1. INTRODUCTION

The Tropical Cyclones (TCs) are termed differently in different parts of the globe. Over the Atlantic and Eastern Pacific, they are termed as ‘Hurricane’ and in western Pacific as ‘Typhoon’. In the North Indian Ocean (NIO) region, they are named as ‘Tropical Cyclone’. A "Cyclonic Storm' or a "Cyclone" is an intense vortex or a whirl in the atmosphere with very strong winds circulating around it in anti-clockwise direction in the Northern Hemisphere and in clockwise direction in the Southern Hemisphere.

The word "Cyclone" is derived from the Greek word 'Cyclos" meaning the coil of a snake. To Henri Piddington, the tropical storms in the Bay of Bengal and in the Arabian Sea appeared like the coiled serpents of the Sea and he named these storms as "Cyclones". Tropical cyclones are also referred to as 'Hurricanes' over Atlantic Ocean, 'Typhoons' over Pacific Ocean, 'Willy-Willies' over Australian Seas and simply as 'Cyclones' over north Indian Ocean (NIO).

The NIO contributes 5-6 TCs every year (~7% of the world’s TCs) (Mohanty and Gupta, 1997). Though the percentage of frequency over the NIO basin is less relative to any other global basins, annual frequency rate is often stable with an average disparity of ±7%. Every year, about 80 TCs causes an average number of 20,000 deaths and a total economic loss of $6-7 billion (Mohanty et al., 2015). Almost all the TCs form within 25° latitudes on both sides of the equator except in the equatorial region of 5° S to 5° N due to negligible Coriolis force

The NIO basin comprised of Bay of Bengal (BoB) and Arabian Sea (AS), also commonly referred as Indian seas, is unique in its nature than any other global basins, as far as the genesis and season are concerned. Mohanty et al. (2011) suggest that the BoB and AS contributes ~76% and ~24%, respectively to the total number of TCs. In other ocean basins, TCs occur around late summer to early fall. However, the genesis of TCs over the NIO basin is very seasonal with

primary maxima in post monsoon season (October-December) and secondary maxima during pre-monsoon season (April-May) (Mohanty, 1997). The TCs over the NIO move usually the west, north-west and northward, sometimes, recurve towards north-east or east

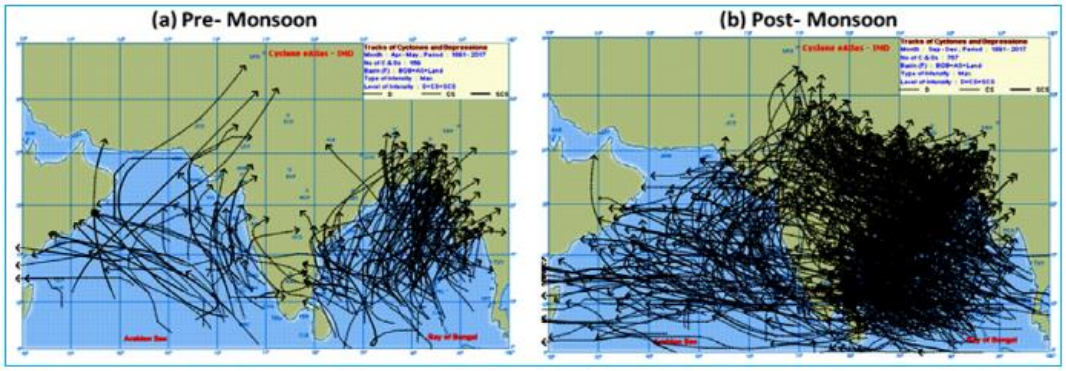

Fig 1.1 Observed tracks of tropical cyclones over the NIO basin for a (a) pre-monsoon and (b) post monsoon season

1.1 Classification of Cyclonic Disturbances

Cyclones are intense low-pressure areas - from the center of which pressure increases outwards. The amount of the pressure drops in the center and the rate at which it increases outwards gives the intensity of the cyclones and the strength of winds. The criteria followed by the India Meteorological Department (IMD) to classify the low-pressure systems in the Bay of Bengal and in the Arabian Sea as adopted by the World Meteorological Organization (W.M.O.) are given in Table 1.1.

S.No	Type of disturbance	Associated maximum sustained wind (MSW)
1.	Low-pressure Area	Not exceeding 17 knots (<31 kmph)
2.	Depression	17 to 27 knots (31-49 kmph)
3.	Deep Depression	28 to 33 Knots (50-61 kmph)
4.	Cyclonic Storm	34 to 47 Knots (62-88 kmph)
5.	Severe Cyclonic Storm	48 to 63 Knots (89-117 kmph)
6.	Very Severe Cyclonic Storm	64 to 90 Knots (118-167 kmph)
7.	Extremely Severe Cyclonic Storm	91 to119 Knots (168-221 kmph)
8.	Super Cyclonic Storm	120 Knots and above (≥222 kmph)

Table 1.1 Criteria for classification of cyclonic disturbances over the North Indian Ocean

1.2 Importance of Identification of Cyclones

Cyclones are one of the most devastating natural disasters on the planet. They are characterized by strong winds, heavy rainfall, and storm surges that can cause widespread damage to infrastructure, homes, and livelihoods. Identifying cyclones is crucial to taking preventative measures and mitigating the damage caused by these storms. In this essay, we will discuss the importance of identifying cyclones and the impact it has on people, property, and the environment

The identification of cyclones is crucial because it helps in predicting their path and intensity. With accurate predictions, authorities can evacuate people from the affected areas and prepare them for the worst-case scenario. This can save many lives and minimize damage to property and infrastructure. In the absence of identification, authorities would not be able to take timely action, resulting in more loss of life and property.

Identification of cyclones is also essential for mitigating their impact on the environment. Cyclones can cause soil erosion, loss of vegetation, and contamination of water sources. With accurate identification, authorities can take preventative measures, such as reinforcing soil and vegetation cover, to minimize the impact of the storm on the environment. This can ensure that the ecological balance of the region is not affected in the long run.

In addition, identifying cyclones is critical for monitoring the changing climate. Cyclones are a natural phenomenon that is influenced by various factors, including temperature, humidity, and wind patterns. Studies of cyclone patterns and frequency can help scientists better understand the impacts of climate change. This knowledge can help in developing effective policies and strategies to mitigate the impact of climate change on vulnerable regions.

In conclusion, the identification of cyclones is essential for protecting human life, property, and the environment. Accurate identification allows authorities to take timely action, allocate resources, and mitigate the impact of the storm on the environment. It also provides valuable data for monitoring the impact of climate change on the planet. Therefore, it is crucial to invest in research and technology that can improve our ability to identify and predict cyclones accurately

The identification of TCs from satellite images is based on their size, position, status, and intensity. Deep learning algorithms have also been used for identification and classification purposes. This has been demonstrated in the use of an ensemble of convolutional neural networks (CNNs) classifiers on the simulated outgoing longwave radiation (OLR) for classifying as TCs and their precursors in [24]. The CNNs were trained with 50 000 images containing TCs and their precursors and 500 000 non-TC data for binary classification, showing success in the WNP region. Also, there have been interesting attempts to estimate TC intensity from satellite data by using CNN in recent times.

This project uses ML & DL techniques to detect TCs from high-resolution satellite images by considering only the shape of the clouds in the images and maximum sustained surface wind speeds from JTWC. Each detection is also provided with a segmentation, allowing any initial analysis based on the shape and size of the detected TC. The pipeline consists of a state-of-the-art mask region-CNN (R-CNN) detector, a wind speed filter, and a CNN classifier

2. LITERATURE SURVEY

Tropical cyclones are among the most devastating weather systems on the planet, and accurate predictions of them are essential to make reliable warnings. For these warnings, the information about the track of the cyclone and its intensity is required and needs to be extracted from forecasts.

Forecasting can be done with three levels of observations:

(i) Surface level

(ii) Upper air level

(iii) Space -Based

A. Surface observations from:
   i. Land synoptic stations
   ii. Ships
   iii. Buoys
   iv. Tide gauges
   v. Aviation meteorological offices
   vi. Automatic Weather Stations (AWSs)

B. Upper air observations from:
   i. Pilot balloon stations
   ii. GPS Sonde/ Radiosonde /Radiowind stations
   iii. Wind profiler stations

C. Radar Observations

D. Satellite observations

E. Aircraft observations

F. Microseism observations

Satellite Imagery

There are three main types of satellite images available:

- Visible
- Infrared
- Water Vapor

Visible Imagery: Visible satellite pictures can only be viewed during the day, since clouds reflect the light from the sun. On these images, clouds show up as white, the ground is normally grey, and water is dark. In winter, snow-covered ground will be

white, which can make distinguishing clouds more difficult. To help differentiate between clouds and snow, looping pictures can be helpful; clouds will move while the snow won't. Snow-covered ground can also be identified by looking for terrain features, such as rivers or lakes. Rivers will remain dark in the imagery as long as they are not frozen. If the rivers are not visible, they are probably covered with clouds. Visible imagery is also very useful for seeing thunderstorm clouds building. Satellite will see the developing thunderstorms in their earliest stages, before they are detected on radar.

Infrared Imagery: Infrared satellite pictures show clouds in both day and night. Instead of using sunlight to reflect off of clouds, the clouds are identified by satellite sensors that measure heat radiating off of them. The sensors also measure heat radiating off the surface of the earth. Clouds will be colder than land and water, so they are easily identified. Infrared imagery is useful for determining thunderstorm intensity. Strong to severe thunderstorms will normally have very cold tops. Infrared imagery can also be used for identifying fog and low clouds. The fog product combines two different infrared channels to see fog and low clouds at night, which show up as dark areas on the imagery.

Water Vapor Imagery: Water vapor satellite pictures indicate how much moisture is present in the upper atmosphere (approximately from 15,000 ft to 30,000 ft). The highest humidities will be the whitest areas while dry regions will be dark. Water vapor imagery is useful for indicating where heavy rain is possible. Thunderstorms can also erupt under the high moisture plumes

Satellite Cloud Imagery Data and Derived products useful in cyclone detection.

Cloud Imagery Data At present IMD is receiving and processing meteorological data from two Indian satellites namely Kalpana-1 and INSAT-3A. Kalpana-1 was launched on 12th September, 2002 and is located at 740 E. INSAT-3A was launched on 10 April, 2003 and is located at 93.50 E. Kalpana-1 and INSAT-3A both have three channel Very High Resolution Radiometer (VHRR) for imaging the Earth in Visible (0.55-0.75 um), Infra-Red (10.5-12.5um) and Water vapour (5.7-7.1um) channels having resolution of 2X2 km. in visible and 8X8 km. in Water vapour (WV) and Infra red (IR) channels. In addition, the INSAT-3A has a three channel Charge Coupled Device (CCD) payload for imaging the earth in Visible (0.62- 0.69um), Near IR (0.77-0.86um) and Short Wave IR (1.55-1.77um) bands of Spectrum.

Indian Meteorological Data Processing System (IMDPS) is processing meteorological data from INSAT VHRR and CCD data and supports all operational activities of the Satellite Meteorology Division on round the clock basis. Cloud Imagery Data are processed and transmitted to forecasting offices of the IMD as well as to the other users in India and foreign countries.

To supplement these observations, cloud imagery data from METEOSAT-5 satellite, which is also located to observe Indian region, from 63 deg E long, are also being received in VIS (0.4-1.1mm), IR (10.5-12.5mm) and Water Vapour channels(5.7-7.1mm). Since all these satellites are geostationary satellites, cloud imagery data from these satellites are frequently ingested. It is 3 hourly in case of INSAT and half hourly in case of Kalpana-1 and ranges from half to one and half-hourly in case of METEOSAT-5 satellite.

From observations (mainly satellite images), tropical cyclones are manually detected by the World Meteorological Organization (WMO) regional specialized meteorological centers (RSMCs) and tropical cyclone warning centers (TCWCs).

TC track forecasting can also be regarded as a classification issue. In the Northwest Pacific, Camargo et al. (2007) used the shape and movement parameters of TC tracks to conduct the K-means cluster analysis (Krishna and Narasimha, 1999), and they pointed out that TC tracks in this region are mainly “westward” and “turning” types. (2008) adopted a dynamic fuzzy clustering method to investigate the TC tracks in the South China Sea from 1960 to 2002. Also, they surveyed the TC-related factors, namely circulation, physical factors and motion characteristics, and then they established a forecast model for summer TC tracks in this region based on the multiple regression algorithm.

Deep Learning (DL) methods, which can efficiently extract the nonlinear features, are used to investigate the highly nonlinear atmospheric systems such as TC. For example, the recurrent neural network (RNN) (Dorffner, 1996) can effectively extract the temporal features from continuous data, so it has been widely used in TC track forecasting (Dong and Zhang, 2016; Alemany et al., 2018). Kordmahalleh et al. (2016) employed a sparse recurrent neural network based on the dynamic time warping to forecast TC tracks in the Caribbean Sea and indicated this network is particularly suitable for modeling of hurricanes which have complex systems with unknown

dynamics. The dynamic time warping can be used to recognize similar TCs so that the RNN can extract common features. However, this method is not suitable for non-single-track TCs. Alemany et al. (2018) considered all types of TC tracks and used the RNN to forecast them. Unlike traditional methods which directly predict latitudes and longitudes, Alemany et al. (2018) divided the Atlantic Ocean into 1 ° × 1 ° grids and numbered the grid points. The wind speed, latitudes, longitudes, travel angles and TC grid numbers were used as inputs, which can effectively reduce the recursive error transfer caused by direct prediction. The RNN performs better in short-term forecasting but not very good for long-term forecasting. Another important method, long short-term memory neural network (LSTM) (Hochreiter et al., 1997) was developed in 1997. Gao et al. (2018) used the TC best track data to train and optimize the LSTM-based deep neural network (DNN), and the results showed that the LSTM has a better performance in TC track forecasting with the leading time of 6–24 h.

TC Track Forecasting Based on Remote Sensing Images

Forecasting TC tracks using ML methods are not only affected by the characteristics of historical TCs, but also by spatial factors. Compared with time-series data, remote sensing images contain more rich spatial information. Early in 2000, Lee and Liu (2000) proposed a TC automatic identification and track mining system based on the neural network, and the forecast errors of this system were reduced by 30% and 18% compared with the one-way interactive TC model and track forecast system, respectively. Thereafter, Kovordá and Roy , in 2009, extracted Dvorak features from remote sensing images of meteorological satellites and input the data, such as TC locations and maximum wind speed, into the neural network to predict TC tracks. This method improved the forecast accuracy by about 30% compared with the numerical model in Guam. In addition, the neural network represented by the convolutional neural networks (CNN) can effectively extract the spatial features from the data. Later in 2020 a method of fused extracted nonlinear features with latitudes and longitudes of TCs, wind speed and air pressure based on the CNN algorithm. The results indicated that the method better predicted the TC tracks over the Eastern Pacific and the Atlantic Ocean and well retained the TC three-dimensional features. Moreover, this method can forecast the genesis of a TC in a few seconds, which is an important asset for real-time forecasts compared to traditional forecasts.

3. SYSTEM ANALYSIS

3.1. EXISTING SYSTEM

The identification of TCs from satellite images is based on their size, position, status, and intensity . This form of identification, based on pattern recognition Accurately estimating tropical cyclone (TC) intensity is one of the most critical steps in TC forecasting and disaster warning/management. For over 40 years, the Dvorak technique (and several improved versions) has been applied for estimating TC intensity by forecasters worldwide.

3.2 Drawbacks of Existing System

The operational Dvorak techniques primarily used in various agencies have several deficiencies, such as inherent subjectivity leading to inconsistent intensity estimates within various basins.

- Limited Availability of Human Resources
- Can cater to only one person at a time
- It has a lower effectiveness.
- It has the lowest accuracy rate.

3.3 PROPOSED SYSTEM

We propose a deep learning approach for TC identification which includes a Deep Convolutional Neural Network of Alex-Net architecture as the classifier. When the result is of a cyclone, the system sends alert notifications and SMS alerts to the users.

The suggested TC identification technique deploys the pre-trained Deep CNN by taking an satellite image as input and pre-processed by resizing and adding filter. If the classification result indicates that the image is of cyclone, then the result will be displayed as “It is a cyclone” and an alert will be sent to the users as SMS messages.

3.4 Advantages of Proposed System

- Deep learning uses these loads of data and runs faster than human.
- Accurately locate the multiple-cyclone
- Capable of filtering out the low-intensity findings
- More efficient than existing approaches

The following diagram shows how a spiral model acts like:

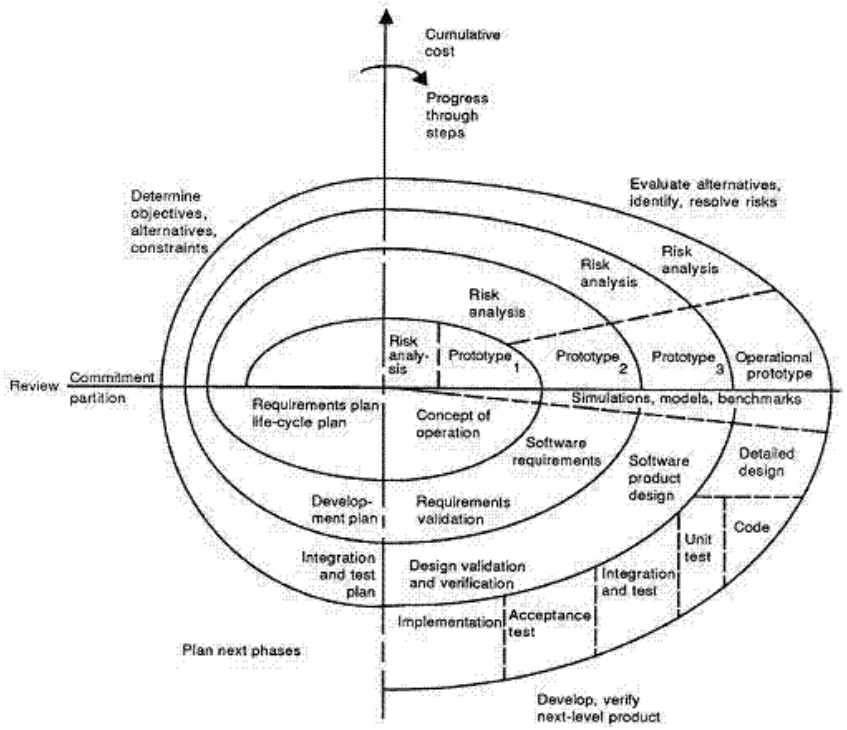

Fig: 3.1 spiral model

Advantages

- Estimates(i.e. budget, schedule etc .) become more relistic as work progresses, because important issues discoved earlier.
- It is more able to cope with the changes that are software development generally entails.
- Software engineers can get their hands in and start woring on the core of a project earlier.

3.5 MODULES

3.5.1 INTRODUCTION

Software Environment:

Python is a programming language. Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently whereas other languages use punctuation, and it has fewer syntactical constructions than other languages.

Fig 6.1 Python Icon

Python Programming Language:

- **Python is Interpreted** - Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** - You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** - Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** - Python is a great language for beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Python Features:

Python’s features include

- **Easy-to-learn** - Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** - Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** - Python's source code is fairly easy-to- maintain.
- **A broad standard library** - Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** - Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** - Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** - you can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** - Python provides interfaces to all major commercial databases.
- **GUI Programming** - Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** - Python provides a better structure and support for large programs than shell scripting.

3.6 Technologies Used:

2.6.1 Tensorflow

Fig 6.2.1 Tensorflow icon

TensorFlow is an open-source software library used for developing and training machine learning models. It was developed by Google Brain Team and released in

2015. TensorFlow has gained significant popularity in the field of machine learning due to its flexibility, scalability, and ease of use.

One of the most significant advantages of TensorFlow is its flexibility. It can be used to develop a wide range of machine learning models, including deep learning, convolutional neural networks, and recurrent neural networks. TensorFlow provides a comprehensive set of tools and libraries that can help developers create models that are specific to their needs. It also allows for customization of models, making it an ideal platform for research and development.

Moreover, TensorFlow is constantly evolving, with new features and updates released regularly. This means that developers can take advantage of the latest advancements in machine learning and artificial intelligence. TensorFlow is also open-source, which means that developers can contribute to its development and improvement. This collaborative approach has resulted in a vibrant community that has created a wide range of resources, including libraries, tools, and tutorials.

In conclusion, TensorFlow is an essential tool in the field of machine learning. Its flexibility, scalability, and user-friendliness make it an ideal platform for developing and training machine learning models. TensorFlow has enabled developers to create innovative solutions that have revolutionized various industries, including healthcare, finance, and transportation. As the field of machine learning continues to evolve, TensorFlow will continue to play a significant role in advancing the capabilities of AI systems.

3.6.2 Keras

Fig 6.2.2 Keras A deep learning library icon

Keras is a popular open source neural network library written in Python. It provides a simple yet powerful interface for building and training deep learning models. Keras is designed to be user-friendly and accessible for beginners while providing

advanced features and flexibility for experienced users. This essay discusses the importance of keras in deep learning.

One of Keras' main advantages is its ease of use. This library provides a high-level API that allows developers to easily build and train deep learning models without worrying about low-level implementation details. Keras also features a modular design, allowing developers to easily swap out different layers and models to create a custom his architecture.

Keras is used in both industry and science for a variety of applications such as image recognition, speech recognition, and natural language processing. Keras is also used in various competitions such as: B. Achieved state-of-the-art performance in the ImageNet Large Scale Visual Recognition Challenge.

In summary, Keras is an important tool in the field of deep learning. Its ease of use, flexibility, and portability make it popular with both beginners and experienced developers. Keras enables developers to build and train deep learning models for various applications, achieving excellence in various competitions. As the demand for deep learning continues to grow in various industries, its use is expected to continue to grow.

3.6.3 Pillow (PIL)

Pillow is a popular open-source Python library that provides a wide range of image processing functionalities. It is a fork of the Python Imaging Library (PIL), which was discontinued in 2011. Pillow is widely used for various image processing tasks, such as image resizing, image filtering, image enhancement, and image manipulation. In this essay, we will discuss the importance of the Pillow library in Python.

One of the key advantages of Pillow is its ease of use. The library provides a simple and intuitive API that makes it easy for developers to manipulate images in Python. The API is well-documented, and the library has a large user community that provides support and guidance to developers.

Pillow provides a wide range of image processing functionalities that can be used to manipulate images in various ways. These include resizing, cropping, flipping, rotating, and adjusting the brightness and contrast of an image. Pillow also provides a

wide range of image filters, such as blur, sharpen, edge detection, and noise reduction. These filters can be used to enhance the quality of images or create artistic effects.

Pillow is widely used in various industries, including digital media, photography, and web development. It is used to create image processing pipelines, generate thumbnails, and manipulate images for web applications. Pillow is also used in scientific research and medical imaging, where it is used to process and analyze images.

3.6.4 OpenCV

OpenCV (Open Source Computer Vision) is a popular open source computer vision and image processing library written in Python. It is widely used in various fields such as robotics, machine learning, and artificial intelligence. This essay discusses the importance of OpenCV in Python.

Fig 6.2.3 OpenCV2 Icon

One of the main advantages of OpenCV is its ability to handle various image and video formats such as JPG, PNG, TIFF and MP4. OpenCV offers a wide range of image processing and computer vision capabilities and algorithms, including image filtering, object detection, motion tracking, and more. These features make OpenCV an essential tool for computer vision and image processing applications.

OpenCV is written in C++, but provides Python bindings that allow developers to use his OpenCV functionality in Python. This makes it easy for developers familiar with Python to use OpenCV without having to learn C++. OpenCV's Python bindings are well documented, and the library has a large user community providing support and guidance to developers.

OpenCV is also cross-platform compatible and works on various operating systems such as Windows, Mac and Linux. This allows developers to easily create applications that work on different platforms without writing platform-specific code.

Another important advantage of OpenCV is its integration with other Python libraries such as NumPy, SciPy and Matplotlib. This makes it easier for developers to combine his OpenCV with other Python libraries for data analysis and visualization. For example, you can use NumPy arrays to store and manipulate images, and Matplotlib to visualize images and data.

In summary, OpenCV is an essential tool in the field of computer vision and image processing in Python. Its ability to handle various image and video formats, cross-platform compatibility, and integration with other Python libraries make it a popular choice among developers. OpenCV makes it easy for developers to create advanced computer vision and image processing applications. Also, as the demand for computer vision and image processing applications continues to grow in various industries, the use of OpenCV is expected to continue to grow.

3.6.5 Numpy

NumPy is a powerful Python library that is used for numerical and scientific computing. It provides a wide range of functionalities for manipulating large arrays and matrices of numeric data. NumPy is a fundamental library for scientific computing in Python and is widely used in academia, research, and industry.

One of the key advantages of NumPy is its speed and efficiency. It is written in C and provides fast and efficient computation for large arrays of data. NumPy also provides a wide range of mathematical and statistical functions, such as linear algebra, Fourier transforms, and random number generation.

NumPy is also compatible with other Python libraries, such as Pandas, SciPy, and Matplotlib, which makes it a versatile tool for data analysis and visualization. Its functionality and compatibility make it an essential tool for data scientists, engineers, and researchers who work with large amounts of numerical data in Python.

3.6.6 Sklearn

Fig 6.2.4 Sci-Kit-learn library icon

scikit-learn (sklearn) is a popular machine learning library in Python that provides a wide range of tools for data preprocessing, function selection, model selection, and model evaluation. One of the key features of scikit-learn is the ability to split the data into training and test sets using the train_test_split() function.

The train_test_split() function is used to split the dataset into two parts, a training set and a test set. The training set is used to train the machine learning model and the test set is used to evaluate the model's performance. The train_test_split() function randomly splits the dataset into two parts, with each set assigned a certain proportion of data points.

The train_test_split() function is important as it helps prevent overfitting of the machine learning model. Overfitting occurs when a machine learning model performs well on the training data set but poorly on the test data set. This can happen when the model memorizes the training set instead of learning the underlying patterns. By splitting the dataset into a training set and a test set, we can evaluate the model's performance on unseen data and generalize well. The train_test_split() function is also important as it helps optimize the hyperparameters of your machine learning model. Hyperparameters are parameters that are set before training a model, such as: B. Number of hidden layers for neural networks, or regularization strength for linear regression models. By splitting the dataset into a training set and a test set, you can evaluate model performance with different hyperparameters and choose the best set of hyperparameters.

The train_test_split() function is easy to use and can be applied to all types of machine learning problems such as classification, regression and clustering. This

function accepts several arguments such as the data set, the proportion of data points assigned to the training set, and the random seed used for randomization. This function returns four arrays: training data, test data, training labels, and test labels.

In summary, the train_test_split() function is an essential tool in scikit-learn for developing and evaluating machine learning models. It helps prevent overfitting, optimize hyperparameters, and ensure generalization of machine learning models. This feature is easy to use and applicable to all types of machine learning problems, making it a powerful tool for machine learning practitioners.

3.6.7 Geopy

Geopy is an open-source Python library that provides geocoding and geolocation functionalities. It allows developers to access various geocoding services, such as Google Maps, Bing Maps, and OpenStreetMap, to obtain latitude and longitude coordinates for specific addresses or locations. Geopy also provides distance and proximity calculations between different locations.

Fig 6.2.5 GeoPy library icon

The library is easy to use and has a simple API that makes it straightforward for developers to integrate geolocation and geocoding functionalities into their applications. Geopy is widely used in various industries, such as logistics, transportation, and healthcare, where accurate geolocation data is essential.

In addition to geocoding and geolocation functionalities, Geopy also provides support for various geographic coordinate systems, such as WGS84 and UTM. This makes it a versatile tool that can handle a wide range of geospatial data. Geopy is an important library for developers who need to work with geolocation and geocoding data in Python.

3.6.8 Tkinter

Tkinter is a popular open source Python library that provides a graphical user interface (GUI) for Python applications. This is the standard Python library that comes with most Python installations and is commonly used for building desktop applications. This essay discusses the importance of Tkinter in Python.

One of Tkinter's main advantages is its ease of use. This library provides a simple and intuitive API that allows a developer to easily create his GUI in Python. The API is well documented and the library has a large user community providing support and guidance to developers.

Tkinter provides a wide range of widgets that can be used to create her GUI such as buttons, labels, textboxes, menus, listboxes. These widgets can be easily customized and combined to create complex GUIs. Tkinter also provides support for event-driven programming, allowing developers to write code that responds to user input such as button clicks and menu selections.

Tkinter also provides theme support so developers can change the look and feel of her GUI. This is useful for creating applications that match the branding or style of a particular organization or project. Tkinter is widely used in various industries such as education, finance, and healthcare. It is used to create desktop applications such as text editors, spreadsheet applications, and image editors. Tkinter is also used in scientific research, where it is used to create data visualization tools and control laboratory equipment.

2.6.9 Twilio

Fig 6.2.6 Twilio API icon

Twilio is a cloud communications platform that provides a set of APIs and tools for building voice, video, and messaging applications. The Twilio Python package

allows developers to use the Twilio API to send and receive text messages, make and receive phone calls, and create video chats and conferencing applications.

The Twilio package in Python is designed to be easy to use and allows developers to quickly integrate Twilio's communication features into their applications. The package provides a set of Python classes that correspond to Twilio's API resources, such as the Message, Call, and Conference resources. Developers can use these classes to send and receive messages, make and receive phone calls, and manage conference calls.

One of the key features of the Twilio package is its ability to handle asynchronous requests using callbacks or async/await syntax. This allows developers to build responsive applications that can handle multiple requests simultaneously without blocking the main thread.

In addition to its core functionality, the Twilio package also provides several helper libraries that make it easy to integrate with popular Python frameworks such as Flask, Django, and Pyramid. These libraries provide pre-built integrations that allow developers to quickly add Twilio functionality to their web applications.

4. FEASIBILITY STUDY

The preliminary investigation examines project feasibility, the likelihood the system are going to be useful to the organization. The main objective of the feasibility study is to check the Technical, Operational, and Economical feasibility for adding new modules and debugging old running systems. All systems are possible if they have unlimited resources and infinite time to do a task. There are aspects within the feasibility study portion of the preliminary investigation:

- Economic Feasibility
- Technical Feasibility
- Operational Feasibility

4.1 Economic Feasibility

As system are often developed technically which are going to be used if installed must still be an honest investment for the organization. In the economic feasibility, the event cost in creating the system is evaluated against the last word benefit derived from the new systems. Financial benefits must equal or exceed the costs. The system is economically feasible. It doesn't require any additional hardware or software. Since the interface for this system is developed using the existing resources and technologies java1.6 open source, there is nominal expenditure and economic feasibility for certain.

4.2 Technical Feasibility

This assessment focuses on the technical resources available to the organization. It helps organizations determine whether the technical resources meet capacity and whether the technical team can convert the ideas into working systems. Technical feasibility also involves evaluation of the hardware, software, and other technology requirements of the proposed system. This assessment is predicated on an overview design of system requirements, to work out whether the corporate has the technical expertise to handle completion of the project.

When writing a feasibility report, the subsequent should be taken to consideration.

- A brief description of the business to assess more possible factors which could affect the study 23
- The part of the business being examined

• The human and economic factor

• The possible solutions to the problem at this level, the concern is whether the proposal is both technically and legally feasible (assuming moderate cost).

The technical feasibility assessment is focused on gaining an understanding of the present technical resources of the organization and their applicability to the expected needs of the proposed system. It is an evaluation of the hardware and software and how it meets the need of the proposed system.

4..3 Operational Feasibility

Proposed projects are beneficial only if they can be turned out into information system. That will meet the organization's operating requirements. Operational feasibility aspects of the project are to be taken as a crucial a part of the project implementation. Some of the important issues raised are to check the operational feasibility of a project includes the following.

- Is there sufficient support for the management from the users?
- Will the system be used and work properly if it is being developed and implemented?
- Will there be any resistance from the user that will undermine the possible application benefits?

This system is targeted to be in accordance with the above-mentioned issues. Beforehand, the management issues and user requirements have been taken into consideration. So, there is no question of resistance from the users that can undermine the possible application benefits. The well-planned design would ensure the optimal utilization of the computer resources and would help in the improvement of performance status.

5. SOFTWARE REQUIREMENT SPECIFICATIONS

5.1 Functional Requirements

The Functional Requirements of the system are as follows

1. Read the input satellite image.
2. Must Pre-process the Input Image.
3. Feature Extraction and Deep CNN Classification.
4. Alert System send alert notifications/SMS messages.

Input the satellite images:

The system should be able to take input from the users.

Image Pre-processing:

The system must be able to apply filters to change the satellite images to Grayscale.

Feature Extraction and Deep CNN Classification

The system should be able to extract features and classify whether if the image contains cyclone or not

Alert System send alert notifications/SMS messages

The system must be able to send SMS messages if the classified class of input image is cyclone**.**

5.2 Software Requirement Specifications

- **Operating System** Windows 8 or above
- **Platform** VSCode or Google Colab
- **Language** Python 3.6 or above
- **Tools** PIP
- **Libraries** TensorFlow, Keras, PIL, Open CV2, Webbrowser, PyPDF2, Tkinter, Geopy, Twilio

5.3Hardware Requirement Specifications

- **Hard Disk** 256Gb or above
- **RAM** 4GB or above
- **Processor** i3 or above

6. SYSTEM DESIGN

6.1 Design Tool Used – Visual Paradigm Tool

Introduction

Visual Paradigm (VP-UML) is a UML CASE Tool supporting UML 2, SysML and Business Process Modeling Notation (BPMN) from the Object Management Group (OMG). In addition to the modeling support, it provides report generation and code engineering capabilities including code generation. It can reverse engineer diagrams from code, and provide round-trip engineering for various programming languages.

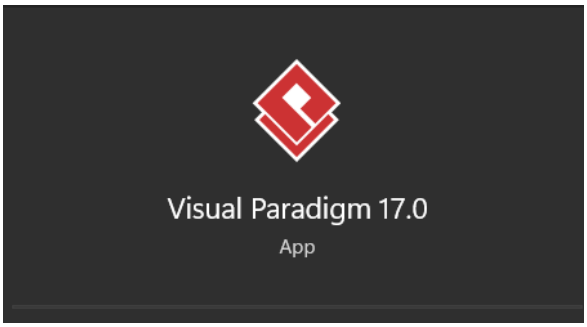

Fig 5.2 Visual Paradigm in Windows 11 Start Menu

Product Editions

Higher-priced editions provide more features. The following editions were available:

- Community Edition
- Modeler Edition
- Standard Edition
- Professional Edition
- Enterprise Edition

UML Modeling

Visual Paradigm supports 13 types of diagrams:

- Class Diagram
- Use Case Diagram
- Sequence Diagram
- Communication Diagram
- State Machine Diagram
- Activity Diagram
- Component Diagram
- Deployment Diagram
- Package Diagram
- Object Diagram
- Composite Structure Diagram
- Profile Diagram
- Timing Diagram
- Interaction Overview Diagram

Requirements Management

- Visual Paradigm supports requirements management including user stories, use cases, SysML requirement diagrams and textual analysis.
- A SysML requirement diagram specifies the capability or condition that must be delivered in the target system. Capability refers to the functions that the system must support. Condition means that the system should be able to run or produce the result given a specific constraint. Visual Paradigm provides a SysML requirement diagram for specifying and analyzing requirements

Business Modeling

Supports BPMN 2.0 for modeling of business processes. The latest version (Aug 2016) also supports Case Management with CMMN

Data Modeling

Visual Paradigm supports both Entity Relationship Diagrams (ERD) and Object Relational Mapping Diagrams (ORMD). ERD is used to model the relational database. ORMD is one of the tools to show the mapping between class from object-oriented world and entity in relational database world.

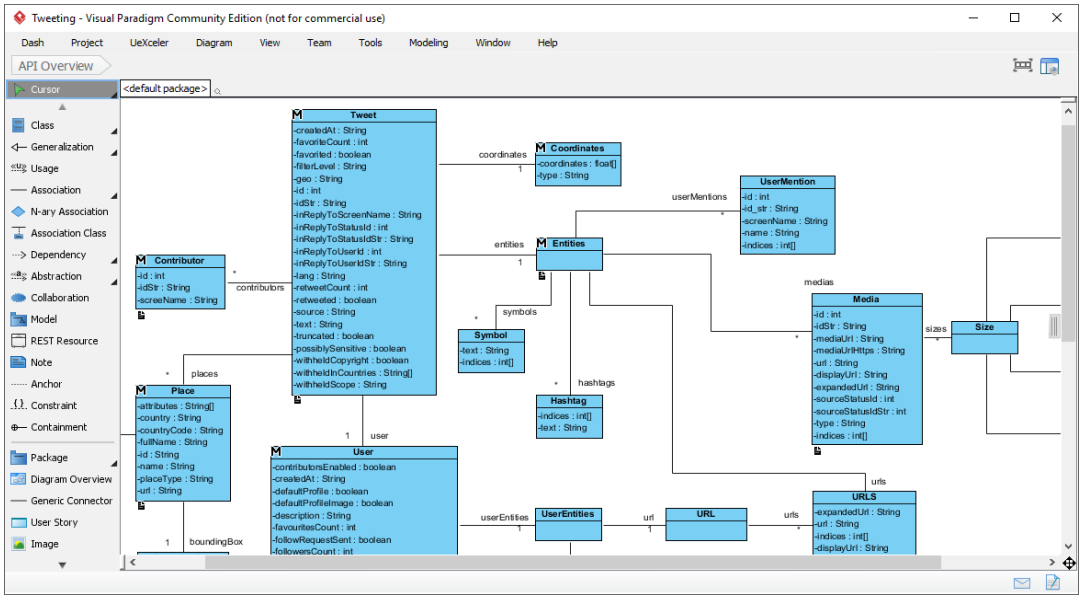

Fig 5.3 Visual Paradigm 17.0 running on Windows 11

Developer(s)	Visual Paradigm International Ltd.
Initial Release	20 June 2002; 15 years ago
Stable Release	17.0 / August 01, 2022
License	Proprietary with Free Community Edition

6.2 UML DIAGRAMS

6.2.1 USECASE DIAGRAM :

It shows the set of use cases, actors & their relationships. In our project we have 2 actors as the user and admin interacting with the chatbot system. The use cases shown are asking query, which could be course related, admissions related, training the model and getting response

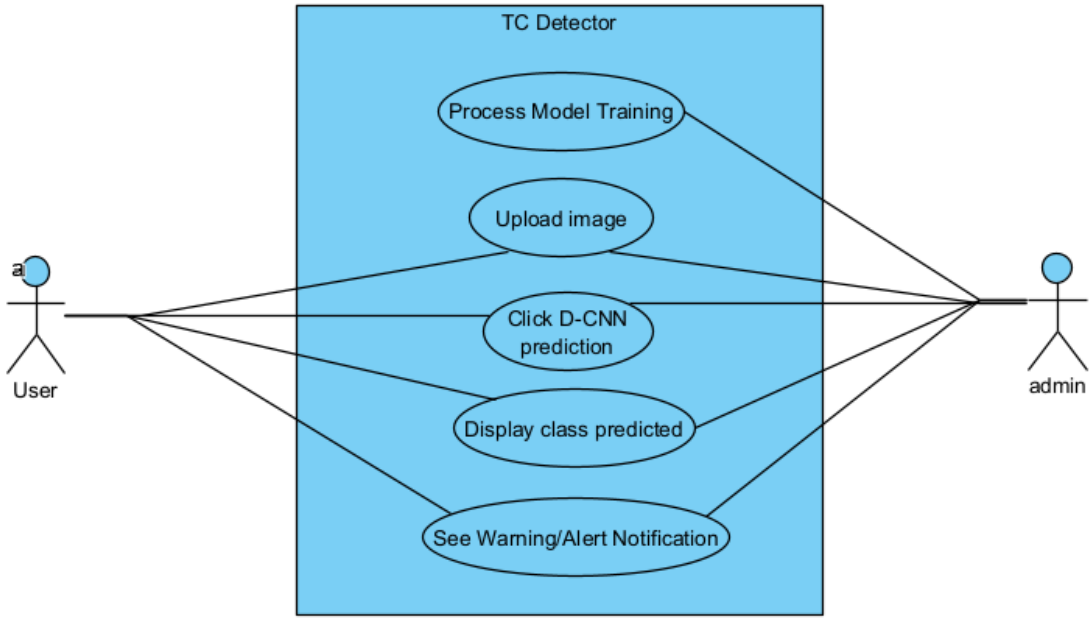

Fig 5.4 Use Case diagram

6.2.2 CLASS DIAGRAM:

A class diagram shows a set of classes, interfaces, and collaborations and their relationships. These diagrams are the most common diagram found in modeling object-oriented systems. Class diagrams address the static design view of a system. Class diagrams that include active classes address the static process view of a system.

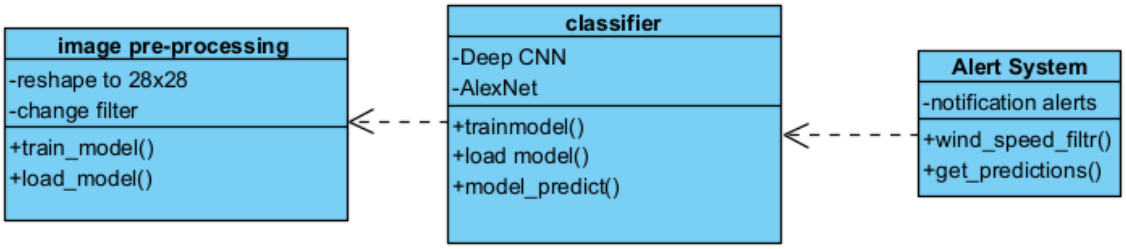

Fig 5.5 Class Diagram

6.2.3 SEQUENCE DIAGRAM:

A *sequence diagram* is an interaction diagram that emphasizes the time-ordering of messages; a *collaboration diagram* is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages. Sequence diagrams and collaboration diagrams are isomorphic, meaning that you can take one and transform it into the other.

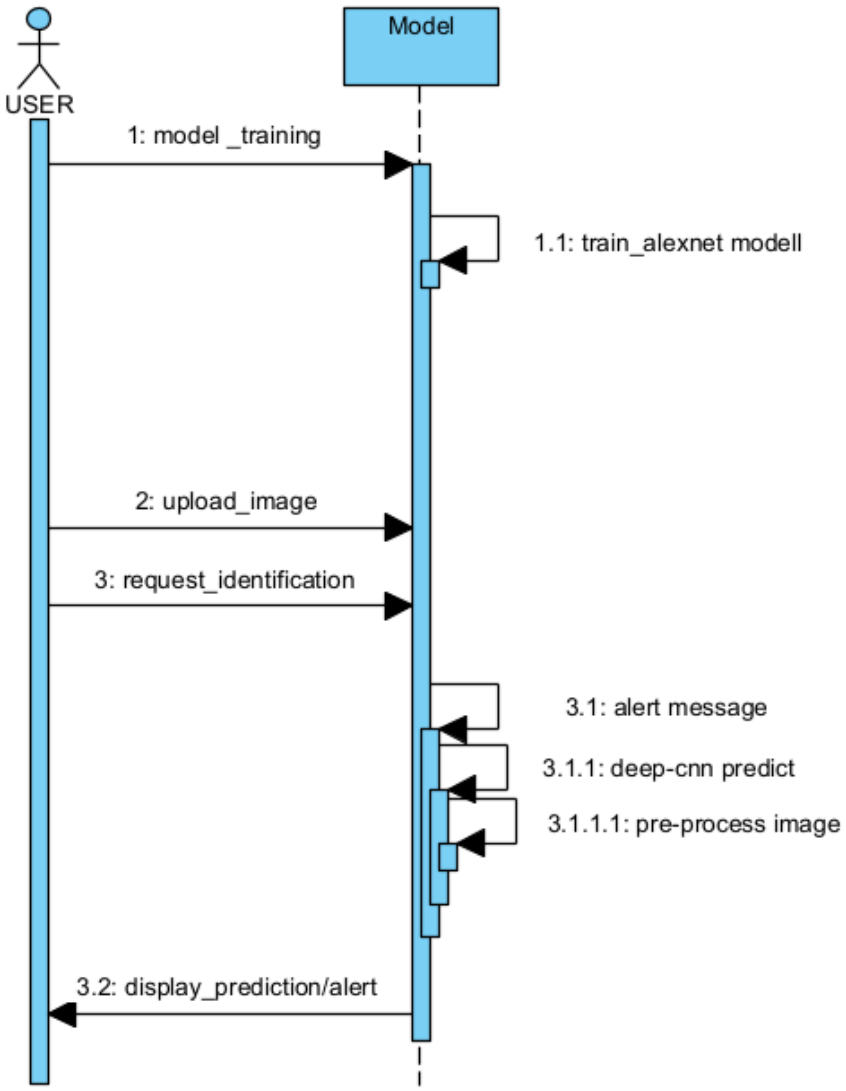

Fig 5.6 Sequence Diagram

6.2.4 ACTIVITY DIAGRAM:

An *activity diagram* is a special kind of a state chart diagram that shows the flow from activity to activity within a system. Activity diagrams address the dynamic view of a system. They are especially important in modeling the function of a system and emphasize the flow of control among objects.

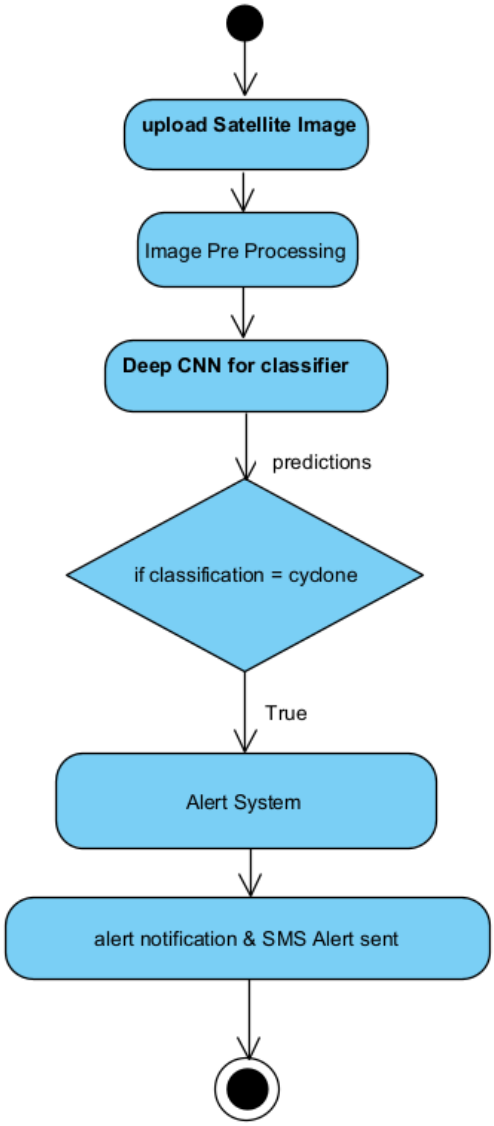

Fig 5.7 Activity Diagram

7. IMPLEMENTATION

7.1 SYSTEM ARCHITECTURE

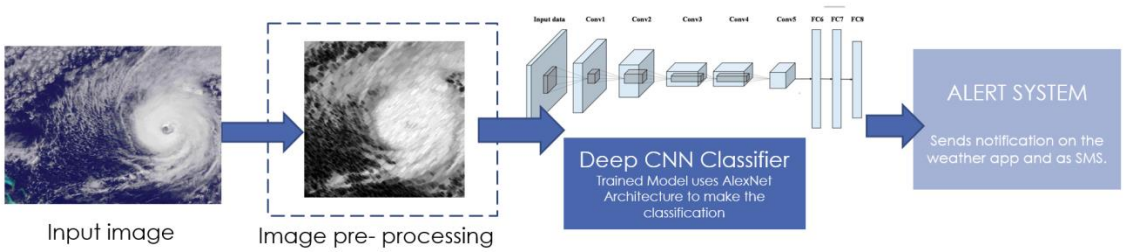

Fig5.1: System Architecture

The system architecture shows a model and the overall workflow of the project. Here, we use the satellite images of the cyclones which are passed to the detector consisting Mask R-CNN layer. The detections obtained from the input image are recorded in detectron2's output format. If more than one bounding boxes are detected, the wind velocity of the timestamp is checked. If the wind speed is more tan 34 knots, it is saved and if less than 34 knots, the image is discarded. If there are more than one prediction made for an image, the classifier of DenseNet is then provided images cropped from the input image using bounding boxes. The image with highest confidence score is chosen as the correct prediction.

7.2 Algorithm

1. Image Pre-processing
2. Feature Extraction & Deep CNN Classification
3. Alert System

7.2.1 Image Pre-processing:

One of the key steps in this process is image pre-processing, which involves transforming raw satellite images into a format that can be used for further analysis. In this context, the images are typically reshaped to 28x28x1 and changed to black and white (B&W) using Python packages such as PIL, OpenCV2, TensorFlow, and Keras.

Reshaping the images to 28x28x1 involves resizing the images to a smaller size to reduce the computational load and to make them easier to handle. The images are typically first converted to grayscale, which reduces the amount of data that needs to be processed while retaining the necessary information for cyclone classification. A grayscale image contains only one channel (i.e., black, white, and various shades of gray), which is represented by a 2D matrix. However, for the purpose of deep learning-based cyclone classification, a 3D matrix is required, which can be achieved by adding a third dimension to the grayscale image.

Another package commonly used for image pre-processing in cyclone classification is OpenCV2, which is an open-source computer vision library that provides a range of image processing functions. OpenCV2 provides several methods for converting images to B&W, including thresholding, which involves setting all pixels above a certain threshold value to white and all pixels below that value to black. Other methods include adaptive thresholding, which adjusts the threshold value based on the local image intensity, and binary thresholding, which converts the image to B&W by setting all pixels above a certain threshold value to white and all pixels below that value to black.

Finally, TensorFlow and Keras are popular Python packages for deep learning-based image classification, and they provide various functions for image pre-processing. These packages allow images to be normalized, augmented, and converted to B&W using a range of functions and methods. For example, the ImageDataGenerator class in Keras provides a range of image pre-processing functions, including rescale, which normalizes pixel values, and color_mode, which allows images to be converted to B&W by setting the value to grayscale.

7.2.2 Feature Extraction & D-CNN Classification:

7.2.2.1 Deep CNN

A deep convolutional neural network (CNN) is a type of neural network used for image recognition and processing tasks. CNNs are designed to automatically identify features in input images and classify them accordingly using multi-level convolution, pooling, and fully connected planes.

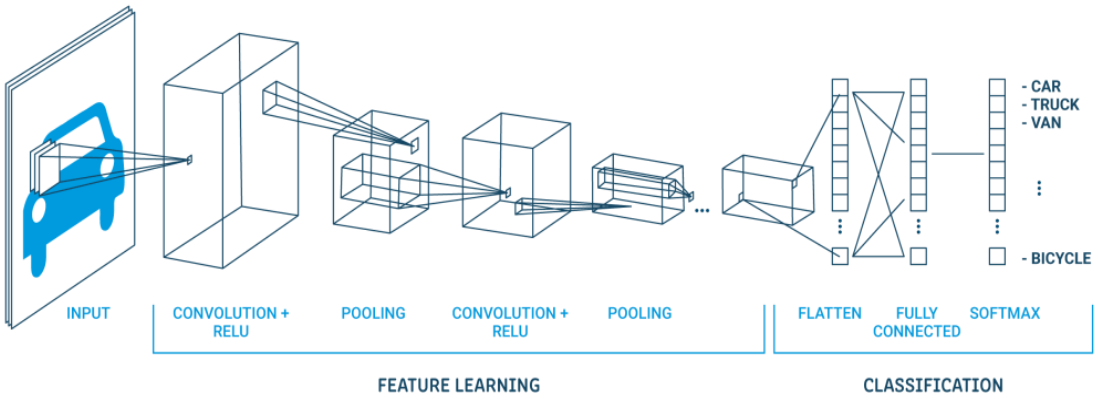

Fig: 6.4.1 Deep Convolutional Neural Networks

A convolutional layer is the core building block of a CNN and consists of multiple filters that perform convolutional operations on the input image. Each filter is responsible for detecting specific features or patterns such as edges and corners. The output of each convolutional layer is a set of feature maps that capture the presence of these features in the input image. Pooling layers are used to reduce the spatial size of feature maps while preserving important information. The most common pooling operation is Max Pooling, which selects the maximum value of each pooling window as output.

At the end of the CNN, a fully connected layer is used to classify the input image into different categories. These layers take the flattened feature map as input and use the weights and biases to output the final probabilities for each category.

The key features of deep Convolutional Neural Networks (CNNs) include the ability to automatically learn hierarchical representations of the input data, the use of convolutional layers to extract local features, pooling layers to reduce spatial dimensions, and the use of non-linear activation functions such as ReLU. They can handle high-dimensional data such as images and audio signals, and are trained with large datasets using stochastic gradient descent and backpropagation algorithms.

CNNs also employ regularization techniques such as dropout and data augmentation to prevent overfitting. With their ability to model complex non-linear relationships, deep CNNs have become a powerful tool for solving a wide range of machine learning problems.

Deep Convolutional Neural Networks (CNNs) have a wide range of applications in computer vision, including image classification, object detection, image segmentation, and image synthesis. They are also used in natural language processing (NLP) tasks such as sentiment analysis, machine translation, and text classification.

In addition, CNNs have been applied in various fields such as medicine, autonomous driving, robotics, and security systems. With their ability to learn and extract features from large datasets, CNNs have become a powerful tool for solving complex problems and making significant contributions to the advancement of artificial intelligence.

7.2.2.2 AlexNet CNN

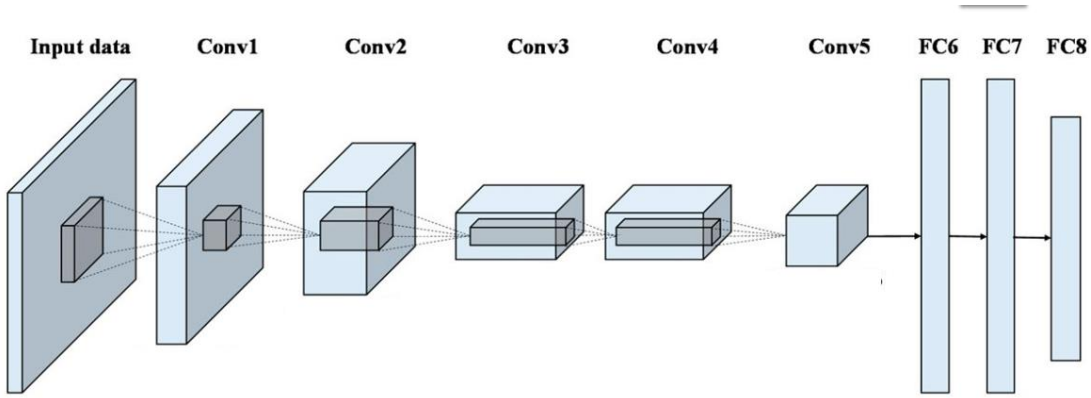

Fig 6.4.2 AlexNet Architecture

AlexNet, a deep convolutional neural network (CNN) architecture designed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton in 2012, asked participants to place images in one of 1,000 categories. AlexNet was able to achieve state-of-the-art results on this task and played an important role in advancing computer vision.

The AlexNet architecture consists of 5 convolutional layers and 3 fully connected layers. The convolution layer is followed by a max pooling layer that downsamples the feature map to reduce the spatial dimensionality of the input. The final layer is a softmax layer that generates probability distributions over 1,000 categories.

The key features of the AlexNet CNN architecture include the use of ReLU activation functions, overlapping max-pooling layers, local response normalization, data augmentation techniques, and dropout regularization. It has five convolutional layers, three fully connected layers, and a softmax output layer for classification. The network

was trained on a large dataset of 1.2 million images, allowing it to learn rich representations of the input data. AlexNet's innovative use of these features enabled it to achieve state-of-the-art performance on the ImageNet dataset, marking a significant breakthrough in the field of computer vision and paving the way for further advancements in deep learning.

One of AlexNet's key innovations is the use of Rectified Linear Units (ReLU) as the activation function. ReLU is a simple and efficient activation function that overcomes the vanishing gradient problem present in previous CNN architectures. Another important feature of AlexNet is using data augmentation techniques such as clipping and mirroring to increase the size of the training data set and reduce overfitting.

AlexNet was trained on a dataset of 1.2 million images, an order of magnitude larger than the dataset used in previous CNN architectures. This allowed AlexNet to learn rich representations of the input images and increase the fidelity of the ImageNet dataset.

7.2.3 Using AlexNet & Deep CNN

AlexNet is a popular deep learning model used for feature extraction. It consists of eight convolutional layers, followed by three fully connected layers, and was first introduced by Alex Krizhevsky in 2012. The model is trained on a large dataset of images and is able to learn complex features, such as edges, curves, and textures, from the input images. The features learned by AlexNet can then be used as input for other models, such as deep CNNs, to perform tasks such as object detection, classification, and segmentation. AlexNet has been widely used in computer vision applications and has been shown to be highly effective in feature extraction tasks

Deep CNNs with AlexNet architecture are commonly used for binary classification tasks in computer vision. In this approach, the AlexNet model is used for feature extraction from input images, followed by a set of fully connected layers to perform binary classification. The extracted features from the AlexNet model are passed through a series of convolutional and pooling layers, which help to reduce the dimensionality of the input data and capture local patterns and features. The final fully connected layers are then used for classification, where the model learns to map the

extracted features to the corresponding output label. This approach has shown high accuracy in binary classification tasks, including cyclone classification, where it has been used to detect cyclones of different intensities with high precision.

The model we have already trained has a total of 13 layers, including five convolutional layers, three max pooling layers, and three fully connected layers. The first convolutional layer has 96 filters of size 11x11, with a stride of 4 and a ReLU activation function. The second convolutional layer has 256 filters of size 5x5, with a stride of 1 and a ReLU activation function. The next three convolutional layers have 384, 384, and 256 filters of size 3x3, respectively, with a stride of 1 and a ReLU activation function. The max pooling layers have varying pool sizes and strides. The first two have a pool size of 2x2 and a stride of 2, while the last one has a pool size of 1x1 and a stride of 2. The fully connected layers have 4096 neurons each, with a ReLU activation function and a dropout rate of 0.5. The output layer has 2 neurons, with a softmax activation function for multi-class classification.essential. Images can be segmented for the purpose of classifying image pixels.

7.4.3 Alert System:

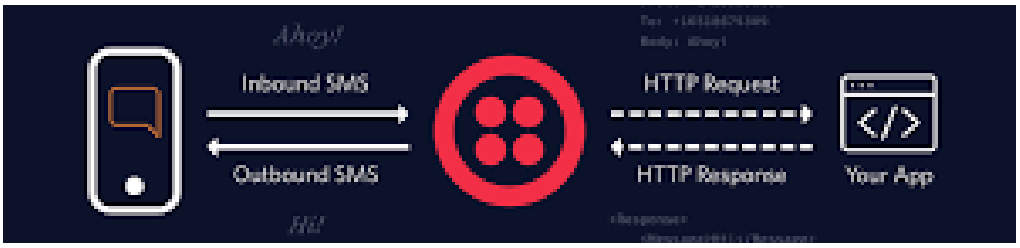

Fig 6.4.3 SMS messages in Python with Twilio

To build a cyclone alert system through SMS, we can use a combination of a classification model to detect the cyclone from input images and the Twilio API to send SMS alerts to designated recipients.

Firstly, we integrate it with our alert system.Next, we need to set up a Twilio account and obtain the necessary credentials, including the Account SID, Auth Token, and a Twilio phone number. We can use the Twilio API for Python to send SMS messages to designated phone numbers.

In the alert system, when an input image is classified as a cyclone, the system will trigger the Twilio API to send SMS alerts to designated recipients with a predefined message. The message can include information about the cyclone.

To implement this system, we wrote a Python script that integrates the classification model and Twilio API. The script should first load the trained model and define the message to be sent in case of a cyclone alert. Then, when a new input image is processed, the script will classify it using the model. If the image is classified as a cyclone, the script will send an SMS alert to designated recipients using the Twilio API.

This cyclone alert system using SMS alerts through Twilio in Python provides a quick and efficient way to inform people of the potential danger of a cyclone, enabling them to take necessary safety measures.

7.3 CODE

```
# -*- coding: utf-8 -*-
"""ri_sp (1).ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1RsPzUeVzUIZeY_z07s-
KdfdoyxHRSqJN
"""

import copy
import numpy as np
import sklearn
import pandas as pd
import matplotlib.pyplot as plt
# from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Activation
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Bidirectional
from tensorflow.keras.layers import RepeatVector
from tensorflow.keras.layers import TimeDistributed
from numpy import hstack
from sklearn.preprocessing import StandardScaler
import datetime
```

```
import time
import joblib
from datetime import timedelta, date
import tensorflow as tf
from sklearn.preprocessing import MinMaxScaler
from numpy import array
from tf.keras.utils import to_categorical
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import f1_score
import matplotlib.pyplot as plt
import os
import seaborn as sns; sns.set_theme()
import errno
from tf.keras.layers.convolutional import Conv1D
from tf.keras.layers.convolutional import MaxPooling1D
from tf.keras.layers import Flatten
from tf.keras.layers import ConvLSTM2D
from tf.keras.models import load_model
import pickle
from sklearn.metrics import accuracy_score, roc_curve, auc,
classification_report, confusion_matrix
from scipy import interp
from imblearn.over_sampling import SMOTE
from collections import Counter
import imblearn
import collections
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from matplotlib import pyplot

def load_data_south_indian(url):
  df = pd.read_csv(url)
  #df.columns = ['id','date','longitude','latitude','speed']
  #df = df.drop(['date'], axis = 1)
  df['category'] = df['Speed(knots)'].apply(lambda x:
  0 if x<=33 else 1  if x<=47 and x>=34 else 2 if x<=63 and x>=48 else 3
if x<=89 and x>=64 else 4 if x<=115 and x>=90 else 5 )
  return df

def load_data_south_pacific(url):
  df = pd.read_csv(url)
  #df.columns = ['id','date','longitude','latitude','speed']
  #df = df.drop(['date'], axis = 1)
  df['category'] = df['Speed(knots)'].apply(lambda x:
```

```
  0 if x<=33 else 1  if x<=47 and x>=34 else 2 if x<=63 and x>=48 else 3
if x<=85 and x>=64 else 4 if x<=107 and x>=86 else 5 )
  return df

ocean = 'south_pacific'  #south_indian or south_pacific
print(ocean)

if ocean == 'south_indian':
    url_data = 'https://raw.githubusercontent.com/sydney-machine-
learning/cyclonedatasets/main/SouthIndian-SouthPacific-
Ocean/South_indian_hurricane.csv'
    function = load_data_south_indian
    hot_encoded_result_file_name = 'south_indian'
    category_result_file_name = 'roc_data_south_indian'
else:
    url_data = 'https://raw.githubusercontent.com/sydney-machine-
learning/cyclonedatasets/main/SouthIndian-SouthPacific-
Ocean/South_pacific_hurricane.csv'
    function = load_data_south_pacific
    hot_encoded_result_file_name = 'south_pacific'
    category_result_file_name = 'roc_data_south_pacific'

df = function(url_data)
speed = df['Speed(knots)'].tolist()
categories = df['category'].tolist()
df.head()

def split_sequence(sequences, n_steps_in, n_steps_out):
    X, y = list(), list()
    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps_in
        out_end_ix = end_ix + n_steps_out
        # check if we are beyond the dataset
        if out_end_ix > len(sequences):
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[i:end_ix, :],
sequences[end_ix:out_end_ix, -1 ]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# split a univariate sequence into samples
def uni_split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
```

```
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

def rmse(pred, actual):
    return np.sqrt(((pred-actual) ** 2).mean())

def categorical(pred, actual):
  cm = confusion_matrix(pred,actual)
  ps = precision_score(pred,actual,average='micro')
  rs = recall_score(pred,actual,average='micro')
  f1 = f1_score(pred,actual, average = 'micro')
  return cm,ps,rs,f1

def make_confusion_matrix_chart(cf_matrix_test):
    plt.figure(1, figsize=(10,5))
    sns.heatmap(cf_matrix_test, annot=True, yticklabels=['0','1'],
                                xticklabels=['0','1'], fmt='g')
    plt.ylabel("Actual")
    plt.xlabel("Pred")
    plt.title('Test data')
    return None

univariate = True # if false, its multivariate case
n_steps_in = 4
n_seq = 2
n_steps_out = 1
n_features_in = 1 #speed
n_features_out = 2 # one hot encoding of category
Hidden = 10
Epochs = 100
Num_Exp = 30

id = df['No. of Cycl'][0]
count = 0
X = []
Y = []
start_index=0
end_index=0
for i in range(1, df.shape[0]):
```

```
  if df['No. of Cycl'][i] == id :
    end_index+=1
  else:
    x,y = uni_split_sequence(speed[start_index:end_index+1], n_steps_in)
    X.append(x)
    Y.append(y)
    id = df['No. of Cycl'][i]
    start_index=i
    end_index=i
  if i == df.shape[0]-1:
    x,y = uni_split_sequence(speed[start_index:end_index+1], n_steps_in)
    X.append(x)
    Y.append(y)

print(len(X), len(Y))
X = [item for sublist in X for item in sublist]
Y = [item for sublist in Y for item in sublist]
print(len(X), len(Y))
print(X[0], Y[0], X[1], Y[1])
print(speed[:10])

intensify_y = []
for i in range(len(X)):
  if Y[i]-X[i][0]>=30:
    intensify_y.append(1)
  else:
    intensify_y.append(0)
print(len(intensify_y))
speed_y = Y
Y=intensify_y

train_limit = int(len(X)*70/100)
train_limit

test_X_original = X[train_limit+1:]
#X_original = X[:train_limit]
#X_original = np.asarray(X_original).astype(float)
test_Y_original = Y[train_limit+1:]
#Y_original = Y[:train_limit]
len(X), len(Y), len(test_X_original), len(test_Y_original)

X = MinMaxScaler().fit_transform(np.asarray(X))

speed_x = X
test_X = X[train_limit+1:]
test_X = np.asarray(test_X).astype(float)
test_Y = Y[train_limit+1:]
```

```
X = X[:train_limit]
X = np.asarray(X).astype(float)
Y = Y[:train_limit]
print(len(test_X), len(test_Y))
len(X), len(Y)

counter_train = Counter(Y)
counter_test = Counter(test_Y)
#counter = collections.OrderedDict(sorted(counter.items()))
print("train data: ", counter_train)
print("test data: ", counter_test)

plt.figure()
#fig = plt.subplots(figsize =(20, 12))
fig, ax = plt.subplots(figsize = (12,8))
plt.bar(range(len(counter_train)), list(counter_train.values()),
align='center')
plt.xticks(range(len(counter_train)), list(counter_train.keys()))
plt.xlabel('Category', size=40)
plt.ylabel('Number', size=40)
#plt.title('ROC' + ' (' + str(no_of_output_steps) + ' steps ahead, ' +
model.capitalize() + ', Time step: ' + str(time_step) + ') - ' +
train_or_test.capitalize() + ' (' + ocean + '_ocean)')
ax.tick_params(axis='both', which='major', labelsize=30)
plt.savefig(ocean + '_class_dist.png', dpi=300, transparent=False,
bbox_inches='tight')
plt.show()

def vanilla(n_steps_in,n_steps_out,n_features_in, n_features_out):
  model = Sequential()
  model.add(LSTM(50, activation='relu', input_shape=(n_steps_in,
n_features_in)))
  model.add(Dense(n_features_out, activation = "softmax"))
  model.compile(optimizer='adam', loss='binary_crossentropy')
  return model

  #model = Sequential()
  #model.add(LSTM(Hidden, activation='relu', input_shape=(n_steps_in,
n_features_in)))
  #model.add(RepeatVector(n_steps_out))
  #model.add(LSTM(Hidden, activation='relu', return_sequences=True))
  #model.add(TimeDistributed(Dense(n_features_out, activation =
"softmax")))
  #model.compile(optimizer='adam', loss='categorical_crossentropy')
  #return model

def bidirectional(n_steps_in,n_steps_out,n_features_in, n_features_out):
```

```
  model = Sequential()
  model.add(Bidirectional(LSTM(Hidden, activation='relu',
input_shape=(n_steps_in, n_features_in))))
  model.add(Dense(n_features_out, activation = "softmax"))
  model.compile(optimizer='adam', loss='binary_crossentropy')
  return model

def cnn_lstm(n_steps_in,n_steps_out,n_features_in, n_features_out,
n_seq):
  model = Sequential()
  model.add(TimeDistributed(Conv1D(filters=64, kernel_size=1,
activation='relu'), input_shape=(None, int(n_steps_in/n_seq),
n_features_in)))
  model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
  model.add(TimeDistributed(Flatten()))
  model.add(LSTM(Hidden, activation='relu'))
  model.add(Dense(n_features_out, activation = "softmax"))
  model.compile(optimizer='adam', loss='binary_crossentropy')
  return model

def conv_lstm(n_steps_in,n_steps_out,n_features_in, n_features_out,
n_seq):
  model = Sequential()
  model.add(ConvLSTM2D(filters=64, kernel_size=(1,2), activation='relu',
input_shape=(n_seq, 1, int(n_steps_in/n_seq), n_features_in)))
  model.add(Flatten())
  model.add(Dense(n_features_out, activation = "softmax"))
  model.compile(optimizer='adam', loss='binary_crossentropy')
  return model

#all models
def MODEL_LSTM(model_name, method, univariate, x_train, x_test, y_train,
y_test, Num_Exp, n_steps_in, n_steps_out, Epochs, Hidden):

    train_acc = np.zeros(Num_Exp)
    test_acc = np.zeros(Num_Exp)

    if model_name == 'vanilla':
      model = vanilla(n_steps_in,n_steps_out,n_features_in,
n_features_out)
    elif model_name == 'bidirectional':
      model = bidirectional(n_steps_in,n_steps_out,n_features_in,
n_features_out)
    elif model_name == 'cnn-lstm':
      model =
cnn_lstm(n_steps_in,n_steps_out,n_features_in,n_features_out,n_seq)
    elif model_name == 'conv-lstm':
```

```
      model =
conv_lstm(n_steps_in,n_steps_out,n_features_in,n_features_out,n_seq)

    model.summary()

    y_predicttest_allruns = np.zeros([Num_Exp, x_test.shape[0],
y_test.shape[1]])

    #print(y_predicttest_allruns.shape, ' shape ')


    Best_f1 = 0  # Assigning a small number
    act_test = [y_test[i].argmax() for i in range(y_test.shape[0])]
    act_train = [y_train[i].argmax() for i in range(y_train.shape[0])]
    start_time = time.time()
    Best_report_train = dict()
    Best_report_test = dict()
    all_report_train=dict()
    all_report_test=dict()
    for run in range(Num_Exp):
        print("Experiment", run + 1, "in progress")
        # fit model
        model.fit(x_train, y_train, epochs=Epochs, batch_size=10,
verbose=0, shuffle=False)
        #scores = model.predict_proba(x_test)
        y_predicttrain = model.predict(x_train)
        y_predicttest = model.predict(x_test)
        #y_predicttest_allruns[run,:,:] = y_predicttest
        #train_acc[run] = rmse(y_predicttrain, y_train)
        #print(train_acc[run], 'train accuracy')
        #test_acc[run] = rmse(y_predicttest, y_test)
        pred_test = [y_predicttest[i].argmax() for i in
range(y_predicttest.shape[0])]
        pred_train = [y_predicttrain[i].argmax() for i in
range(y_predicttrain.shape[0])]
        report_train = classification_report(act_train, pred_train,
labels=[0,1], output_dict=True)
        report_test = classification_report(act_test, pred_test,
labels=[0,1], output_dict=True)
        #test_acc[run] = f1_score(pred,act, average = 'binary')
        all_report_train[run] = report_train
        all_report_test[run] = report_test
        test_acc[run] = report_test['1']['f1-score']
        print("train acc: ", report_train['1']['f1-score'])
        print("test acc: ", test_acc[run])
        if test_acc[run] > Best_f1:
            Best_f1 = test_acc[run]
```

```
            Best_Predict_Test = y_predicttest
            Best_report_train, Best_report_test = report_train,
report_test
    model.save("model_" + ocean+"_"+model_name+"_"+method+'.h5')
    train_std = np.std(train_acc)
    test_std = np.std(test_acc)
    print("Total time for", Num_Exp, "experiments", time.time() -
start_time)
    print("f1 scores for test data: ", test_acc)
    print("mean: ", np.mean(test_acc), "std dev: ", test_std)
    return train_acc, test_acc, train_std, test_std, Best_Predict_Test,
y_predicttrain, y_predicttest, all_report_train, all_report_test

#idx = np.random.permutation(len(X_smote))
idx = np.random.permutation(len(X))
print(len(idx))
x_shuffled = []
y_shuffled = []
for i in idx:
  #x_shuffled.append(X_smote[i])
  #y_shuffled.append(Y_smote[i])
  x_shuffled.append(X[i])
  y_shuffled.append(Y[i])

Y_hot_encoded_train =  np.asarray(to_categorical(y_shuffled))
#Y_hot_encoded_train = Y_hot_encoded_train.reshape(len(y_shuffled),
n_steps_out, n_features_out)

Y_hot_encoded_test =  np.asarray(to_categorical(test_Y))
#Y_hot_encoded_test = Y_hot_encoded_test.reshape(len(test_Y),
n_steps_out, n_features_out)

print(Y_hot_encoded_train.shape, Y_hot_encoded_test.shape)

x_shuffled[0], test_X[0]

#models = ['vanilla', 'bidirectional', 'cnn-lstm', 'conv-lstm']
models = ['vanilla']
predictions_train = dict()
actual_train = dict()
predictions_test = dict()
actual_test = dict()
metrics_train = dict()
metrics_test = dict()
test_acc_all = dict()
test_stddev = dict()
```

```
for j in range(1):
    predictions_train_per_step = dict()
    actual_train_per_step = dict()
    predictions_test_per_step = dict()
    actual_test_per_step = dict()
    metrics_train_per_step = dict()
    metrics_test_per_step = dict()
    test_acc_per_step = dict()
    test_stddev_per_step = dict()
    n_steps_out = j+1
    print('------------------------------------------------------------')
    print('no of steps out: ', n_steps_out)

    for i in models:
        print("for " + i + ":")

        if i == 'vanilla' or i=='bidirectional':
            x_train, y_train = np.asarray(x_shuffled),
np.asarray(Y_hot_encoded_train)
            x_test, y_test = np.asarray(test_X),
np.asarray(Y_hot_encoded_test)
            x_train = x_train.reshape((x_train.shape[0],
x_train.shape[1], n_features_in))
            x_test = x_test.reshape((x_test.shape[0], x_test.shape[1],
n_features_in))
        elif i == 'cnn-lstm':
            x_train, y_train = np.asarray(x_shuffled),
np.asarray(Y_hot_encoded_train)
            x_test, y_test = np.asarray(test_X),
np.asarray(Y_hot_encoded_test)
            x_train = x_train.reshape((x_train.shape[0], n_seq,
int(n_steps_in/n_seq), n_features_in))
            x_test = x_test.reshape((x_test.shape[0], n_seq,
int(n_steps_in/n_seq), n_features_in))
        elif i=='conv-lstm':
            x_train, y_train = np.asarray(x_shuffled),
np.asarray(Y_hot_encoded_train)
            x_test, y_test = np.asarray(test_X),
np.asarray(Y_hot_encoded_test)
            x_train = x_train.reshape((x_train.shape[0], n_seq, 1,
int(n_steps_in/n_seq), n_features_in))
            x_test = x_test.reshape((x_test.shape[0], n_seq, 1,
int(n_steps_in/n_seq), n_features_in))

        #print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
        train_acc, test_acc, train_std_dev, test_std_dev,
Best_Predict_Test, y_predicttrain, y_predicttest, report_train,
```

```
report_test = MODEL_LSTM(i,'original',
univariate,x_train,x_test,y_train,y_test,Num_Exp,n_steps_in,n_steps_out,E
pochs, Hidden)
        predictions_train_per_step[i] = y_predicttrain
        actual_train_per_step[i] = y_train
        predictions_test_per_step[i] = Best_Predict_Test
        actual_test_per_step[i] = y_test
        metrics_train_per_step[i] = report_train
        metrics_test_per_step[i] = report_test
        test_acc_per_step[i] = test_acc
        test_stddev_per_step[i] = test_std_dev
    predictions_train[str(j+1)] = predictions_train_per_step
    actual_train[str(j+1)] = actual_train_per_step
    predictions_test[str(j+1)] = predictions_test_per_step
    actual_test[str(j+1)] = actual_test_per_step
    metrics_train[str(j+1)] = metrics_train_per_step
    metrics_test[str(j+1)] = metrics_test_per_step
    test_acc_all[str(j+1)] = test_acc_per_step
    test_stddev[str(j+1)] = test_stddev_per_step

with open("predictions_" + ocean + '_original' + '.pkl', 'wb') as f:

pickle.dump([predictions_train,actual_train,predictions_test,actual_test,
metrics_train,metrics_test,test_acc,test_stddev], f)

def make_confusion_matrix_chart2(cf_matrix_test, name):
    #plt.figure(figsize=(20,12))
    sns.set(font_scale=2.5)
    fig, ax = plt.subplots(figsize = (20,12))
    sns.heatmap(cf_matrix_test, annot=True, yticklabels=['0','1'],
                                xticklabels=['0','1'], fmt='g')
    plt.ylabel("Actual", size=30)
    plt.xlabel("Pred", size=30)
    ax.tick_params(axis='both', which='major', labelsize=25)
    plt.savefig(name + '.png', dpi=300, transparent=False,
bbox_inches='tight')
    return None

y = [i.argmax() for i in actual_test_per_step['vanilla']]
pred = [i.argmax() for i in predictions_test_per_step['vanilla']]
cf_matrix_test = confusion_matrix(y, pred)
make_confusion_matrix_chart2(cf_matrix_test, ocean +
'_vanilla_cm_original')

precision0=[]
precision1=[]
precisionacc=[]
```

```
precisionmacavg=[]
precisionweighavg=[]
recall0=[]
recall1=[]
recallacc=[]
recallmacavg=[]
recallweighavg=[]
f10=[]
f11=[]
f1acc=[]
f1macavg=[]
f1weighavg=[]
for i in range(Num_Exp):

precision0.append(metrics_test_per_step['vanilla'][i]['0']['precision'])

precision1.append(metrics_test_per_step['vanilla'][i]['1']['precision'])
  precisionacc.append(metrics_test_per_step['vanilla'][i]['accuracy'])
  precisionmacavg.append(metrics_test_per_step['vanilla'][i]['macro
avg']['precision'])
  precisionweighavg.append(metrics_test_per_step['vanilla'][i]['weighted
avg']['precision'])
  recall0.append(metrics_test_per_step['vanilla'][i]['0']['recall'])
  recall1.append(metrics_test_per_step['vanilla'][i]['1']['recall'])
  recallacc.append(metrics_test_per_step['vanilla'][i]['accuracy'])
  recallmacavg.append(metrics_test_per_step['vanilla'][i]['macro
avg']['recall'])
  recallweighavg.append(metrics_test_per_step['vanilla'][i]['weighted
avg']['recall'])
  f10.append(metrics_test_per_step['vanilla'][i]['0']['f1-score'])
  f11.append(metrics_test_per_step['vanilla'][i]['1']['f1-score'])
  f1acc.append(metrics_test_per_step['vanilla'][i]['accuracy'])
  f1macavg.append(metrics_test_per_step['vanilla'][i]['macro avg']['f1-
score'])
  f1weighavg.append(metrics_test_per_step['vanilla'][i]['weighted
avg']['f1-score'])

print(str(round(np.mean(precision0),4)) + "±" +
str(round(np.std(precision0),4))," & " + str(round(np.mean(recall0),4)) +
"±" + str(round(np.std(recall0),4)), " & " + str(round(np.mean(f10),4)) +
"±" + str(round(np.std(f10),4)))
print(str(round(np.mean(precision1),4)) + "±" +
str(round(np.std(precision1),4))," & " + str(round(np.mean(recall1),4)) +
"±" + str(round(np.std(recall1),4)), " & " + str(round(np.mean(f11),4)) +
"±" + str(round(np.std(f11),4)))
print(str(round(np.mean(precisionacc),4)) + "±" +
str(round(np.std(precisionacc),4))," & " +
```

```
str(round(np.mean(recallacc),4)) + "±" + str(round(np.std(recallacc),4)),
" & " + str(round(np.mean(f1acc),4)) + "±" + str(round(np.std(f1acc),4)))
print(str(round(np.mean(precisionmacavg),4)) + "±" +
str(round(np.std(precisionmacavg),4))," & " +
str(round(np.mean(recallmacavg),4)) + "±" +
str(round(np.std(recallmacavg),4)), " & " +
str(round(np.mean(f1macavg),4)) + "±" + str(round(np.std(f1macavg),4)))
print(str(round(np.mean(precisionweighavg),4)) + "±" +
str(round(np.std(precisionweighavg),4))," & " +
str(round(np.mean(recallweighavg),4)) + "±" +
str(round(np.std(recallweighavg),4)), " & " +
str(round(np.mean(f1weighavg),4)) + "±" +
str(round(np.std(f1weighavg),4)))

oversample = SMOTE()
X_smote, Y_smote = oversample.fit_resample(X, Y)
X_smote = X_smote.astype(np.float)

X.shape, test_X.shape, X_smote.shape

counter = Counter(Y_smote)
print(counter)
print(len(X_smote), len(X))
print("old smote data after 10000 index: ", Counter(Y_smote[10000:]))

idx = np.random.permutation(len(X_smote))
print(len(idx))

x_shuffled = []
y_shuffled = []
for i in idx:
  x_shuffled.append(X_smote[i])
  y_shuffled.append(Y_smote[i])

print("shuffled smote data after 10000 index: ",
Counter(y_shuffled[10000:]))
Y_hot_encoded_train =  np.asarray(to_categorical(y_shuffled))
#Y_hot_encoded_train = Y_hot_encoded_train.reshape(len(y_shuffled),
n_steps_out, n_features_out)

Y_hot_encoded_test =  np.asarray(to_categorical(test_Y))
#Y_hot_encoded_test = Y_hot_encoded_test.reshape(len(test_Y),
n_steps_out, n_features_out)

Y_hot_encoded_train.shape, Y_hot_encoded_test.shape
```

```
x_train, y_train = np.asarray(x_shuffled),
np.asarray(Y_hot_encoded_train)
x_test, y_test = np.asarray(test_X), np.asarray(Y_hot_encoded_test)
x_train = x_train.reshape((x_train.shape[0], x_train.shape[1],
n_features_in))
x_test = x_test.reshape((x_test.shape[0], x_test.shape[1],
n_features_in))

train_acc, test_acc, train_std_dev, test_std_dev, Best_Predict_Test,
y_predicttrain, y_predicttest, report_train, report_test =
MODEL_LSTM('vanilla', 'smote',
univariate,x_train,x_test,y_train,y_test,Num_Exp,n_steps_in,n_steps_out,E
pochs, Hidden)

"""**SMOTE Results**"""

y = [i.argmax() for i in y_test]
pred = [i.argmax() for i in Best_Predict_Test]
cf_matrix_test = confusion_matrix(y, pred)
make_confusion_matrix_chart2(cf_matrix_test, ocean + '_vanilla_cm_smote')

with open("predictions_" + ocean + '_smote' + '.pkl', 'wb') as f:
    pickle.dump([y_predicttrain, y_train, Best_Predict_Test, y_test,
report_train, report_test, test_acc, test_std_dev], f)

precision0=[]
precision1=[]
precisionacc=[]
precisionmacavg=[]
precisionweighavg=[]
recall0=[]
recall1=[]
recallacc=[]
recallmacavg=[]
recallweighavg=[]
f10=[]
f11=[]
f1acc=[]
f1macavg=[]
f1weighavg=[]
for i in range(Num_Exp):
  precision0.append(report_test[i]['0']['precision'])
  precision1.append(report_test[i]['1']['precision'])
  precisionacc.append(report_test[i]['accuracy'])
  precisionmacavg.append(report_test[i]['macro avg']['precision'])
  precisionweighavg.append(report_test[i]['weighted avg']['precision'])
  recall0.append(report_test[i]['0']['recall'])
```

```
  recall1.append(report_test[i]['1']['recall'])
  recallacc.append(report_test[i]['accuracy'])
  recallmacavg.append(report_test[i]['macro avg']['recall'])
  recallweighavg.append(report_test[i]['weighted avg']['recall'])
  f10.append(report_test[i]['0']['f1-score'])
  f11.append(report_test[i]['1']['f1-score'])
  f1acc.append(report_test[i]['accuracy'])
  f1macavg.append(report_test[i]['macro avg']['f1-score'])
  f1weighavg.append(report_test[i]['weighted avg']['f1-score'])
print(str(round(np.mean(precision0),4)) + "±" +
str(round(np.std(precision0),4))," & " + str(round(np.mean(recall0),4)) +
"±" + str(round(np.std(recall0),4)), " & " + str(round(np.mean(f10),4)) +
"±" + str(round(np.std(f10),4)))
print(str(round(np.mean(precision1),4)) + "±" +
str(round(np.std(precision1),4))," & " + str(round(np.mean(recall1),4)) +
"±" + str(round(np.std(recall1),4)), " & " + str(round(np.mean(f11),4)) +
"±" + str(round(np.std(f11),4)))
print(str(round(np.mean(precisionacc),4)) + "±" +
str(round(np.std(precisionacc),4))," & " +
str(round(np.mean(recallacc),4)) + "±" + str(round(np.std(recallacc),4)),
" & " + str(round(np.mean(f1acc),4)) + "±" + str(round(np.std(f1acc),4)))
print(str(round(np.mean(precisionmacavg),4)) + "±" +
str(round(np.std(precisionmacavg),4))," & " +
str(round(np.mean(recallmacavg),4)) + "±" +
str(round(np.std(recallmacavg),4)), " & " +
str(round(np.mean(f1macavg),4)) + "±" + str(round(np.std(f1macavg),4)))
print(str(round(np.mean(precisionweighavg),4)) + "±" +
str(round(np.std(precisionweighavg),4))," & " +
str(round(np.mean(recallweighavg),4)) + "±" +
str(round(np.std(recallweighavg),4)), " & " +
str(round(np.mean(f1weighavg),4)) + "±" +
str(round(np.std(f1weighavg),4)))

# report_df = pd.DataFrame(report_test).transpose()
# report_df = report_df.reset_index()
# model_eval  =
report_df[report_df['index'].str.contains('1')][['precision','recall','f1
-score']]
# model_eval['accuracy']  =
list(report_df[report_df['index'].str.contains('accuracy')]['support'])
# report_df

# for i in range(report_df.shape[0]):
#   tmp=""
#   for j in ['precision', 'recall', 'f1-score', 'support']:
#     tmp=tmp+ str(round(report_df[j][i],4)) + " & "
#   print(tmp)
```

```
"""**Simple GAN**"""

import torch
from torch import nn
from tqdm.auto import tqdm
from torchvision import transforms
from torchvision.utils import make_grid
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
torch.cuda.empty_cache()

x_shuffled = X
y_shuffled = Y

len(x_shuffled), len(X), len(y_shuffled)

t2 = np.asarray(x_shuffled).shape
X_oversampled = torch.from_numpy(np.asarray(x_shuffled))

def get_generator_block(input_dim, output_dim):
    return nn.Sequential(
        nn.Linear(input_dim, output_dim),
        nn.BatchNorm1d(output_dim),
        nn.ReLU(inplace=True),
    )
class Generator(nn.Module):

    def __init__(self, z_dim=t2[1], im_dim=t2[1], hidden_dim=128):
        super(Generator, self).__init__()
        self.gen = nn.Sequential(
            get_generator_block(z_dim, hidden_dim),
            get_generator_block(hidden_dim, hidden_dim * 2),
            get_generator_block(hidden_dim * 2, hidden_dim * 4),
            get_generator_block(hidden_dim * 4, hidden_dim * 8),
            nn.Linear(hidden_dim * 8, im_dim),
            nn.Sigmoid()
        )
    def forward(self, noise):
        return self.gen(noise)


    def get_gen(self):

        return self.gen
def get_discriminator_block(input_dim, output_dim):
    return nn.Sequential(
```

```
        nn.Linear(input_dim, output_dim),
        nn.LeakyReLU(0.2, inplace=True)
    )
class Discriminator(nn.Module):
    def __init__(self, im_dim=t2[1], hidden_dim=128):
        super(Discriminator, self).__init__()
        self.disc = nn.Sequential(
            get_discriminator_block(im_dim, hidden_dim * 4),
            get_discriminator_block(hidden_dim * 4, hidden_dim * 2),
            get_discriminator_block(hidden_dim * 2, hidden_dim),
            nn.Linear(hidden_dim, 1)
        )

    def forward(self, image):

        return self.disc(image)

    def get_disc(self):

        return self.dis
def get_noise(n_samples, z_dim, device='cuda'):

    return torch.randn(n_samples,z_dim,device=device)

criterion = nn.BCEWithLogitsLoss()
n_epochs = 1000
z_dim = t2[1]
batch_size = 128
lr = 0.00001
display_step = 1
device = 'cuda'
gen = Generator(z_dim).to(device)
gen_opt = torch.optim.Adam(gen.parameters(), lr=lr)
disc = Discriminator().to(device)
disc_opt = torch.optim.Adam(disc.parameters(), lr=lr)

def get_disc_loss(gen, disc, criterion, real, num_images, z_dim, device):

    fake_noise = get_noise(num_images, z_dim, device=device)
    fake = gen(fake_noise)
    disc_fake_pred = disc(fake.detach())
    disc_fake_loss = criterion(disc_fake_pred,
torch.zeros_like(disc_fake_pred))
    disc_real_pred = disc(real)
    disc_real_loss = criterion(disc_real_pred,
torch.ones_like(disc_real_pred))
    disc_loss = (disc_fake_loss + disc_real_loss) / 2
```

```
    return disc_loss

def get_gen_loss(gen, disc, criterion, num_images, z_dim, device):

    fake_noise = get_noise(num_images, z_dim, device=device)
    fake = gen(fake_noise)
    disc_fake_pred = disc(fake)
    gen_loss = criterion(disc_fake_pred, torch.ones_like(disc_fake_pred))
    return gen_loss

li=[]
for i in range(len(y_shuffled)):
    if int(y_shuffled[i])==1:
        li.append(x_shuffled[i])

len(y_shuffled), len(li)

X_real=np.array(li)
t3=X_real.shape
li2=[1]*(t3[0])
y_real=np.array(li2)
y_real.shape

from torch.utils.data import TensorDataset, DataLoader
tensor_x = torch.Tensor(X_real)
tensor_y = torch.Tensor(y_real)
my_dataset = TensorDataset(tensor_x,tensor_y)
dataloader = DataLoader(
    my_dataset,
    batch_size=batch_size,
    shuffle=True)

cur_step = 0
mean_generator_loss = 0
mean_discriminator_loss = 0
test_generator = True
gen_loss = False
error = False

samples_to_generate = X_oversampled.shape[0]-X_real.shape[0]
print(samples_to_generate)

#epochs = [100,500,1000,2000,3000,4000,5000]
epochs = [10000]

original_gan_data = dict()
```

```
for no_epoch in epochs:
  print("no of epochs: ", no_epoch)
  for epoch in range(no_epoch):


    for real, _ in tqdm(dataloader, disable=True):
        cur_batch_size = len(real)


        real = real.view(cur_batch_size, -1).to(device)



        disc_opt.zero_grad()


        disc_loss = get_disc_loss(gen, disc, criterion, real,
cur_batch_size, z_dim, device)


        disc_loss.backward(retain_graph=True)


        disc_opt.step()


        if test_generator:
            old_generator_weights = gen.gen[0][0].weight.detach().clone()

        gen_opt.zero_grad()
        gen_loss = get_gen_loss(gen, disc, criterion, cur_batch_size,
z_dim, device)
        gen_loss.backward()
        gen_opt.step()

        if test_generator:
            try:
                assert lr > 0.0000002 or
(gen.gen[0][0].weight.grad.abs().max() < 0.0005 and epoch == 0)
                assert torch.any(gen.gen[0][0].weight.detach().clone() !=
old_generator_weights)
            except:
                error = True
                print("Runtime tests have failed")
```

```
        mean_discriminator_loss += disc_loss.item() / display_step


        mean_generator_loss += gen_loss.item() / display_step

        if epoch%500==0:
            print(f"Epoch {epoch}: Step {cur_step}: Generator loss:
{mean_generator_loss}, discriminator loss: {mean_discriminator_loss}")

        if cur_step % display_step == 0 and cur_step > 0:
            mean_generator_loss = 0
            mean_discriminator_loss = 0
        cur_step += 1
  fake_noise = get_noise(samples_to_generate, z_dim, device=device)
  res=gen(fake_noise)
  fres=res.cpu().detach().numpy()
  X_old=X
  finX=np.concatenate((X_old, fres), axis=0)
  y_fake = np.ones(samples_to_generate)
  Y_old=np.asarray(Y)
  finY = np.append(Y_old, y_fake, axis=0)
  print(finX.shape, finY.shape)
  #idx = np.random.permutation(len(X_smote))
  idx = np.random.permutation(finX.shape[0])
  print(len(idx))
  x_shuffled = []
  y_shuffled = []
  xy=dict()
  for i in idx:
    #x_shuffled.append(X_smote[i])
    #y_shuffled.append(Y_smote[i])
    x_shuffled.append(finX[i])
    y_shuffled.append(finY[i])
  xy['x_shuffled']=x_shuffled
  xy['y_shuffled']=y_shuffled
  original_gan_data[no_epoch]=xy

with open(ocean + '_original_gan_data' + '.pkl', 'wb') as f:
    pickle.dump([original_gan_data], f)

len(original_gan_data[10000]['x_shuffled']),
len(original_gan_data[10000]['y_shuffled'])

x_shuffled = original_gan_data[10000]['x_shuffled']
y_shuffled = original_gan_data[10000]['y_shuffled']

len(test_Y), test_X.shape
```

```
Y_hot_encoded_train =  np.asarray(to_categorical(y_shuffled))
#Y_hot_encoded_train = Y_hot_encoded_train.reshape(len(y_shuffled),
n_steps_out, n_features_out)

Y_hot_encoded_test =  np.asarray(to_categorical(test_Y))
#Y_hot_encoded_test = Y_hot_encoded_test.reshape(len(test_Y),
n_steps_out, n_features_out)

print(Y_hot_encoded_train.shape, Y_hot_encoded_test.shape)

test_X

x_shuffled[0], y_shuffled[0]

x_train, y_train = np.asarray(x_shuffled),
np.asarray(Y_hot_encoded_train)
x_test, y_test = np.asarray(test_X), np.asarray(Y_hot_encoded_test)
x_train = x_train.reshape((x_train.shape[0], x_train.shape[1],
n_features_in))
x_test = x_test.reshape((x_test.shape[0], x_test.shape[1],
n_features_in))

train_acc, test_acc, train_std_dev, test_std_dev, Best_Predict_Test,
y_predicttrain, y_predicttest, report_train, report_test =
MODEL_LSTM('vanilla', 'gan',
univariate,x_train,x_test,y_train,y_test,Num_Exp,n_steps_in,n_steps_out,E
pochs, Hidden)

with open("predictions_" + ocean + '_gan' + '.pkl', 'wb') as f:
    pickle.dump([y_predicttrain, y_train, Best_Predict_Test, y_test,
report_train, report_test, test_acc, test_std_dev], f)

"""**GAN Results**"""

precision0=[]
precision1=[]
precisionacc=[]
precisionmacavg=[]
precisionweighavg=[]
recall0=[]
recall1=[]
recallacc=[]
recallmacavg=[]
recallweighavg=[]
f10=[]
f11=[]
```

```
f1acc=[]
f1macavg=[]
f1weighavg=[]
for i in range(Num_Exp):
  precision0.append(report_test[i]['0']['precision'])
  precision1.append(report_test[i]['1']['precision'])
  precisionacc.append(report_test[i]['accuracy'])
  precisionmacavg.append(report_test[i]['macro avg']['precision'])
  precisionweighavg.append(report_test[i]['weighted avg']['precision'])
  recall0.append(report_test[i]['0']['recall'])
  recall1.append(report_test[i]['1']['recall'])
  recallacc.append(report_test[i]['accuracy'])
  recallmacavg.append(report_test[i]['macro avg']['recall'])
  recallweighavg.append(report_test[i]['weighted avg']['recall'])
  f10.append(report_test[i]['0']['f1-score'])
  f11.append(report_test[i]['1']['f1-score'])
  f1acc.append(report_test[i]['accuracy'])
  f1macavg.append(report_test[i]['macro avg']['f1-score'])
  f1weighavg.append(report_test[i]['weighted avg']['f1-score'])
print(str(round(np.mean(precision0),4)) + "±" +
str(round(np.std(precision0),4))," & " + str(round(np.mean(recall0),4)) +
"±" + str(round(np.std(recall0),4)), " & " + str(round(np.mean(f10),4)) +
"±" + str(round(np.std(f10),4)))
print(str(round(np.mean(precision1),4)) + "±" +
str(round(np.std(precision1),4))," & " + str(round(np.mean(recall1),4)) +
"±" + str(round(np.std(recall1),4)), " & " + str(round(np.mean(f11),4)) +
"±" + str(round(np.std(f11),4)))
print(str(round(np.mean(precisionacc),4)) + "±" +
str(round(np.std(precisionacc),4))," & " +
str(round(np.mean(recallacc),4)) + "±" + str(round(np.std(recallacc),4)),
" & " + str(round(np.mean(f1acc),4)) + "±" + str(round(np.std(f1acc),4)))
print(str(round(np.mean(precisionmacavg),4)) + "±" +
str(round(np.std(precisionmacavg),4))," & " +
str(round(np.mean(recallmacavg),4)) + "±" +
str(round(np.std(recallmacavg),4)), " & " +
str(round(np.mean(f1macavg),4)) + "±" + str(round(np.std(f1macavg),4)))
print(str(round(np.mean(precisionweighavg),4)) + "±" +
str(round(np.std(precisionweighavg),4))," & " +
str(round(np.mean(recallweighavg),4)) + "±" +
str(round(np.std(recallweighavg),4)), " & " +
str(round(np.mean(f1weighavg),4)) + "±" +
str(round(np.std(f1weighavg),4)))

y = [i.argmax() for i in y_test]
pred = [i.argmax() for i in Best_Predict_Test]
cf_matrix_test = confusion_matrix(y, pred)
make_confusion_matrix_chart2(cf_matrix_test, ocean + 'vanilla_cm_gan')
```

```
torch.cuda.empty_cache()

x_shuffled = X_smote
y_shuffled = Y_smote
print(len(x_shuffled), len(X), len(y_shuffled))
t2 = np.asarray(x_shuffled).shape
X_oversampled = torch.from_numpy(np.asarray(x_shuffled))

def get_generator_block(input_dim, output_dim):
    return nn.Sequential(
        nn.Linear(input_dim, output_dim),
        nn.BatchNorm1d(output_dim),
        nn.ReLU(inplace=True),
    )
class Generator(nn.Module):

    def __init__(self, z_dim=t2[1], im_dim=t2[1], hidden_dim=128):
        super(Generator, self).__init__()
        self.gen = nn.Sequential(
            get_generator_block(z_dim, hidden_dim),
            get_generator_block(hidden_dim, hidden_dim * 2),
            get_generator_block(hidden_dim * 2, hidden_dim * 4),
            get_generator_block(hidden_dim * 4, hidden_dim * 8),
            nn.Linear(hidden_dim * 8, im_dim),
            nn.Sigmoid()
        )
    def forward(self, noise):
        return self.gen(noise)


    def get_gen(self):

        return self.gen
def get_discriminator_block(input_dim, output_dim):
    return nn.Sequential(
        nn.Linear(input_dim, output_dim),
        nn.LeakyReLU(0.2, inplace=True)
    )
class Discriminator(nn.Module):
    def __init__(self, im_dim=t2[1], hidden_dim=128):
        super(Discriminator, self).__init__()
        self.disc = nn.Sequential(
            get_discriminator_block(im_dim, hidden_dim * 4),
            get_discriminator_block(hidden_dim * 4, hidden_dim * 2),
            get_discriminator_block(hidden_dim * 2, hidden_dim),
            nn.Linear(hidden_dim, 1)
```

```
        )

    def forward(self, image):

        return self.disc(image)

    def get_disc(self):

        return self.dis
def get_noise(n_samples, z_dim, device='cuda'):

    return torch.randn(n_samples,z_dim,device=device)

criterion = nn.BCEWithLogitsLoss()
n_epochs = 1000
z_dim = t2[1]
batch_size = 128
lr = 0.00001
display_step = 1
device = 'cuda'
gen = Generator(z_dim).to(device)
gen_opt = torch.optim.Adam(gen.parameters(), lr=lr)
disc = Discriminator().to(device)
disc_opt = torch.optim.Adam(disc.parameters(), lr=lr)

def get_disc_loss(gen, disc, criterion, real, num_images, z_dim, device):

    fake_noise = get_noise(num_images, z_dim, device=device)
    fake = gen(fake_noise)
    disc_fake_pred = disc(fake.detach())
    disc_fake_loss = criterion(disc_fake_pred,
torch.zeros_like(disc_fake_pred))
    disc_real_pred = disc(real)
    disc_real_loss = criterion(disc_real_pred,
torch.ones_like(disc_real_pred))
    disc_loss = (disc_fake_loss + disc_real_loss) / 2

    return disc_loss

def get_gen_loss(gen, disc, criterion, num_images, z_dim, device):

    fake_noise = get_noise(num_images, z_dim, device=device)
    fake = gen(fake_noise)
    disc_fake_pred = disc(fake)
    gen_loss = criterion(disc_fake_pred, torch.ones_like(disc_fake_pred))
    return gen_loss
```

```
li=[]
for i in range(len(y_shuffled)):
    if int(y_shuffled[i])==1:
        li.append(x_shuffled[i])

print(len(y_shuffled), len(li))

X_real=np.array(li)
t3=X_real.shape
li2=[1]*(t3[0])
y_real=np.array(li2)
y_real.shape

from torch.utils.data import TensorDataset, DataLoader
tensor_x = torch.Tensor(X_real)
tensor_y = torch.Tensor(y_real)
my_dataset = TensorDataset(tensor_x,tensor_y)
dataloader = DataLoader(
    my_dataset,
    batch_size=batch_size,
    shuffle=True)

cur_step = 0
mean_generator_loss = 0
mean_discriminator_loss = 0
test_generator = True
gen_loss = False
error = False

samples_to_generate = X_oversampled.shape[0]-X_real.shape[0]
print(samples_to_generate)

#epochs = [100,500,1000,2000,3000,4000,5000]
epochs = [10000]

smote_gan_data = dict()

for no_epoch in epochs:
  print("no of epochs: ", no_epoch)
  for epoch in range(no_epoch):


    for real, _ in tqdm(dataloader, disable=True):
        cur_batch_size = len(real)


        real = real.view(cur_batch_size, -1).to(device)
```

```
        disc_opt.zero_grad()


        disc_loss = get_disc_loss(gen, disc, criterion, real,
cur_batch_size, z_dim, device)


        disc_loss.backward(retain_graph=True)


        disc_opt.step()


        if test_generator:
            old_generator_weights = gen.gen[0][0].weight.detach().clone()

        gen_opt.zero_grad()
        gen_loss = get_gen_loss(gen, disc, criterion, cur_batch_size,
z_dim, device)
        gen_loss.backward()
        gen_opt.step()

        if test_generator:
            try:
                assert lr > 0.0000002 or
(gen.gen[0][0].weight.grad.abs().max() < 0.0005 and epoch == 0)
                assert torch.any(gen.gen[0][0].weight.detach().clone() !=
old_generator_weights)
            except:
                error = True
                print("Runtime tests have failed")


        mean_discriminator_loss += disc_loss.item() / display_step


        mean_generator_loss += gen_loss.item() / display_step

        if epoch%500==0:
            print(f"Epoch {epoch}: Step {cur_step}: Generator loss:
{mean_generator_loss}, discriminator loss: {mean_discriminator_loss}")

        if cur_step % display_step == 0 and cur_step > 0:
            mean_generator_loss = 0
```

```
            mean_discriminator_loss = 0
        cur_step += 1
  fake_noise = get_noise(samples_to_generate, z_dim, device=device)
  res=gen(fake_noise)
  fres=res.cpu().detach().numpy()
  X_old=X
  finX=np.concatenate((X_old, fres), axis=0)
  y_fake = np.ones(samples_to_generate)
  Y_old=np.asarray(Y)
  finY = np.append(Y_old, y_fake, axis=0)
  print(finX.shape, finY.shape)
  #idx = np.random.permutation(len(X_smote))
  idx = np.random.permutation(finX.shape[0])
  print(len(idx))
  x_shuffled = []
  y_shuffled = []
  xy=dict()
  for i in idx:
    #x_shuffled.append(X_smote[i])
    #y_shuffled.append(Y_smote[i])
    x_shuffled.append(finX[i])
    y_shuffled.append(finY[i])
  xy['x_shuffled']=x_shuffled
  xy['y_shuffled']=y_shuffled
  smote_gan_data[no_epoch]=xy

with open(ocean + '_smote_gan_data' + '.pkl', 'wb') as f:
    pickle.dump([smote_gan_data], f)

print(len(smote_gan_data[10000]['x_shuffled']),
len(smote_gan_data[10000]['y_shuffled']))
x_shuffled = smote_gan_data[10000]['x_shuffled']
y_shuffled = smote_gan_data[10000]['y_shuffled']
print(len(test_Y), test_X.shape)

Y_hot_encoded_train =  np.asarray(to_categorical(y_shuffled))
#Y_hot_encoded_train = Y_hot_encoded_train.reshape(len(y_shuffled),
n_steps_out, n_features_out)

Y_hot_encoded_test =  np.asarray(to_categorical(test_Y))
#Y_hot_encoded_test = Y_hot_encoded_test.reshape(len(test_Y),
n_steps_out, n_features_out)

print(Y_hot_encoded_train.shape, Y_hot_encoded_test.shape)

test_X
```

```
x_shuffled[0], y_shuffled[0]

x_train, y_train = np.asarray(x_shuffled),
np.asarray(Y_hot_encoded_train)
x_test, y_test = np.asarray(test_X), np.asarray(Y_hot_encoded_test)
x_train = x_train.reshape((x_train.shape[0], x_train.shape[1],
n_features_in))
x_test = x_test.reshape((x_test.shape[0], x_test.shape[1],
n_features_in))

train_acc, test_acc, train_std_dev, test_std_dev, Best_Predict_Test,
y_predicttrain, y_predicttest, report_train, report_test =
MODEL_LSTM('vanilla', 'smote_gan',
univariate,x_train,x_test,y_train,y_test,Num_Exp,n_steps_in,n_steps_out,E
pochs, Hidden)

with open("predictions_" + ocean + '_smote_gan' + '.pkl', 'wb') as f:
    pickle.dump([y_predicttrain, y_train, Best_Predict_Test, y_test,
report_train, report_test, test_acc, test_std_dev], f)

"""**SMOTE GAN results**"""

precision0=[]
precision1=[]
precisionacc=[]
precisionmacavg=[]
precisionweighavg=[]
recall0=[]
recall1=[]
recallacc=[]
recallmacavg=[]
recallweighavg=[]
f10=[]
f11=[]
f1acc=[]
f1macavg=[]
f1weighavg=[]
for i in range(Num_Exp):
  precision0.append(report_test[i]['0']['precision'])
  precision1.append(report_test[i]['1']['precision'])
  precisionacc.append(report_test[i]['accuracy'])
  precisionmacavg.append(report_test[i]['macro avg']['precision'])
  precisionweighavg.append(report_test[i]['weighted avg']['precision'])
  recall0.append(report_test[i]['0']['recall'])
  recall1.append(report_test[i]['1']['recall'])
  recallacc.append(report_test[i]['accuracy'])
  recallmacavg.append(report_test[i]['macro avg']['recall'])
```

```
  recallweighavg.append(report_test[i]['weighted avg']['recall'])
  f10.append(report_test[i]['0']['f1-score'])
  f11.append(report_test[i]['1']['f1-score'])
  f1acc.append(report_test[i]['accuracy'])
  f1macavg.append(report_test[i]['macro avg']['f1-score'])
  f1weighavg.append(report_test[i]['weighted avg']['f1-score'])
print(str(round(np.mean(precision0),4)) + "±" +
str(round(np.std(precision0),4))," & " + str(round(np.mean(recall0),4)) +
"±" + str(round(np.std(recall0),4)), " & " + str(round(np.mean(f10),4)) +
"±" + str(round(np.std(f10),4)))
print(str(round(np.mean(precision1),4)) + "±" +
str(round(np.std(precision1),4))," & " + str(round(np.mean(recall1),4)) +
"±" + str(round(np.std(recall1),4)), " & " + str(round(np.mean(f11),4)) +
"±" + str(round(np.std(f11),4)))
print(str(round(np.mean(precisionacc),4)) + "±" +
str(round(np.std(precisionacc),4))," & " +
str(round(np.mean(recallacc),4)) + "±" + str(round(np.std(recallacc),4)),
" & " + str(round(np.mean(f1acc),4)) + "±" + str(round(np.std(f1acc),4)))
print(str(round(np.mean(precisionmacavg),4)) + "±" +
str(round(np.std(precisionmacavg),4))," & " +
str(round(np.mean(recallmacavg),4)) + "±" +
str(round(np.std(recallmacavg),4)), " & " +
str(round(np.mean(f1macavg),4)) + "±" + str(round(np.std(f1macavg),4)))
print(str(round(np.mean(precisionweighavg),4)) + "±" +
str(round(np.std(precisionweighavg),4))," & " +
str(round(np.mean(recallweighavg),4)) + "±" +
str(round(np.std(recallweighavg),4)), " & " +
str(round(np.mean(f1weighavg),4)) + "±" +
str(round(np.std(f1weighavg),4)))

y = [i.argmax() for i in y_test]
pred = [i.argmax() for i in Best_Predict_Test]
cf_matrix_test = confusion_matrix(y, pred)
make_confusion_matrix_chart2(cf_matrix_test, ocean +
'vanilla_cm_smote_gan')

for i in range(report_df.shape[0]):
  tmp=""
  for j in ['precision', 'recall', 'f1-score', 'support']:
    tmp=tmp+ str(round(report_df[j][i],4)) + " & "
  print(tmp)

"""**Recall Precision curves**"""

with open(ocean + '_original' + '.pkl', 'rb') as f:
```

```
predictions_train,actual_train,predictions_test,actual_test,metrics_train
,metrics_test,test_acc,test_stddev=pickle.load(f)

lr_probs = predictions_test['1']['vanilla']
lr_probs = lr_probs[:, 1]
print(lr_probs)
testy= [i.argmax() for i in actual_test['1']['vanilla']]
yhat= [i.argmax() for i in predictions_test['1']['vanilla']]
lr_precision, lr_recall, _ = precision_recall_curve(testy, lr_probs)
lr_f1, lr_auc = f1_score(testy, yhat), auc(lr_recall, lr_precision)
# summarize scores
print('Logistic: f1=%.3f auc=%.3f' % (lr_f1, lr_auc))

with open(ocean + '_smote' + '.pkl', 'rb') as f:
    y_predicttrain, y_train, Best_Predict_Test, y_test, report_train,
report_test, test_acc, test_stddev = pickle.load(f)

lr_probs = Best_Predict_Test
lr_probs = lr_probs[:, 1]
print(lr_probs)
testy= [i.argmax() for i in y_test]
yhat= [i.argmax() for i in Best_Predict_Test]
lr_precision_smote, lr_recall_smote, _ = precision_recall_curve(testy,
lr_probs)
lr_f1_smote, lr_auc_smote = f1_score(testy, yhat), auc(lr_recall_smote,
lr_precision_smote)
# summarize scores
print('Logistic: f1=%.3f auc=%.3f' % (lr_f1_smote, lr_auc_smote))

with open(ocean + '_gan' + '.pkl', 'rb') as f:
    y_predicttrain, y_train, Best_Predict_Test, y_test, report_train,
report_test, test_acc, test_stddev = pickle.load(f)

lr_probs = Best_Predict_Test
lr_probs = lr_probs[:, 1]
print(lr_probs)
testy= [i.argmax() for i in y_test]
yhat= [i.argmax() for i in Best_Predict_Test]
lr_precision_gan, lr_recall_gan, _ = precision_recall_curve(testy,
lr_probs)
lr_f1_gan, lr_auc_gan = f1_score(testy, yhat), auc(lr_recall_gan,
lr_precision_gan)
# summarize scores
print('Logistic: f1=%.3f auc=%.3f' % (lr_f1_gan, lr_auc_gan))

with open(ocean + '_smote_gan' + '.pkl', 'rb') as f:
```

```
    y_predicttrain, y_train, Best_Predict_Test, y_test, report_train,
report_test, test_acc, test_stddev = pickle.load(f)

lr_probs = Best_Predict_Test
lr_probs = lr_probs[:, 1]
print(lr_probs)
testy= [i.argmax() for i in y_test]
yhat= [i.argmax() for i in Best_Predict_Test]
lr_precision_smote_gan, lr_recall_smote_gan, _ =
precision_recall_curve(testy, lr_probs)
lr_f1_smote_gan, lr_auc_smote_gan = f1_score(testy, yhat),
auc(lr_recall_smote_gan, lr_precision_smote_gan)
# summarize scores
print('Logistic: f1=%.3f auc=%.3f' % (lr_f1_smote_gan, lr_auc_smote_gan))

# plot the precision-recall curves
fig, ax = pyplot.subplots(figsize = (20,12))
no_skill = 135 / len(testy)    #no of one/total
pyplot.plot(lr_recall, lr_precision, marker='.',lw=4, label='Original
data: ' + str(round(lr_f1,3)))
pyplot.plot(lr_recall_smote, lr_precision_smote,lw=4, marker='.',
label='SMOTE: ' + str(round(lr_f1_smote,3)))
pyplot.plot(lr_recall_gan, lr_precision_gan,lw=4, marker='.', label='GAN:
' + str(round(lr_f1_gan,3)))
pyplot.plot(lr_recall_smote_gan, lr_precision_smote_gan,lw=4, marker='.',
label='SMOTE-GAN: ' + str(round(lr_f1_smote_gan,3)))
# axis labels
pyplot.xlabel('Recall', size=40)
pyplot.ylabel('Precision', size=40)
ax.tick_params(axis='both', which='major', labelsize=30)
# show the legend
pyplot.legend(loc="upper right", fontsize=30)
#save the plot
plt.savefig(ocean + '_precision_recall_curve.png', dpi=300,
transparent=False, bbox_inches='tight')
plt.show()
# show the plot
pyplot.show()
```

8. SYSTEM TESTING

8.1 TESTING PLAN

The software engineering process can be viewed as a spiral. Initially system engineering defines the role of software and leads to software requirement analysis where the information domain, functions, behavior, performance, constraints and validation criteria for software are established. Moving inward along the spiral, we come to design and finally to coding. To develop computer software we spiral in along streamlines that decrease the level of abstraction on each turn.

A strategy for software testing may also be viewed in the context of the spiral. Unit testing begins at the vertex of the spiral and concentrates on each unit of the software as implemented in source code. Testing progress by moving outward along the spiral to integration testing, where the focus is on the design and the construction of the software architecture. Talking another turn on outward on the spiral we encounter validation testing where requirements established as part of software requirements analysis are validated against the software that has been constructed. Finally we arrive at system testing, where the software and other system elements are tested as a whole.

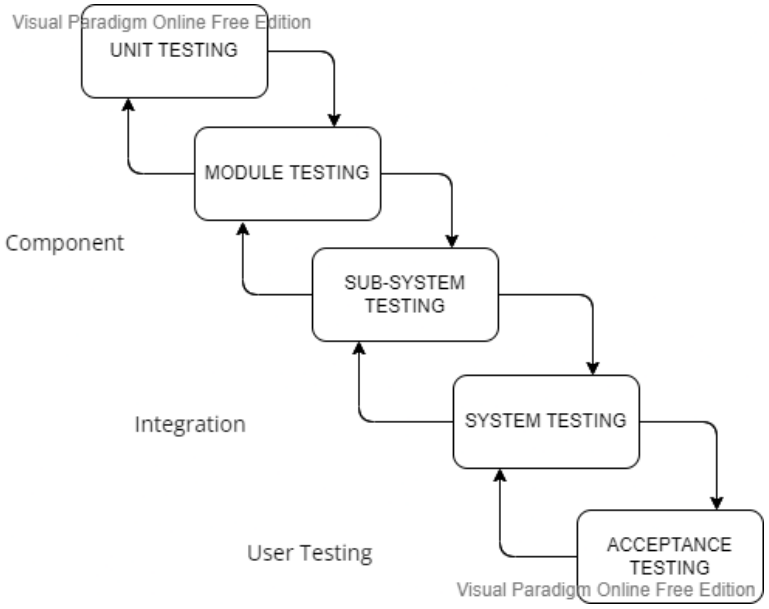

Figure: 7.1 Testing Plan

8.2 SOFTWARE TESTING

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.

8.2.1 TESTING METHODOLOGIES

The following are the Testing Methodologies:

- Unit Testing.
- Integration Testing.
- User Acceptance Testing.
- Output Testing.
- Validation Testing.
- Black box Testing.
- White box Testing.

8.2.1.1 Unit Testing

Unit testing focuses verification effort on the smallest unit of Software design that is the module. Unit testing exercises specific paths in a module's control structure to ensure complete coverage and maximum error detection. This test focuses on each module individually, ensuring that it functions properly as a unit. Hence, the naming is Unit Testing. During this testing, each module is tested individually and the module interfaces are verified for the consistency with design specification. All important processing path are tested for the expected results. All error handling paths are also tested.

8.2.1.2 Integration Testing

Integration testing addresses the issues associated with the dual problems of verification and program construction. After the software has been integrated a set of high order tests are conducted. The main objective in this testing process is to take unit tested modules and builds a program structure that has been dictated by design.

8.2.1.3 User Acceptance Testing

User Acceptance of a system is the key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with the prospective system users at the time of developing and making changes wherever required. The system developed provides a friendly user interface that can easily be understood even by a person who is new to the system.

8.2.1.4 Output Testing

After performing the validation testing, the next step is output testing of the proposed system, since no system could be useful if it does not produce the required output in the specified format. Asking the users about the format required by them tests the outputs generated or displayed by the system under consideration. Hence the output format is considered in 2 ways – one is on screen and another in printed format.

8.2.1.5 Validation Testing

Validation checks are performed on the following fields.

- **Text Field:**
  - The text field can contain only the number of characters lesser than or equal to its size. The text fields are alphanumeric in some tables and alphabetic in other tables. Incorrect entry always flashes and error message.
- **Numeric Field:**
  - The numeric field can contain only numbers from 0 to 9. An entry of any character flashes an error messages. The individual modules are checked for accuracy and what it has to perform. Each module is subjected to test run along with sample data. The individually tested modules are integrated into a single system. Testing involves executing the real data information is used

in the program the existence of any program defect is inferred from the output. The testing should be planned so that all the requirements are individually tested. A successful test is one that gives out the defects for the inappropriate data and produces and output revealing the errors in the system.

- **Preparation of Test Data**
  - Taking various kinds of test data does the above testing. Preparation of test data plays a vital role in the system testing. After preparing the test data the system under study is tested using that test data. While testing the system by using test data errors are again uncovered and corrected by using above testing steps and corrections are also noted for future use.

8.2.1.6 White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

8.2.1.7 Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

TESTING STRATEGY:

A strategy for system testing integrates system test cases and design techniques into a well planned series of steps that results in the successful construction of software. The testing strategy must co-operate test planning, test case design, test execution, and the resultant data collection and evaluation. A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high level tests that validate major system functions against user requirements. Software testing is a critical element of software quality assurance and represents the ultimate review of specification design and coding. Testing represents an interesting anomaly for the software. Thus, a series of testing are performed for the proposed system before the system is ready for user acceptance testing.

8.3 TEST CASES

Test Case Id	Test Case Description	Expected Results	Actual Result	Result
01	User Interface	User should able to upload and view the result images on the system	Users are able to interact with UI and also able to upload and view the output images	Pass
02	Image Browsing	The User should able to browser and upload the image	The User is able to browse and upload the image	Pass
03	Image Classify	Once if the user clicks the classify button he should be able to view the classified output	The user is able to hit the classify button and view the classified button.	Pass

Table 8.1 : Test Cases for User Interface

Test Case ID	Test Case Description	Expected Output	Actual Output	Result
01	User Interface Testing	User should be able to upload and view the result images on the system	Users can interact with UI and also able to upload and view the output images	Pass
02	Image with a cyclone	'It is a Cyclone' message and SMS notification	'It is a Cyclone Message'	Fail
02	Image with a cyclone	'It is a Cyclone' message and SMS notification	'It is a Cyclone' message and SMS notification	Pass
03	Image without a cyclone.	'It is not a Cyclone' message and NO SMS notification	'It is not a Cyclone' message and NO SMS notification	Pass
04	Image unrelated to aerial imagery	'It is not a Cyclone' message and NO SMS notification	'It is not a Cyclone' message and NO SMS notification	Pass

Table 8.2 :Test Cases for Classification Model

7.4 BUG REPORT

S.no	Steps	Description
1.	**Bug_id**	Bug_1
2.	**Bug Description**	Invalid Image
3.	**Expected output**	Proper Detected Text
4.	**Actual output**	Invalid Image
5.	**Priority**	High
6.	**Severity**	High

Table 8.3 Bug Report 1

S.no	Steps	Description
1.	**Bug_id**	Bug_2
2.	**Bug Description**	Could not able to Upload Image
3.	**Expected output**	Visible of Image on the Screen
4.	**Actual output**	Image is not visible on the screen
5.	**Priority**	High
6.	**Severity**	High

Table 8.4 Bug Report 2

S.no	Steps	Description
1.	**Bug_id**	Bug_3
2.	**Bug Description**	Could not able to Load the Deep CNN model
3.	**Expected output**	Proper Image Classification
4.	**Actual output**	Deep CNN model could not able to detect the image.
5.	**Priority**	High
6.	**Severity**	High

Table 8.5 Bug Report 3

9. RESULT SCREENS

9.1 Application

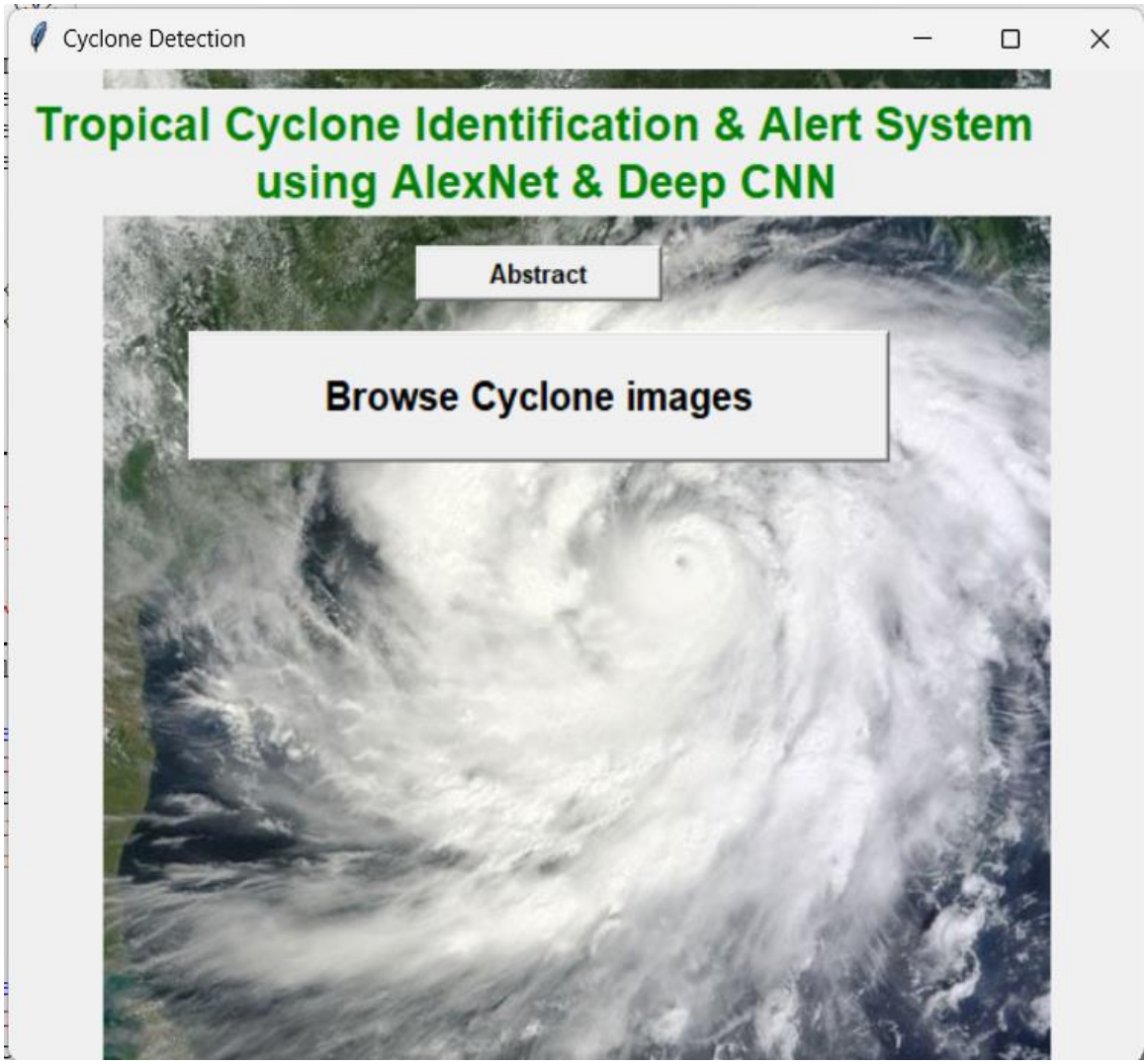

Fig 9.2.1 Graphic User interface

Navigation:

- Once we click “Browse Cyclone image” button, file explorer window opens to let you select/ upload image for classification.

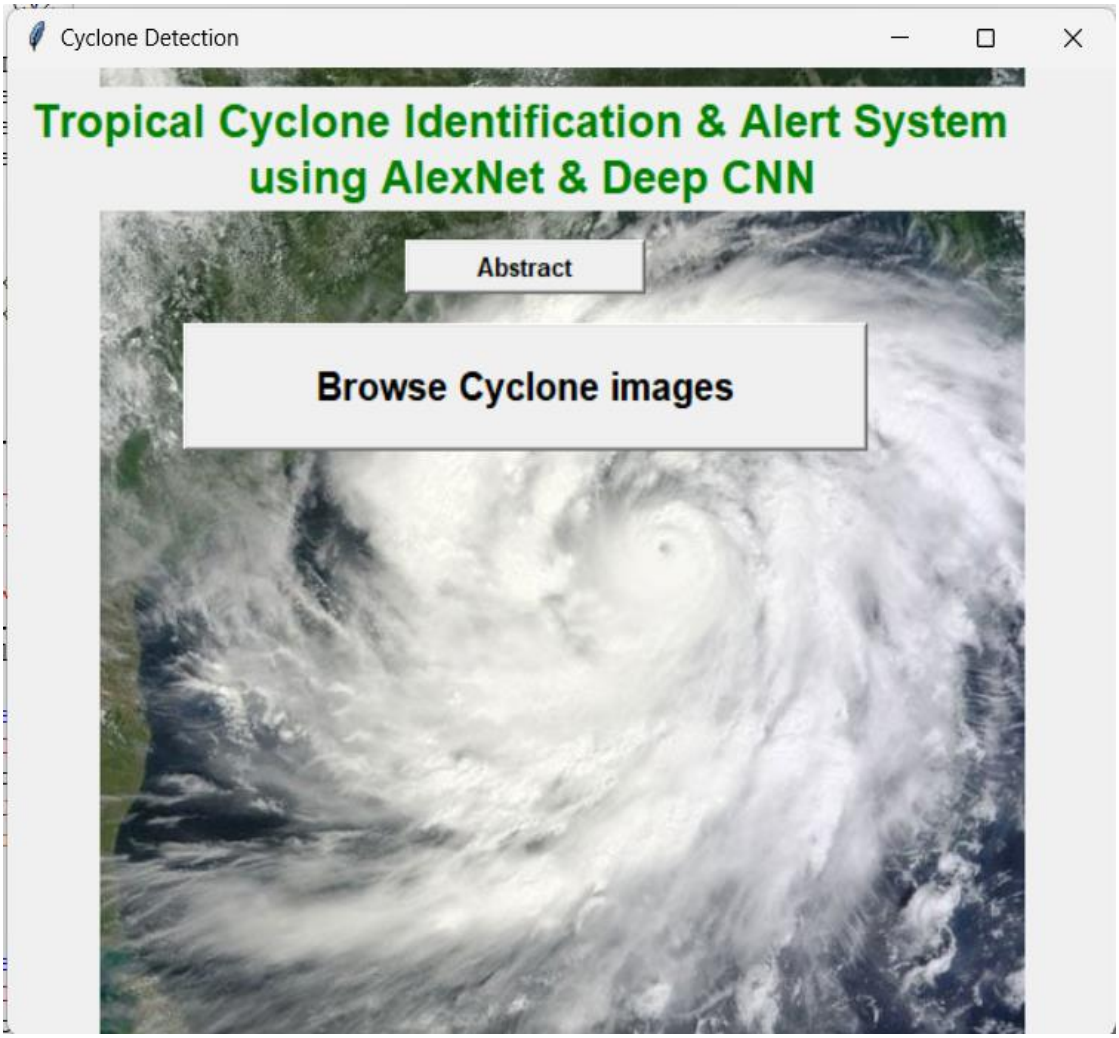

Fig 9.2.3 Graphic User interface

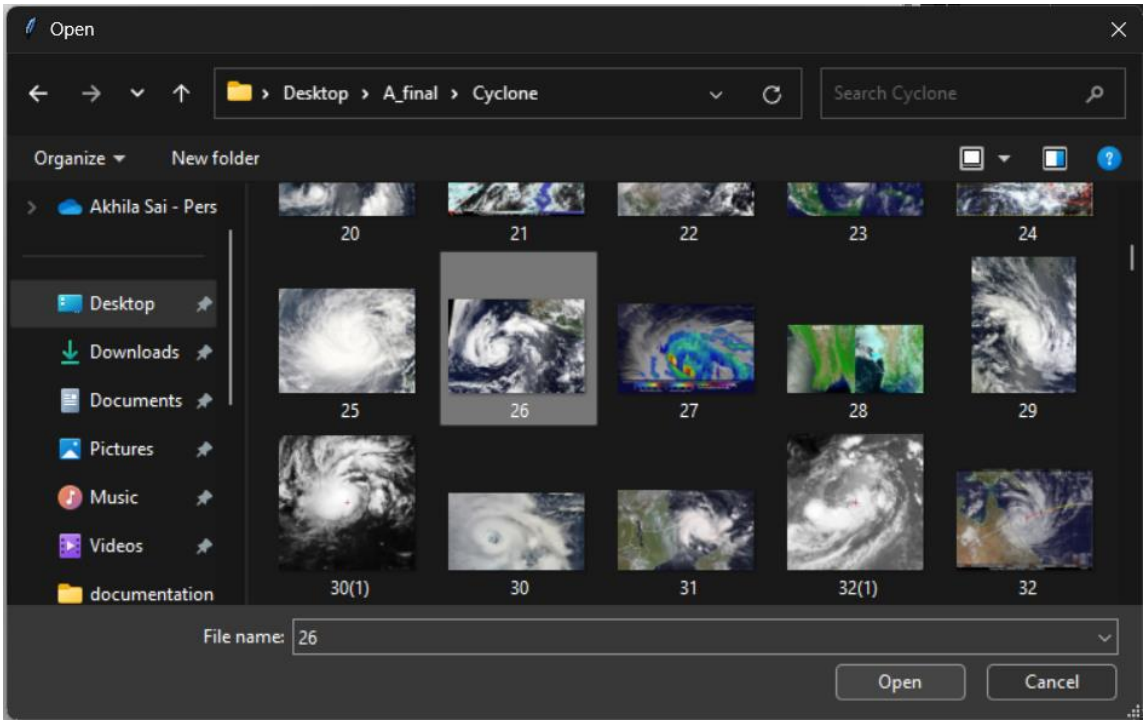

Fig 9.2.4 File explorer window to select/ upload image for classification

- Once you select the image, the selected image is displayed along with the classification result

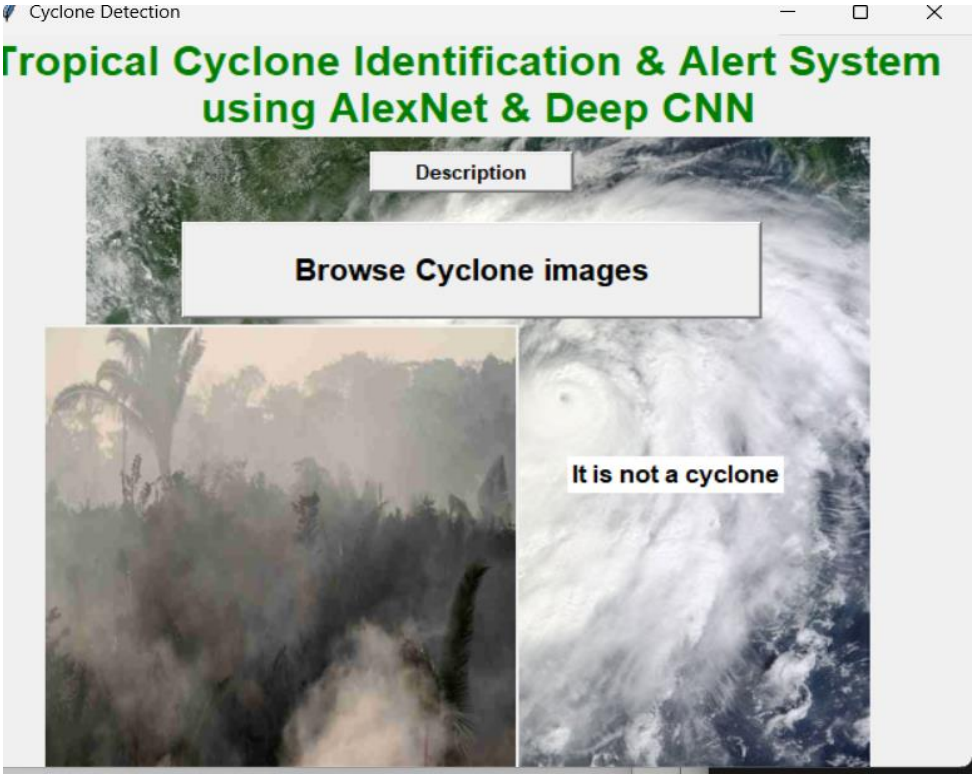

Fig 9.3 When given an image which is not a cyclone

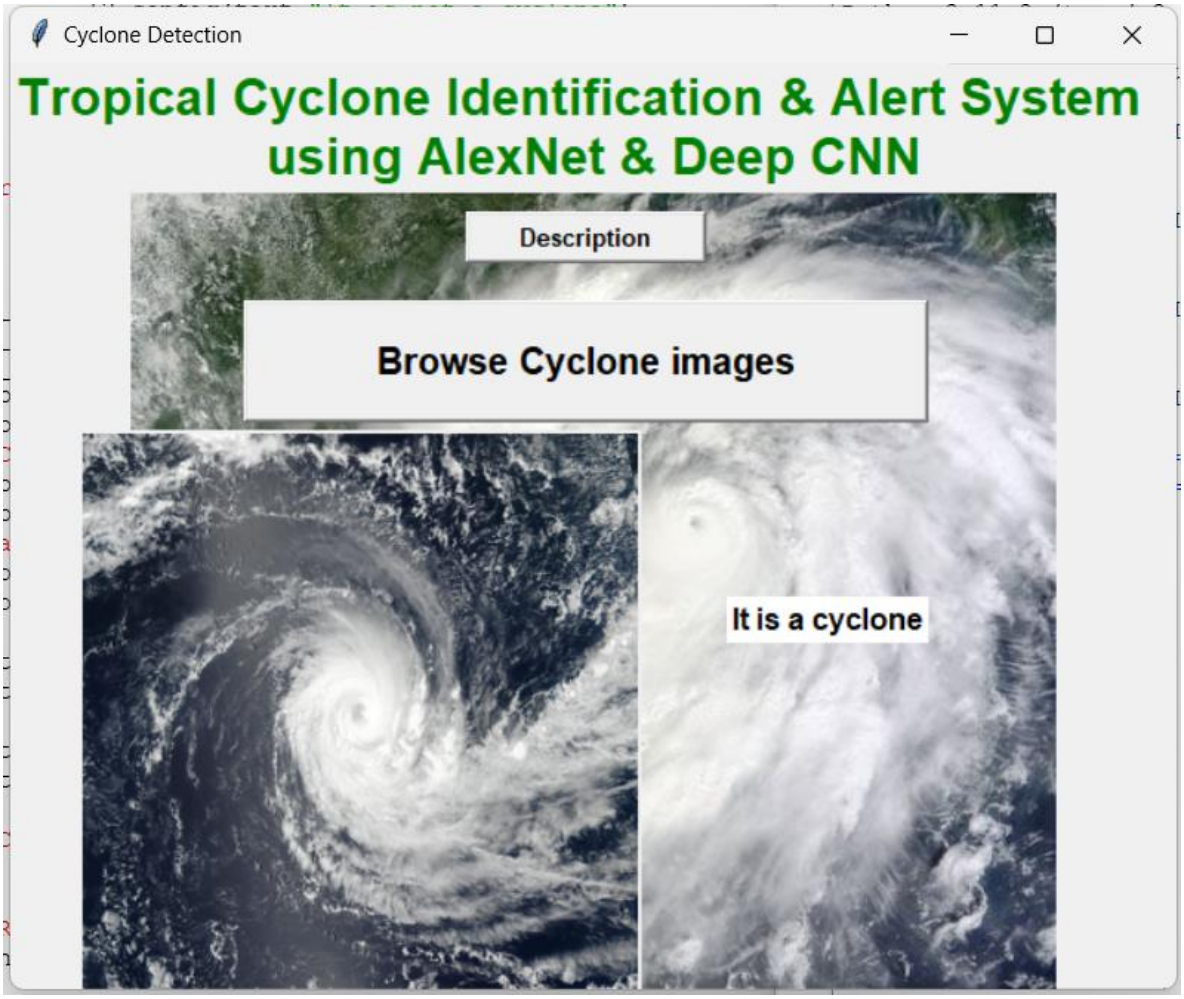

Fig 9.4 When given an image which is a cyclone

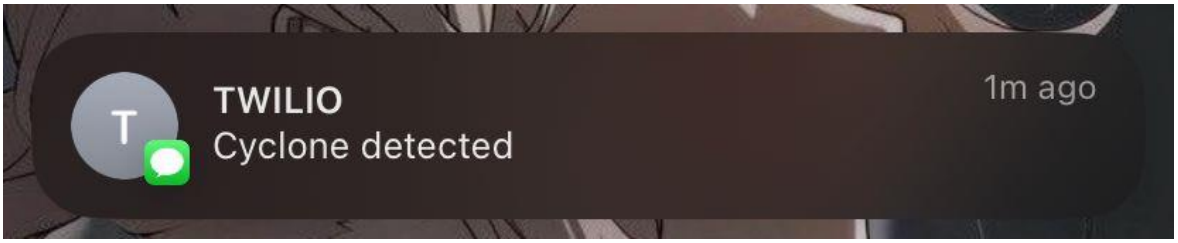

Fig 9.5 Notifications from Alert System

10. CONCLUSION AND FUTURE SCOPE

The development of a Tropical Cyclone Identification and Alert System using AlexNet and Deep CNN with satellite images has the potential to significantly improve the accuracy and timeliness of tropical cyclone tracking and forecasting. By leveraging the power of deep learning algorithms and satellite imagery, the system can quickly and accurately identify tropical cyclones, track their movement, and forecast their intensity, allowing for more timely and effective emergency response measures. This has the potential to save lives and reduce the damage caused by these devastating natural disasters. As technology continues to advance, the use of deep learning algorithms in weather forecasting and disaster management is likely to become increasingly prevalent. In the future, the system could be enhanced by incorporating additional data sources, such as atmospheric data, real-time data feeds, and data analytics to provide more accurate and up-to-date information on the trajectory and intensity of tropical cyclones. Additionally, the development of more sophisticated deep learning algorithms could improve the system's ability to predict the path and intensity of tropical cyclones. A mobile application could also be developed to provide users with real-time tropical cyclone data and alerts on their smartphones, while expanding the system's geographical coverage could make it more accessible and user-friendly. These enhancements could significantly improve the effectiveness and usability of the system, helping to mitigate the damage caused by tropical cyclones and saving lives in the process.

11. REFERENCES

[1] K. A. Emanuel and D. Nolan, “Tropical cyclone activity and globalclimate,” in Proc. 26th Conf. Hurricanes Tropical Meteorol., Miami,FL, USA, 2004, pp. 240–241.

[2] J. P. Kossin, K. R. Knapp, T. L. Olander, and C. S. Velden, “Globalincrease in major tropical cyclone exceedance probability over thepast four decades,” Proc. Nat. Acad. Sci. USA, vol. 117, no. 22, pp. 11975–11980, Jun. 2020.

[3] R. S. Pandey and Y.-A. Liou, “Decadal behaviors of tropical storm tracksin the North West Pacific Ocean,” Atmos. Res., vol. 246, Dec. 2020, Art. no. 105143.

[4] Y.-S. Lee, Y.-A. Liou, J.-C. Liu, C.-T. Chiang, and K.-D. Yeh, “Formation of winter supertyphoons Haiyan (2013) and Hagupit (2014)through interactions with cold fronts as observed by multifunctional transport satellite,” IEEE Trans. Geosci. Remote Sens., vol. 55, no. 7, pp. 3800–3809, Jul. 2017.

[5] Y.-A. Liou, J.-C. Liu, C. P. Liu, and C.-C. Liu, “Season-dependent distributions and profiles of seven super-typhoons (2014) in the Northwestern Pacific Ocean from satellite cloud images,” IEEE Trans. Geosci. Remote Sens., vol. 56, no. 5, pp. 2949–2957, May 2018.

[6] C. W. Landsea, G. A. Vecchi, L. Bengtsson, and T. R. Knutson, “Impact of duration thresholds on Atlantic tropical cyclone counts,” J. Climate, vol. 23, no. 10, pp. 2508–2519, May 2010.

[7] J. A. Leese, C. S. Clark, and C. S. Novak, “An automated technique for obtaining cloud motion from geosynchronous satellite data using crosscorrelation,” J. Appl. Meteorol., vol. 10, no. 1, pp. 118–132, Feb. 1971.

[8] J. Schmetz, “Cloud motion winds from METEOSAT: Performanceof an operational system,” Global Planet. Change, vol. 4, nos. 1–3, pp. 151–156, Jul. 1991.

[9] W. L. Woodley, C. G. Griffith, J. S. Griffin, and S. C. Stromatt, “The inference of GATE convective rainfall from SMS-1 imagery,” J. Appl. Meteorol., vol. 19, no. 4, pp. 388–408, Apr. 1980.

[10] M. Williams and R. A. Houze, “Satellite-observed characteristics of winter Monsoon cloud clusters,” Monthly Weather Rev., vol. 115, no. 2, p. 505–519, Feb. 1987.

[11] Y. Arnaud, M. Desbois, and J. Maizi, “Automatic tracking and characterization of African convective systems on Meteosate