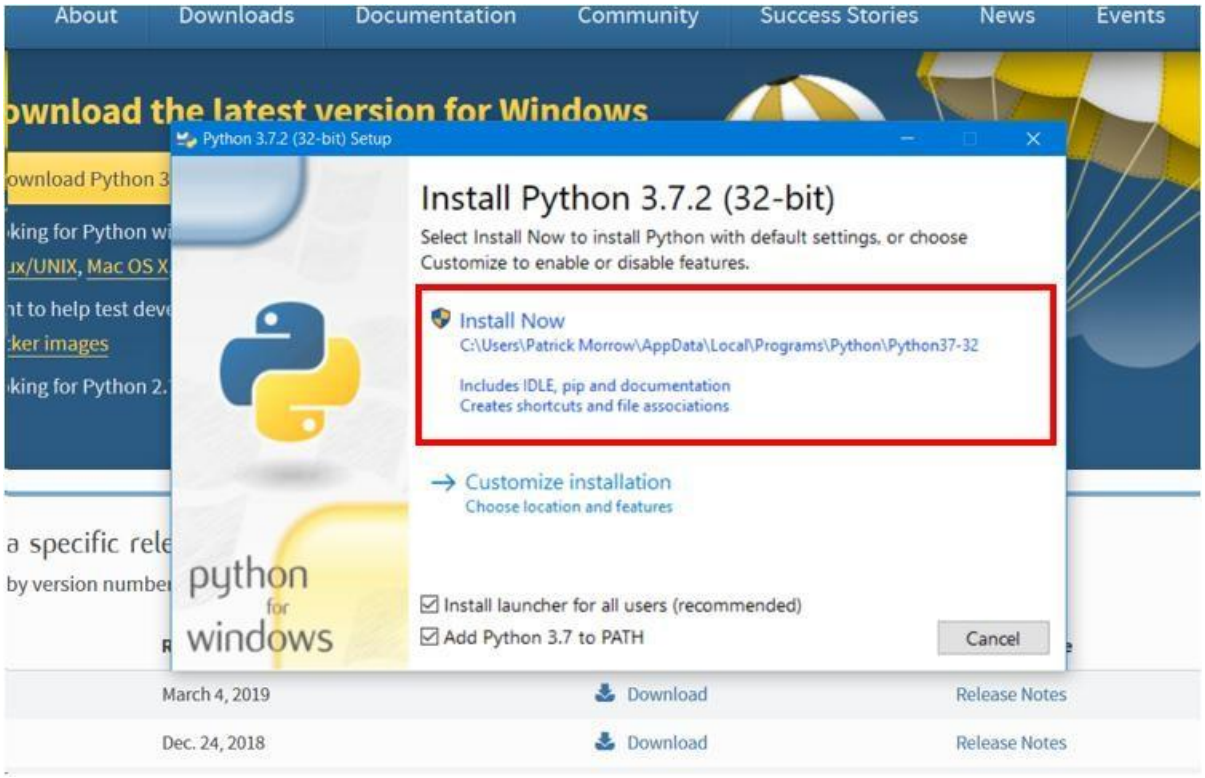


Fig 6.1.3: setup window

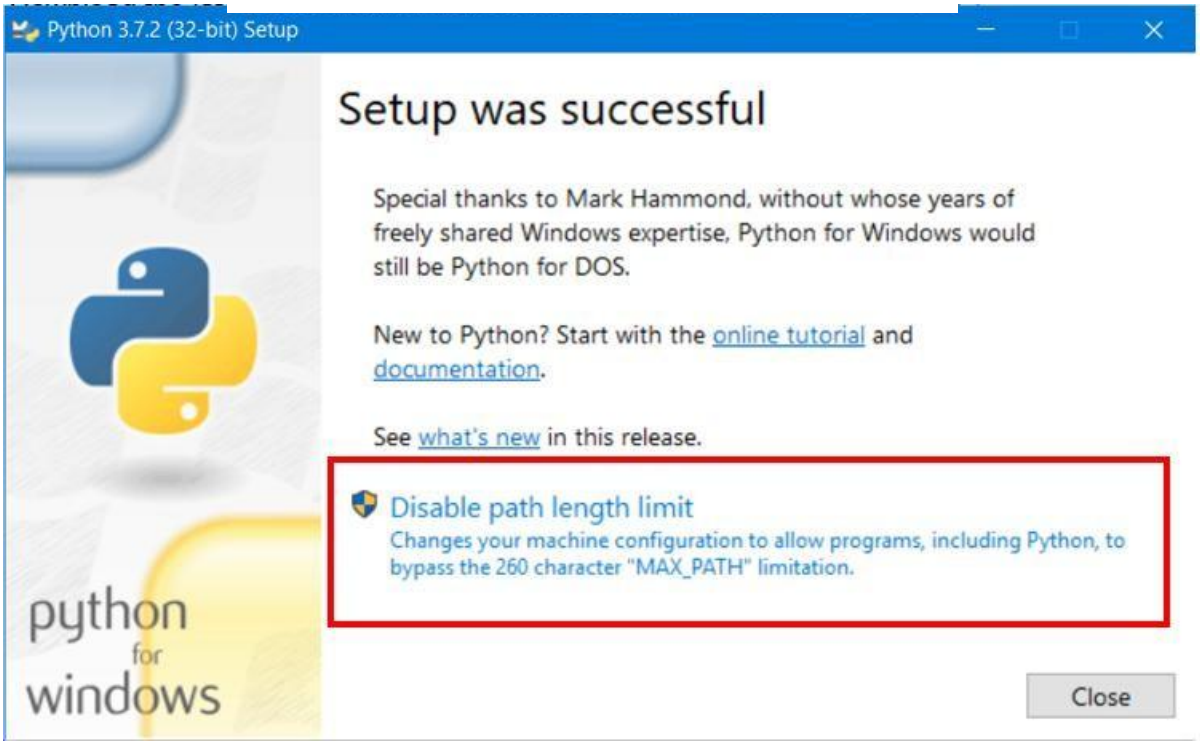


Fig 6.1.4: successfully setup python

Download Install PyCharm

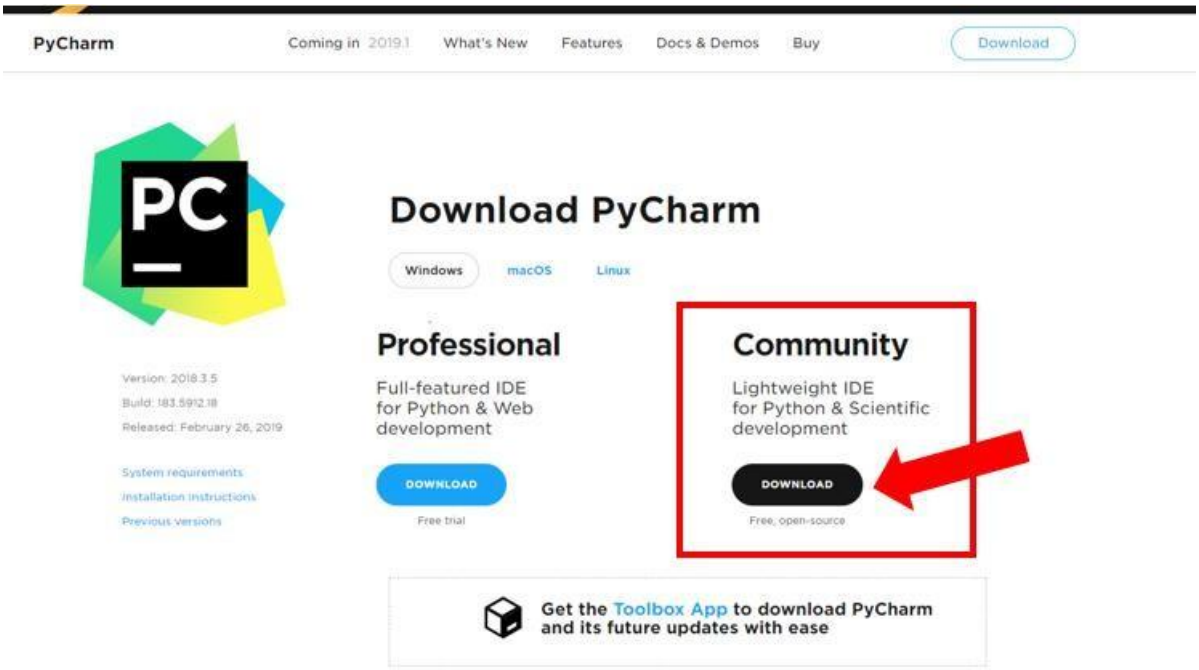


Fig 6.1.5: download pycharm

Fig 6.1.6: pycharm setup

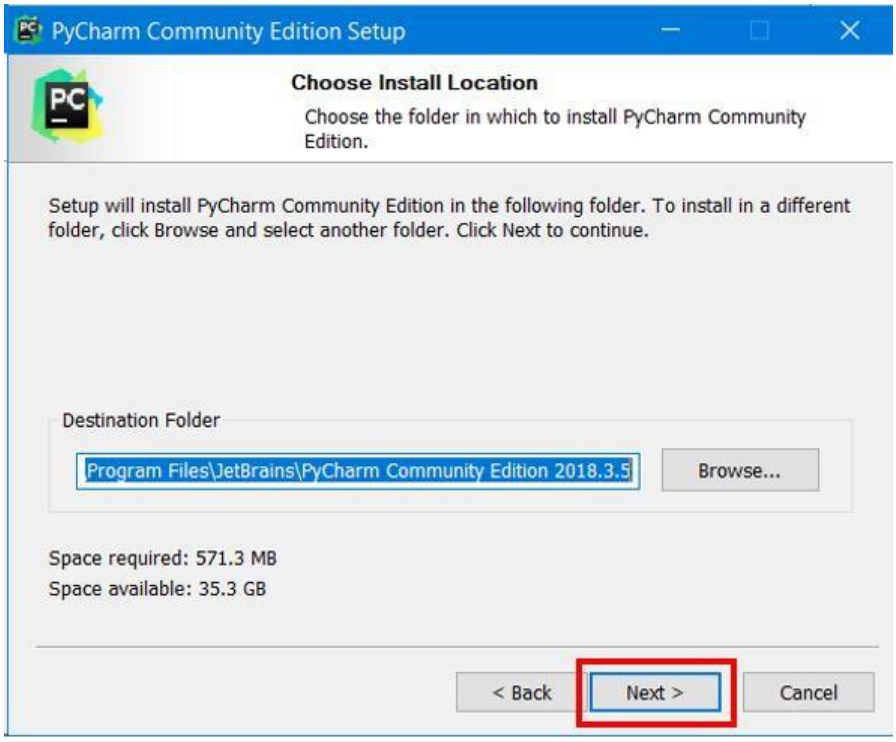


Fig 6.1.7: Setup window

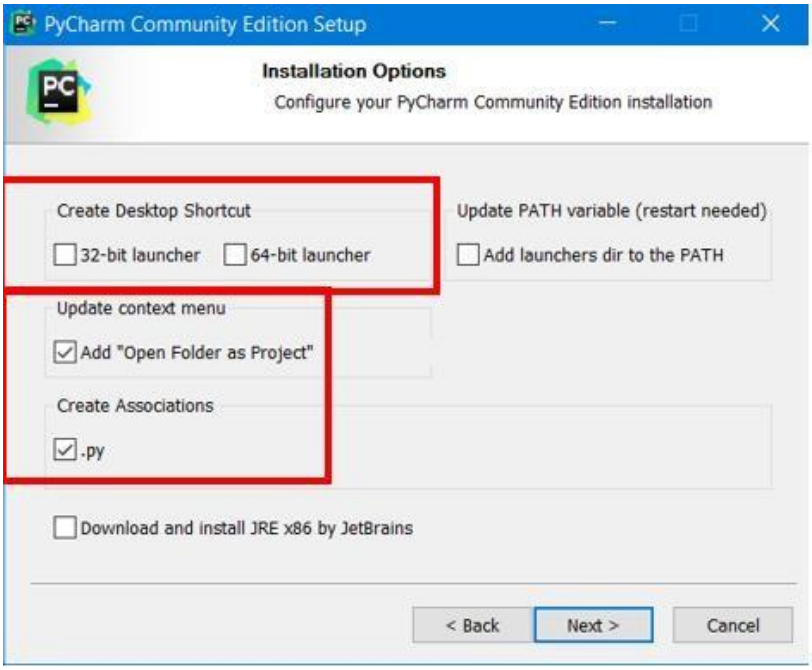


Fig 6.1.8: Installation pycharm

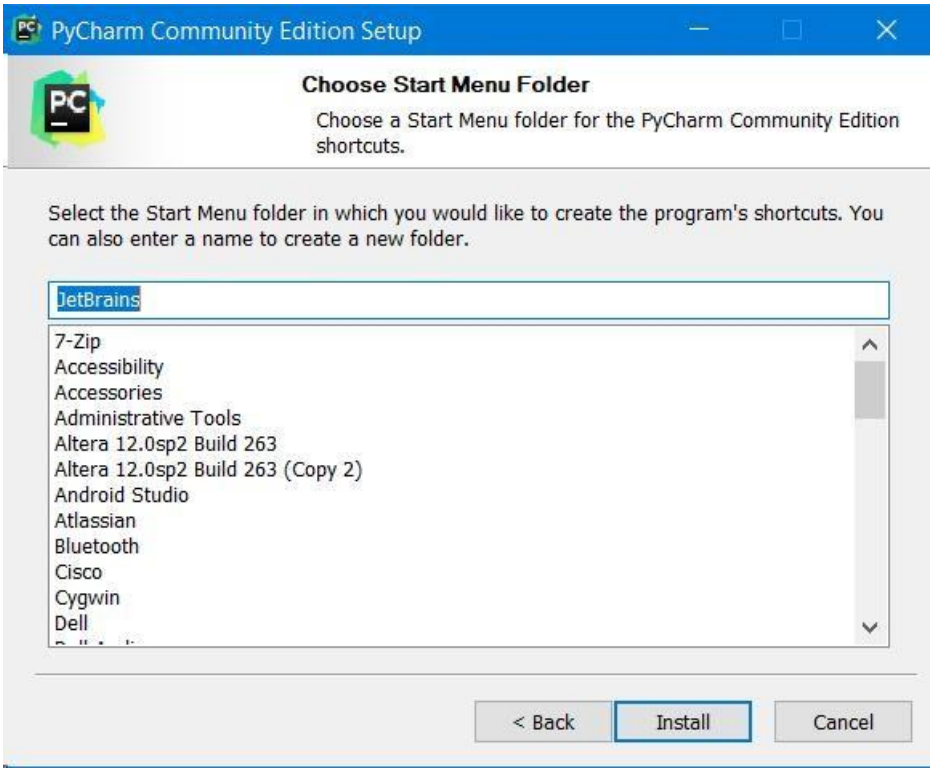


Fig 6.1.9: Start menu folder

Fig 6.1.10: setup finish

PyCharm--Setup

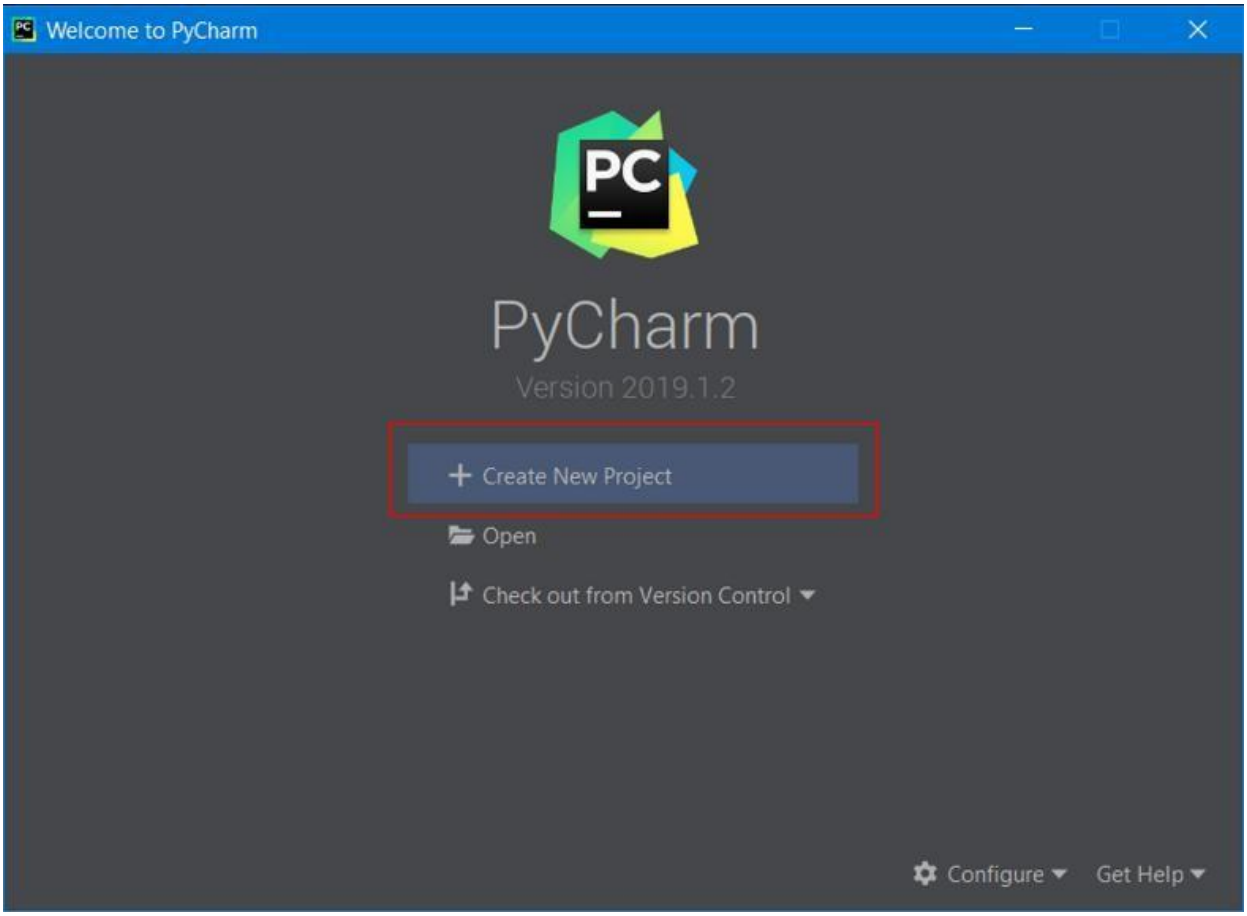


Fig 6.1.11: setup completion

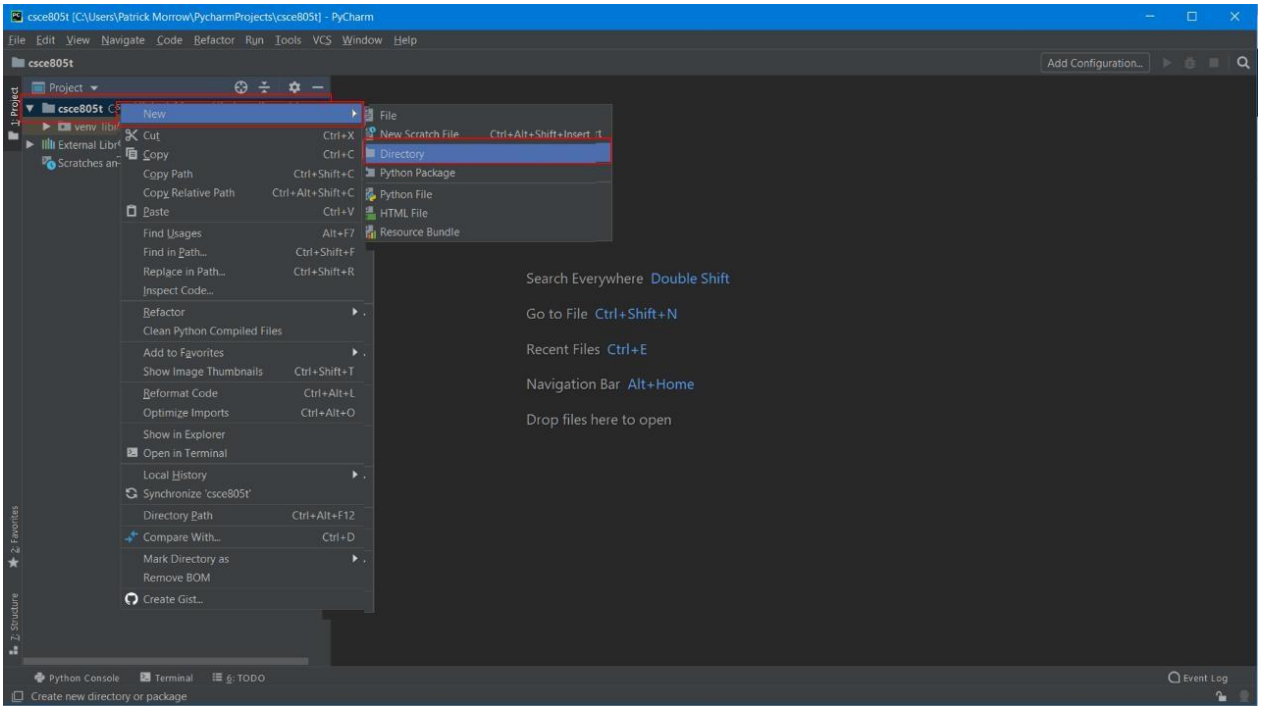


Fig 6.1.12: creating Directory

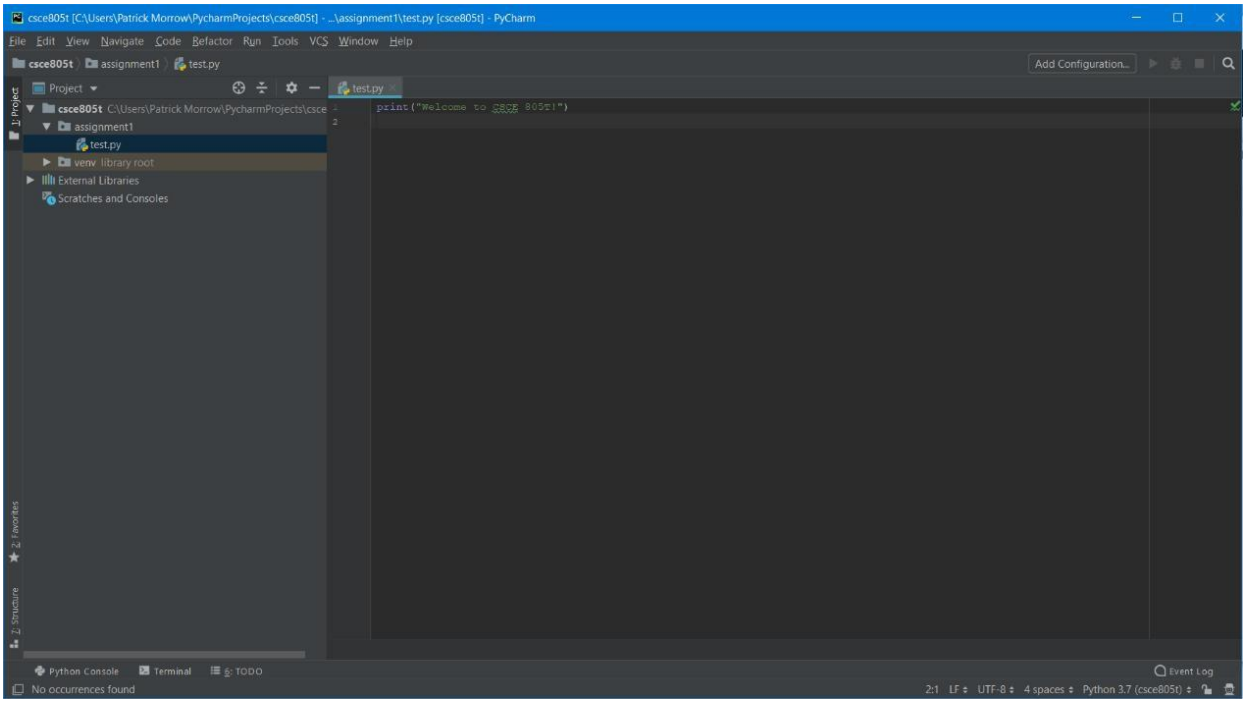


Fig 6.1.13: new directory created

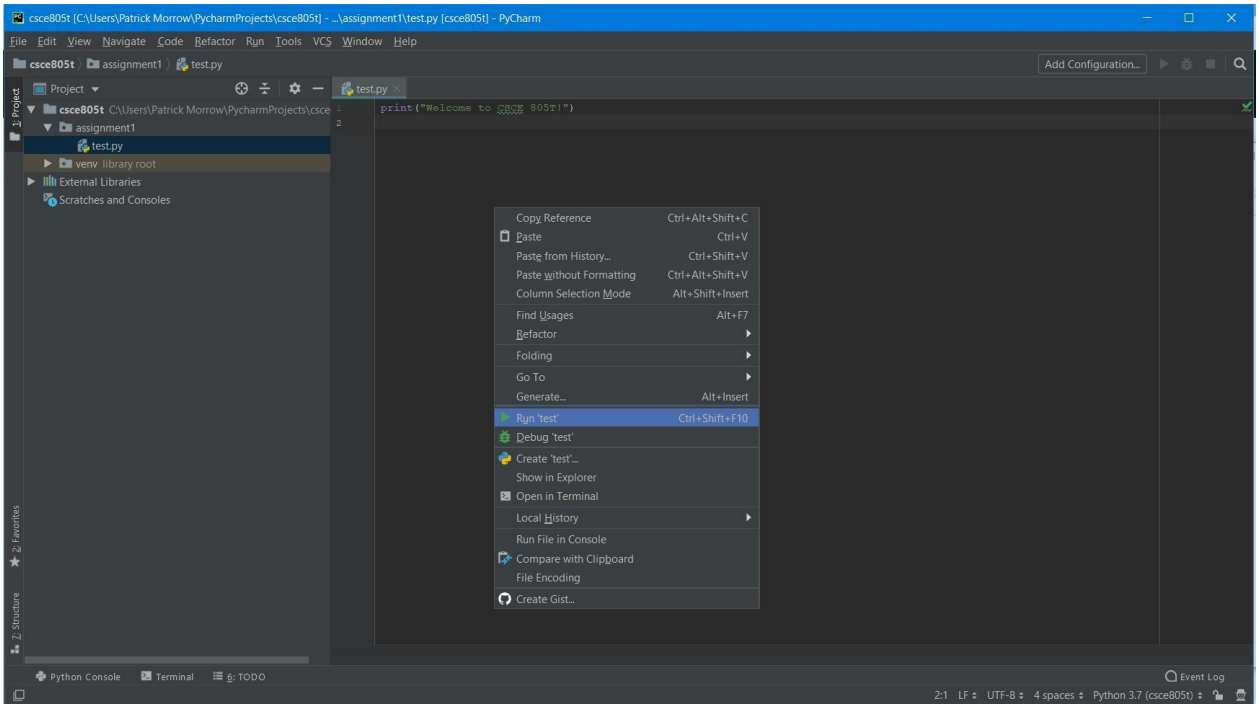


Fig 6.1.14: testing pycharm

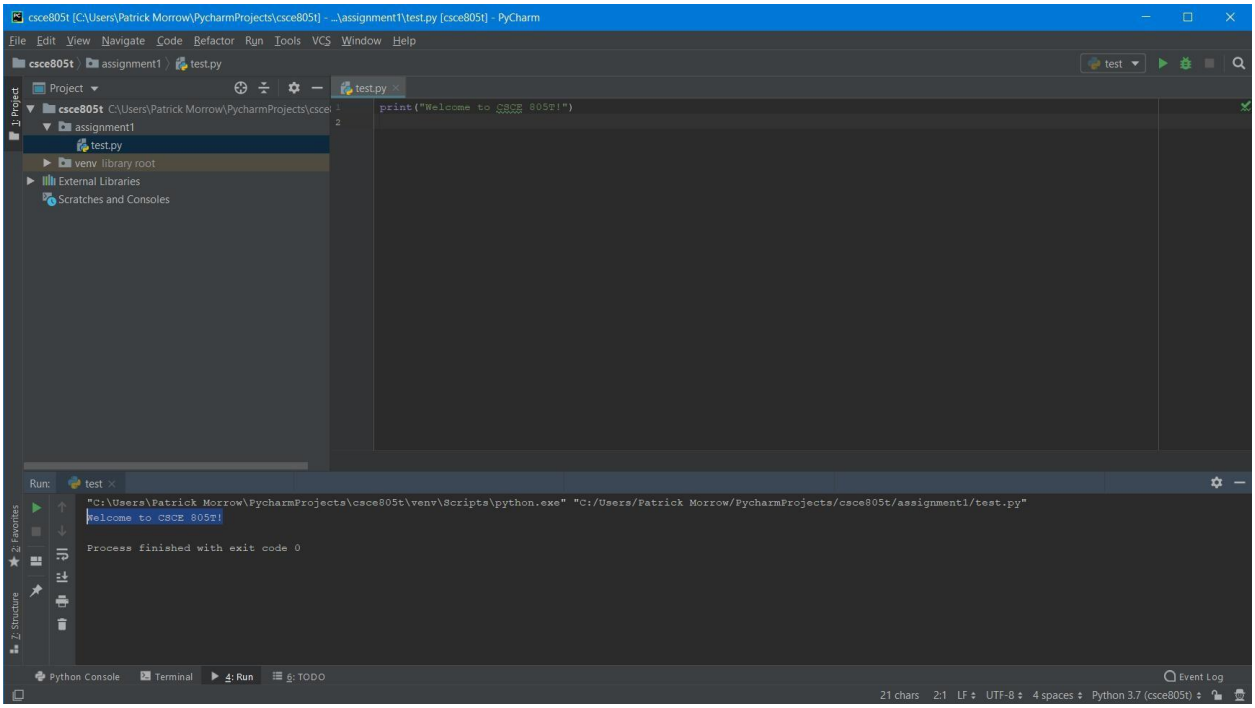


Fig 6.1.15: test successful

Installing Anaconda Navigator

Fig 6.1.16: installing Anaconda3

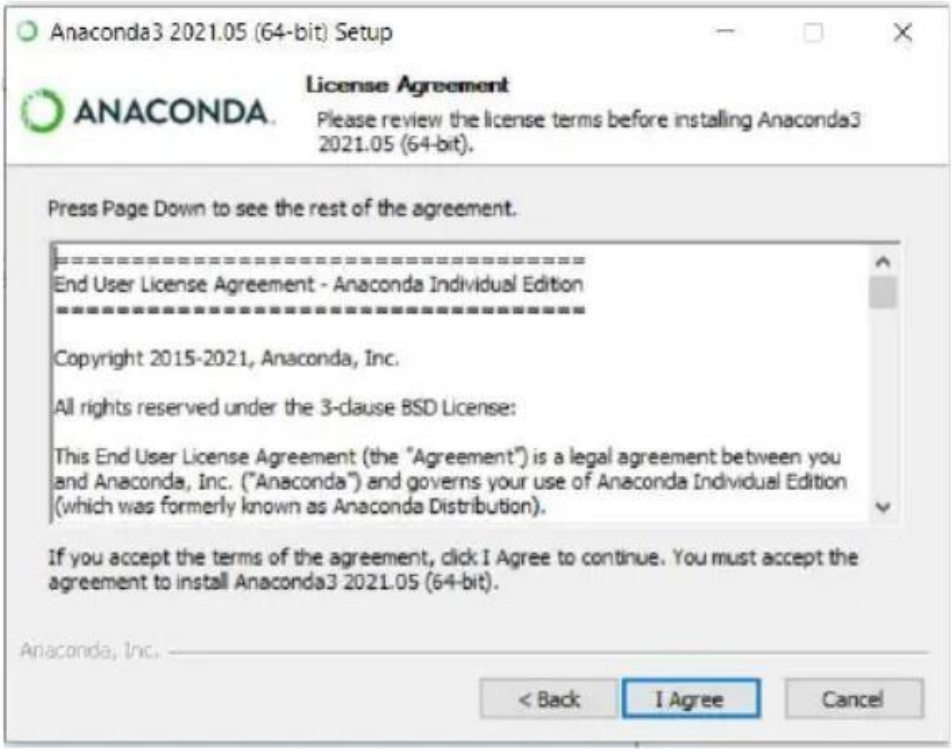


Fig 6.1.17: setup window

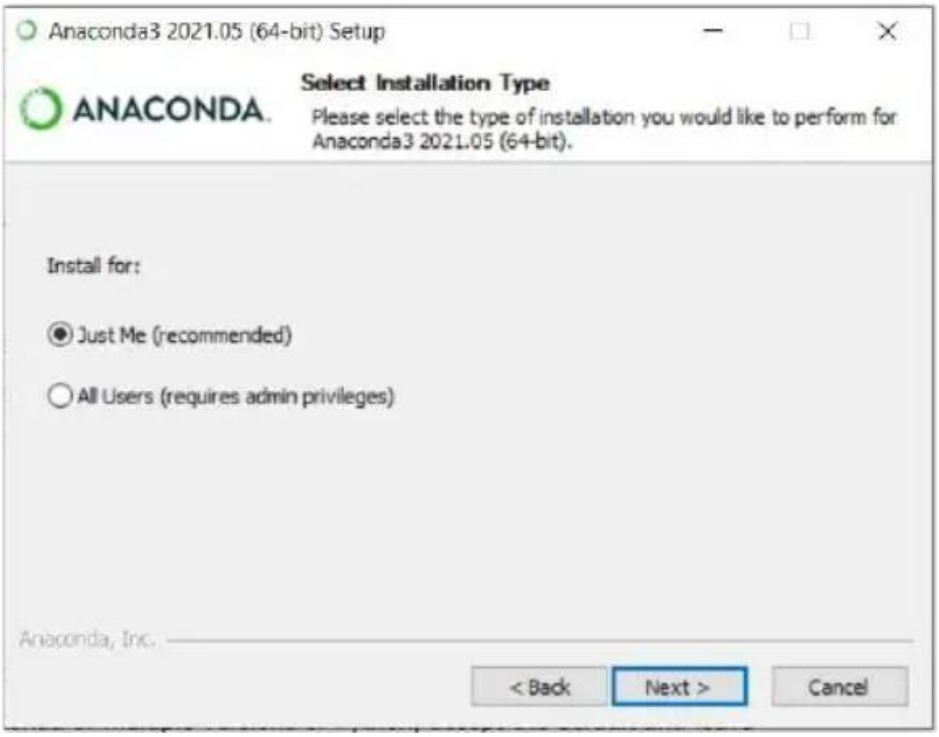


Fig 6.1.18: Installation type

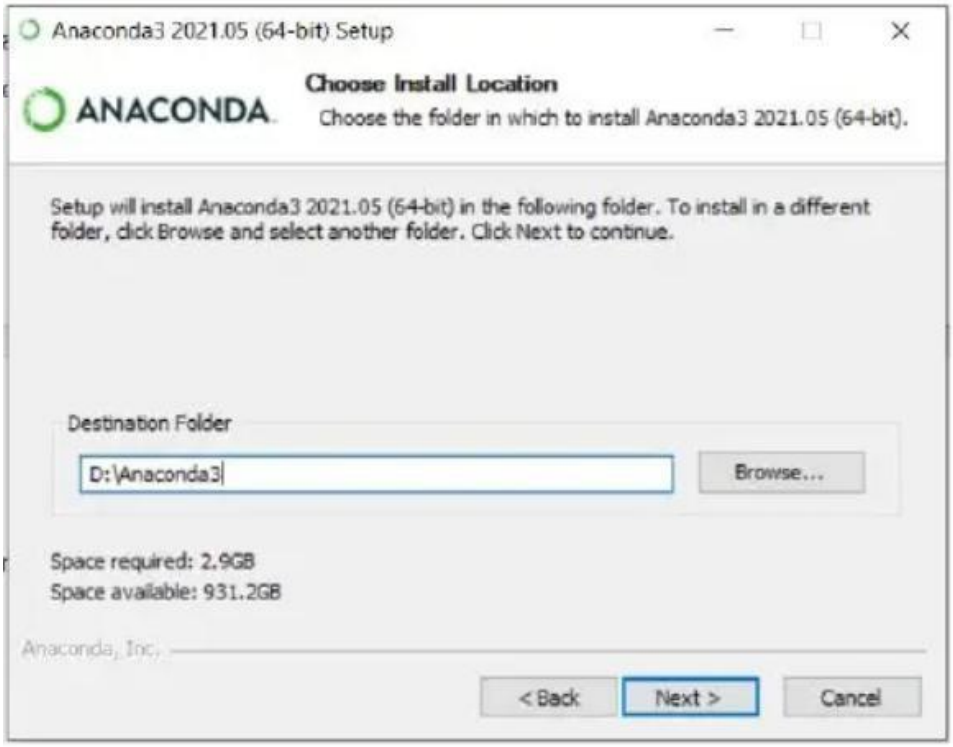


Fig 6.1.19: Installation location

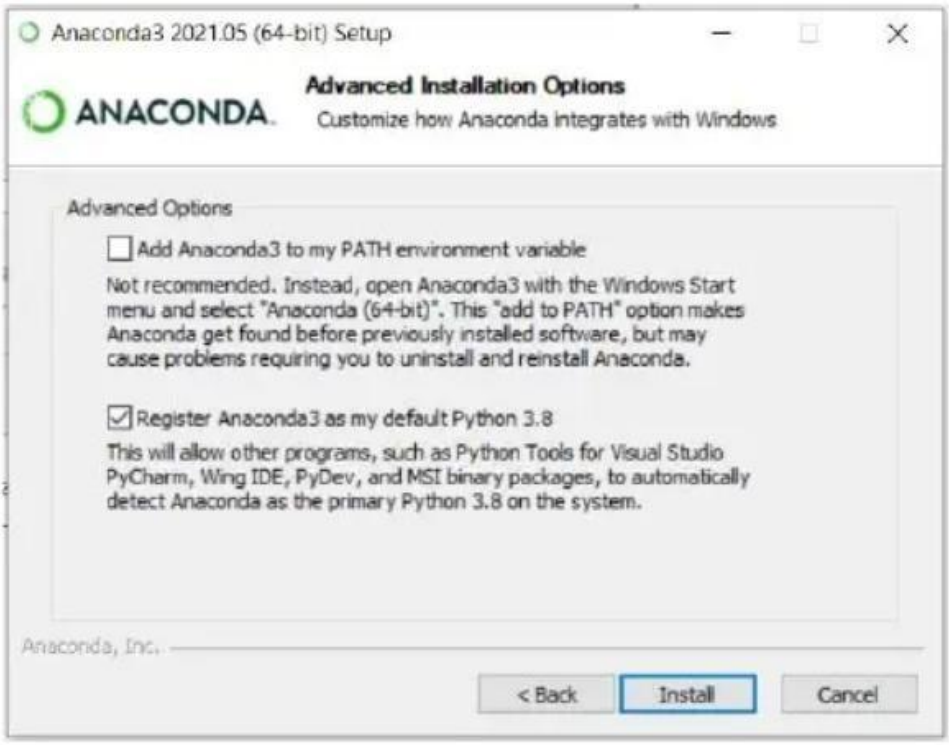


Fig 6.1.20: Installation options

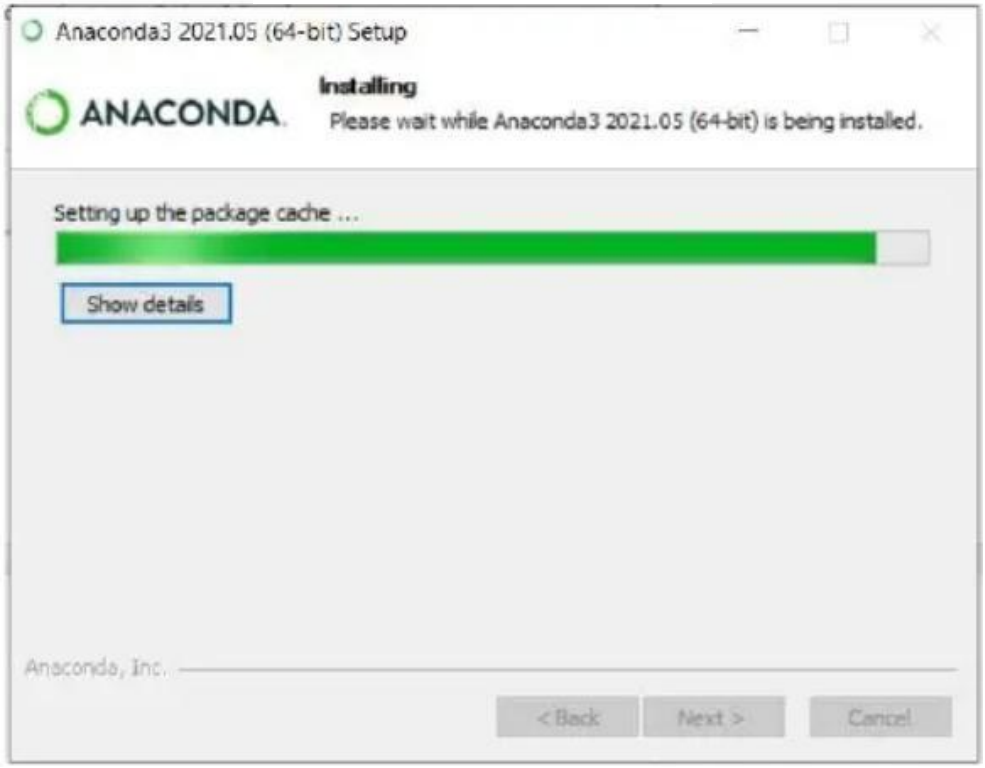


Fig 6.1.21: Installing

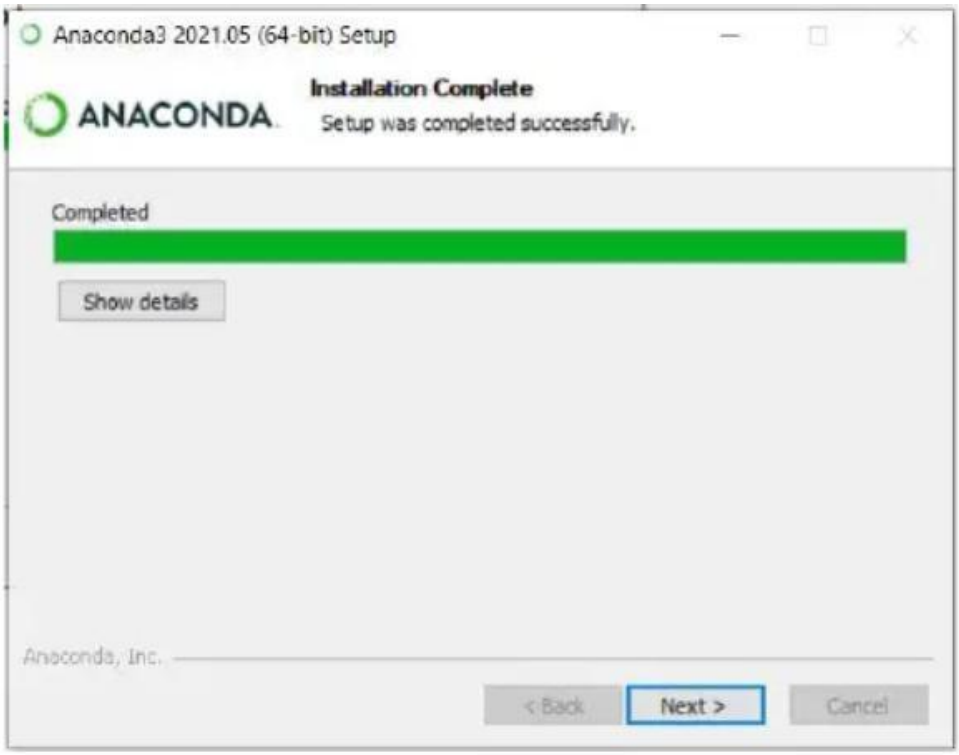


Fig 6.1.22: Installation completed

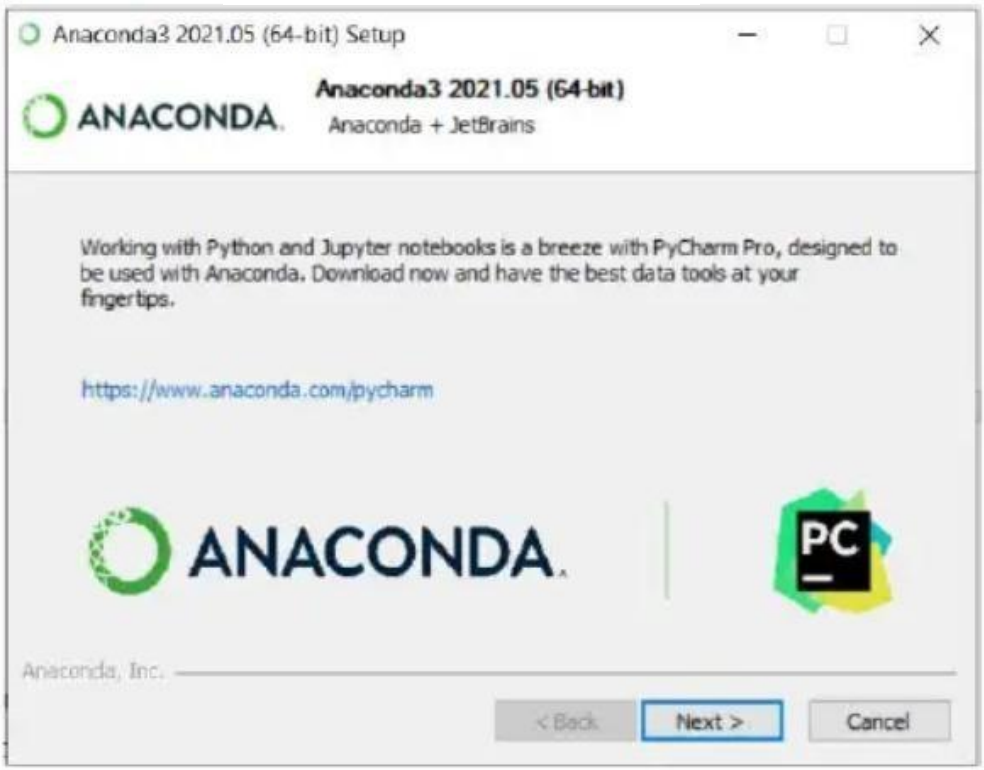


Fig 6.1.23: successful installation

6.2 HISTORY OF PYTHON

The Python programming language was created by Guido van Rossum in the late 1980s at the Centrum Wiskunde & Informatica (CWI) in the Netherlands. It was developed as a successor to the ABC language, incorporating features from other languages such as C, Modula-3, and Unix shell scripting. The name "Python" was inspired by the British comedy series "Monty Python's Flying Circus," reflecting van Rossum's desire to make the language fun and accessible. Python 0.9.0 was officially released on February 20, 1991, and it already included many features that still define the language today, such as functions, exceptions, modules, and classes with inheritance.

Python continued to evolve with the release of the Python 1.x series throughout the 1990s, gaining popularity for its simplicity and readability. In October 2000, Python 2.0 was released, introducing significant features like list comprehensions, garbage collection, and limited Unicode support. Python 2.7, released in 2010, became the final version of the 2.x series and remained in use for many years until official support ended on January 1, 2020.

To address various design limitations and to modernize the language, Python 3.0 was launched in December 2008. This version was not backward compatible with Python 2, and it introduced major changes such as the use of the print() function instead of a statement, better Unicode handling, and the removal of outdated features. Since then, Python 3.x has continued to grow, with recent versions like Python 3.10, 3.11, and 3.12 introducing enhancements such as structural pattern matching**,** improved type hinting, and performance improvements.

6.3 FEATURES OF PYTHON

1. Simple and Easy to Learn

Python has a clear and readable syntax that resembles the English language, making it easy for beginners to understand and write code quickly.

2. Open Source and Free

Python is open-source, meaning its source code is freely available. Anyone can download, modify, and distribute it without paying any license fee.

3. Cross-Platform Compatibility

Python is a cross-platform language, which means code written in Python can run on various operating systems like Windows, Linux, and macOS without modification.

4. Supports Multiple Programming Paradigms

Python supports object-oriented, procedural, and functional programming, providing flexibility in writing code based on the problem or developer's preference.

5. Extensive Standard Library

Python includes a large standard library with built-in modules and functions for tasks such as file handling, regular expressions, database access, and internet protocols.

6. Dynamic Typing and Automatic Memory Management

In Python, you don’t need to declare the type of variable. It uses dynamic typing and manages memory allocation and garbage collection automatically.

7. Rich Third-Party Libraries

Python has a vast ecosystem of third-party libraries like NumPy, Pandas, TensorFlow, Django, Flask, and more, which support tasks in data science, machine learning, web development, etc.

8. Easy Integration

Python can easily integrate with other languages like C, C++, and Java. This makes it ideal for extending applications and building complex systems.

9. Strong Community Support

Python has a large and active community that contributes to its development and provides a wealth of resources like tutorials, forums, and documentation for learning and problem-solving.

CHAPTER 7

IMPLEMENTATION

7. IMPLEMENTATION AND SNAPSHOTS

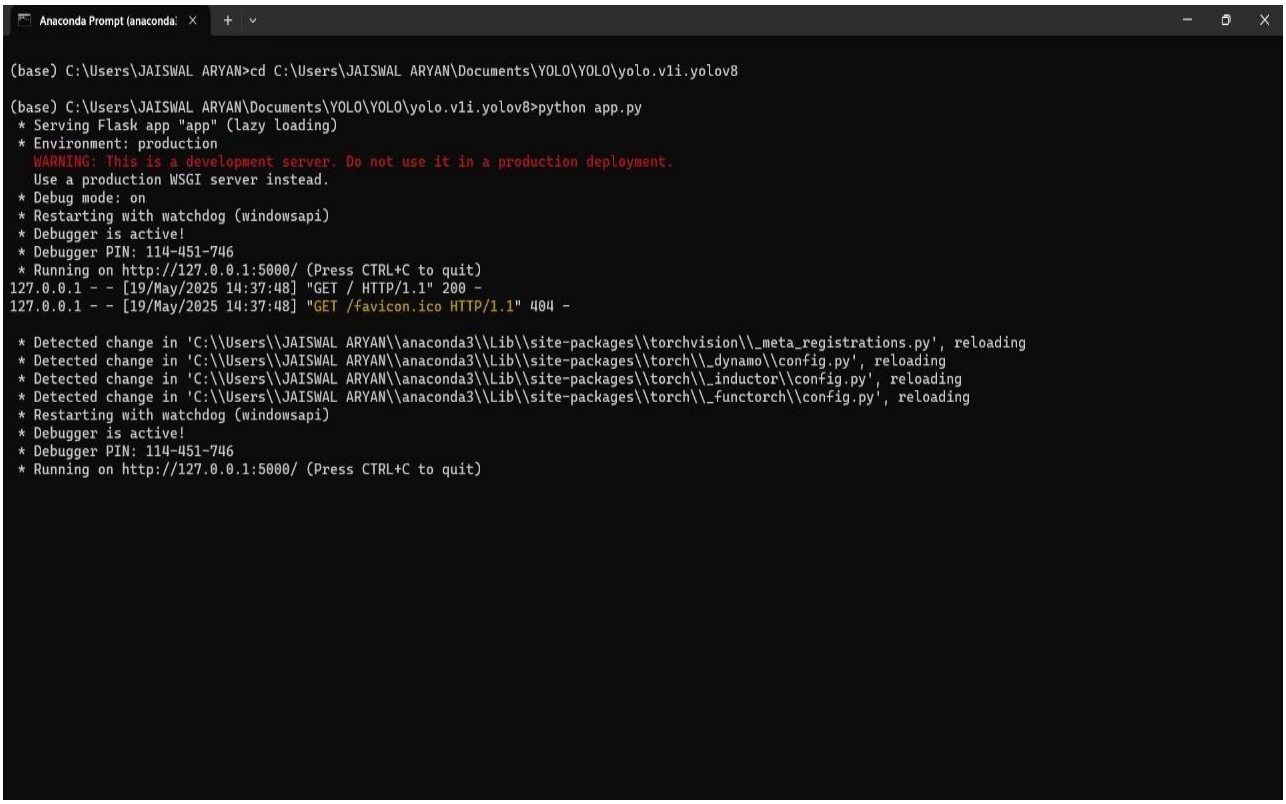


Fig 7.1 Anaconda Prompt

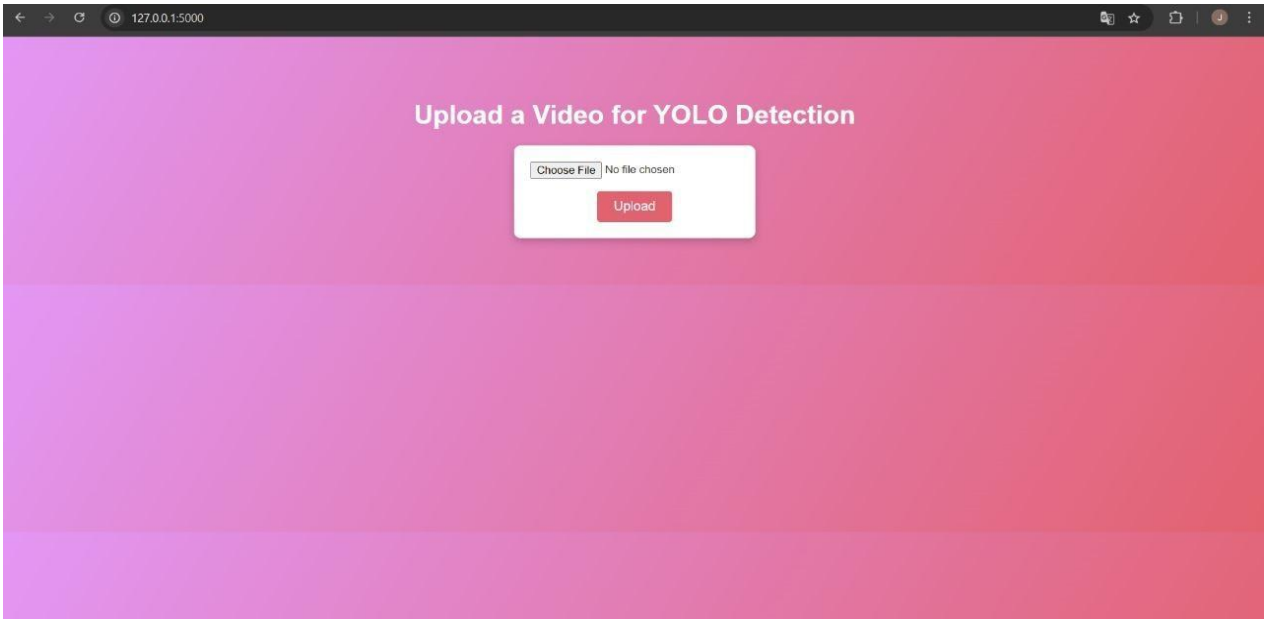


Fig 7.2: Home page

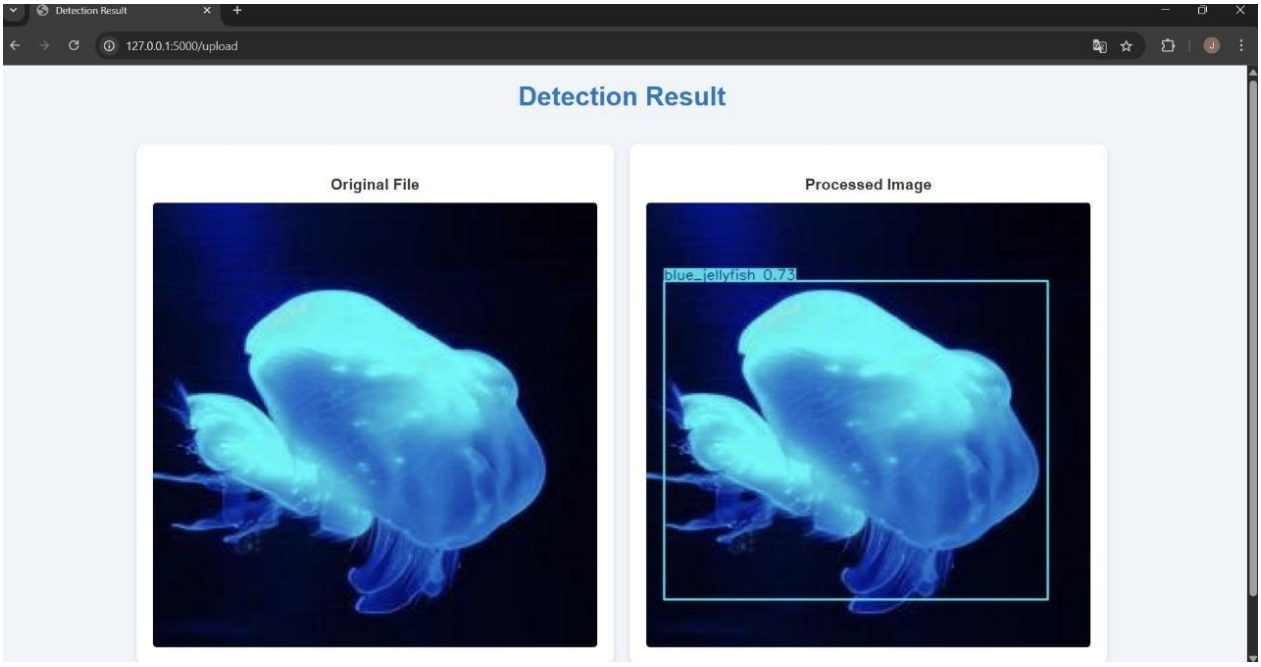


Fig 7.3: Result page

CHAPTER 8
SOFTWARE TESTING

8.1 GENERAL

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

8.2 DEVELOPING METHODOLOGIES

The test process is initiated by developing a comprehensive plan to test the general functionality and special features on a variety of platform combinations. Strict quality control procedures are used. The process verifies that the application meets the requirements specified in the system requirements document and is bug free. The following are the considerations used to develop the framework from developing the testing methodologies.

8.3 TYPES OF TESTS

8.3.1 UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program input produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

8.3.2 FUNCTIONAL TEST

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals. Functional testing is centered on the following items:

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.

8.3.3 SYSTEM TEST

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

8.3.4 PERFORMANCE TEST

The Performance test ensures that the output be produced within the time limits, and the time taken by the system for compiling, giving response to the users and request being send to the system for to retrieve the results.

8.3.5 INTEGRATION TESTING

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

8.3.6 ACCEPTANCE TESTING

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Acceptance testing for Data Synchronization

- The Acknowledgements will be received by the Sender Node after the Packets are received by the Destination Node
- The Route add operation is done only when there is a Route request in need
- The Status of Nodes information is done automatically in the Cache Updation process

8.2.7 BUILD THE TEST PLAN

Any project can be divided into units that can be further performed for detailed processing. Then a testing strategy for each of this unit is carried out. Unit testing helps to identity the possible bugs in the individual component, so the component that has bugs can be identified and can be rectified from errors.

CHAPTER 9

FUTURE ENHANCEMENT

9.1 FUTURE ENHANCEMENTS

Future enhancements for jellyfish detection using YOLOv11 could focus on several key areas to further improve accuracy, robustness, and deployment efficiency. One potential enhancement would be the integration of multi-modal data sources, such as combining optical imagery with sonar or thermal imaging, to improve jellyfish detection in diverse environments, especially in low-visibility or deep-sea scenarios. Additionally, domain adaptation techniques could be explored to fine-tune the model for specific marine environments, enabling better generalization across different oceanic regions and conditions. Another promising direction is the incorporation of real-time adaptive learning capabilities, where the model can continuously learn from new data collected during deployment, improving its performance without requiring manual retraining. Enhancing multi-object tracking would also be valuable, allowing YOLOv11 to not only detect jellyfish but also track their movement over time, which would aid in studying their behavior and migration patterns. Finally, reducing model complexity while maintaining high accuracy could be an important future enhancement to make YOLOv11 more suitable for deployment on edge devices, such as underwater drones or autonomous marine vehicles, ensuring scalable and efficient real-time detection in the field. These advancements would further solidify YOLOv11's position as a powerful tool for marine research and conservation efforts. Further advancements of YOLO versions can lead to higher accuracy and precise detections of an object.

CHAPTER 10

CONCLUSION AND REFERENCES

10.1 CONCLUSION

In conclusion, this study represents a significant advancement in YOLO versions and while detecting the objects in any scenarios with precise confidence scores. YOLOv11 represents a significant leap forward in the detection of jellyfish, particularly in the challenging and dynamic underwater environment. By integrating advanced techniques like Enhanced Attention Mechanisms, CoordCov Modules, and Global Attention Mechanisms, YOLOv11 enhances both the accuracy and efficiency of detecting jellyfish, especially in cluttered and low-visibility scenarios. Its optimized computational pipeline ensures real-time performance, making it a viable tool for practical applications in marine research and conservation. While YOLOv11 shows considerable promise, future advancements in multi-modal data integration, adaptive learning, and multi-object tracking could further enhance its capabilities. As the field of underwater object detection continues to evolve, YOLOv11 stands poised to play a pivotal role in the study and monitoring of jellyfish populations, contributing to more effective marine ecosystem management and conservation strategies.

10.2 REFERENCES

[1] L. Yan, S. Li, and F. Ding, ‘‘The preliminary studies on the dynamics of macro-jellyfish resources and their relationship with fisheries in the East China Sea and yellow sea,’’ Mar. Fisheries, vol. 1, pp. 10–14, Aug. 2004.

[2] J. Dong et al., ‘‘The morphology and structure of jellyfish (Cyanea nozakii kishinouye),’’ Fish Sci., vol. 24, no. 2, pp. 22–23, 2005.

[3] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, ‘‘Object detection in 20 years: A survey,’’ Proc. IEEE, vol. 111, no. 3, pp. 257–276, Mar. 2023.

[4] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, ‘‘LLaMA: Open and efficient foundation language models,’’ 2023, arXiv:2302.13971.

[5] H. Touvron et al., ‘‘Llama 2: Open foundation and fine-tuned chat models,’’ 2023, arXiv:2307.09288.

[6] I. de Zarzà, J. de Curtò, G. Roig, and C. T. Calafate, ‘‘LLM multimodal traffic accident forecasting,’’ Sensors, vol. 23, no. 22, p. 9225, Nov. 2023, doi: 10.3390/s23229225.

[7] H. Liu, C. Li, Q. Wu, and Y. J. Lee, ‘‘Visual instruction tuning,’’ 2023, arXiv:2304.08485.

[8] 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.

[9] R. Girshick, ‘‘Fast R-CNN,’’ in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Santiago, Chile, Dec. 2015, pp. 1440–1448.

[10] S. Ren, K. He, R. Girshick, and J. Sun, ‘‘Faster R-CNN: Towards realtime object detection with region proposal networks,’’ IEEE Trans. Pattern Anal. Mach. Intell., vol. 39, no. 6, pp. 1137–1149, Jun. 2017.

[11] 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.

[12] W. Liu et al., ‘‘SSD: Single shot multibox detector,’’ in Proc. 14th Eur. Conf. Amsterdam, The Netherlands: Springer, Oct. 2016, pp. 21–37.

[13] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, ‘‘Focal loss for dense object detection,’’ in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Oct. 2017, pp. 2999–3007.

[14] B. Alexey, C.-Y. Wang, and H.-Y. Liao, ‘‘YOLOV4: Optimal speed and accuracy of object detection,’’ in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Apr. 2020, pp. 1–17.

[15] G. Jocher. (2020). YOLOV5. [Online]. Available: https://github.com/ultralytics/yolov5

[16] C. Li, L. Li, H. Jiang, K. Weng, Y. Geng, L. Li, Z. Ke, Q. Li, M. Cheng, W. Nie, Y. Li, B. Zhang, Y. Liang, L. Zhou, X. Xu, X. Chu, X. Wei, and X. Wei, ‘‘YOLOV6: A single-stage object detection framework for industrial applications,’’ 2022, arXiv:2209.02976.

[17] (2023). YOLOV8. [Online]. Available: https://github.com/ultralytics/ultralytics

[18] J. Redmon and A. Farhadi, ‘‘YOLOV3: An incremental improvement,’’ 2018, arXiv:1804.02767.

[19] M. Martin-Abadal. (2019). Jellyfish Object Detection. Accessed: Mar. 17, 2020. [Online]. Available: https://github.com/srv/ jf_object_detection

[20] T.-N. Pham, V.-H. Nguyen, and J.-H. Huh, ‘‘Integration of improved YOLOV5 for face mask detector and auto-labeling to generate dataset for fighting against COVID-19,’’ J. Supercomput., vol. 79, no. 8, pp. 8966–8992, May 2023.

[21] H. Kim, D. Kim, S. Jung, J. Koo, J.-U. Shin, and H. Myung, ‘‘Development of a UAV-type jellyfish monitoring system using deep learning,’’ in Proc. 12th Int. Conf. Ubiquitous Robots Ambient Intell. (URAI), Oct. 2015, pp. 495–497.

[22] J. Koo, S. Jung, and H. Myung, ‘‘A jellyfish distribution management system using an unmanned aerial vehicle and unmanned surface vehicles,’’ IEEE Underwater Technol. (UT), vol. 2017, no. 1, pp. 1–5, Jun. 2017.

[23] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, ‘‘Gradient-based learning applied to document recognition,’’ Proc. IEEE, vol. 86, no. 11, pp. 2278–2324, Nov. 1998. VOLUME 12, 2024 87847 T.-N. Pham et al.: Improved YOLOv5 Based Deep Learning System for Jellyfish Detection

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, ‘‘ImageNet classification with deep convolutional neural networks,’’ in Proc. Adv. Neural Inf. Process. Syst. (NIPS), 2012, pp. 1097–1105.

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, ‘‘Going deeper with convolutions,’’ inProc. IEEE Conf. Comput. V is. Pattern Recognit. (CVPR), Boston, MA,
USA, Jun. 2015, pp. 1–9.