CHAPTER 1

INTRODUCTION

Cyclone is a kind of disastrous weather system with tremendous destructive power. The early warning of the cyclone is a focus of meteorological operations and also wide concern. The location of the cyclone is crucial information for intensity estimation.

The trajectory of cyclones is an important area of study in meteorology, as it can have a significant impact on the safety and well-being of people living in affected areas. Advances in satellite technology and machine learning have made it possible to track and predict the trajectory of cyclones with increasing accuracy and efficiency.

Machine learning is a subfield of artificial intelligence that involves developing algorithms and models that can learn from data and make predictions or decisions based on that learning. In the context of cyclone trajectory prediction, machine learning techniques can be used to analyse satellite data and identify patterns and trends that can be used to make accurate predictions about the path of the storm.

One popular machine learning technique used in cyclone trajectory prediction is deep learning. Deep learning involves using artificial neural networks to learn from large amounts of data and make predictions based on that learning.

Other machine learning techniques that can be used for cyclone trajectory prediction. These techniques can be used to analyse satellite data and identify features that are predictive of the trajectory of a cyclone.

Cyclone in the satellite video is identified first, and then a more accurate center positioning method is performed. In recent years, with the rapid development of deep learning, feature extraction has been delivered to a deep convolutional neural network (CNN) with powerful feature learning capabilities instead of designing handcrafted features. Some scholars generated the bounding box based on the given “center,” and then performed a CNN-based binary classification to categorize cloud data in this bounding box. However, they mainly addressed the classification of cyclone, but not their localization. Since the Cyclone region depends on the given “center,” in practical application, the problem of how to determine the suitable candidate

"center" and how to deal with all "centers" on the same Cyclone should be solved. Therefore, an algorithm that directly detects Cyclone from the data or image instead of using the "center" is necessary.

PROBLEM STATEMT:

To get the trajectory of the Cyclone and to predict the point of contact of the Cyclone on the land surface in Satellite Video using Advanced Machine Learning Techniques.

OBJECTIVE:

- Identifying Cyclone in the Satellite Video.
- Getting the epicentre (eye) of the cyclone.
- Getting the Trajectory of the cyclone.

MAJOR CONTRIBUTIONS:

Deep learning is introduced into Cyclone detection, center location framework based on convolutional network and algorithms is proposed, which can handle different multiple cyclones.

An OS detection framework with the multiscale feature detection module is applied. Multiscale fusion features on two different scales are constructed by the deep convolutional network to process multiple cyclones.

CNN can extract cyclone features with powerful feature learning capabilities instead of designing hand crafted features. Algorithms can determine the center position based on the Cyclone morphology. The accurate Cyclone center is located by threshold segmentation, edge detection, and circle detection, which improves the accuracy of specific types of Cyclones.

CHAPTER 2

LITERATURE REVIEW

2.1 Machine Learning (ML) algorithm:

There are plenty of machine learning algorithms. The choice of the algorithm is based on the objective. Machine learning algorithms are computational methods that enable machines to learn from data without being explicitly programmed. These algorithms are designed to identify patterns in data and make predictions or decisions based on those patterns.

2.2 Challenges and Limitations:

The primary challenge of machine learning is the lack of data or the diversity in the dataset. A machine cannot learn if there is no data available. Besides, a dataset with a lack of diversity gives the machine a hard time. A machine needs to have heterogeneity to learn meaningful insight. It is rare that an algorithm can extract information when there are no or few variations. It is recommended to have at least 20 observations per group to help the machine learn. This constraint leads to poor evaluation and prediction.

2.3 Applications of Machine Learning:

There are several applications in machine learning some of the applications of the ML are augmentation, automation, finance industry, government industry, organization, marketing etc. Machine learning, which assists humans with their day-to-day tasks, personally or commercially without having complete control of the output. Such machine learning is used in different ways such as Virtual Assistant, Data analysis, software solutions. The primary user is to reduce errors due to human bias.

2.4 Example of application of Machine Learning in Supply Chain

Machine learning gives terrific results for visual pattern recognition, opening up many potential applications in physical inspection and maintenance across the entire supply chain network. Unsupervised learning can quickly search for comparable patterns in the diverse dataset. In turn, the machine can perform quality inspection throughout the logistics hub, shipment with damage and wear.

2.5 Importance of Machine Learning

Machine learning is the best tool so far to understand and identify a pattern in the data. One of the main ideas behind machine learning is that the computer can be trained to automate tasks that would be exhaustive or impossible for a human being. The clear breach from the traditional analysis is that machine learning can take decisions with minimal human intervention.

Take the following example for this ML tutorial; a retail agent can estimate the price of a house based on his own experience and his knowledge of the market. A machine can be trained to translate the knowledge of an expert into features. The features are all the characteristics of a house, neighbourhood, economic environment, etc. that make the price difference. For the expert, it took him probably some years to master the art of estimate the price of a house. His expertise is getting better and better after each sale.

For the machine, it takes millions of data, (i.e., example) to master this art. At the very beginning of its learning, the machine makes a mistake, somehow like the junior salesman. Once the machine sees all the example, it got enough knowledge to make its estimation. At the same time, with incredible accuracy. The machine is also able to adjust its mistake accordingly.

2.6 Overview:

Machine learning involves computers discovering how they can perform tasks without being explicitly programmed to do so. It involves computers learning from data provided so that they carry out certain tasks. For simple tasks assigned to computers, it is possible to program algorithms telling the machine how to execute all steps required to solve the problem at hand; on the computer's part, no learning is needed. For more advanced tasks, it can be challenging for a human to manually create the needed algorithms. In practice, it can turn out to be more effective to help the machine develop its own algorithm, rather than having human programmers specify every needed step.

The discipline of machine learning employs various approaches to teach computers to accomplish tasks where no fully satisfactory algorithm is available. In cases where vast numbers of potential answers exist, one approach is to label some of the correct answers as valid. This can then be used as training data for the computer to improve the algorithm(s) it uses to

determine correct answers. For example, to train a system for the task of digital character recognition, the MNIST dataset of handwritten digits has often been used.

2.7 Machine learning approaches:

Machine learning approaches are traditionally divided into three broad categories, depending on the nature of the "signal" or "feedback" available to the learning system:

2.7.1 Supervised Learning:

This type of algorithm learns from labelled data and uses this information to make predictions on new, unlabelled data. Examples of supervised learning algorithms include linear regression, logistic regression, decision trees, and neural networks.

2.7.2 Unsupervised Learning:

This type of algorithm learns from unlabelled data and tries to identify patterns or groupings within the data. Examples of unsupervised learning algorithms include k-means clustering, principal component analysis (PCA), and self-organizing maps (SOM).

2.7.3 Reinforcement Learning:

This type of algorithm learns by trial and error and is often used in games and simulations. The algorithm receives feedback in the form of rewards or punishments and adjusts its behaviour accordingly.

2.7.4 Deep Learning: This type of algorithm uses neural networks with multiple layers to learn hierarchical representations of data. Deep learning algorithms have achieved state-of-the-art performance in a variety of tasks, including image and speech recognition.

2.7.5 Semi-supervised learning:

Semi-supervised learning falls between unsupervised learning (without any label training data) and supervised learning (with completely label training data). Some of the training examples are missing training labels, yet many machine-learning researchers have found that data, when used in conjunction with a small amount of label data, can produce a considerable improvement in learning accuracy. In weakly supervised learning, the training labels are noisy, limited, or imprecise.

CHAPTER 3

DATA SOURCE

The data source is collected from MOSDAC (Meteorological and Oceanographic Satellite Data Archival Center) is a national data center under the Indian Space Research Organization (ISRO) that is responsible for the archival, processing, and dissemination of meteorological and oceanographic satellite data. It was established in 2012.MOSDAC primarily handles data from Indian remote sensing satellites, meteorological satellites, and oceanographic satellites.

The center receives real-time data from these satellites and processes it to create value-added products such as cloud motion vectors, sea surface temperature, and ocean colour. MOSDAC provides these products and data to various organizations and users, including the Indian Meteorological Department (IMD), National Remote Sensing Centre (NRSC), and academic institutions. The center also maintains a web portal that allows users to access and download satellite data and products.

MOSDAC is responsible for archiving and processing data from various Indian and international satellite missions, including the Indian National Satellite (INSAT) system, the Indian Remote Sensing (IRS) system, and international missions such as the NOAA, NASA, and European Space Agency (ESA) satellites.

The center is also involved in developing algorithms and tools for data processing, quality control, and analysis.

The data products generated by MOSDAC are used by various organizations in India and around the world for weather forecasting, disaster management, agriculture, water resource management, and climate studies. MOSDAC also provides training and support to users of its data products. Some of the key services provided by MOSDAC include: Data archiving and retrieval: MOSDAC archives data from various satellite missions and makes it available to users through its web portal.

Data processing and analysis: MOSDAC processes satellite data and generates various data products such as cloud cover, sea surface temperature, ocean color, and vegetation indices.

Quality control: MOSDAC performs quality control checks on the data products to ensure that they are accurate and reliable.

Capacity building: MOSDAC provides training and support to users of its data products, including workshops, seminars, and online training programs.

Overall, MOSDAC plays a crucial role in supporting weather forecasting, disaster management, and other applications that require meteorological and oceanographic satellite data.

MOSDAC providing satellite data and products that are used for weather forecasting, climate research, and other applications related to meteorology and oceanography in India.

Object detection in cyclones using satellite video involves using computer vision algorithms to automatically detect and track objects in satellite video footage of cyclones.

This can help meteorologists and disaster management authorities to monitor the movement and intensity of cyclones, as well as identify potential hazards such as storm surges, flooding, and wind damage.

DATA DESCRIPTION:

S.No	Name of the cyclone	Duration	Region
1.	Asani	May 7-11,2022	Bay of Bengal
2.	Jawad	Dec 3-6,2021	Bay of Bengal
3.	Gulab	Sep 24-Oct 4,2021	Arabian sea
4.	Yaas	May 24-27,2021	Bay of Bengal
5.	Tauktae	May 14-19,2021	Arabian sea
6.	Sitrang	Oct 25-29,2022	Bay of Bengal
7.	Mandous	Dec 7-10,2022	Bay of Bengal

Table 3: Data collection

CHAPTER 4

OBJECT DETECTION

Pre-processing the video footage enhance features such as edges and corners. Applying a feature extraction algorithm to identify key features of the objects in the video frames. Training a machine learning algorithm, such as a neural network, to recognize and classify the objects based on the extracted features. Applying a tracking algorithm to follow the movement of the objects over time. Some of the key challenges in object detection in cyclones include dealing with low-contrast and low-light conditions, detecting objects of different sizes and shapes, and distinguishing between different types of objects such as buildings, trees, and vehicles. To address these challenges, researchers have developed various techniques such as multi-scale object detection, adaptive thresholding, and fusion of multiple data sources. Additionally, some research groups are exploring the use of unmanned aerial vehicles (UAVs) equipped with cameras to collect high-resolution imagery of cyclones and improve object detection accuracy.

Object detection is a computer vision technique that involves identifying and localizing objects in an image or video and classifying them into predefined categories. It is a crucial component of many modern computer vision applications, such as autonomous vehicles, surveillance systems, robotics, and augmented reality.

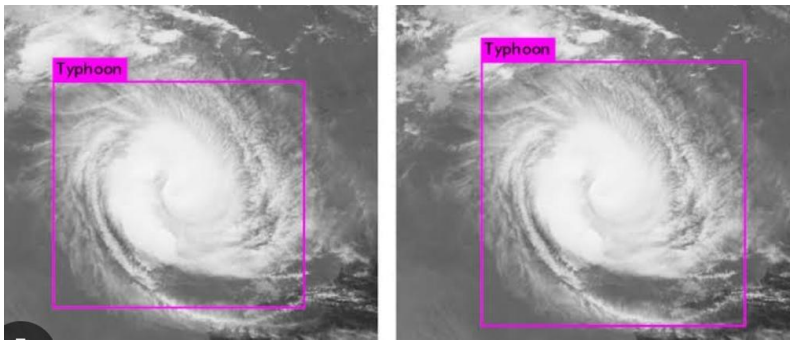

Fig 4: Object detection

The object detection process typically involves the following steps:

Image pre-processing: The input image is first pre-processed to enhance its quality and remove any noise that could interfere with object detection.

Object localization: The image is then analysed to identify regions that contain potential objects of interest, usually through the use of edge detection, segmentation, or feature extraction techniques.

Object classification: Once regions of interest are identified, the objects in those regions are classified into predefined categories using machine learning algorithms such as deep neural networks.

Post-processing: The classification results are then post-processed to refine the object boundaries and eliminate false positives.

There are several popular object detection algorithms, such as Faster R-CNN, YOLO, and SSD. Each of these algorithms has its own strengths and weaknesses and can be used in different applications depending on the specific requirements of the task.

Object detection is a challenging problem in computer vision, and researchers are continuously working to improve the accuracy and speed of object detection algorithms to enable more advanced applications.

Overall, object detection in cyclones using satellite video has the potential to improve our understanding of these natural disasters and enable more effective disaster management strategies.

CHAPTER 5

IMAGE PROCESSING

5.1 IMAGE CLASSIFICATION IN OBJECT DETECTION:

Image classification is a computer vision task that involves assigning a single label or category to an entire image. On the other hand, object detection is a task that involves detecting and localizing multiple objects within an image and classifying them into different categories.

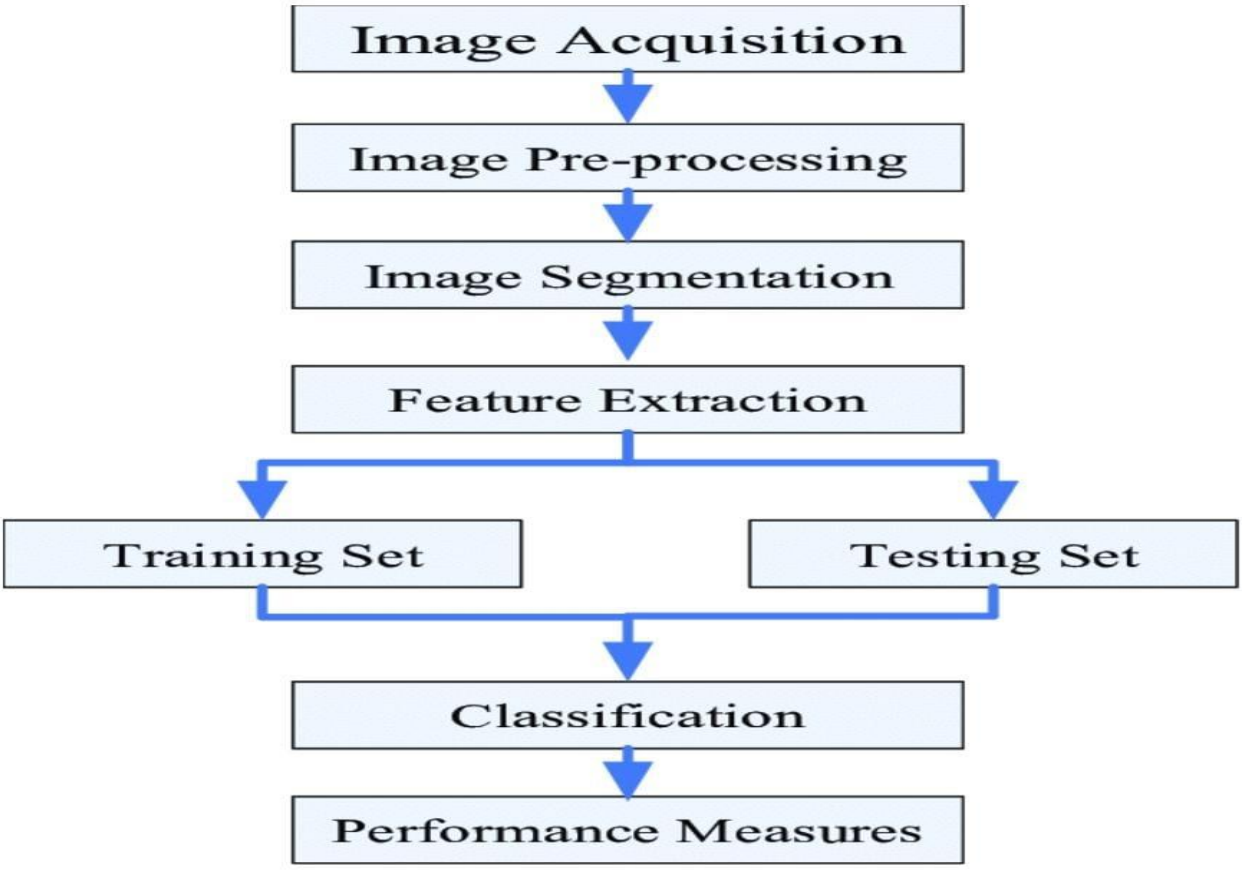

Fig 5.1: Block diagram for image classification

In object detection, image classification is typically performed as a sub-task, where each detected object is classified into a specific category. Object detection models often use a combination of object localization and object classification techniques to achieve this.

There are different approaches to perform image classification in object detection, including:

Region-based classification: In this approach, the object detection algorithm first generates object proposals or regions of interest (ROIs) in the image, and then each proposal is classified into a specific category. This approach is used in popular object detection models like Faster R-CNN, Mask R-CNN, and Retina Net.

Single-stage classification: In this approach, the object detection algorithm directly predicts the bounding box coordinates and the class label of each object in a single step. This approach is used in models like YOLO (You Only Look Once) and SSD (Single Shot Detector).

In both approaches, image classification is an integral part of the object detection task, and the accuracy of the object detection model depends on the accuracy of the image classification component.

CHAPTER 6
OBJECT DETECTION

6.1 OBJECT DETECTION USING ROBO FLOW:

Roboflow is an online platform that provides tools for building computer vision models, including object detection models. It offers a user-friendly interface that makes it easy to prepare and annotate data, train models, and deploy them to various applications. Here's how you could use Roboflow for object detection in cyclones using satellite video:

Data preparation: First, you would need to gather the satellite video footage of the cyclone that you want to analyses. You could then upload the video to Roboflow and use its annotation tools to label the objects in the video frames that you want to detect, such as clouds, land, and water bodies. You can also use Roboflow's data augmentation tools to generate more training data by flipping, rotating, or adjusting the brightness and contrast of the video frames.

Model training: Once you have annotated and prepared your data, you can use Roboflow to train an object detection model using popular deep learning frameworks such as TensorFlow or YOLOv5.

Roboflow provides pre-trained models that can be fine-tuned on your annotated data, or you can train your own custom model from scratch.

Model evaluation: After training your model, you can evaluate its performance on a validation set to determine its accuracy and to fine-tune its hyperparameters.

Roboflow provides metrics such as precision, recall, and F1 score to help you evaluate your model's performance.

Deployment: Once you're satisfied with your model's performance, you can deploy it to various applications, such as mobile or web-based applications, to perform object detection on new satellite video footage of cyclones in real-time.

Fig 6.1: Roboflow Process

Roboflow provides an end-to-end solution for object detection in cyclones using satellite video, from data preparation to model training and deployment. Its user-friendly interface and pre-built tools make it easy to get started with building computer vision models for a wide range of applications.

6.2 OBJECT DETECTION USING YOLOV5 MODEL:

YOLOv5 is a state-of-the-art object detection algorithm and a variant of the YOLO (You Only Look Once) family of object detection models. It was developed by Ultralytics and released in June 2020. YOLOv5 is a deep learning-based algorithm that uses convolutional neural networks (CNNs) to detect objects in images or video frames.

The YOLOv5 algorithm is based on a single-stage architecture, which means that it directly predicts bounding boxes and class probabilities for all objects in an image in a single forward pass. This makes it faster and more efficient than traditional two-stage object detection algorithms such as Faster R-CNN or SSD.

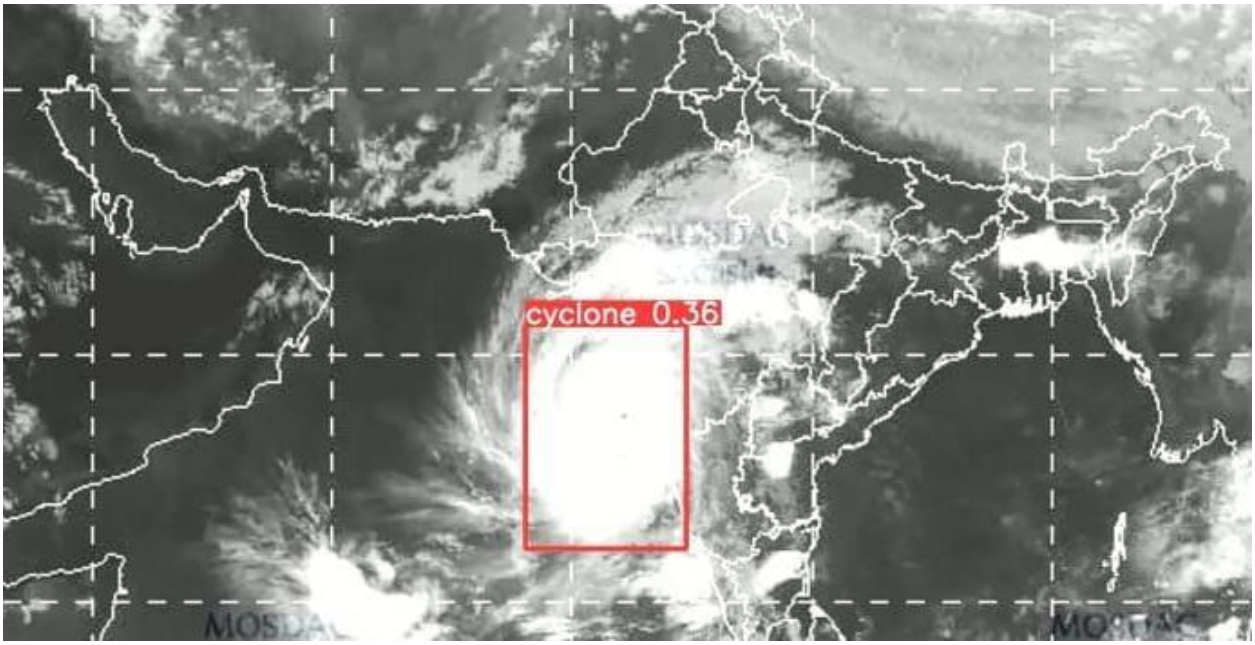

Fig 6.2: Object detection using yolov5

The YOLOv5 architecture consists of a backbone network that extracts features from the input image and a detection head that predicts the bounding boxes and class probabilities for the objects in the image. YOLOv5 uses a novel CSP Darknet architecture for the backbone network, which improves its accuracy and speed compared to previous versions of YOLO.

YOLOv5 is available in several variants, including YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x, each with a different number of layers and computational complexity. The different variants allow users to choose the optimal balance between accuracy and speed depending on their specific application requirements.

Overall, YOLOv5 is a highly accurate and efficient object detection algorithm that has been widely adopted in various computer vision applications such as self-driving cars, robotics, surveillance systems, and medical imaging.

CHAPTER 7

EPICENTER

The "eye" or center of a cyclone can be identified in satellite images by looking for a circular or roughly circular area of calm weather surrounded by a band of intense convection or thunderstorms. The eye typically has a clear appearance with low clouds and is surrounded by the eyewall, which is characterized by high winds and heavy rain.

The "eye" of a cyclone or hurricane is the center of the storm and is typically characterized by calm winds and clear skies. It is surrounded by a circular band of thunderstorms known as the eyewall, where the strongest winds and heaviest rainfalls occur.

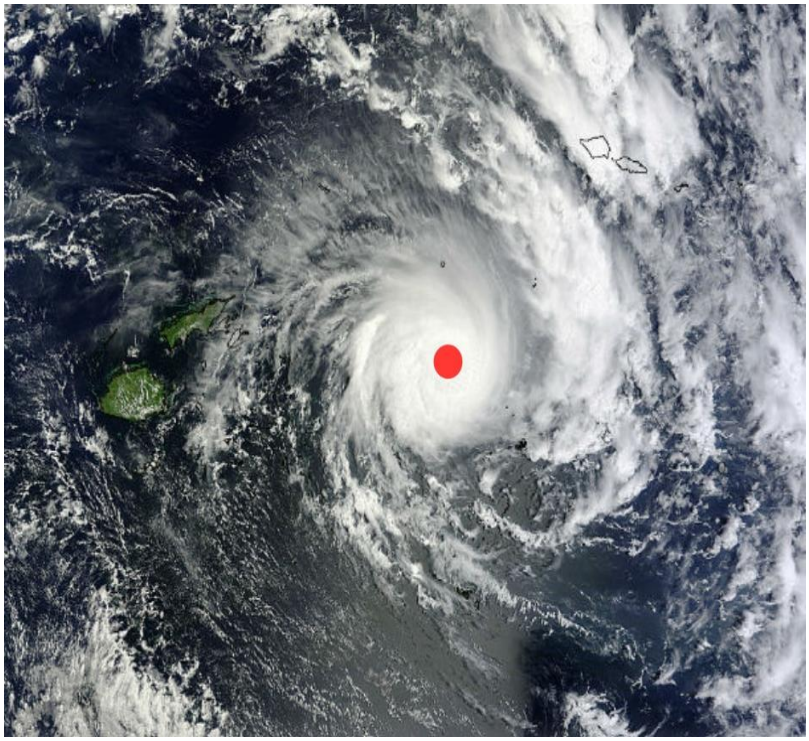

Fig 7:Center of the cyclone(epicenter)

Determining the exact location of the cyclone's eye or center is important for forecasting the storm's track and intensity. There are several ways to determine the position of the cyclone's center, including:

Satellite imagery: Satellites provide a valuable tool for monitoring cyclones and hurricanes from space. They can capture high-resolution images of the storm, allowing forecasters to track its movement and determine the location of its center.

Radar: Radar systems can be used to measure the speed and direction of the winds within the storm, allowing forecasters to estimate the position of the cyclone's center.

Aircraft reconnaissance: Reconnaissance aircraft can fly into the storm to gather data on its structure, winds, and other characteristics. This information can be used to determine the position of the cyclone's center with greater accuracy.

Surface observations: Observations from ground stations, buoys, and ships can provide valuable information on the storm's location, wind speed, and pressure.

To identify the center of a cyclone in satellite imagery, meteorologists use a variety of techniques, including:

Visual inspection: Experts carefully examine the satellite images to identify the location of the center of the storm. They look for the telltale signs of the eye, such as a clear center surrounded by intense thunderstorms.

Computer algorithms: Automated algorithms can be used to analyse satellite imagery and identify the center of the storm. These algorithms may use features such as cloud temperature, wind speed, and other meteorological parameters to make their determinations.

Combination of visual inspection and computer algorithms: Often, a combination of visual inspection and automated algorithms is used to identify the center of the storm. Experts will examine the images and then use the output of the algorithms to refine their determinations.

Overall, determining the position of the cyclone's center is a critical task for forecasting and monitoring these dangerous storms, and requires the use of a variety of tools and techniques.

7.1 IMAGE SEGMENTATION:

Image segmentation is a computer vision technique that involves dividing an image into multiple regions or segments, each of which corresponds to a specific object or part of the image.

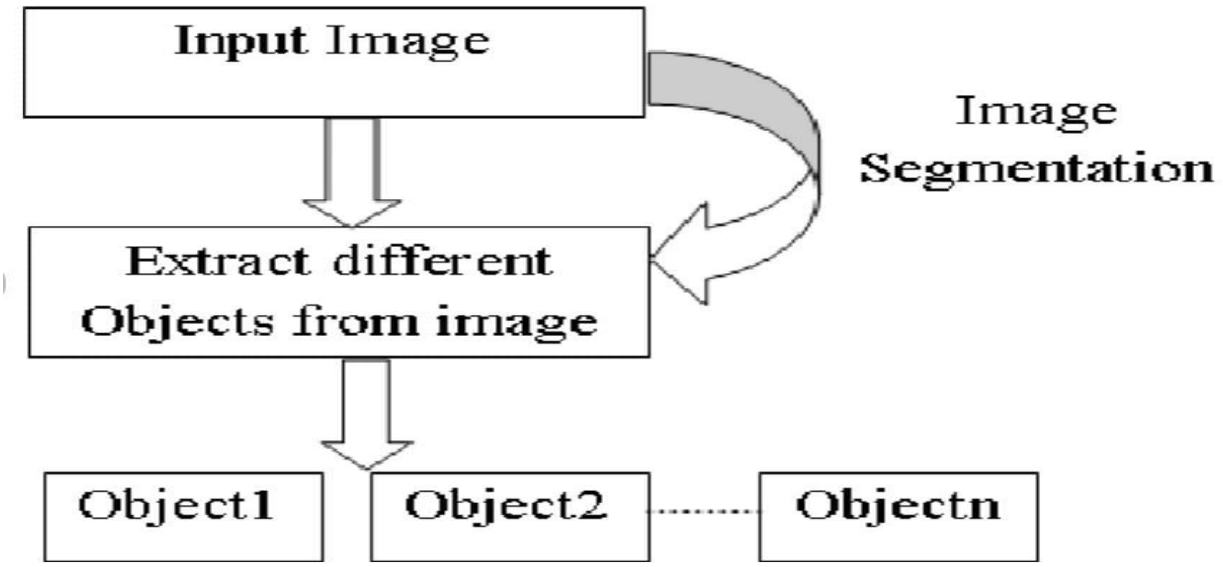

Fig 7.1: Block of Image Segmentation

Image segmentation is the process of dividing an image into multiple segments or regions based on certain criteria such as color, texture, or intensity. This technique is commonly used in image processing and computer vision applications, such as object recognition, medical imaging, and autonomous vehicles

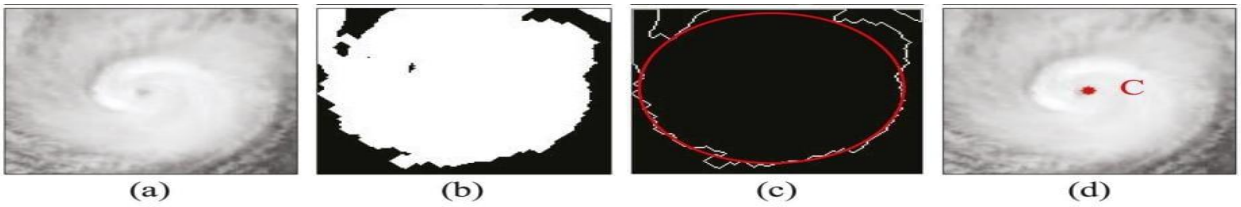

Fig 7.1: Image segmentation

There are several methods of image segmentation, including:

a. Object Detection
b. Binary Segmentation

c. Edge and circle detection
d. Eye detection based on concentric circles

Thresholding: This method involves setting a threshold value for a certain parameter (such as color or intensity), and then separating the pixels into two groups based on whether they are above or below the threshold.
Region growing: This technique starts with a seed pixel and grows the region around it by adding adjacent pixels that meet certain criteria.
Edge detection: This method identifies the edges of objects in an image, which can then be used to separate them into distinct regions.
Clustering: This approach groups pixels that have similar characteristics (such as color or texture) into clusters, which can then be separated into segments.
Watershed transformation: This technique is based on the idea of simulating water flow over an image, with the valleys representing the boundaries between regions.

Overall, image segmentation is a powerful tool for analyzing and processing digital images, allowing for the identification and isolation of specific regions or objects within the image.

7.1.1 OBJECT DETECTION:

Object detection is a computer vision technique that involves identifying and locating objects within an image or video stream. This technique is used in a wide range of applications, such as autonomous vehicles, surveillance, and robotics.Object detection involves two main tasks: object localization and object classification. Object localization involves identifying the location of objects within an image or video stream, typically through the use of bounding boxes or masks.

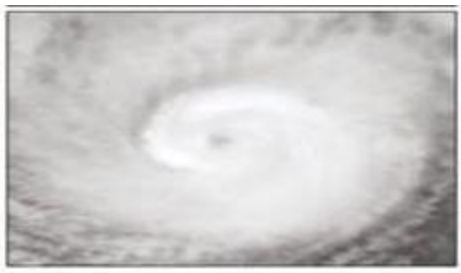

Fig 7.1.1: Object Detection

There are several techniques used for object detection, including:

Haar cascades: This is a machine learning-based approach that uses a set of features to identify objects within an image. Haar cascades are commonly used for face detection.

Feature-based detection: This method involves identifying specific features within an image, such as edges or corners, and using those features to identify objects.

Template matching: This technique involves comparing an image to a pre-defined template in order to identify objects.

Deep learning-based approaches: This is the most commonly used approach for object detection, which involves training a neural network to identify and classify objects within an image. Popular deep learning-based object detection frameworks include YOLO, Faster R-CNN, and SSD.

7.1.2 BINARY SEGMENTATION:

Binary segmentation is a type of image segmentation that divides an image into two regions or classes based on a certain threshold value. The two regions are typically a foreground region and a background region. In binary segmentation, each pixel in the image is classified as either foreground or background based on whether its value is above or below a certain threshold.

The goal of binary segmentation is to separate an image into two parts in order to extract meaningful information from it. Binary segmentation is a widely used technique in image processing and computer vision applications, such as object detection, medical image analysis, and document processing.

Fig 7.1.2: Binary Segmentation

Binary segmentation can be performed using various techniques, including global thresholding, adaptive thresholding, and Otsu's method. These techniques involve selecting a threshold value for the entire image or for local regions based on pixel statistics, histogram analysis, or other image features.

Binary segmentation can be performed using several techniques, including:

Global thresholding: This involves selecting a threshold value for the entire image based on the histogram of pixel values. All pixels above the threshold are classified as foreground, while all pixels below the threshold are classified as background.

Adaptive thresholding: This involves selecting a threshold value for each local region of the image based on the local pixel statistics. This approach is particularly useful when the illumination varies across the image.

Otsu's method: This is a commonly used thresholding technique that automatically calculates the optimal threshold value based on the image histogram.

7.1.3 EDGE AND CIRCLE DETECTION:

Edge detection is a computer vision technique that involves identifying and highlighting the edges in an image. Edges in an image are defined as regions where the intensity of the image changes sharply or discontinuously. Edge detection is an important step in many image processing and computer vision applications, such as object detection, image segmentation, and feature extraction. Edge detection can be performed using various techniques, including gradient-based methods, Laplacian-based methods, and Canny edge detection. These techniques involve analyzing the image to detect changes in intensity or color and highlighting the edges accordingly.

Fig 7.1.3: Edge and Circle Detection

Circle detection is a specific type of object detection that involves identifying circles within an image. Circle detection is often used in computer vision applications, such as robotics, where identifying circular objects like wheels can be important.

Circle detection can be performed using various techniques, including the Hough transform and the Randomized Hough transform. These techniques involve analyzing the image to identify groups of pixels that form circular patterns, and then fitting circles to these patterns.

Both edge detection and circle detection are important techniques in image processing and computer vision that allow for the identification and analysis of specific features within an image.

7.1.4 EYE DETECTION ON CONCENTRIC CIRCLE:

Eye detection based on concentric circles is a computer vision technique that involves identifying the location and shape of eyes within an image by analyzing concentric circles that surround the pupils. This technique is commonly used in facial recognition and biometric applications, as the eyes are an important feature for identifying individuals.

The concentric circle approach to eye detection involves analyzing the image to identify areas of high contrast around the pupils, which typically appear as dark regions surrounded by lighter areas. The technique then applies concentric circles around the pupils and uses the resulting circles to locate the position of the eyes.

The concentric circle approach can be combined with other computer vision techniques, such as Haar cascades and Viola-Jones object detection, to improve the accuracy and efficiency of eye detection. It can also be used in conjunction with other facial recognition techniques to identify individuals based on their unique eye features.

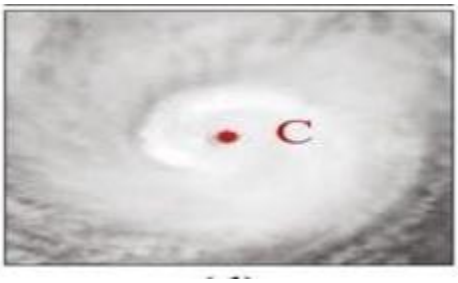

Fig7.1.4: Eye detection and concentric circles

Eye detection based on concentric circles is a powerful technique in computer vision and facial recognition that allows for the accurate and efficient identification of eyes within an image.

7.2 CONCENTRIC CIRCLES:

Concentric circles are a set of circles that share the same center point. Each circle in the set has a different radius, but they all share the same center point.

Concentric circles are commonly used in geometry and math to study properties of circles and to perform calculations involving circles. In computer vision, concentric circles are often used in

object detection and recognition applications, such as detecting the location of eyes or circular objects within an image.

Concentric circles can also be used to create visual patterns and designs in graphic design and art. They can be used to create symmetrical designs or to draw attention to a particular element in a composition.

Concentric circles are a versatile and useful mathematical concept that has applications in various fields, including computer vision, math, and art.

7.3 BOUNDING BOX METHOD:

The bounding box method is a computer vision technique used to locate and identify objects within an image. The technique involves drawing a rectangle or box around the object of interest in the image.

In the bounding box method, the object of interest is typically identified using object detection techniques, such as edge detection, color-based segmentation, or feature extraction. Once the object is identified, a rectangle or box is drawn around it to highlight its location and boundaries.

The bounding box method is commonly used in object detection and recognition applications, such as pedestrian detection, vehicle detection, and face detection. The bounding box can be used to crop the image around the object of interest, making it easier to analyse and classify the object.

The size and shape of the bounding box can be adjusted based on the object being detected and the application requirements. For example, in some applications, a tight bounding box may be required to accurately identify the object, while in others, a looser bounding box may be more appropriate.

The bounding box method is a powerful and widely used technique in computer vision that allows for the identification and analysis of objects within an image by drawing a rectangle or box around them.

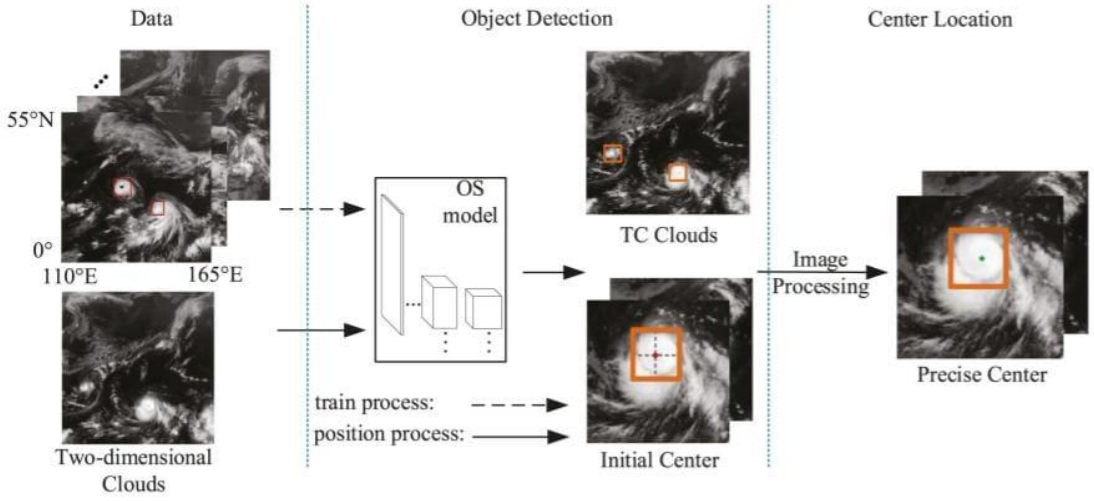

Fig 7.3: Flowchart of the OSIP algorithm

7.4 LABELLING METHOD FOR BOUNDING BOX METHOD:

Labeling methods for the bounding box method involve assigning a label or class to an object of interest within an image or video, and then drawing a bounding box around that object. This allows an object detection algorithm to identify and locate the object within the image or video.

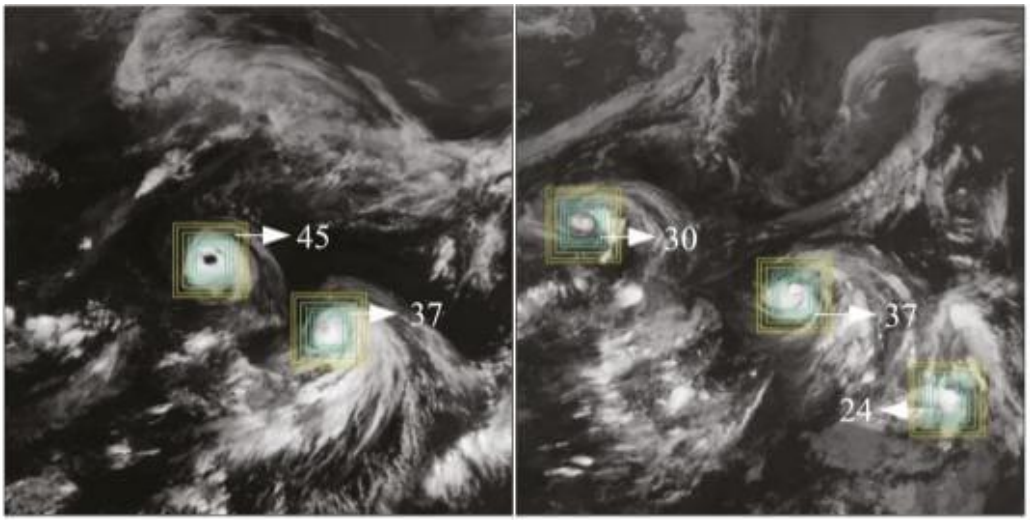

Fig 7.4: Examples of bounding box labelling

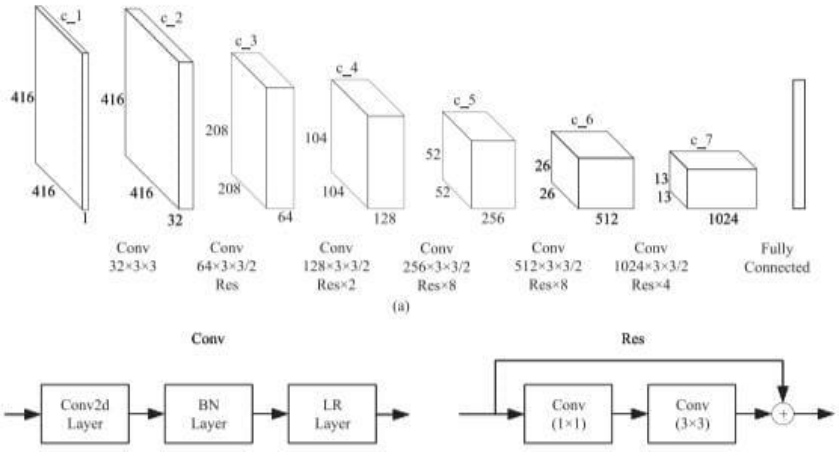

Fig7.4.1: Structure of Darknet

Manual Labeling: This involves manually annotating each object of interest within an image or video frame. The annotator draws a bounding box around the object and assigns it a class label.

Semi-Automatic Labeling: This involves using software to assist with the labeling process. The software may use object tracking algorithms to automatically draw bounding boxes around objects of interest, which can then be reviewed and refined by a human annotator.

Active Learning Labeling: This involves using a machine learning model to iteratively select the most informative samples for labeling. The model selects the samples that it is most uncertain about and presents them to a human annotator for labeling.

Crowdsourcing Labeling: This involves outsourcing the labeling task to a group of people, often through an online platform. The annotators may be paid or unpaid, and the quality of the labels may vary depending on the level of expertise and attention to detail of the annotators.

Regardless of the labeling method used, it is important to ensure that the labels are accurate and consistent across all images or video frames. This can help to ensure that the object detection algorithm can accurately locate and classify objects of interest within the data.

7.5 GETTING EPICENTRIC OF CYCLONE:

Getting the epicenter of a cyclone typically involves analyzing satellite images or radar data to locate the center of rotation of the storm. There are several methods for doing this, including: 7.

Eye Wall Replacement Cycle (EWRC) method:

This method involves tracking the changes in the eye wall of the cyclone as it undergoes an EWRC. The center of rotation is located where the old eye wall disappears and the new eye wall forms.

Maximum Wind Speed method:

This method involves tracking the maximum wind speed of the storm as it moves across the ocean. The center of rotation is typically located at the point where the maximum wind speeds are recorded.

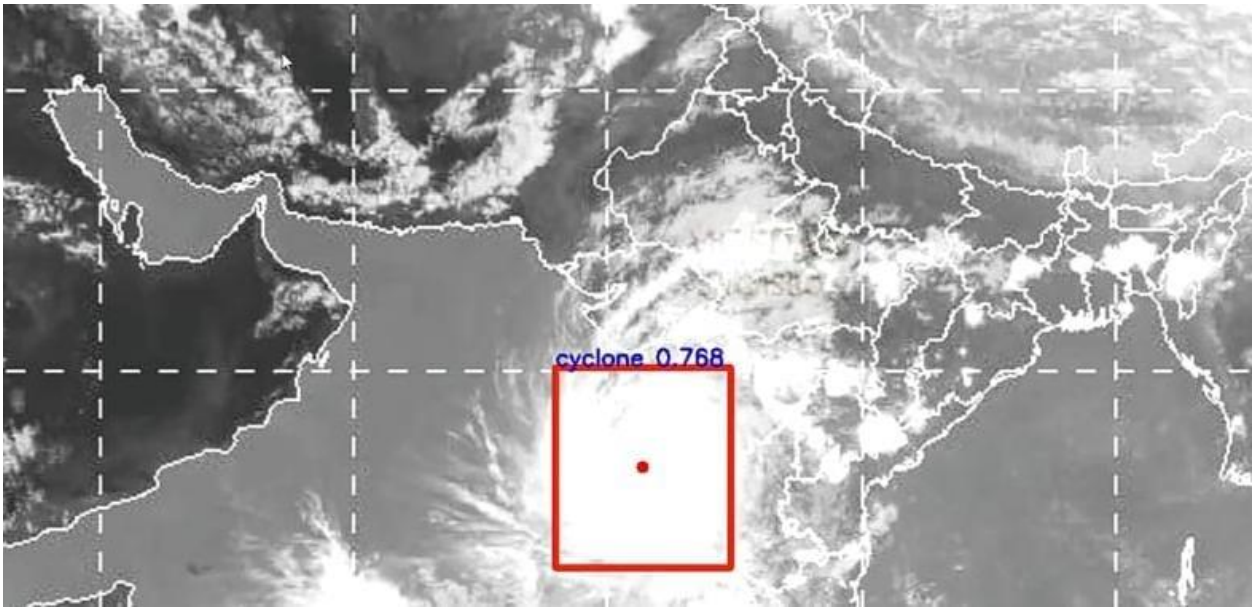

Fig 7.6: EPICENTER OF CYCLONE IN SATELLITE IMAGE

Pressure Drop method: This method involves tracking the changes in air pressure within the storm. The center of rotation is typically located where the lowest air pressure is recorded.

Spiral Banding method: This method involves tracking the spiral bands of clouds that spiral around the center of rotation. The center of rotation is typically located where the spiral bands converge.

In all of these methods, the location of the center of rotation is estimated by analyzing various data points related to the cyclone, such as wind speeds, air pressure, or cloud patterns. Once the location of the center of rotation is identified, this point is typically referred to as the "epicenter" of the cyclone.

It's important to note that the accuracy of these methods can vary depending on the quality and quantity of the data available, as well as other factors such as the size and intensity of the storm.

7.6 GETTING EPICENTRIC OF CYCLONE USING YOLOV7:

There is no such thing as YOLO v7, as the latest version of YOLO (You Only Look Once) is YOLOv5. However, I can provide a general description of how object detection techniques such as YOLO can be used to locate the epicenter of a cyclone in an image or video.

Object detection in YOLO involves dividing the image into a grid and predicting bounding boxes for objects within each grid cell. To locate the epicenter of a cyclone, the first step would be to train a YOLO model on a dataset of cyclone images, with the ground truth labels indicating the location of the epicenter.

Once the model is trained, it can be used to detect the location of the epicenter in new images or video frames. This may involve adjusting the size and shape of the bounding boxes predicted by the model to fit the shape of the cyclone and to accurately locate the epicenter.Another approach to locating the epicenter of a cyclone using YOLO is to use feature extraction to identify patterns in the image that correspond to the shape of the cyclone, such as circular patterns or areas of high intensity. These features can then be used to locate the epicenter within the image.

YOLO can be a powerful tool for object detection and can be used to locate the epicenter of a cyclone in an image or video by training a model on a dataset of cyclone images and using feature extraction to identify patterns in the image that correspond to the shape of the cyclone.

7.7 **CODE:**

```
import cv2

import time

import requests

import random

import numpy as np

import onnxruntime as ort

from PIL import Image

from pathlib import Path

from collections import OrderedDict, namedtuple

# test_img = r"C:\Users\jagan\Downloads\project\four.jpg"

img = cv2.VideoCapture(r"C:\Users\janardhan\Downloads\test.mp4")

out = cv2.VideoWriter('output.avi',-1,20.0,(640,480))

# img = cv2.imread(test_img)

# print(type(img))

# cv2.imshow("hi",img)

# cv2.waitKey(40)
```

```
onnx_mld = r"C:\Users\janardhan\Downloads\last.onnx"

cuda = False

# whT = 320

# confThreshold = 0.5

# nmsThreshold = 0.3

# onx = cv2.dnn.readTensorFromONNX(onnx_mld)

# print(onnx_mld)

# providers = ['CUDAExecutionProvider', 'CPUExecutionProvider'] if cuda else
['CPUExecutionProvider']

providers = ['CPUExecutionProvider']

session = ort.InferenceSession(onnx_mld, providers=providers)

def letterbox(im, new_shape =(640, 640), color =(114, 114, 114), auto=True, scaleup=True,
stride=32):

    # Resize and pad image while meeting stride-multiple constraints

    shape = im.shape[:2] # current shape [height, width]

    if isinstance(new_shape, int):

        new_shape = (new_shape, new_shape)

# Scale ratio (new / old)
```

```
    r = min(new_shape[0] / shape[0], new_shape[1] / shape[1])

    if not scaleup:  # only scale down, do not scale up (for better val mAP)

        r = min(r, 1.0)

# Compute padding

    new_unpad = int(round(shape[1] * r)), int(round(shape[0] * r))

    dw, dh = new_shape[1] - new_unpad[0], new_shape[0] - new_unpad[1]  # wh padding



    if auto:  # minimum rectangle

        dw, dh = np.mod(dw, stride), np.mod(dh, stride)  # wh padding

dw /= 2  # divide padding into 2 sides

    dh /= 2

if shape[::-1] != new_unpad:  # resize

        im = cv2.resize(im, new_unpad, interpolation=cv2.INTER_LINEAR)

    top, bottom = int(round(dh - 0.1)), int(round(dh + 0.1))

    left, right = int(round(dw - 0.1)), int(round(dw + 0.1))

    im = cv2.copyMakeBorder(im, top, bottom, left, right, cv2.BORDER_CONSTANT,
value=color)  # add border

    # mid = ((top+bottom)/2,(left+right)/2)
```

```
    return im, r, (dw, dh)

# names = ['person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck', 'boat', 'traffic
light',

#         'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow',

#         'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee',

#         'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball glove', 'skateboard',
'surfboard',

#         'tennis racket', 'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple',

#         'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair', 'couch',

#         'potted plant', 'bed', 'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell
phone',

#         'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors', 'teddy
bear',

#         'hair drier', 'toothbrush']

names = ['cyclone']

colors = {name: [random.randint(0, 255) for _ in range(3)] for i, name in enumerate(names)}

# print(colors)

# ret,img = img.read()

if (img.isOpened()==False):
```

```
    print("error")

while True:

    # print(type(img))

    # try:

    img1 = img

    success, img1 = img1.read()

 if success == False:

        break

# except Exception:

    #     print(Exception)

 # print(type(img))

    img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)

    # print(img1)

    # cv2.imshow("imshow", img1)

    # cv2.waitKey(3000)

image = img1

    image, ratio, dwdh = letterbox(image, auto=False)
```

```
    image = image.transpose((2, 0, 1))

    image = np.expand_dims(image, 0)

    image = np.ascontiguousarray(image)

    key = 0

    im = image.astype(np.float32)

    im /= 255

    # im.shape

outname = [i.name for i in session.get_outputs()]

    # outname

  inname = [i.name for i in session.get_inputs()]

    # inname

 inp = {inname[0]: im}

outputs = session.run(outname, inp)[0]

    # print(outputs)

ori_images = [img1.copy()]

for i, (batch_id, x0, y0, x1, y1, cls_id, score) in enumerate(outputs):

        image = ori_images[int(batch_id)]
```

```
        box = np.array([x0, y0, x1, y1])

        box -= np.array(dwdh * 2)

        box /= ratio

        box = box.round().astype(np.int32).tolist()

        cls_id = int(cls_id)

        score = round(float(score), 3)

        name = names[cls_id]

        color = colors[name]

        # color = (255,0,0)

        name += ' ' + str(score)

        thickness = 5

        # print(tuple(box[:2]), tuple(box[2:]))

        xb, yu = box[:2]

# print(mid1)

        xu, yb = box[2:]

        mid1 = (xb + xu) / 2

        mid2 = (yu + yb) / 2
```

```
        cv2.rectangle(image, pt1=tuple(box[:2]), pt2=tuple(box[2:]), color=color,
thickness=thickness)

        cv2.putText(image, name, (box[0], box[1] - 2), cv2.FONT_HERSHEY_SIMPLEX, 0.75,
[225, 0, 0], thickness=2)

        # print(mid)

        mid = (int(mid1),int(mid2))

        print("mid vales:",mid)

        image = cv2.circle(image, mid, radius=5, color=(0, 0, 255), thickness=-1)

        key = 1

        # cv2.imwrite("hello.jpg", image)

        # cv2.imshow('Image', image)

Image.fromarray(ori_images[0])

    frame = cv2.flip(image, 0)

    out. write(frame)

    # image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)

    cv2.imshow("image", image)

    if cv2.waitKey(1) & 0xFF == ord('q'):

        break
```

```
# After the loop release the cap object

img.release()

out.release()

# Destroy all the windows

cv2.destroyAllWindows()

    # if key == 1:

    #     image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

    #     cv2.imshow("image", image)

    #     key = 0

    # else:

    #     img1 = cv2.cvtColor(img1, cv2.COLOR_RGB2BGR)

    #     cv2.imshow("image", img1)

    # cv2.waitKey(1)

    # cv2.waitKey(0)

# img.release()

# while True:

#     ret, frame = img.read()
```

```
#     cv2.imshow("frame", frame)

# cv2.imshow("test",img)

# cv2.waitkey(0)

# vid = cv2.VideoCapture(0)

# vid = img

# while (True):

#     # Capture the video frame

#     # by frame

#     ret, frame = vid.read()

#     # Display the resulting frame

#     cv2.imshow('frame', frame)

#     # the 'q' button is set as the

#     # quitting button you may use any

#     # desired button of your choice

#     if cv2.waitKey(1) & 0xFF == ord('q'):

#         break

# vid.release()
```

CHAPTER 8

TRAJECTORY OF THE CYCLONE

The trajectory of a cyclone refers to the path it takes as it moves across the ocean. Cyclones typically move in a westward or north-westward direction and can be influenced by a variety of factors such as wind patterns, ocean currents, and temperature gradients. Predicting the trajectory of a cyclone is important for issuing warnings to areas that may be affected by the storm. Data is collected from various sources, including satellite images, radar data, and weather buoys, and then analysed to create computer models that can simulate the path of the storm. The accuracy of cyclone trajectory predictions has improved over the years due to advancements in technology and modeling techniques.

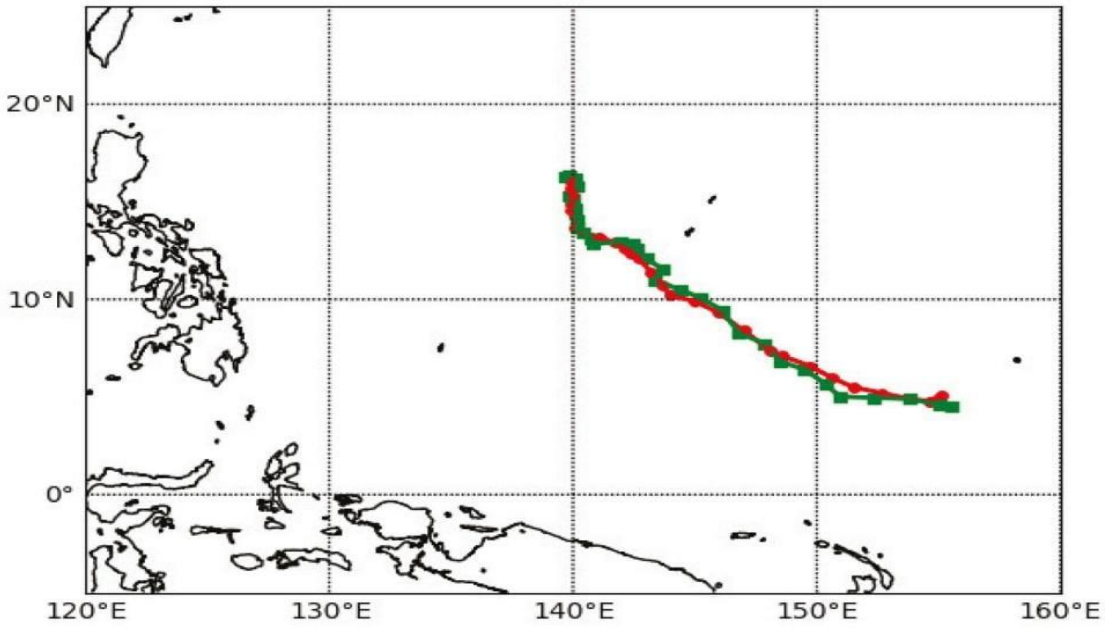

Fig 8: Trajectory of the cyclone

Satellite images can be used to track the trajectory of a cyclone by observing its movement over time. The satellite images can show the location of the cyclone and the direction it is moving. By analyzing the sequence of satellite images, the trajectory of the cyclone can be determined.

Satellite images are collected from various sources, such as polar orbiting and geostationary satellites. Polar orbiting satellites provide high-resolution images that can be used to track the movement of the cyclone over a smaller area, while geostationary satellites provide wider coverage of the storm and can track its movement over a larger area.

To track the trajectory of a cyclone using satellite images, meteorologists use a technique called motion estimation. This involves comparing the position of the cyclone in two or more consecutive satellite images and calculating the distance and direction it has moved. By repeating this process with subsequent images, the trajectory of the cyclone can be estimated.

The accuracy of cyclone trajectory estimates from satellite images can be influenced by factors such as cloud cover, image resolution, and the frequency of image captures. However, advancements in satellite technology have led to improved accuracy in recent years, allowing for more accurate predictions of the path of the cyclone.

The trajectory of a cyclone can be marked at different points depending on the specific context. In general, the trajectory of a cyclone is marked at regular intervals along its path to track its

movement over time and predict where it may be headed.

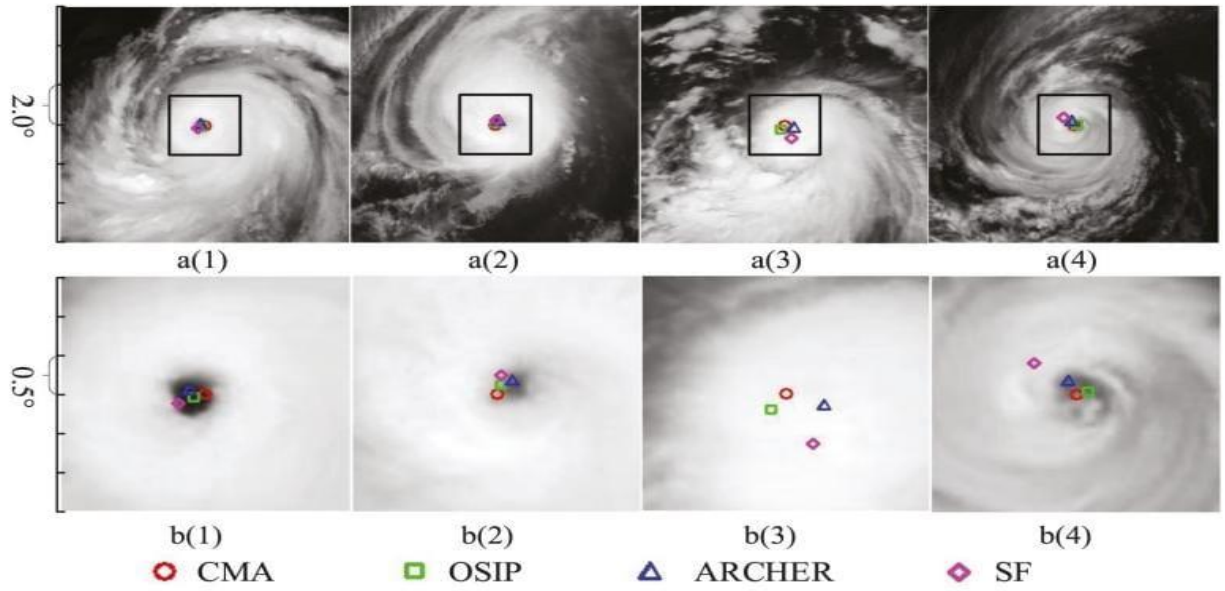

Fig 8.2: The Colours marks are the position results

The points on the trajectory can be marked based on the specific tracking method used, such as satellite imagery or ground-based observations. For example, in satellite imagery, the position of the cyclone can be marked based on the center of the storm or the location of its outer bands. In ground-based observations, the position of the cyclone can be marked based on its location relative to specific landmarks or weather stations.

In addition to marking the current position of the cyclone, other points on its trajectory can be marked to track its movement and predict its future path. These points may include the initial point of formation, where the cyclone first began to develop, and the point of landfall, where the storm makes land contact and causes potential damage to coastal areas.

Other important points on the trajectory of a cyclone include the points of maximum intensity, where the storm reaches its peak strength, and points where the storm undergoes significant changes in direction or speed. By marking these points and tracking the movement of the cyclone over time, meteorologists can provide accurate forecasts and issue warnings to areas that may be in the path of the storm.

8.1 CODE:

```
import cv2

import time

import requests

import random

import numpy as np

import onnxruntime as ort

from PIL import Image

from pathlib import Path

from collections import OrderedDict, namedtuple

# test_img = r"C:\Users\jagan\Downloads\project\four.jpg"

img = cv2.VideoCapture(r"C:\Users\janardhan\Downloads\test.mp4")

out = cv2.VideoWriter('output.avi',-1,20.0,(640,480))

# img = cv2.imread(test_img)

# print(type(img))

# cv2.imshow("hi",img)

# cv2.waitKey(40)
```

```
onnx_mld = r"C:\Users\janardhan\Downloads\last.onnx"

cuda = False

# whT = 320

# confThreshold = 0.5

# nmsThreshold = 0.3

# onx = cv2.dnn.readTensorFromONNX(onnx_mld)

# print(onnx_mld)

# providers = ['CUDAExecutionProvider', 'CPUExecutionProvider'] if cuda else
['CPUExecutionProvider']

providers = ['CPUExecutionProvider']

session = ort.InferenceSession(onnx_mld, providers=providers)

def letterbox(im, new_shape=(640, 640), color=(114, 114, 114), auto=True, scaleup=True,
stride=32):

    # Resize and pad image while meeting stride-multiple constraints

    shape = im.shape[:2]  # current shape [height, width]

    if isinstance(new_shape, int):

        new_shape = (new_shape, new_shape)

# Scale ratio (new / old)
```

```
    r = min(new_shape[0] / shape[0], new_shape[1] / shape[1])

    if not scaleup:  # only scale down, do not scale up (for better val mAP)

        r = min(r, 1.0)

# Compute padding

    new_unpad = int(round(shape[1] * r)), int(round(shape[0] * r))

    dw, dh = new_shape[1] - new_unpad[0], new_shape[0] - new_unpad[1]  # wh padding



    if auto:  # minimum rectangle

        dw, dh = np.mod(dw, stride), np.mod(dh, stride)  # wh padding

dw /= 2  # divide padding into 2 sides

    dh /= 2

if shape[::-1] != new_unpad:  # resize

        im = cv2.resize(im, new_unpad, interpolation=cv2.INTER_LINEAR)

    top, bottom = int(round(dh - 0.1)), int(round(dh + 0.1))

    left, right = int(round(dw - 0.1)), int(round(dw + 0.1))

    im = cv2.copyMakeBorder(im, top, bottom, left, right, cv2.BORDER_CONSTANT,
value=color)  # add border

    # mid = ((top+bottom)/2,(left+right)/2)
```

```
    return im, r, (dw, dh)

# names = ['person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck', 'boat', 'traffic
light',

#         'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow',

#         'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee',

#         'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball glove', 'skateboard',
'surfboard',

#         'tennis racket', 'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple',

#         'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair', 'couch',

#         'potted plant', 'bed', 'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell
phone',

#         'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors', 'teddy
bear',

#         'hair drier', 'toothbrush']

names = ['cyclone']

colors = {name: [random.randint(0, 255) for _ in range(3)] for i, name in enumerate(names)}

# print(colors)

# ret,img = img.read()

mid_val = []
```

```
if (img.isOpened()==False):
    print("error")
while True:
    # print(type(img))
    # try:
    img1 = img
    success, img1 = img1.read()


    if success == False:
        break
# except Exception:
    # print(Exception)
# print(type(img))
    img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
    # print(img1)
    # cv2.imshow("imshow", img1)
    # cv2.waitKey(3000)
```

```
Image = img1
    image, ratio, dwdh = letterbox(image, auto=False)
    image = image.transpose((2, 0, 1))
    image = np.expand_dims(image, 0)
    image = np.ascontiguousarray(image)
    key = 0
    im = image.astype(np.float32)
    im /= 255
    # im.shape
outname = [i.name for i in session.get_outputs()]
    # outname
inname = [i.name for i in session.get_inputs()]
    # inname
 inp = {inname[0]: im}
outputs = session.run(outname, inp)[0]
    # print(outputs)
ori_images = [img1.copy()]
```

```
for i, (batch_id, x0, y0, x1, y1, cls_id, score) in enumerate(outputs):
    image = ori_images[int(batch_id)]
    box = np.array([x0, y0, x1, y1])
    box -= np.array(dwdh * 2)
    box /= ratio
    box = box.round().astype(np.int32).tolist()
    cls_id = int(cls_id)
    score = round(float(score), 3)
    name = names[cls_id]
    color = colors[name]
    # color = (255,0,0)
    name += ' ' + str(score)
    thickness = 5
    # print(tuple(box[:2]), tuple(box[2:]))
    xb, yu = box[:2]
# print(mid1)
    xu, yb = box[2:]
```

```
        mid1 = (xb + xu) / 2

        mid2 = (yu + yb) / 2

        cv2.rectangle(image, pt1=tuple(box[:2]), pt2=tuple(box[2:]), color=color,
thickness=thickness)

        cv2.putText(image, name, (box[0], box[1] - 2), cv2.FONT_HERSHEY_SIMPLEX, 0.75,
[225, 0, 0], thickness=2)

        # print(mid)

        mid = (int(mid1),int(mid2))

        mid_val.append(mid)

        print("mid vales:",mid)

        # image = cv2.circle(image, mid, radius=5, color=(0, 0, 255), thickness=-1)

        for l in mid_val:

            cv2.circle(image, l, radius=5, color=(0, 0, 255), thickness=-1)

        key = 1

        # cv2.imwrite("hello.jpg", image)

        # cv2.imshow('Image', image)

 Image.fromarray(ori_images[0])

    frame = cv2.flip(image, 0)
```

```
    out.write(frame)
    # image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
    cv2.imshow("image", image)
    if cv2.waitKey(30) & 0xFF == ord('q'):
        break
# After the loop release the cap object
img.release()
out.release()
# Destroy all the windows
cv2.destroyAllWindows()
    # if key == 1:
    #     image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    #     cv2.imshow("image", image)
    #     key = 0
    # else:
    #     img1 = cv2.cvtColor(img1, cv2.COLOR_RGB2BGR)
    #     cv2.imshow("image", img1)
```

```
    # cv2.waitKey(1)

    # cv2.waitKey(0)

# img.release()

# while True:

#     ret, frame = img.read()

#     cv2.imshow("frame", frame)



# cv2.imshow("test",img)

# cv2.waitkey(0)

# vid = cv2.VideoCapture(0)

# vid = img

# while (True):

#     # Capture the video frame

#     # by frame

#     ret, frame = vid.read()

#     # Display the resulting frame

#     cv2.imshow('frame', frame)
```

```
#     # the 'q' button is set as the

#     # quitting button you may use any

#     # desired button of your choice

#     if cv2.waitKey(1) & 0xFF == ord('q'):

#         break

# # After the loop release the cap object

# vid.release()

# # Destroy all the windows

# cv2.destroyAllWindows()
```

OUTPUT:

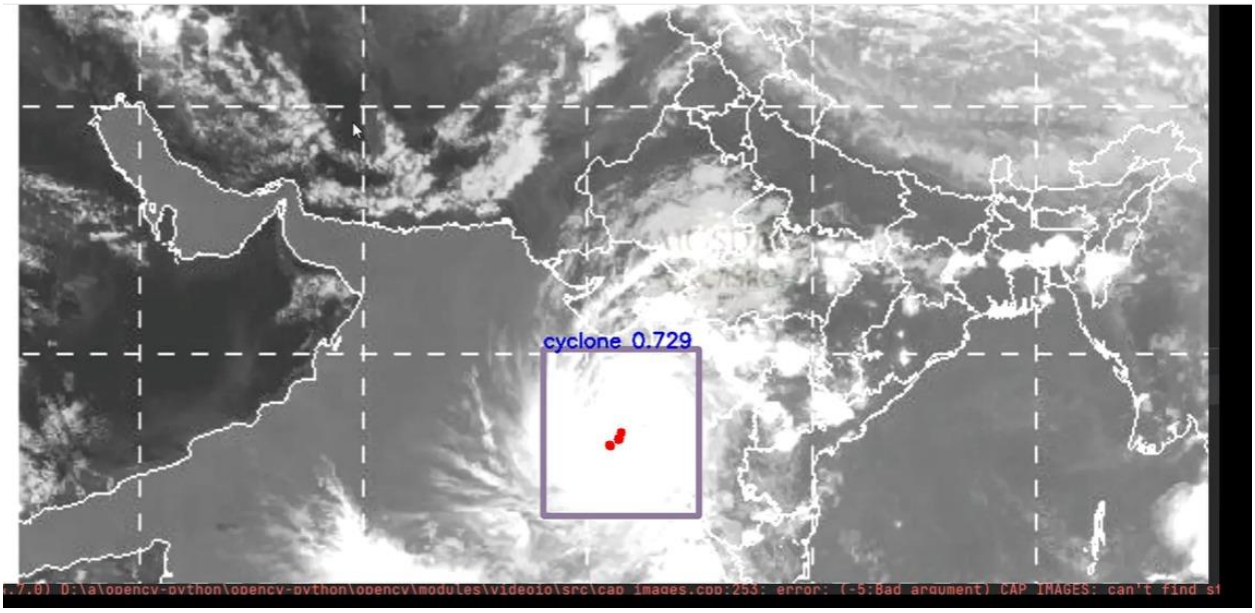

Fig 8.1: Trajectory of the cyclone in satellite image

CHAPTER 9

CONCLUSION

By analyzing the features of TCs such as structure andmorphology in satellite IR images, a two-step method called OSIP based on deep learning network object detection and IP isdesigned to locate TC center in this article. First, Darknet-53 isused to extract the features at different scales. According to thescale distribution of TCs, three hierarchical features are selected for two-scale fusion. The object detection model can consider both the general characteristics of details (low-level feature) and overall morphological attributes (high-level feature). Thus, thismodel shows good adaptability to the complex and changeable flexible TCs. The detection results make the center location getrid of background interference and create conditions for the position algorithm to focus on rational physical characteristics. Second, the characteristics of the eye and its surrounding mor-phological features are considered comprehensively to achieve the accurate location of the TC center. Based on the circle detection, a comprehensive decision is made by multiple information such as the outer circle, the inner circle, and the center of the object detection box. Experimental results show that the proposed algorithm has better position accuracy compared with the existing methods, especially for TCs without typical structural and morphological features. Because OSIP does not need to use the ergodic algorithm for parameter estimation, it has high efficiency and universality. In future work, accurate central information can be used to construct a more perfect model to complete the difficult task of estimating the TC intensity in real time.

REFERENCES

1. T. L. Olander, C. S. Velden, and J. Kossin, "The advanced objective dvorak\ technique (AODT)–latest upgrades and future directions," in Proc. 26th Conf. Hurricanes Tropical Meteorol., 2004, pp. 294–295.
2. P. Shamsolmoali, M. Zareapoor, R. Wang, H. Zhou, and J. Yang, "A novel deep structure U-Net for sea-land segmentation in remote sensing images," IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens., vol. 12, no. 9, pp. 3219–3232, Sep. 2019.
3. W. Fang et al., "Recognizing global reservoirs from landsat 8 images: A deep learning approach," IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens., vol. 12, no. 9, pp. 3168–3177, Sep. 2019
4. A. J. Wimmers and C. S. Velden, "Objectively determining the rotational center of tropical cyclones in passive microwave satellite imagery," J. Appl. Meteorol. Climatol., vol. 49, no. 9, pp. 2013–2034, 2010.
5. K. Y. Wong, C. L. Yip, and P. W. Li, "Automatic tropical cyclone eye fix using genetic algorithm," Expert Syst. Appl., vol. 34, no. 1, pp. 643–656, 2008.
6. C.-J. Zhang and Q. Luo, "Tropical cyclones objection detection based on faster r-CNN and infrared satellite cloud images," in Proc. Int. Conf. Image Video Process., Artif. Intell., 2018, vol. 10836, Art. no. 108360G.
7. K. Y. Wong and C. L. Yip, "An intelligent tropical cyclone eye fix system using motion field analysis," in Proc. IEEE Int. Conf. Tools Artif. Intell., 2005, pp. 652–656.
8. M. Ying et al., "An overview of the China Meteorological Administration tropical cyclone database," J. Atmospheric Oceanic Technol., vol. 31, no. 2, pp. 287–301, 2014.
9. N. Otsu, "A threshold selection method from gray-level histograms," IEEE Trans. Syst., Man, Cybern., vol. 9, no. 1, pp. 62–66, Jan. 1979.
10. J. Canny, "A computation approach to edge detection," IEEE Trans. Pattern Anal. Mach. Intell., vol. 8, no. 6, pp. 670–700, Nov. 1986.